Knowledge-based Approach for VLSI Structured Design

Shigeru Takagi, Yasuyoshi Okada, Mitsukazu Washisaka

NTT Electrical Communications Laboratories
3-9-11, Midori-cho, Musashino-shi, Tokyo, 180, Japan

Abstract

This paper presents a knowledge-based approach for a design automation system which synthesizes VLSIs based on register transfer level behavior specifications.
By dividing the VLSI design problem into several design stages and introducing top-down and structured design knowledge at each stage, high-quality logic circuits can be synthesized with little computation. The ultimate goal of the knowledge-based design automation system architecture is also proposed.

1 Introduction

To reduce the VLSI design time and design costs, there is a strong need for a design automation system which can synthesize VLSIs based on high level specifications. Considerable effort has been devoted to the study of the fundamental logic synthesis technique and considerable progress has been made in the following areas:

- Logic minimization of Boolean expressions[1][2].
- Mapping of Boolean expressions into circuits[3].
- Local transformation of circuits[4].

However, these technique only covers an elementary part of the logic design knowledge. For large and complex VLSI design, the following more advanced design knowledge must be clarified and formalized.

- VLSI architecture (data paths and control organization) design knowledge.
- Knowledge for expanding high level functions into Boolean expressions.
- Design process control knowledge.

This paper presents this advanced design knowledge in top-down and structured design frameworks. Introducing this top-down and structured design knowledge in the design automation system improves the design quality and decreases the computation time.

Section 2 gives a brief summary of the knowledge based design model. Section 3 introduces a hardware description language. Section 4 overviews the VLSI design expert system under development. Section 5 formulates the architecture design knowledge. Section 6 deals with the knowledge for expanding high level function blocks into Boolean expressions. Section 7 shows a layout method.

2 Knowledge-based Design Model

Human designers are quite flexible and can:

- Design high-quality objects based on abstract specifications.
- Modify or redesign already-designed objects to meet new, slightly changed specifications.
- Backtrack to the appropriate design step if a constraint violation or design deadlock occurs.
- Explain design results and design decisions.
- Improve their design skill from design experience.

Unfortunately, it is not yet possible to make a design automation system with these abilities. However, we have decided on a knowledge-based VLSI design model as shown in Figure 1 and have attempted to formalize each knowledge base. The model consists of four subsystems:

- The design plan knowledge base contains knowledge for planning, scheduling, and focusing subparts to be designed, modified or redesigned. A design plan is generated and executed, and then the result is evaluated.
- The design operation knowledge base consists of design operators for synthesis, local modification, optimaization, analysis, and evaluation.
- The design context stores all information related to the design object. It includes specifications, designed objects, design decisions for the design operators choice and selection reasons, and the evaluation value.
- The designed instance base stores all designed instances. Each instance keeps all information copied from the design context.

In this paper, we focus on design operator knowledge and shows how we formalized this problem.

3 Hardware Specification

A successful design requires a clear idea of the end product. A VLSI is a processor which executes an algorithm. Therefore, a VLSI processor can be most concisely and clearly specified by the algorithm it executes. This algorithm can be formally described as a finite state machine in some specification language such as the Digital System Design Language (DDL)[5].

Figure 2 shows a VLSI specification example. The processor has five registers (GR0, GR1, SC, BR, IR), a Memory-read-data input terminal, and a Memory-address output terminal. It has three control states (Instruction-fetch, Memory-read, Decode-execute) . An automaton takes exactly one control state for each cycle. During each control state, several data transfer operations and execution operations are performed, followed by a state transition to the next control state. An execution operation refers to hardware resources , applies a particular function, and sends the result to a destination resource. For example,

```
GRO <-  GRO + GR1
```

means that the contents of GR0 and GR1 are added and the result is set to GR0.

4 System Overview

Because there is a great semantic gap between the behavior specifications and the VLSI mask pattern, it is difficult to design the mask directly from the specifications. Therefore, intermediate layers must be provided, and then abstract specifications must be successively refined into more concrete objects.

For a well structured design, these intermediate layers should be arranged according to the following criteria:

- Each layer must be concrete enough to have its own information representation language.
- The semantic gaps must be small enough to allow mapping operators to join the layers.

Figure 3 is an overview of the VLSI design expert system (SAVE) under development. SAVE is implemented on the Krine (Knowledge Representation and Inference Environment) system[6], which integrates AI programming paradigms such as LISP procedures, rules, logic programming, frames, and so one. The knowledge is described in a mixture of these paradigms.
The design process is as follows:

- The behavior of a VLSI is described as a finite state machine in the digital system design language (DDL). The DDL translator summarizes the DDL specifications into a table listing operations and conditions which must be satisfied for the operation to be executed.
- Based on this table, the VLSI architecture is designed. The VLSI architecture consists of several function blocks such as registers, arithmetic and logic units, and multiplexers. The architecture is designed so that the component number would be a minimum and the resulting hardware would be reasonable in a practical sense.
- Each architecture component (function block) is then expanded into Boolean expressions by the logic design subsystem. Function blocks are classified into one of several function block types. For each function block type, there are several circuits which differ in cost and speed. Design methods for generating these circuits are integrated in this subsystem.

- Boolean expressions are then converted to a network of real circuits such as the CMOS standard cell. If the constraint on circuit cost or circuit performance is not satisfied, a procedure for improving current circuits is applied.
- Finally, circuits are laid out and routed.

5 Architecture Design

This section shows how to break a complex design problem into several smaller, simpler problems in a structured way (Figure 4). For example, the original problem X is divided into sub-problems A and B. Sub-problem A is further divided into sub-sub-problems AA and AB. Although some problems can be designed in an arbitrary order, most sub-problems are not independent. For example, sub-problem B must be solved after sub-problem A ,because B uses information obtained from A. These dependencies are shown as dashed lines in Figure 4.

The method of problem division is not unique. The depth and excellence of the method determines the design quality and the design time.

This paper discusses the division method shown in Figure 4. A VLSI consists of a data path subsystem and a control subsystem. The data path subsystem is divided into an arithmetic unit subsystem and data transfer subsystem. The data transfer subsystem is further divided into buses, multiplexers, tri-state gates, and wires.

When the model is complete, optimum design knowledge for each sub-problem must be clarified. Correct, optimum, and efficient design needs theoretical background, design philosophy, and heuristics.

In VLSI architecture design, information about parallelism between operations can be used as theoretical background. The design philosophy is to merge hardware resources, which do not work simultaneously, into a single unit.

The rest of this section shows the parallelism analysis method and the optimum design knowledge.

5.1 Parallelism analysis

The DDL specification is summarized in an operation table. Each entry in the operation table consists of an operation and the condition which must be satisfied for the operation to be executed[8].

```
/execute-condition/  sink <- operand-1 operator operand-2
```

An automaton takes exactly one control state for each cycle. Therefore the logical product of any two control states S_k and ,S_l is zero.

$S_k * S_l = 0$ —— Relation1

Let two operations be OP_i and OP_j and their conditions be C_i and Cj respectively. Then, the analysis of the parallelism can be reduced to calculating the logical product of C_i and C_j under the above relation. That is:

(1) If $C_i * C_j = 0$ then OP_i and OP_j never work simultaneously.
(2) If $C_i * C_j \neq 0$ then OP_i and OP_j work simultaneously when $C_i * C_j$ becomes true.

Using this rule, we can infer parallelism between any two operations.

5.2 ALU Allocation

If two hardware resources never work simultaneously, they can be merged into a single block. Therefore, operations which do not work in parallel can be merged into an ALU. This merging procedure reduces hardware costs, and suggests the following rule:

- Merge as many hardware functions, that do not work in parallel, as possible.

The optimum ALU subsystem is obtained using the following graph procedure:

1. First, a graph is defined where nodes represent operations and edges exist only between operations which do not work simultaneously.
2. Then, the graph is partitioned into completely connected subgraphs.
3. Finally, an ALU is allocated for each subgraph.

After roughly formalizing the problem, heuristic knowledge for reducing computation time and maintaining reasonable ALU hardware design is collected. Figure 5 shows an example of the ALU allocation rule. The rule first checks the parallelism between an operation and the ALU under construction, then merges the operation into the ALU. By this procedure, an ALU becomes a multi-function unit.

During the ALU allocation, logical information about data transfer (e.g., source, sink, transfer condition) is also derived. This information becomes the specification for the data transfer path.

5.3 Data Transfer Path Design

Based on the logical information about data transfer, physical data transfer paths are designed. The data transfer path design stages and the knowledge for each stage are as follows.

1. Bus design stage: A bus is assigned for a set of logical transfers, which don not work in parallel or whose sources are the same. A minimum number of sets are obtained by the graph partitioning procedure as shown in the ALU design. During bus design, information about logical data transfer is modified and becomes the specification to the rest of the data transfer path design.

2. Multiplexer design stage: If there are multiple logical data transfers, whose destinations are the same, a resource input multiplexer is generated between the sources and the destination.

3. Tri-state gate design stage: Tri-state gates are inserted between the source and the bus.

Figure 6 shows an example of the non-structured design data paths and the structured design data paths obtained from the same specifications.

At the architecture design level, details of each function block are not yet fixed. The next step is to realize each function block with lower level components.

6 Logic Design

The knowledge for function block design is also modeled in top-down and hierarchical style[9]. The problem division method depends on the function block type. If the function block is very complex and large, the division tree becomes wide and deep. If the function block is small and simple, it can be directly realized with gates. This section gives two examples: a multi-function block and a fixed-function block.

6.1 Multi-function Block Design

An arithmetic and logic unit (ALU) is a multi-function unit. The design of an ALU is difficult because of its varying functions. An ALU has both data input/output terminals and control input terminals for selecting functions. If control signals are supplied to the control input terminals, the ALU will execute the designated operation between input data and then send the result to an output terminal. Therefore, ALU specifications can be expressed in terms of data input-output relations and control signals for specifying each input-output relation. For example

S4*S3*S2*S1*S0 F = A plus Cin

means that if control signals S0 through S4 are all "1", the ALU executes the arithmetic sum of A and Cin.

There are several ways to divide an ALU into sub blocks. Figure 7 is an example of a division method (called ALU template) devised for medium-speed computers. The philosophy behind this template is as follows:

1. ALU logic is structured as output-stage logic, carry-predict logic, input logic which generates G and P signals, and decoder logic. Here, the G signal implies nth-bit-carry generation, and the P signal indicates the nth-bit-partial sum.

2. To perform an arithmetic sum function, the carry is predicted from the G and P signals and is applied, along with partial sum P, to the output-stage logic.

3. To perform a logical function, the carry is set to zero, and P is controlled to produce the required logical function.

An ALU template outlines the ALU logical substructure. Details of each sub-block are not given at this level. When an ALU specification is given, sub-blocks are synthesized according to the following steps:

1. The G, P, and Carry-gate logic is initialized to "0".
2. The syntax of each function specification is analyzed and sub-blocks are modified according to the synthesis rules.

   ```
   [Rule example]
   IF the function syntax is "Control-code F = X plus Y plus Cin"
   Then DO P = P + Control-code * (X @ Y)
        DO G = G + Control-code * (X * Y)
        DO Carry-gate = carry-gate + Control-code
   ```

3. The G, P and Carry-gate logic is minimized.
4. The G, P and output logic is repeated n times (data bbit width).
5. Carry logic is synthesized.

Figure 8 shows the effect of the ALU structured design.

6.2 Fixed-function Block Design

Problem division of a fixed-function block such as an adder, multiplier, and divider is straight forward. Figure 9 shows an example of problem division. The floating-point add/subtract block is hierarchically broken down. These fixed-function blocks are easily specified by a few parameters such as the amount of input data and the data bit-width. When these block parameters are given, parameters for each sub-block are derived and given to the sub-blocks. Then each sub-block is divided in turn.

6.3 Optimization Knowledge

The top-down, structured division method can realize functionally correct circuits. However, global constraints such as circuit costs or circuit performance may not be satisfied. This problem is addressed with the following two approaches:

1. Generate and test:
   There are several ways to divide a function block . The easiest way is to try each method to see if the constraints are satisfied.
2. Controlled local-transformation of circuits.
   When given logic circuits do not satisfy the global constraint, the designer usually tries to modify the existing circuits. This process can be simulated with local-transformation rules and application control of these rules. A local-transformation rule replaces a small circuit subgraph with another subgraph which is functionally equivalent but differs according to some global measure such as cost or performance. These rules are categorized according to the measure.

   When some global constraint is not satisfied, circuits are analyzed, the focus area candidate is decided, and local transformation rules which may reduce the unsatisfied measure are selected and applied.

   For example, if circuit performance is unacceptable, critical paths are analyzed and local transformation rules which decreases the logical depth are applied.

7 Layout Knowledge

The top-down, structured design method produces circuits with a regular structure. In our experience, the chip area of regular circuits decreases if they are laid out regularly. Therefore, in our system, this regular layout knowledge is programmed for each function block type[10].

Figure 10 shows an example of a regular layout. This is a multiplier layout with about 6000 gates.

8 Conclusion and Future Work

This paper proposed a knowledge-based approach for VLSI structured design. By properly arranging design stages and integrating top-down, structured design knowledge for each layer, high-quality logic circuits can be synthesized.

Future work will focus on clarifying the mechanism for design process control, redesign, using designed instances, and design skill acquisition.

References

[1] S.J.Hong, G.R.Cain and D.L.Ostapko :"MINI; A heuristic Approach for Logic Minimization", IBM J.RES.DEVELOP.,PP.443-458(1974)

[2] D.W.Brown :"A State-machine Synthesizer", Proc. 18th DA conf.,PP.301-305(1981)

[3] T.Shinsha, T.Kubo, M.Hikosaka, K.Akiyama and K.Ishihara :"Polaris: Polarity Propagation Algorithm for Combinational Logic Synthesis", Proc. 21th DA Conf. PP.543-549 (1984)

[4] J.A.Darringer :"A New Look at Logic Synthesis", Proc. 17th DA Conf. PP.543-549(1981)

[5] J.R.Duley and D.L.Dietmeyer :"A Digital System Design Language (DDL)", IEEE Trans. Comput. VOL. C-17, No.9, PP850-861.

[6] U.Ogawa,K.Shima,T.Sugawara and S.Takagi :"Knowledge Representation and Inference Environment: KRINE", Proc. FGCS'84, PP.643-651 (1984)

[7] M.Endo,T.Hoshino and O.Karatsu :"Optimization Technique of Logic Synthesis System:ANGEL (in Japanese)", DA23-5, IPSJ (1984)

[8] S.Takagi :"Rule Based Synthesis, Verification and Compensation of Data Paths", Proc. ICCD'84 PP.133-138 (1984)

[9] S.Takagi :"Design Method Based Logic Synthesis", CHDL'85 PP.49-63(1985)

[10] S.Takagi :"A Placement Method Based on Logic Circuit Structure information", Trans. IECE Jpn, Vol.J70-CNo.1,PP.11-20(1987)

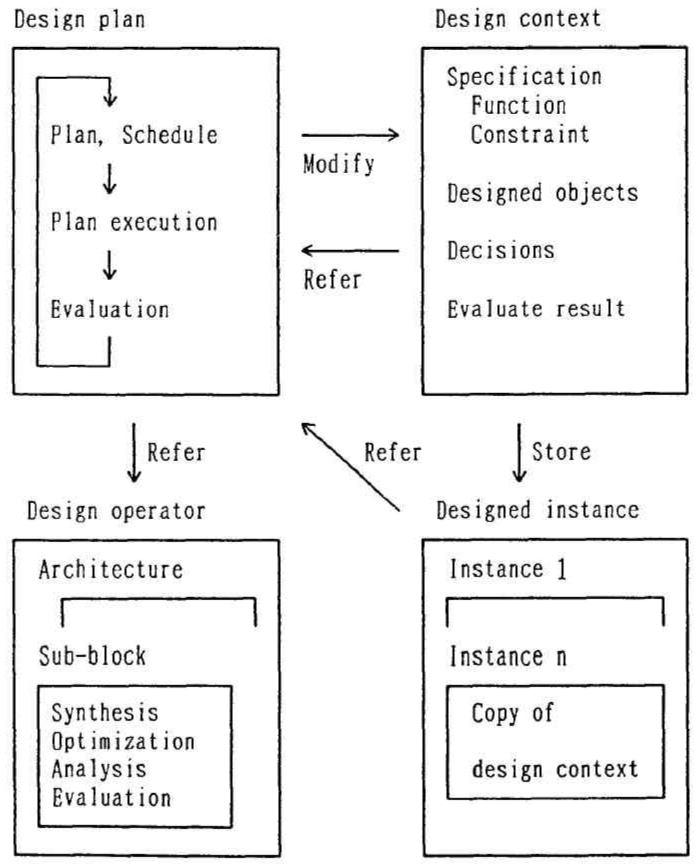


FIG.1 Model of the knowledge-based design expert system

```
<SYSTEM> SAMPLE
  <REGISTER> GR0 GR1 SC BR IR;
  <INPUT> MEMORY-READ-DATA;
  <OUTPUT> MEMORY-ADDRESS;
  <AUTOMATON> CONTROL;
    <STATE>
    INSTRUCTION-FETCH: CO-BEGIN
                         MEMORY-ADDRESS <- SC,
                         GOTO MEMORY-READ  END;
    MEMORY-READ:       CO-BEGIN
                         IR <- MEMORY-READ-DATA,
                         SC <- 1+ SC,
                         GOTO DECODE-EXECUTE END;
    DECODE-EXECUTE: CASE IR ADD
                           CO-BEGIN
                             GR0 <- GR0 + GR1,
                             GOTO INSTRUCTUIN-FETCH END;
                          SUBTRACT
                           CO-BEGIN
                             GR0 <- GR0 - GR1,
                             GOTO INSTRUCTION-FETCH END;
                          EXCLUSIVE-OR
                           CO-BEGIN
                             GR0 <- GR0 @ GR1,
                             GOTO INSTRUCTION-FETCH END;
                 :
                 :
                 :
                    END-CASE;
    END <STATE>;
  END CONTROL;
END SAMPLE;
```

FIG.2 DDL specification example

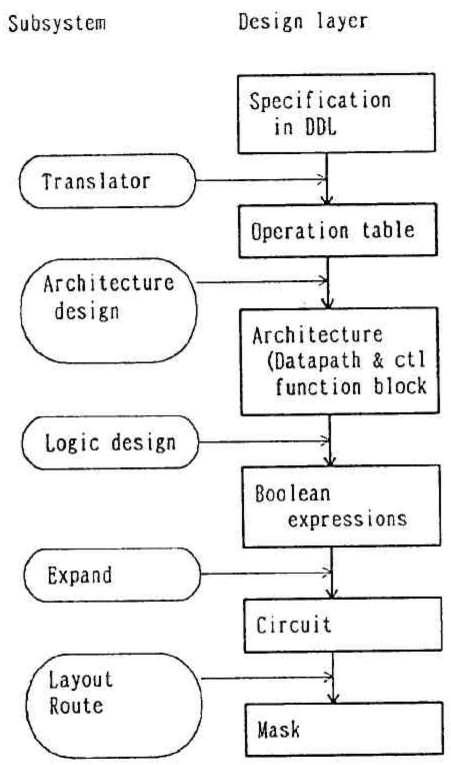


FIG. 3 System overview

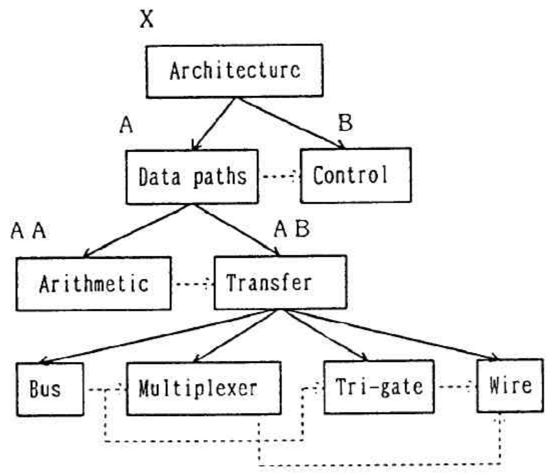
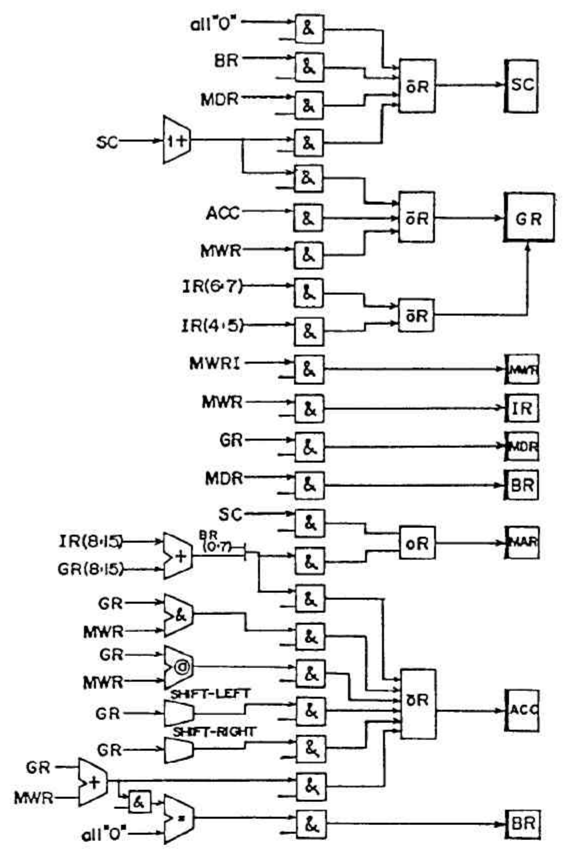


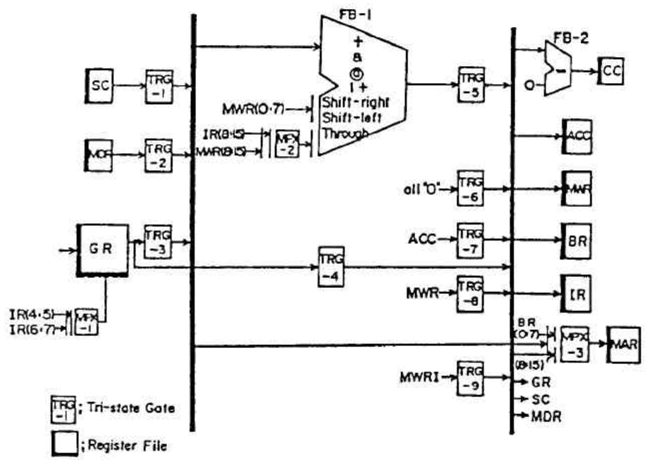
Fig. 4 Architecture design method

```
IF
  1)there is an ALU,
  2)there is an operation,
  3)the ALU and the operation never work simultaneously,
  4)the ALU does not include a function of the operation,
  5)the function of the operation is mergeable with the ALU,
  6)the left input port of the ALU is already connected
        to the first operand of the operation,
  7)the right input port of the ALU is already connected
        to the second operand of the operation, and
  8)the output port of the ALU is not connected to the
        sink of the operation,
THEN
  1)add the required function with its condition to the ALU,
  2)update the transfer condition from the first operand,
  3)update the transfer condition from the second operans, and
  4)add a new path from the output port of the ALU
        to the sink with its transfer condition.
```

Fig. 5 ALU allocation rule example



non-structured design data paths



structured design data paths

Fig. 6 Data paths design example

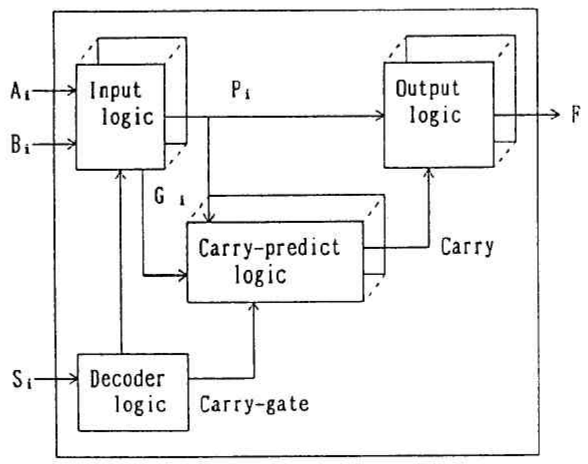


Fig. 7 ALU template example

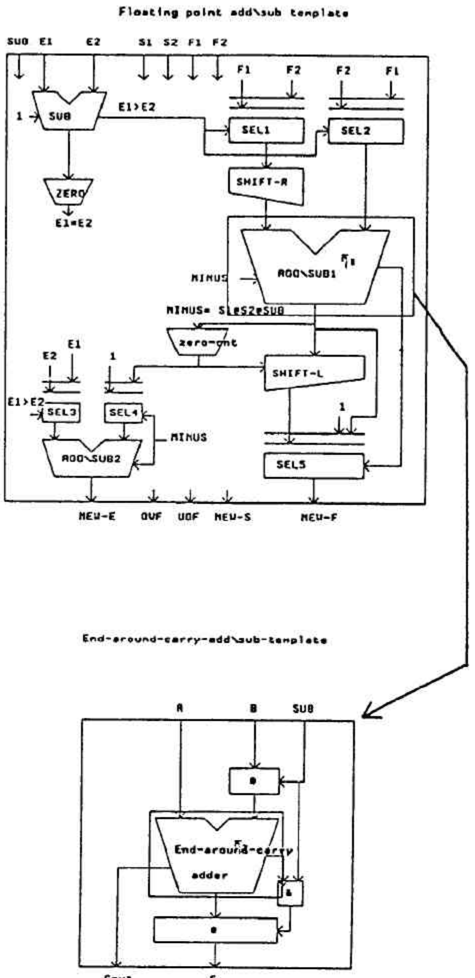


Fig. 9 Fixed-function block design (FLP adder)

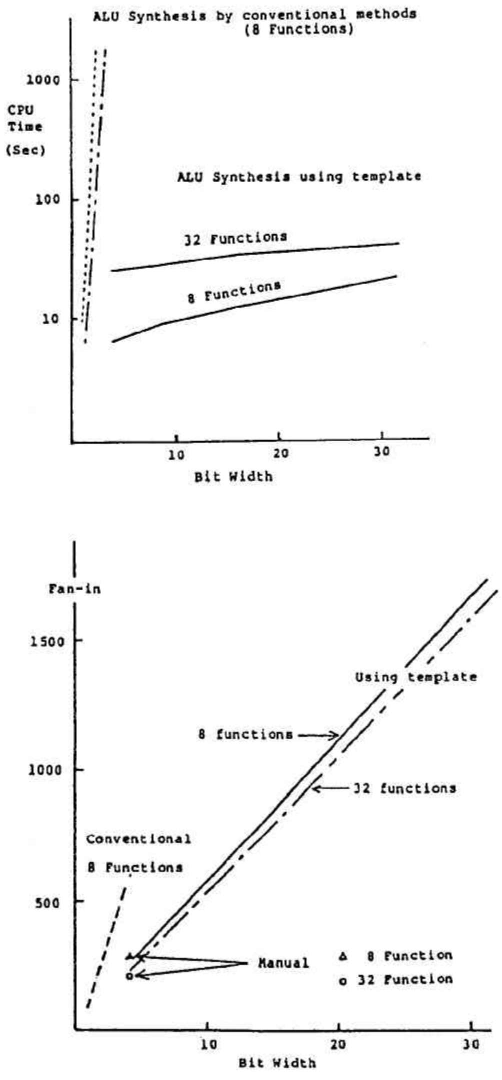


Fig. 8 Evaluation of synthesized ALU

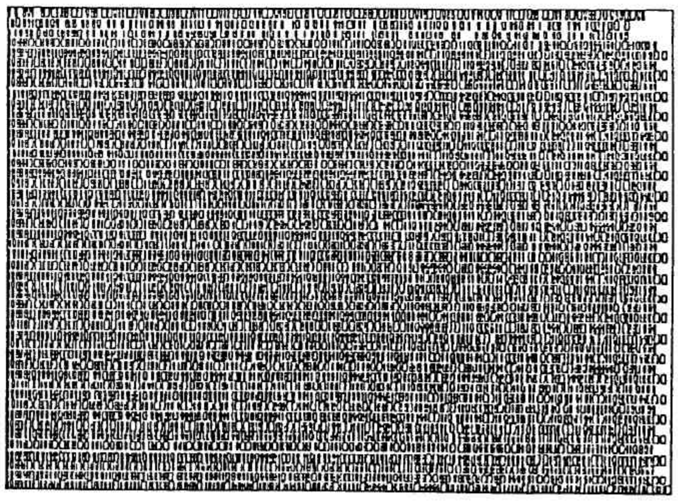


Fig. 10 Regular layout example (multiplier)