PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

co-LODEX: A COOPERATIVE EXPERT SYSTEM FOR LOGIC DESIGN

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

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

ABSTRACT

Distributed artificial intelligence (DAI) deals with cooperative solutions by distributed agents. Motivated by the conventional distinction between datapath and control design, we are developing a DAI system for VLSI logic design. Assumption-based reasoning is used to automate logic design, which inevitably involves tentative decisions.

Our cooperative expert system for logic design, co-LODEX, features DAI and assumption-based reasoning. co-LODEX consists of three distributed agents, two of which are for datapath and control design. Both agents iterate the refine-evaluate-redesign cycle, under the constraints of area and time. An agent unable to clear a constraint asks the other to make a change. That is, the agents cooperate in an attempt to satisfy all given constraints.

We treat design decisions as assumptions because they are sometimes tentative. We consider redesign to be a contradiction resolution because we view constraint violations as contradictions. When a constraint violation is detected, the redesign mechanism, which is based on assumption-based reasoning, is invoked. Assumption-based reasoning is more important when cooperation is involved, because decisions must be retracted. Justifications for constraint violations, called nogood justifications (NJs), play a central role in redesign. co-LODEX implements redesigns by expanding and generating NJs in a hierarchy that represents the circuit being designed.

1 INTRODUCTION

CAD systems that can produce quality designs quickly are needed for the expanding VLSI market. Although rule-based expert systems have great potential, they are still inferior to experienced designers. One of the most pressing problems is the lack of a means to integrate different types of knowledge and the iterative design cycle.

Distributed artificial intelligence (DAI) deals with cooperative solutions by distributed agents (Smith 1985). It is particularly effective for problems without a global goal. Logic design involves mutually conflicting criteria such as area and time, which is where DAI systems for logic design come in. Previous attempts at this approach include ULYSSES (Bushnell and Director 1986) and the work of Brewer and Gajski (1986).

ULYSSES is a design environment which uses existing CAD tools as knowledge sources (KSs). KSs communicate through files in a global database, which is called the blackboard. ULYSSES is a realistic approach to integrating existing tools, but it cannot be used to automate the iterative design cycle.

Brewer and Gajski's view is that the design at a high level becomes a specification for the lower levels, and they propose a design based on a set of communicating expert systems, each of which corresponds to a different level of abstraction. Although there are two directions of communication -- down for constraint propagation and up for failure reporting -- design is limited to a single stream and thus is not flexible enough.

We use two streams of design: datapath and control. Datapath design begins with a block diagram. Each component is designed hierarchically using either the top-down or bottom-up method. That is, component and subcomponents are split up into subsubcomponents. Control design begins with a behavioral specification. It then establishes finite-state machines conforming to the specification, represents them with flip-flops, and designs circuits that generate the control signals. Design progresses with these two streams interacting. The cooperative expert system for logic design we propose, co-LODEX, contains two agents which correspond to datapath and control design. Both agents iterate the refine-evaluate-redesign cycle, under area and time constraints. An agent unable to clear a constraint asks the other to make a change. When requested, an agent tries making changes to coopera-

tively satisfy all given constraints.

The consequences of a design decision are not always clear when made. Later evaluation may show that the decision was incorrect and must be retracted. The design does not progress until a decision is made, and the best alternative at the time is usually selected. With the results of the decision added to the design, the next decision is made. When the design can be evaluated, it is done so against the constraints.

Assumption-based reasoning uses both facts and assumptions that can be retracted (de Kleer 1986). Justification, originally introduced for truth maintenance (Doyle 1979), is the key to manipulating information containing assumptions. Since justification is a logical concept, the foundations of assumption-based reasoning are given in logic (Reiter and de Kleer 1987). Finger and Genesereth (1985) propose a direct application of logic to design.

We incorporate assumption-based reasoning into co-LODEX for three reasons:
(1) Design decisions are sometimes tentative.
(2) Some decisions must be retracted during cooperation.
(3) Constraints are also subject to change.
Decisions and constraints are treated as assumptions in co-LODEX. We think of redesign as contradiction resolution by considering a constraint violation to be a contradiction. When a constraint violation is detected during evaluation, the redesign mechanism is invoked. Justifications for constraint violations, called nogood justifications (NJs), play a central role in redesign. They are conjunctions of assumptions and conditions about area or time. co-LODEX redesigns by expanding and generating NJs in the hierarchy representing the circuit under design.

The next section gives an overview of co-LODEX, focusing on its CAD aspects. Section 3 discusses its DAI aspects and distributed agents. Section 4 discusses the redesign mechanism. Section 5 gives our current status and conclusions.

2 co-LODEX OVERVIEW

co-LODEX inputs a behavioral specification and generates a collection of CMOS standard cells. It also accepts global constraints on area and time. Design is done in two interactive streams -- datapath and control. After combining their results, co-LODEX optimizes and outputs a CMOS standard cell circuit description.

The specification language for behavior used in co-LODEX is an extension of DDL (Duley and Dietmeyer 1968), and is based on temporal logic (Moszkowski 1986). Figure 1 shows the specification for the greatest common divisor (Camposano 1987).

```
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 of behavioral specification

Two intervals, idle and loop, have counterparts in DDL states, but are not limited to one clock cycle. STOP(rst=0) means that interval idle is finished when rst equals 0. <- means register transfer and := means terminal connection. The rest is self-explanatory.

The user can enter a block diagram of the datapath on which co-LODEX will base its design. One possible datapath for Figure 1 is shown in Figure 2. SUB is a subtracter and MUX is a multiplexer.

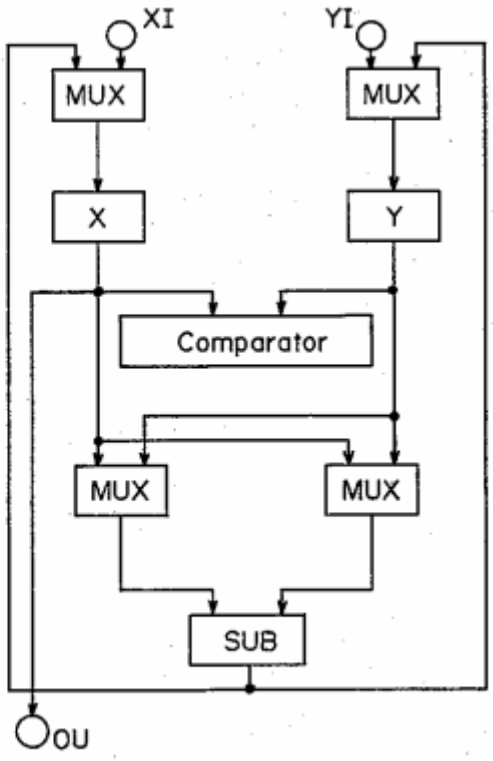

Figure 2 Block diagram

Constraints on area are expressed as inequalities in the basic cell count, for example, "The total basic cell count must not exceed 1300." Constraints on time are expressed as inequalities in the delay or clock cycle. Constraints themselves are subject to change, because design is exploratory and depends on the actual circumstances. The designer might want to explore other possibilities by tightening or relaxing certain constraints. co-LODEX lets the user add, retract, or restore constraints by storing all given constraints. When the constraints change, it provides another solution by redesigning only the portions of the circuit affected by the change. This gives fast turnaround.

Figure 3 shows the co-LODEX configuration. There are three agents: control design, datapath design, and the user interface. The items in the figure are detailed in subsequent sections.

The datapath design agent produces the whole datapath as a partial solution, with each component implemented using CMOS standard cells. The control design agent designs as another partial solution circuits that control the flow of execution and each component of the datapath. These two partial solutions are linked at the control terminals as shown in Figure 4.

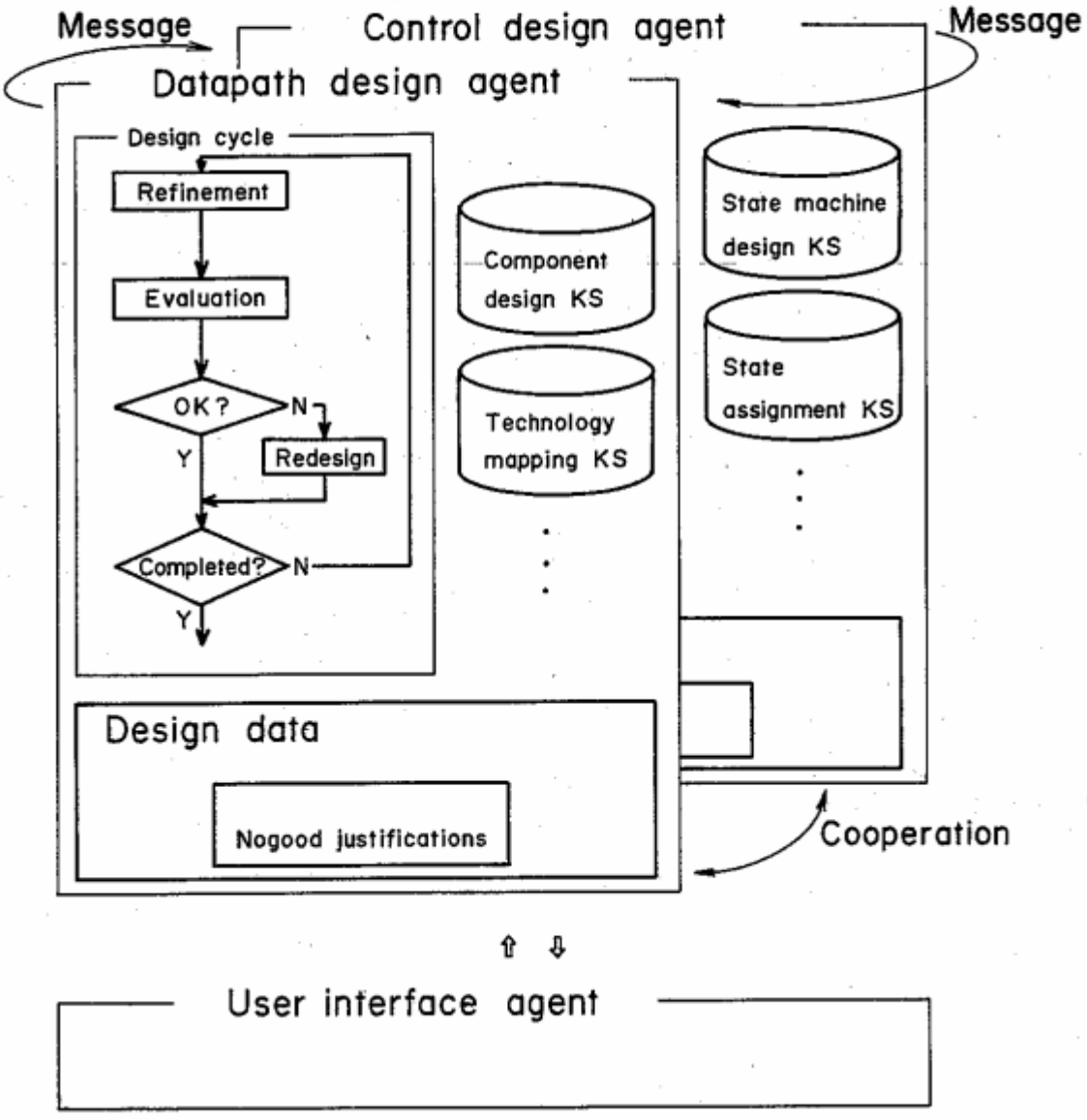

Figure 3 co-LODEX configuration

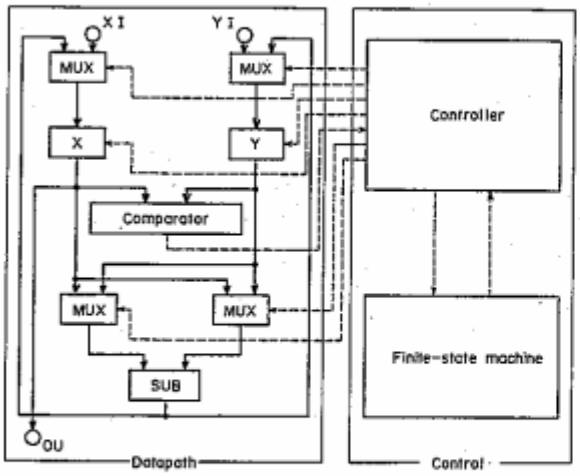

Figure 4 Linkage between datapath and control

Figure 5 diagrams one of the results. The only constraint was that the total basic cell count not exceed 1400. The design results of the greatest common divisor are varied by making constraints stronger or weaker; the datapath is replaced automatically (Fig. 6). The process began with the point farthest to the right. Different results were achieved as the area constraint got stronger. After the smallest circuit was designed, the time constraint was strengthened and the area constraint weakened. It ended up back at the first design, the fastest.

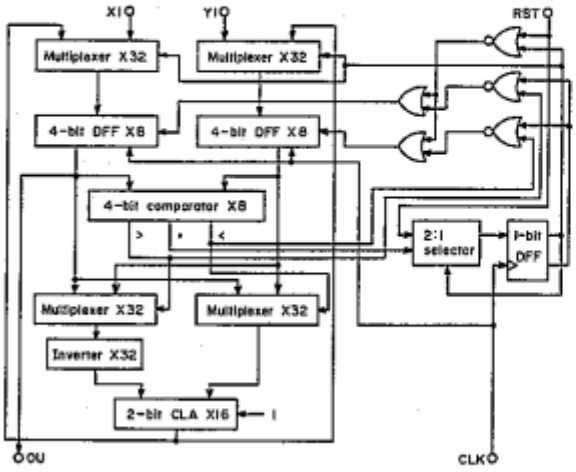

Figure 5 Example of design result

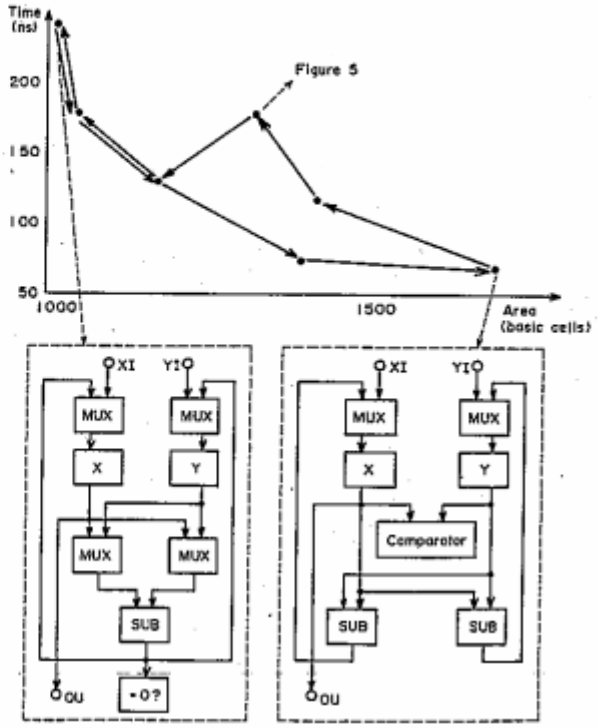

Figure 6 Alternative constraint designs

3 COOPERATIVE DISTRIBUTED AGENTS

Logic design is evaluated based on conflicting criteria, e.g., area and time, and there is no global goal. This means that it is impossible to serialize designs which conform to the specification. Instead, the designer must come up with a design within the allowed region.

Each distributed agent of co-LODEX produces a partial solution on its own, based on the relevant global constraints. These partial solutions are combined to form the complete design. Any adjustments still needed are made by cooperation between the agents.

3.1 Communication

The two agents influence each other as follows. If a delay constraint is so strict that the components on a path cannot satisfy it, the control design agent asks the datapath design agent to redesign some of those components.

If the datapath design agent cannot make a

design fast enough to satisfy the clock cycle constraint, it asks the control design agent to either give up the current path or to break the operation into suboperations.

If one of the duplicate components must be removed because of the area constraint, the datapath design agent asks the control design agent to change control so that the new set of components satisfies all performance requirements.

To cooperate, the two agents exchange four types of information (Fig. 7).

1. A request for change is issued when the datapath design agent is unable to satisfy a constraint. The control design agent decides what to change.
2. The control design agent is informed of an alternative datapath when the datapath design agent selects it in place of the current datapath.
3. Since the control design agent determines the timing of each operation by establishing finite-state machines conforming to the specification, it generates internal time constraints. Though it may seem logical that the control design agent should evaluate the design, this is difficult because it involves the transfer of a large amount of data. Instead, the control design agent sends the internal time constraints to the datapath design agent for evaluation.
4. The new datapath is sent to the datapath design agent, that has been made possible by change of control.

One agent comes up with a partial solution and makes the other check it. In the meantime, the agent proceeds to the next task. When the check is successful, the partial solution will be considered valid until contradiction occurs. If the check is not successful, the agent comes back and tries to find another solution. For example, the control design agent can tell the datapath design agent to evaluate "critical paths" to determine as soon as possible whether its design is feasible, while it begins to implement it. A negative result comes as an urgent message, which causes a different control.

3.2 Datapath Design Agent

The datapath design agent's purpose is to design all components forming the datapath. This is done hierarchically, under constraints on basic cell count and delay or clock cycle. Any synthesis tool can be used, ranging from rule-based KSs to algorithms. Figure 8 shows a rule example for designing an n-bit subtracter with an n-bit adder and an n-bit one's complement. In the last stage, the CMOS standard cell library must be referenced.

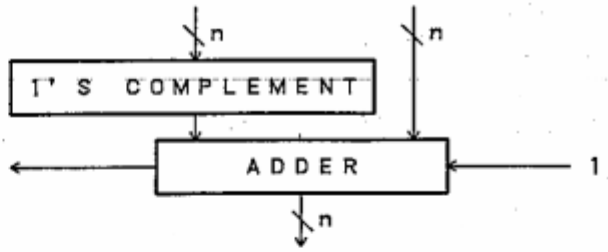

Figure 8 Rule for subtracter in component design KS

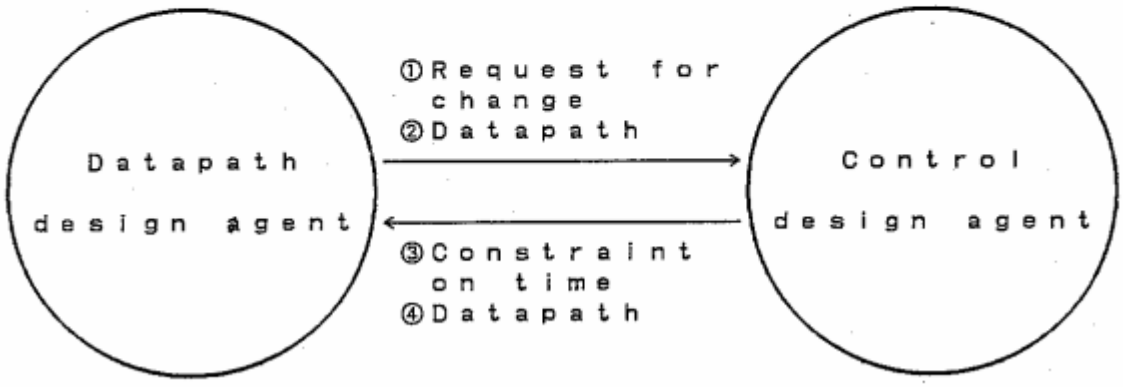

Figure 7 Communication

Figure 9 shows an example of mapping a 4n-bit adder into n 4-bit carry-lookahead-adder (CLA) cells.

3.3 Control Design Agent

The control design agent establishes finite-state machines conforming to the specification. It breaks each interval into states. Usually, it would be best to have as few states as possible but, when there are strong area constraints, time slicing is required because of the limited components. At this point, every detail of synchronous timing is determined. The control design agent generates internal time constraints by examining which path must be covered within one clock cycle. It sends these constraints to the datapath design agent. It then constructs the finite-state machines with flip-flops and designs circuits that generate the control signals. As for the datapath design agent, any synthesis tool can be used.

3.4 User Interface Agent

The user interface agent is responsible for transferring the following:

1. The behavioral specifications are input as text.
2. The user can enter a block diagram of the datapath through the graphics editor of the user interface agent.
3. The user interface agent provides a menu-driven dedicated window for specifying constraints as inequalities. It also enables the user to change them by storing all given constraints.
4. The user can design or modify any component through the graphics editor of the user interface agent and force co-LODEX to use it.

There should be cooperation between the user and co-LODEX through the user interface agent. The user can explore several possibilities by strengthening or relaxing constraints, and by designing some of the components.

4 REDESIGN USING ASSUMPTION-BASED REASONING

As mentioned earlier, design decisions are regarded as assumptions. The assumption-based truth maintenance system (ATMS) (de Kleer 1986) enumerates all assumptions in advance and examines all combinations. In design, however, we are not interested in all combinations because a decision's importance depends on decisions made earlier. In Fig. 2, for example, how to construct an adder is unimportant if the subtracter is designed without using adders.

The area a circuit requires and its delay are the sum of their constituent parts. The delay, for example, can be attributed to that of the components along its path. This fact lets us break a global condition into local conditions. Hierarchical structure is useful for this.

We propose a redesign mechanism based on NJs. They are represented as conjunctions of assumptions and conditions. co-LODEX redesigns by expanding and generating NJs in the hierarchy representing the circuit under design.

4.1 Hierarchical Design Description

Design objects are represented in a hierarchy. Figure 10 shows part of the hierarchy corresponding to Fig. 2. There are two types of nodes: component nodes (ovals) and alternative nodes (rectangles). A component node associates alternative nodes as possibilities of implementation. There is a special component node called the datapath node that corresponds to the whole datapath. An alternative node contains information about the connection between subcomponents and has the subcomponent nodes as children. An alternative is called either *in* or *out* based on whether it is adopted or discarded. Each component node has at most one *in* alternative node. Other alternative nodes are stored in the *out* alternative list to be recalled later, if necessary. Figure 10, which shows only *in* alternative nodes, means the following: The subtracter consists of an adder and a one's complement (Fig. 8), and the 32-bit adder consists of eight 4-bit CLA cells connected serially (Fig. 9). The current datapath, DATAPATH1, is shown in Figure 2. Current *out* alternatives might include a serial connection of 16 2-bit CLA cells and a serial connection of 32 single-bit adder cells.

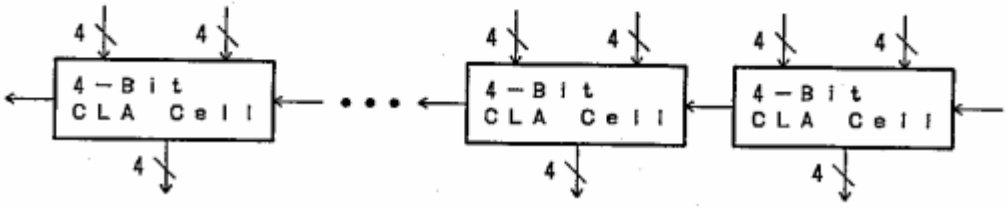

Figure 9 Mapping a 4n-bit adder into n 4-bit CLA cells

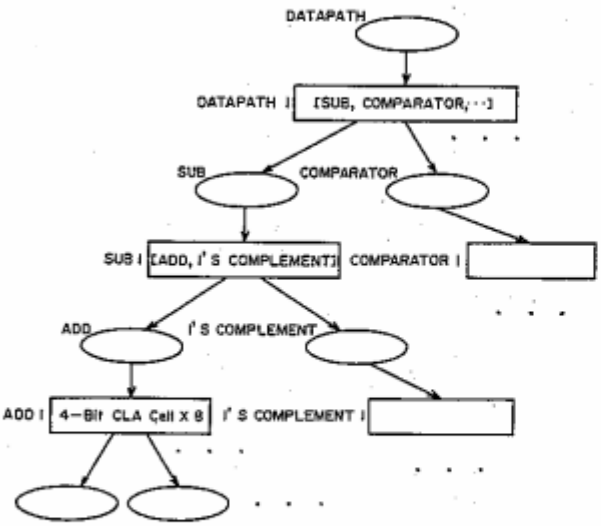

Figure 10 Hierarchical design description

4.2 Nogood Justification (NJ)

An NJ is a logical expression that must not hold during design. Satisfying an NJ means a constraint violation and invokes the redesign mechanism. NJs have conjunctive forms:

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.

There are also default NJs. 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 Fig. 10:

(# of basic cells $\leq$ 1300) & DATAPATH1
& (SUB + COMPARATOR + ... > 1300) (1)

The first conjunct is a constraint, the second an alternative, and the third a condition about the total sum of the basic cells of the components. The above NJ is the same as the original constraint in that any design violating the constraint in DATAPATH1 satisfies it. Default NJs let us reduce the amount of evaluation.

4.3 NJ Expansion

NJ expansion is used to narrow the scope down to resolve contradictions. NJ expansion is defined as follows:

1. Removing the component's contribution from every condition with the component in it
2. Making an NJ the logical product of the conjunction obtained above, the component's *in* alternative, and the subcomponents' conditions
3. Putting the resulting NJ at the alternative node of the component

For example, expanding (1) for SUB (Fig. 10) may give us the following NJ at SUB1:

(# of basic cells $\leq$ 1300) & DATAPATH1
& (COMPARATOR + ... > 868) & SUB1
& (ADD + 1'S-COMPLEMENT $\geq$ 432) (2)

This happens when the design exceeds 1300 basic cells and the subtracter is selected as the part of the circuit to be changed. Expanding (2) for ADD will give us the following NJ at ADD1; the 4-bit CLA cell consists of 50 basic cells:

(# of basic cells $\leq$ 1300) & DATAPATH1
& (COMPARATOR + ... > 868) & SUB1
& (1'S-COMPLEMENT $\geq$ 32) & ADD1 (3)

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

(# of basic cells $\leq$ 1300) & DATAPATH1
& (COMPARATOR + ... > 868) & SUB1
& ADD1 (4)

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

(# of basic cells $\leq$ 1300) & DATAPATH1
& (COMPARATOR + ... > 868) & SUB1

4.4 NJ Generation

If every alternative of a component causes a constraint violation, NJ generation enables us to get an NJ with no reference to the component from a set of NJs, which contain each alternative as a conjunct. If, for example, there are only alternatives a, b, and c for component X, and we have NJs, A & a, B & b, and C & c, then NJ A & B & C will be generated at the alternative node of X's parent node. This procedure is justified by resolution (Robinson 1965). The generated NJ suggests that a component to be changed be selected among those it references.

Suppose, in addition to (4), we have NJs 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 $\leq$ 1300) & DATAPATH1
& (COMPARATOR + ... > 1012) & SUB1
& ADD2 (5)
(# of basic cells $\leq$ 1300) & DATAPATH1
& (COMPARATOR + ... > 1012) & SUB1
& ADD3 (6)

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

(# of basic cells $\leq$ 1300) & DATAPATH1
& (COMPARATOR + ... > 1012) & SUB1 (7)

This shows that the design in Fig. 8 is not possible under the circumstances specified by the first three conjuncts. co-LODEX would have to change either the subtracter or the circumstances.

4.5 Redesign Algorithm

The redesign algorithm uses NJ expansion and generation. Redesign is invoked when a default NJ turns out to be true. The redesign algorithm begins with the NJ, from the alternative node where the NJ resides.

Step 1: If this alternative should be kept intact and one of the subcomponents should assume the responsibility, the algorithm selects it, expands the NJ for it, goes down the hierarchy one level, and executes Step 1 again.

Step 2: If there is an alternative available that makes no NJs at the ancestor nodes (including itself) true, it modifies the current alternative and exits.

Step 3: It generates an NJ and goes up the hierarchy one level. If the NJ generated contains only constraints (fail!), it issues a request for change to the other agents. Otherwise, it returns to Step 1. □

The decision to replace an alternative and which subcomponent to select in Step 1 is based on heuristics which are separated from the above algorithm. An example of 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 part of the design process. It is important for the user to be able to adjust 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 NJ 7. If the user comes up with a smaller adder and enters it in the system, the system adopts it as an *in* alternative and makes the assumption false. It means NJ 7 is not satisfied and the subtracter is restored to the way it was before.

5 CONCLUSION

We are implementing co-LODEX on personal sequential inference machines (PSIs) in ESP (Chikayama 1984). Our work focuses on the cooperation between distributed agents (datapath design and control design agents) and the redesign mechanism using assumption-based reasoning. Possible extensions include addition of an agent for DFT (design for testability) and to have agents responsible for each partition of the whole circuit. Assumption-based reasoning proves to be more effective when used for DAI. It is also important to work on individual CAD tools, such as logic minimization and synthesis which consider delays.

ACKNOWLEDGMENTS

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

REFERENCES

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

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

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

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

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

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

(Duley and Dietmeyer 1968) 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 1986) Moszkowski, B., "Executing Temporal Logic Programs" Cambridge University Press (1986).

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

(Robinson 1965) 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 1984) Chikayama, T., "Unique Features of ESP" Proc. of FGCS'84, pp. 292-298 (1984).