ICOT Technical Report: TR-567

TR-567

A Cooperative Logic Design Expert System on a Multiprocessor

by
Y. Minoda, F. Maruyama, S. Sawada,
Y. Takizawa & N. Kawato (Fujitsu)

July, 1990

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

A Cooperative Logic Design Expert System on a Multiprocessor

Yoriko Minoda, Fumihiro Maruyama, Shuho Sawada,
Yuka Takizawa, and Nobuaki Kawato
FUJITSU LIMITED
Kamikodanaka, Nakahara-ku, Kawasaki 211, Japan
phone: (044) 777-1111
E-mail: pro114@flab.fujitsu.co.jp

Area: Expert systems

Keywords: logic design, assumption-based reasoning, cooperation, constraint

Abstract

CAD systems that can quickly produce quality designs are needed for the expanding VLSI market. This paper presents a cooperative design mechanism in a cooperative logic design expert system on a multiprocessor, co-LODEX. co-LODEX accepts constraints on area and speed, and outputs a CMOS standard cell netlist that satisfies the constraints. Short turnaround is expected through the combination of parallel processing by several processors and their cooperation.

The cooperative design mechanism is based on an evaluation-redesign mechanism using assumption-based reasoning within a single processor. Design alternatives are considered as assumptions and constraint violations as contradictions. Redesign is implemented as contradiction resolution. The evaluate-redesign cycle repeats itself until the design satisfies the specified constraints.

co-LODEX divides the whole circuit to be designed into subcircuits in advance and designs each subcircuit on each processor to take advantage of parallel processing. Global evaluation-redesign takes place by processors exchanging design results in terms of gate counts and delays (in case of success) or justifications for constraint violations (in case of failure).

Experimental results on the cooperative design algorithm suggest that the number of iterations is considerably reduced. It is observed that the number of iterations through cooperation is less than half that for the sequential evaluation-redesign algorithm.

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 iterate evaluate-redesign cycle until the design satisfies all given constraints. Without it, it would be impossible to design a quality circuit with the desired characteristics by looking at the design from a global point of view.

Turnaround time seems to be another key issue. Short turnaround allows designers to rapidly implement a variety of architectural choices and choose the solution best suited for their specific situation by comparing area and speed characteristics. Designers can thus explore their options in a way that has not been practical before.

Since design decisions may be retracted after later evaluation, they can be thought of as assumptions. 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 concept to manipulating information containing assumptions. In de Kleer's Assumption-based Truth Maintenance System (ATMS), all assumptions are enumerated in advance and all combinations are examined. In design, however, we are not interested in all combinations. This is because a decision's importance depends on the decisions made earlier. We can prune a considerable number of combinations.

Finger and Genesereth propose a new approach to deductive design synthesis, the Residue

Approach, in which designs are represented as sets of constraints [Finger and Genesereth 1985]. Since we are interested in the characteristics (area and speed) of the resulting circuit, we can think of constraints regarding the above characteristics.

We proposed an evaluation-redesign mechanism using assumption-based reasoning [Maruyama 1988]. In our evaluation-redesign mechanism, design alternatives are considered as assumptions and constraint violations as contradictions. Redesign is implemented as contradiction resolution. Justifications for violations, called nogood justifications (NJs), play a central role in the mechanism.

In this paper, we present a cooperative logic design expert system on a multiprocessor, co-LODEX. co-LODEX divides the whole circuit to be designed into subcircuits in advance and designs each subcircuit on each processor to exploit parallel processing. Global evaluate-redesign takes place by processors exchanging design results (in case of success) or NJs (in case of failure). A cooperative design algorithm makes this possible. Short turnaround is expected through the combination of parallel processing by several processors and their cooperation.

The next section gives an overview of co-LODEX. Section 3 describes its evaluation-redesign mechanism within an agent or a processor. Section 4 discusses its cooperative design algorithm among agents. We give some experimental results in Section 5 and concluding remarks in Section 6.

2 co-LODEX Overview

2.1 Inputs and Outputs

The user specifies a behavioral specification, a block diagram of the datapath, and constraints on area or speed. co-LODEX outputs a CMOS standard cell netlist that satisfies all given constraints.

The specification language for behavior used in co-LODEX is UHDL [Fujisawa 1989], an extension of DDL [Duley and Dietmeyer 1969]. Figure 1 shows the specification for a circuit that calculates the greatest common divisor (GCD) between two integers using the Euclidian algorithm [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 rest equals 0. <- means register transfer and := means terminal connection. The rest is self-explanatory.

A block diagram of the datapath is shown in Figure 2. The boxes signify functional blocks. COMP, SUB, 2CPL, MUX, and X and Y are a comparator, a subtracter, a two's-complement, a multiplexer, and registers. The triangles signify input/output buffers.

Constraints on area are expressed as inequalities in the gate count, for example, "The total gate count must not exceed 1400." The user can specify as an area constraint the maximum gate count that could be squeezed into a given LSI device. Constraints on time are expressed as inequalities in the propagation delay, for example, "The maximum delay must not be longer than 200 ns." The user can specify as a timing constraint the clock cycle the LSI device should operate with.

The resulting netlist can be input to an automatic place-and-route system.

2.2 Brief Overview

Figure 3 gives an overview of co-LODEX. Each agent is given one of the subcircuits of the whole circuit. Figure 4 shows the six subcircuits for the GCD example and the agents in charge. It should be noted that the control circuit, CTRL, is included. co-LODEX establishes a finite-state machine from the behavioral specification and extracts the specifications for the control circuit in terms of logical expressions. It then divides the whole circuit so that the blocks along critical path candidates are distributed to as few agents as possible.

Each agent designs given functional blocks hierarchically using the top-down method. It keeps splitting up functional block and subblocks into sub-subblocks until all given blocks are implemented with CMOS standard cells. This is done by referring to the library that includes knowledge about functional block design, knowledge about technology mapping, and standard cells data. Then it counts the number of gates and estimates delays for evaluating the implemented circuit against constraints on area and time.

An agent usually designs its subcircuit independently and in parallel with the other agents. However, since the design results of the other agents are necessary for evaluation against global constraints, agents exchange their results every time they finish design/redesign. An agent redesigns when it detects a constraint violation for which it is responsible, for example when a path passing through it is too slow. If it designs a standard cell netlist that satisfies all the local constraints specified by stored NJs, it notifies the resulting gate count and delays. If it cannot, it notifies an NJ.

3 Redesign Mechanism Using Assumption-based Reasoning

As mentioned earlier, we regard design decisions as assumptions. ATMS 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 Figure 2, for example, how to construct an adder is unimportant if the subtracter is designed without using adders. Moreover, we are interested in the characteristics (area and speed) of the resulting circuit.

The area a circuit requires and its delay are the sum of their constituent parts. The delay of a path, for example, can be attributed to that of the components along it. This fact lets us break a global condition into local conditions. A hierarchical structure is useful for this. We explain a redesign mechanism using assumption-based reasoning, which operates on a hierarchical design description.

3.1 Hierarchical Design Description

Design objects are represented in a hierarchy. Figure 5 shows part of the hierarchy corresponding to Figure 4. There are three types of nodes; agent nodes (capsules), component nodes (ovals) and alternative nodes (rectangles). An agent node is responsible for one or more component nodes. A component node associates alternative nodes as possibilities of implementation. There is a special component node called the chip node that corresponds to the whole chip. 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 5, which shows only *in* alternative nodes, means the following:

·The whole chip (Chip) consists of a control circuit (CTRL), a comparator (COMP), a subtracter (SUB), a multiplexer (MUX), a register (Y), and other parts.

·Agent3 is responsible for a subtracter; other agents are responsible for the respective components.

·SUB consists of an adder (ADD) and a one's-complement (1CPL).

·ADD, the 32 bit adder, consists of eight 4-bit CLA (carry-lookahead adder) cells connected serially. Current *out* alternatives might include a serial connection of 16 2-bit CLA adder cells and 32 single-bit adder cells.

3.2 Justifications for Constraint Violations (NJs)

An NJ (nogood justification) is a logical expression that must not hold during design. Satisfying an NJ means a constraint violation and invokes the redesign mechanism.

The following default NJ at Chip (in Figure 5) is equivalent to the original constraint on gate count in that any design violating the constraint satisfies it.

$$\mathrm{CTRL} + \mathrm{COMP} + \mathrm{SUB} + \mathrm{2CPL} + \mathrm{MUX} + X + \mathrm{MUX} + Y > \mathrm{CHIP} \qquad (1)$$

This says that if the total gate count of the control circuit,the comparator, the subtracter, and so on, exceeds the value of variable CHIP, it means a constraint violation. CHIP is the variable that refers to the currently valid constraint value on gate count, for example 1400. co-LODEX transforms each constraint specified by the designer into default NJs.

A timing constraint in terms of the clock cycle is transformed into a set of default NJs, among which is an inequality representing that the sum of the delays of the components along a path from source to destination exceeds the constraint value. For example, one of the default NJs, the path from Y via SUB, 2CPL, MUX, back to Y is longer than the clock cycle. It is as follows:

$$Y(P1) + \mathrm{SUB}(P2) + \mathrm{2CPL}(P1) + \mathrm{MUX}(P2) + Y(P2) > \mathrm{CLOCK} \qquad (2)$$

The form, "component (P number)", represents a path within each component. CLOCK is the variable that refers to the currently valid constraint value on clock cycle, for example 200.

Starting from default NJs, new NJs are added during redesign through NJ expansion and generation. NJs save us doing direct evaluation against constraints. All we have to do is to check to see if any NJ is satisfied.

3.3 NJ Expansion

NJ expansion is used to narrow the scope and go down the hierarchy to resolve contradictions, or constraint violations. Formally, NJ expansion is defined in the following three steps. The NJ to be expanded is one that is satisfied at the moment.

Step 1: Select a component appearing in the NJ to be expanded. Call it C.

Step 2: Replace C in the NJ with its *in* alternative's subcomponents. If the *in* alternative is at the leaf of the hierarchical structure (at the standard cell level), replace C with its actual gate count or its delay value.

Step 3:Go down the hierarchy to the alternative node and store the NJ obtained in Step 2.

(End)

An example of heuristics for selecting a component in Step 1 would be to select the largest or the slowest component, the one that is the most responsible.

Suppose default NJ (1) turns out to be true. That is, a constraint violation on gate count has been detected. In NJ expansion, the largest component, the subtracter, is selected in (1) and is replaced with its *in* alternative's subcomponents, the adder and the one's-complement.

$$\mathrm{CTRL} + \mathrm{COMP} + \underline{\mathrm{ADD} + \mathrm{1CPL}} + \mathrm{2CPL} + \mathrm{MUX} + \mathrm{X} + \mathrm{MUX} + \mathrm{Y} > \mathrm{CHIP} \qquad (3)$$

(3) is put at SUB's *in* alternative node, SUB1. Again, the largest component between the adder and the one's-complement, the adder, is selected. Since it is implemented with standard cells, eight 4-bit CLA cells, variable ADD is replaced with 400, which is the actual gate count for the adder.

$$\mathrm{CTRL} + \mathrm{COMP} + \underline{400} + \mathrm{1CPL} + \mathrm{2CPL} + \mathrm{MUX} + \mathrm{X} + \mathrm{MUX} + \mathrm{Y} > \mathrm{CHIP} \qquad (4)$$

(4) is put at ADD1. Then, alternative ADD1 is going to be changed to another alternative. (4) is a condition under which constructing a 32-bit adder with eight 4-bit CLA cells is inhibited.

3.4 NJ Generation

If every alternative of a component causes a constraint violation, NJ generation enables us to get a new NJ, the logical product of the NJs corresponding to each alternative. The generated NJ does not refer to that component. It is put at the alternative node one level up. This procedure is justified by resolution [Robinson 1965].

In the above example, suppose neither ADD2 (with 2-bit CLA cells: 256 gates) nor ADD3 (with 1-bit adder cells: 256 gates) satisfies the gate count constraint. The following two NJs should be put at ADD2 and ADD3, respectively:

$$\mathrm{CTRL} + \mathrm{COMP} + \underline{256} + \mathrm{1CPL} + \mathrm{2CPL} + \mathrm{MUX} + \mathrm{X} + \mathrm{MUX} + \mathrm{Y} > \mathrm{CHIP} \qquad (5)$$

$$\mathrm{CTRL} + \mathrm{COMP} + \underline{256} + \mathrm{1CPL} + \mathrm{2CPL} + \mathrm{MUX} + \mathrm{X} + \mathrm{MUX} + \mathrm{Y} > \mathrm{CHIP} \qquad (6)$$

The gate counts of ADD2 and ADD3 happen to be the same. If no other alternative is available, NJ generation gives us a new NJ from (4), (5) and (6):

$$\mathrm{CTRL} + \mathrm{COMP} + \underline{256} + \mathrm{1CPL} + \mathrm{2CPL} + \mathrm{MUX} + \mathrm{X} + \mathrm{MUX} + \mathrm{Y} > \mathrm{CHIP} \qquad (7)$$

(7) is put at SUB1. If no more alternatives are available for the one's-complement, it is replaced with its actual cell count, 32, and we have the following NJ:

$$\mathrm{CTRL} + \mathrm{COMP} + \underline{288} + \mathrm{2CPL} + \mathrm{MUX} + \mathrm{X} + \mathrm{MUX} + \mathrm{Y} > \mathrm{CHIP} \qquad (8)$$

(8) is put at SUB1. This shows that alternative SUB1 is not possible under the circumstances specified by this NJ. We would have to change the subtracter. (8) shows that constructing a subtracter with an adder and a one's-complement requires at least 288 gates.

Although only an example of NJ generation from NJs about gate count only was shown, the generated NJ, in general, is a logical product of NJs about gate count and NJs about delay.

3.5 Evaluation-Redesign Algorithm within Each Agent

The redesign algorithm within each agent uses NJ expansion and generation. Redesign is invoked when an NJ turns out to be true.

Step 1: Set ALT to the agent node and proceed to Step 2.

Step 2: Check to see if there is any satisfied NJ at the ancestor alternative nodes (including itself) of ALT. If so, set ALT to the alternative node where the satisfied NJ is put, and proceed to Step 3. Otherwise, go to Step 7.

Step 3: If there is a subcomponent of ALT appearing in the NJ, proceed to Step 4. Otherwise, go to Step 5.

Step 4: Expand the NJ. Set ALT to the current alternative node and return to Step 3.

Step 5: Make ALT *out*. Select another alternative node that is not inhibited by an NJ, make it *in*, set ALT to it, and go to Step 2. If every alternative is inhibited by NJs, proceed to Step 6.

Step 6: Generate an NJ. Set ALT to the current alternative node and go to Step 3. If there is no alternative node one level up, output the generated NJ and exit (Fail!).

Step 7: If there is no component node whose alternative nodes are all *out*, exit. (Succeed!). Otherwise, select an alternative node that is not inhibited by NJs, make it *in*, set ALT to it, and go to Step 2.

(End)

In Step 5, selection is done either by recalling an *out* alternative or by generating a new implementation.

The above algorithm starts when an agent receives information from the other agents. Once the algorithm terminates in success or failure, the agent sends information to the other agents.

4 Cooperative Design Mechanism on a Multiprocessor

We propose a cooperative design mechanism on a multiprocessor. It is based on the redesign mechanism within each agent. Moreover, (1) exchanging design results and NJs among agents and (2) combining the NJs received from the other agents are necessary.

Agents exchange the design results (gate counts and delays) of subcircuits when they succeed in design. They exchange the resulting NJs when they fail to design subcircuits without any stored NJ satisfied.

4.1 Combining NJs

When an agent fails in redesign with the evaluation-redesign algorithm described in Section 3.5, it generates an NJ and sends it out to the other agents. Each agent "combines" the NJs received from the other agents and makes a new NJ out of them. Considering an NJ from an agent as a condition where design is impossible for the agent, the combined NJ can be seen as a condition where design is impossible for the agents other than the recipient agent. Agents are required to design without any combined NJ satisfied.

For example, suppose Agent3 received the following two NJs originated from default NJ (2) from Agent4 and Agent6:

$$\text{Agent3(P2)} + 61.1 + \text{Agent6(P2)} > \text{CLOCK} \qquad (9)$$

$$\text{Agent3(P2)} + \text{Agent4(P1)} + 15.6 > \text{CLOCK} \qquad (10)$$

Where Agent6(P2) represents Y(P1) + MUX(P2) + Y(P2). Agent3 combines the above NJs and makes a new NJ:

$$\text{Agent3(P2)} + 76.7 > \text{CLOCK} \qquad (11)$$

(11) is added to Agent3. (11) works as a local constraint imposing that the delay of subtracter's P2 must not be longer than (CLOCK minus 76.7) ns. This is a simple example. Combined NJs are usually logical products.

4.2 Cooperative Design Algorithm

We propose a cooperative design algorithm by describing the procedure for each agent.

Step 1: Design its subcircuit. Repeat redesign by the evaluation-redesign algorithm. The gate counts and delays of the other subcircuits are assumed to be 0. If any agent fails, the algorithm terminates in failure. Otherwise, proceed to Step 2.

Step 2: Exchange the design results, that is the gate counts and delays of the subcircuits, with the other agents. Proceed to Step 3.

Step 3: Set the gate counts and delays of the other subcircuits to the design results received in Step 2. If no stored NJ is satisfied, go to Step 9. If some of the stored NJs are satisfied and the design results of each agent are the same as in the previous cycle (caught in a loop), go to Step 7. Otherwise, proceed to Step 4.

Step 4: Redesign its subcircuit. If at least one agent succeeds in redesign without any stored NJ satisfied, go to Step 2. Otherwise (all agents fail), proceed to Step 5

Step 5: Exchange the generated NJs with the other agents. Proceed to Step 6.

Step 6: Combine the NJs received in Step 5. Go to Step 1.

Step 7: Set a temporary constraint and proceed to Step 8.

Step 8: Design its subcircuit. Repeat redesign by the evaluation-redesign algorithm until all the constraints including the temporary one are met. The gate counts and delays of the other subcircuits are assumed to be 0. If all the agents fail, the algorithm terminates in failure. Otherwise, go to Step 2.

Step 9: Put together all the subcircuits. The algorithm terminates in success.

(End)

Initially only default NJs are stored. As the algorithm proceeds, new generated NJs and combined NJs are added. In Step 7, select one of the violated constraints with the fewest agents related, and set the current value corresponding to that constraint as a temporary constraint.

Once the above algorithm terminates in success or failure (In Step 1, Step 8,and Step 9), the design run is finished, and the user can retry by changing the constraints. The user can look for a faster circuit by tightening the delay constraint, or can rerun by relaxing the constraints in case of failure. When the constraints are changed, the system updates them and re-evaluates by checking all the stored NJs. As more NJs are accumulated, the efficiency of the algorithm is further improved.

5 Experimental Results

We implemented the cooperative portion of co-LODEX on Multi-PSI [Taki 1988] in KL1 [Ueda 1986] to evaluate the performance of the cooperative design mechanism. We had an extra agent that synchronized the six agents in charge of subcircuits and took statistics.

Figure 6 shows some of the results for the GCD example. First, the area constraint was 1400 and the timing constraint was 200. We obtained the circuit shown at the upper left. As the timing constraint was strengthened and the area constraint weakened, different results were achieved. It ended up the fastest circuit of all, shown at the lower right. Figure 7 shows the circuit in detail. When the area constraint is 1550 and the timing constraint is 140, the design fails with NJ, 142.8>CLOCK & 1572>CHIP. This means that design is impossible if the specified set of constraints satisfies the NJ; we must relax either of the constraints so that the above NJ is not true any more.

The primary statistical results are encouraging. Some results suggest that the number of iterations is considerably reduced. It is observed that the number of iterations through cooperation is less than half that for the sequential evaluation-redesign algorithm. The reason seems to be that combining NJs helps narrow the design space.

6 Conclusion

We presented a cooperative logic design expert system on a multiprocessor, co-LODEX. co-LODEX divides the whole circuit to be designed into subcircuits in advance and designs each subcircuit on each processor to take advantage of parallel processing. Global evaluate-redesign takes place by processors exchanging design results or NJs. A cooperative design algorithm based on assumption-based reasoning makes this possible. Short turnaround is expected through the combination of parallel processing by several processors and their cooperation.

We are implementing co-LODEX on Multi-PSI in KL1. Our future plans include working on parallel processing of design, evaluation, and redesign within an agent. It is also important to work on load balancing among processors.

Acknowledgments

This work has been done as part of the Fifth Generation Project of Japan. We would like to thank Mr.Ikoma, manager of the Fifth Laboratory of ICOT, for his support.

References

[de Kleer 1986] J.de Kleer: "An Assumption-Based Truth maintenance System," Artificial Intelligence 28, pp.127-162 (1986).

[Doyle 1979] J.Doyle: "A Truth Maintenance System," Artificial Intelligence 24 (1986).

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

[Maruyama 1988] F.Maruyama et al.: "co-LODEX: a cooperative expert system for logic design," Proc.of FGCS'88, pp,1299-1306 (1988).

[Fujisawa 1989] H.Fujisawa et al.: "UHDL (Unified Hardware Description Language) and its support tools," Int.J.Computer Aided VLSI Design (1989).

[Duley and Dietmeyer 1969] J.R.Duley and D.L.Dietmeyer: "A digital system design language (DDL)," IEEE Trans.Computers,Vol.C-17,No.19, pp.850-861 (1968).

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

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

[Taki 1988] K.Taki: "The Parallel Software Research and Development Tool: Multi-PSI system," Programming of Future Generation Computers (1988).

[Ueda 1986] K.Ueda: "A Parallel Logic Programming Language with the Concept of a Guard," ICOT Technical Report, TR-208 (1986).

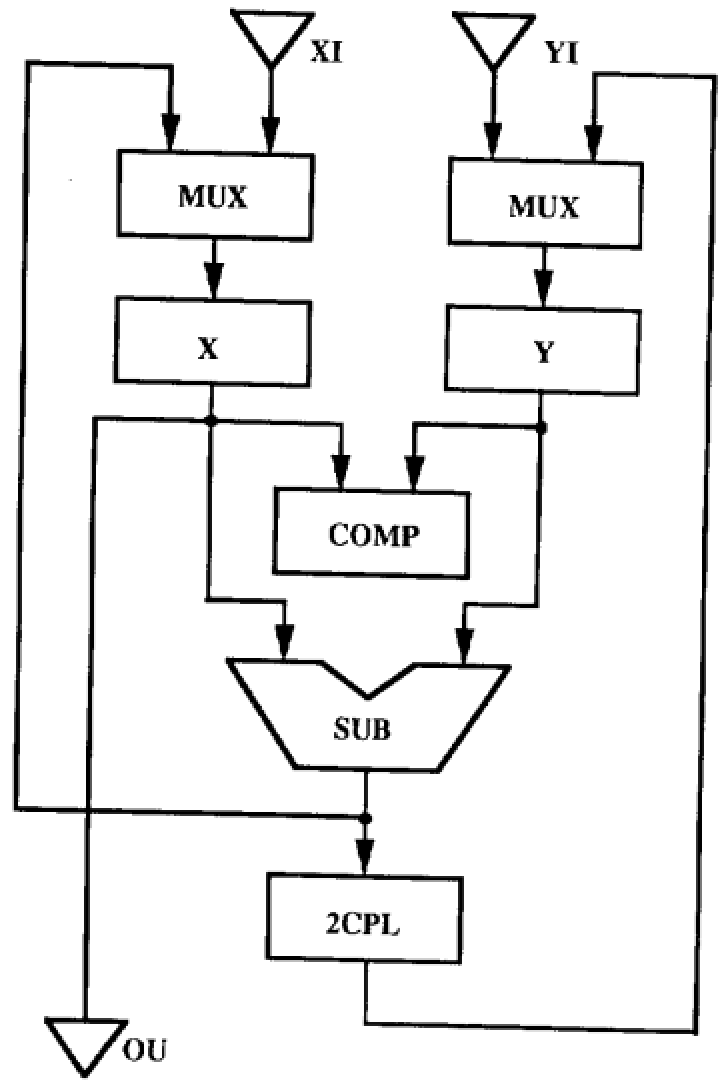

Figure 2. Block diagram

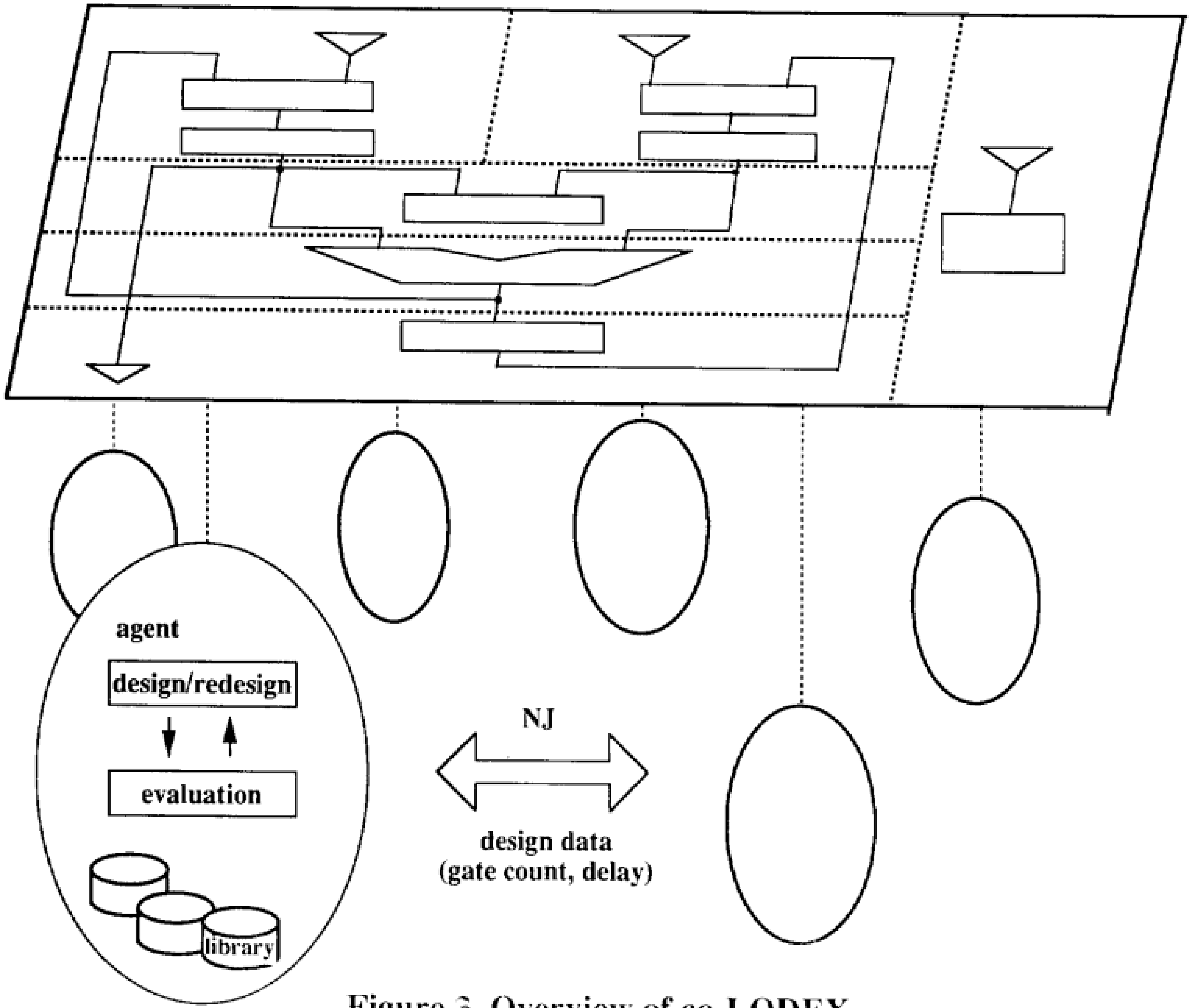

Figure 3. Overview of co-LODEX

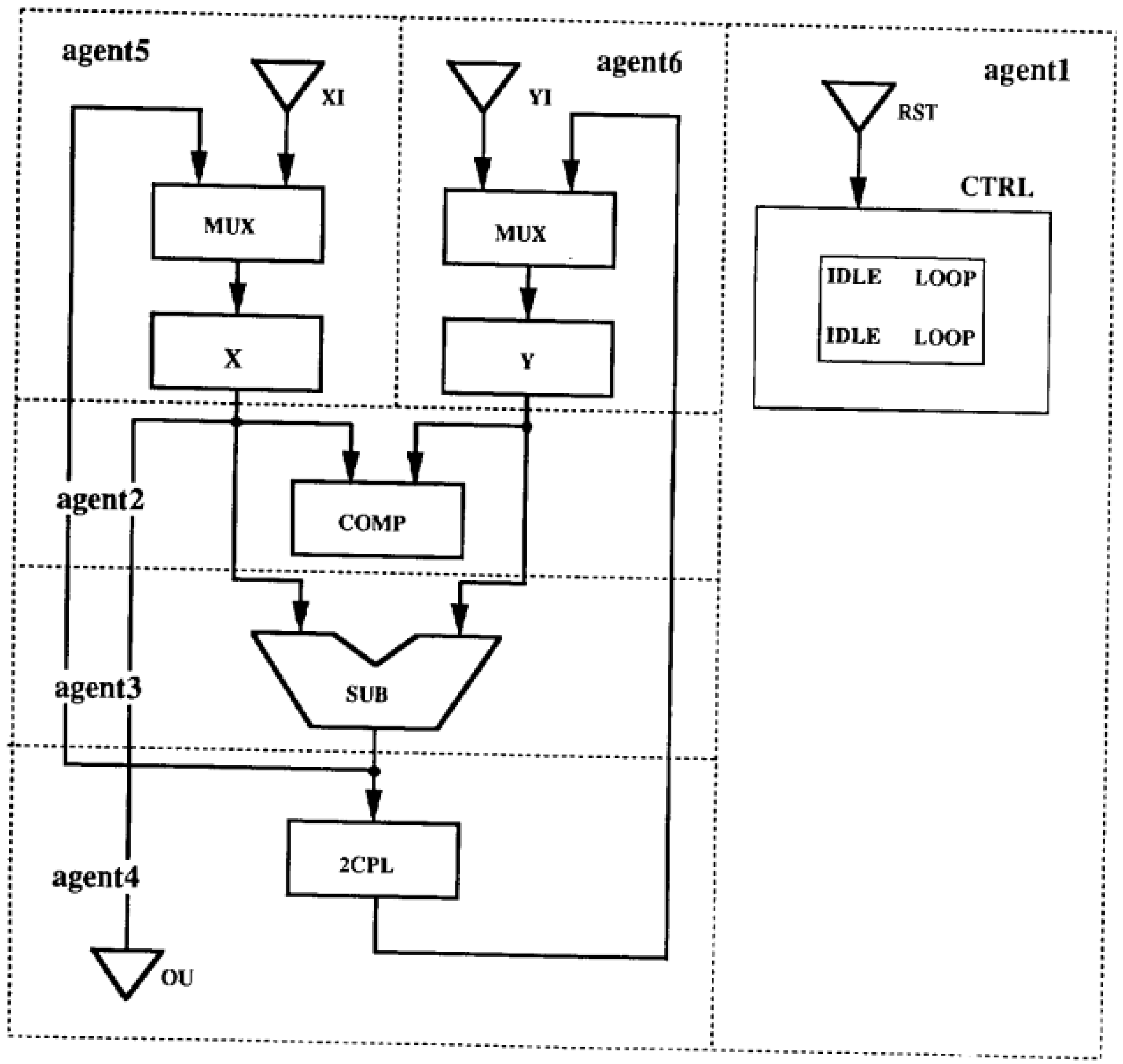

Figure 4. Subcircuits and agents

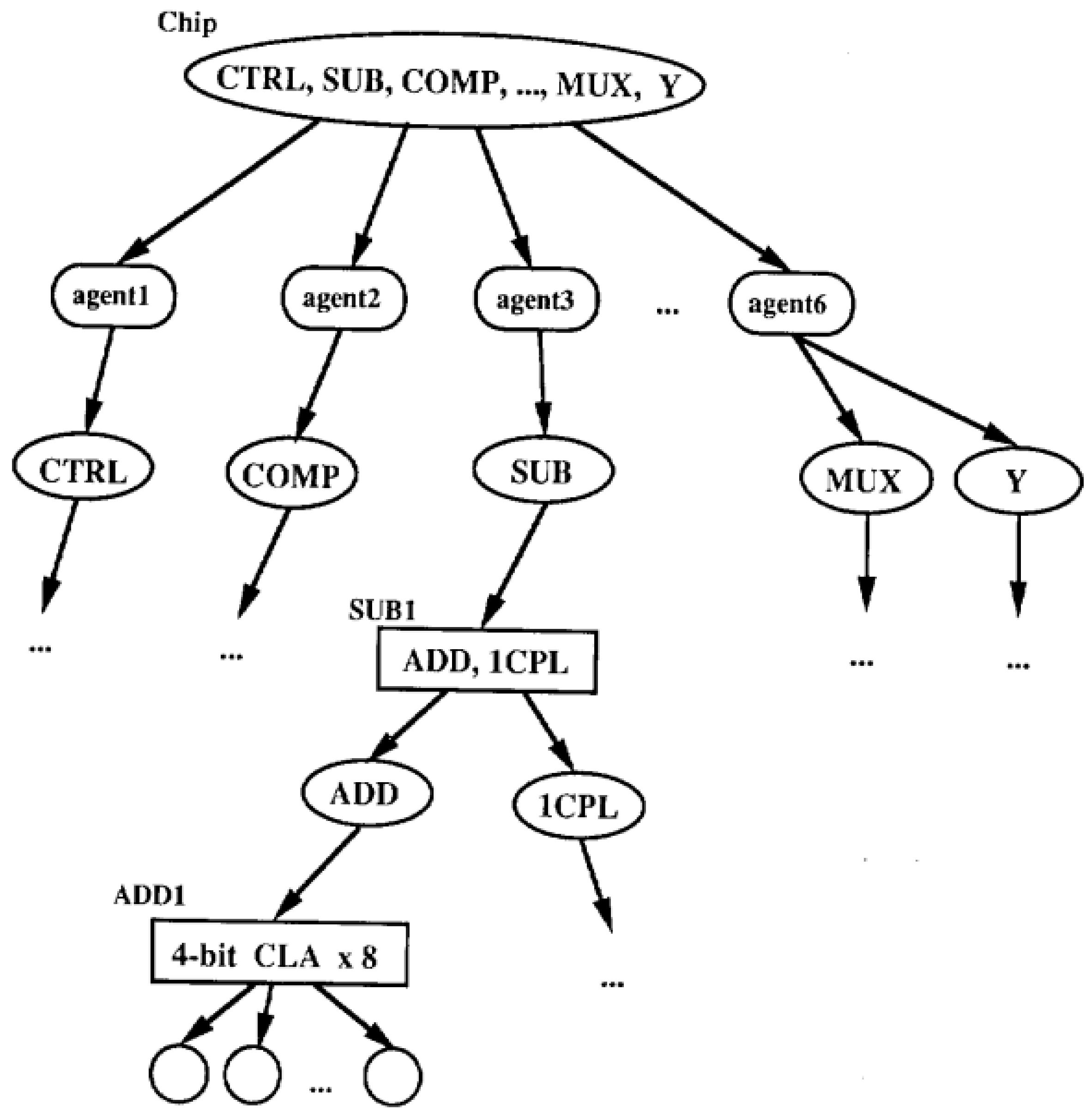

Figure 5. Hierarchical design description

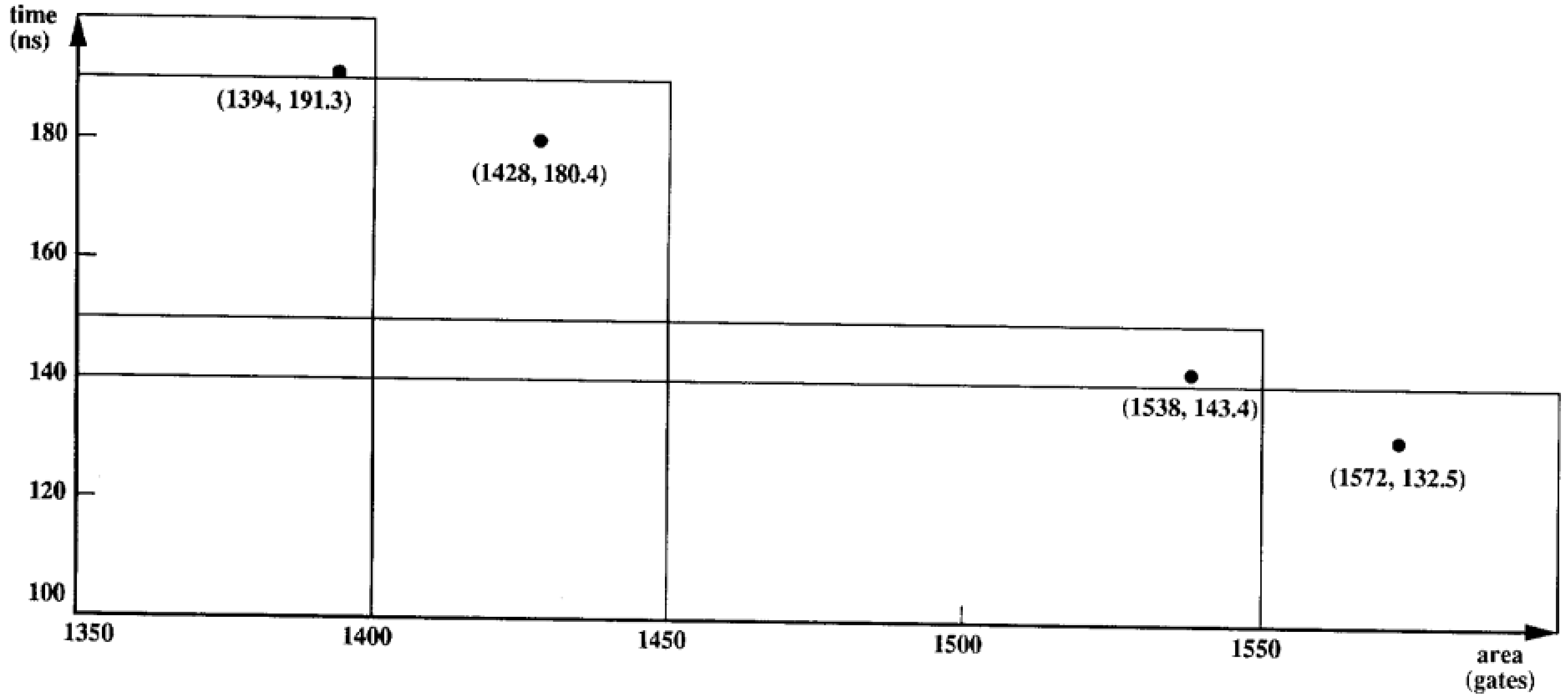

Figure 6. Experimental results

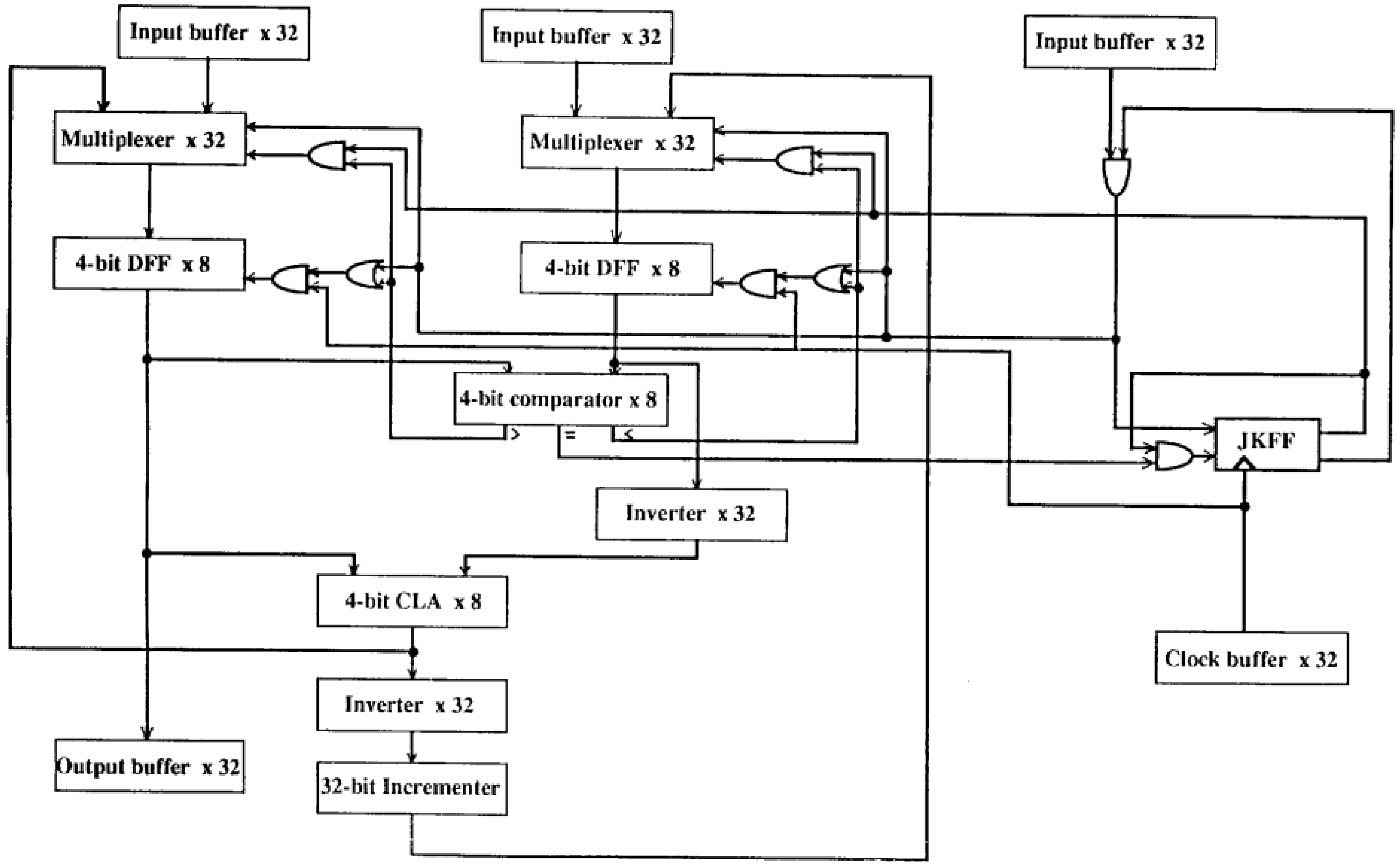

Figure 7. Example of design result