3rd NASA Symposium on VLSI Design 1991

Automated ILA Design for Synchronous Sequential Circuits

M. N. Liu, K. Z. Liu, G. K. Maki and S. R. Whitaker
NASA Space Engineering Research Center for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **This paper presents an ILA architecture for synchronous sequential circuits. This technique utilizes linear algebra to produce the design equations. The ILA realization of synchronous sequential logic can be fully automated with a computer program. A programmable design procedure is proposed to fulfill the design task and layout generation. A software algorithm in the C language has been developed and tested to generate 1 um CMOS layouts using the Hewlett-Packard FUNGEN module generator shell.**

1 Introduction

The design of sequential circuits presents a major task for most digital systems. As Very Large Scale Integrated (VLSI) technology advances, developing an architecture to maximize the efficiencies of all the design steps becomes a major goal in the research of sequential circuit design.

This paper introduces the Iterative Logic Array (ILA) as a new architecture for synchronous sequential circuits. This architecture realizes a sequential circuit by replicating simple basic modules. With an ILA architecture, a sequential machine can be built into a very regular form automatically by a computer program with a single type of ILA module. The simplicity and programmability of the ILA architecture significantly reduce the design task in all stages of VLSI implementation, from logic design, circuit design, artwork generation to verification.

2 ILA Architecture

Iterative Logic Arrays (ILA) have been described in the literature for quite some time [1,2]. An ILA circuit consists of an array of identical cells. Generally, as shown in Figure 1, each ILA cell contains two sets of input signals. One set of inputs are applied in parallel, while the other set of inputs are driven by adjacent cells. Signals normally propagate in only one direction between cells, and outputs are derived only from the serial outputs of the last cell.

In an ILA architecture for sequential circuits, the next state of each state variable is generated by a slice of concatenated ILA cells. A sequential network is then constructed by placing the ILA slices side by side.

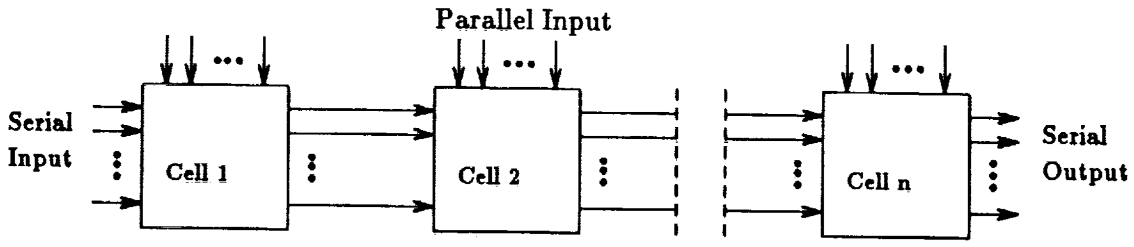


Figure 1: A slice of ILA circuit

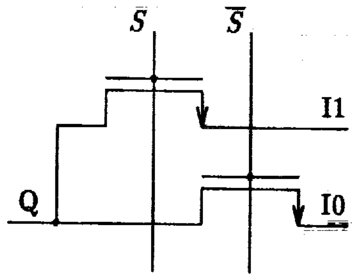


Figure 2: Pass transistor 2-to-1 MUX

The basic cell of an ILA sequential network consists of a 2-to-1 multiplexer (MUX) and a next state forming logic. A MUX cell has a select line S, its complement $\overline{S}$ and two data inputs I_0 and I_1, and a logic function defined by Equation 1.

$$Q = S * I_1 + \overline{S} * I_0 \tag{1}$$

The simplest way to implement the MUX function is to use a pass transistor circuit. Basically, the pass transistor MUX, excluding level restoration logic, is a module of two pass transistors, which functions as two simple switches. Figure 2 shows the circuit of two inputs I_1 and I_0 and one output Q controlled by two control lines S and $\overline{S}$ which are assumed to be asserted exclusively such that only one of two inputs I_1 and I_0 can be passed to Q at a given time.

Some details in pass transistor transmission characteristics are omitted here. Design considerations, such as level restoration, are assumed to be handled by the output buffers. The circuit design considerations have been discussed in [3,4,5].

3 Operational Function

In this research, the one-hot-code is utilized as the state assignment for a synchronous flow table. With the one-hot-code assignment, there is a unique state variable corresponding to each state. That makes it possible to express the design function using the states in the flow table explicitly. A new form of mathematical expression is proposed next which describes a flow table directly by flow table states.

Definition 1 *The set of operational functions is the behavior description of a synchronous flow table of n rows and m columns. Each function is an equation for a next state $\hat{S}_i$ in the flow table.*

$$\hat{S}_i = \sum_{p=1}^{m} s_{ip} I_p \tag{2}$$

where s_{ip} is an OR function of the states $S_j, \forall j = 1, \cdots, n$, which have S_i as the next entry under input I_p.

It can be shown that there is a one-to-one mapping between the next state equation

$$Y_i = \sum_{p=1}^{m} f_{ip} I_p \tag{3}$$

and the operational function. With the one-hot-code state assignment, each τ-partition can be expressed as

$$\tau = \{S_i; S\}$$

which partitions a single state S_i from the rest states S in the flow table. The number of state variables is equal to the number of states. Next state η-partitions can be formed using known procedures [6]. If an η-partition η_i is

$$\eta_i = \{S_1 S_2 \cdots S_i; S\}$$

then it is well known that

$$f_{ip} = y_1 + y_2 + \cdots + y_i.$$

On the other hand, Equation 2

$$\hat{S}_i = \sum_{p=1}^{m} s_{ip} I_p$$

can be mapped into a next state equation as Equation 3 if the one-hot-code assignment is used where f_{ip} are sum of the state variables y_j corresponding to s_{ip} in Equation 2. Therefore, there is a one-to-one mapping between Equation 2 and Equation 3.

Since the operational function is a direct representation of the flow table, they can be derived by inspection. For each state in the next state entry, there is a product term of the present state and input state in the operational function. If a synchronous machine is specified by a state diagram, the state diagram may need to be converted to a flow table, though it will not be too hard for an experienced designer to derive the operational functions from the state diagram directly.

Table 1 is the flow table of a state machine with four states. For example, State S_a appears as the next state entry of states S_b, S_c and S_d under I_1. Therefore, the operational function for S_a is

$$\hat{S}_a = (S_b + S_c + S_d) I_1.$$

For State S_b, it appears as the next state under both I_1 and I_2. So the operational function for state S_b is

$$\hat{S}_b = S_a I_1 + (S_b + S_c) I_2.$$

	I_1	I_2	I_3
S_a	S_b	S_d	S_c
S_b	S_a	S_b	S_c
S_c	S_a	S_b	S_d
S_d	S_a	S_d	S_d

Table 1: A synchronous flow table for the state diagram

The operational functions for state S_c and S_d can be derived in the same way. All together, the operational functions for Table 1 are as follows:

$$\begin{aligned} \hat{S}_a &= (S_b + S_c + S_d)I_1 \\ \hat{S}_b &= S_a I_1 + (S_b + S_c)I_2 \\ \hat{S}_c &= (S_a + S_b + S_d)I_3 \\ \hat{S}_d &= (S_a + S_d)I_2 + (S_c + S_d)I_3 \end{aligned} \tag{4}$$

4 ILA Architecture for Synchronous Sequential Circuits

A simple regular ILA structure requires:

- The design equation is convertible to a pass logic function where each control variable passes a single pass variable or a constant.
- The control variables are shared with each pass logic function.

With such a structure, if the pass variables in each equation are the same, the signal bus to each slice of ILA circuit can be minimized to a single wire.

If state S_i is used as the control and S_i appears as a next state under only one input I_p, then I_p can be the only pass variable in the design equation for $\hat{S}_i$. For example, the equation for $\hat{S}_a$ in Equation 4 can be converted into a pass logic function with input I_1 as a pass variable:

$$\hat{S}_a = S_b(I_1) + S_c(I_1) + S_d(I_1)$$

From the definition of the operational function in Equation 2, if S_i appears only under input I_p, then Equation 2 can be rewritten as:

$$\hat{S}_i = s_{ip} I_p \tag{5}$$

where s_{ip} is an OR function of the states $S_k, k \in \{1, 2, \cdots, n\}$. Therefore, Equation 5 can be written into:

$$\hat{S}_i = I_p \sum_{k=1}^{n} g_{ik} S_k \tag{6}$$

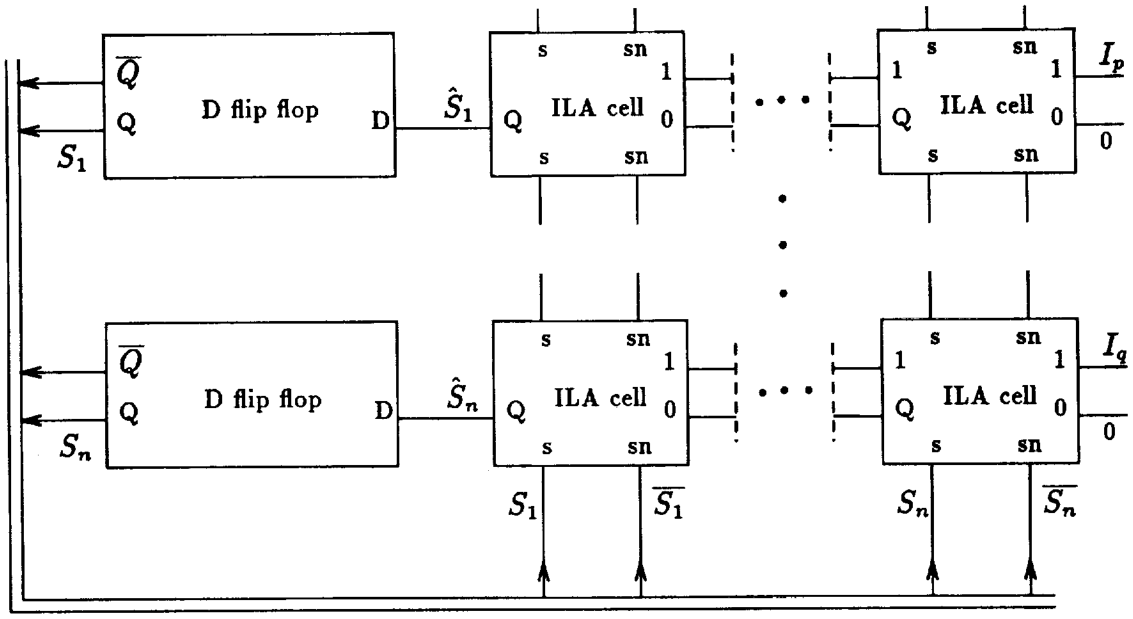


Figure 3: The general ILA structure for synchronous logic

or in a form of pass logic expression:

$$\hat{S}_i = S_1(g_{i1}I_p) + \overline{S_1}(\cdots S_k(g_{ik}I_p) + \overline{S_k}(\cdots(S_n(g_{in}I_p) + \overline{S_n}(0)\cdots). \tag{7}$$

where

$$g_{ik} = \begin{cases} 1 & \text{if } S_i \text{ is the next state of } S_k \text{ under } I_p \\ 0 & \text{if } S_i \text{ is not the next state of } S_k \text{ under } I_p \end{cases}$$

Theorem 1 *The architecture depicted in Figure 3 is a proper model for a synchronous sequential circuit.*

Proof: The proof follows directly from one-hot-code assignment that one and only one state variable are active at a time and Equation 6 contains only one input state. Clearly, the architecture realizes Equation 6 by placing a multiplexer under S_k where $g_{ik} = 1$ and a wire under S_k where $g_{ik} = 0$.
□

To accomplish the ILA structure, S_i must be restricted to appear in a flow table under only one input I_p. If a state S_i appears as a next state under both I_p and I_q, S_i has to be split into two different states. For example, in Table 1, state S_b appears under both input I_1 and I_2. It is necessary to distinguish S_b with two unique states S_{b1} and S_{b2} where S_{b1} represents the S_b under I_1 and S_{b2} represents the S_b under I_2. Similarly, state S_d needs to be split into S_{d2} and S_{d3}. A revised flow table can then be obtained by splitting all states under different columns. Table 2 shows the result.

After updating the flow table, the operational function for each state can be derived in the same way as before. For example, S_{b2} is the state under I_2 only. Therefore, its operational function is

$$\hat{S}_{b2} = 0 + S_{b1}I_2 + S_{b2}I_2 + S_cI_2 + 0 + 0.$$

	I_1	I_2	I_3
S_a	S_{b1}	S_{d2}	S_c
S_{b1}	S_a	S_{b2}	S_c
S_{b2}	S_a	S_{b2}	S_c
S_c	S_a	S_{b2}	S_{d3}
S_{d2}	S_a	S_{d2}	S_{d3}
S_{d3}	S_a	S_{d2}	S_{d3}

Table 2: A revised flow table

All other operational functions are also in the same form. The results are shown as follows:

$$\begin{array}{llllll}
\hat{S}_a = 0 & + S_{b1}I_1 & + S_{b2}I_1 & + S_cI_1 & + S_{d2}I_1 & + S_{d3}I_1 \\
\hat{S}_{b1} = S_aI_1 & + 0 & + 0 & + 0 & + 0 & + 0 \\
\hat{S}_{b2} = 0 & + S_{b1}I_2 & + S_{b2}I_2 & + S_cI_2 & + 0 & + 0 \\
\hat{S}_c = S_aI_3 & + S_{b1}I_3 & + S_{b2}I_3 & 0 & + 0 & + 0 \\
\hat{S}_{d2} = S_aI_2 & + 0 & + 0 & + 0 & + S_{d2}I_2 & + S_{d3}I_2 \\
\hat{S}_{d3} = 0 & + 0 & + 0 & + S_cI_3 & + S_{d2}I_3 & + S_{d3}I_3
\end{array}$$

Splitting states in a flow table allows all of the pass variables in an operational function to be the same. The disadvantage of splitting states is that it generates additional next state equations. Increasing the number of equations implies increasing the area in silicon. It is a trade off by gaining programmability and regularity of the ILA realization versus cost. An automated sequential circuit design will significantly reduce the design effort and speedup the process of implementation.

5 The Matrix Expression

The operational functions discussed in previous sections can be efficiently expressed with matrices. The matrix will also help to implement the function in silicon. With Equation 6, a synchronous sequential circuit can be expressed with a set of equations:

$$\hat{S}_1 = I_p \sum_{k=1}^{n} g_{1k} S_k$$

$$\cdots$$

$$\hat{S}_n = I_q \sum_{k=1}^{n} g_{nk} S_k$$

Such a set of equations are equivalent to a matrix expression:

$$\hat{\mathbf{S}} = \mathbf{A} \times \mathbf{G} \times \mathbf{S} \tag{8}$$

where matrices $\hat{\mathbf{S}}$ are $\mathbf{S}$ are column vectors

$$\hat{\mathbf{S}} \equiv \begin{pmatrix} \hat{S}_1 \\ \hat{S}_2 \\ \vdots \\ \hat{S}_n \end{pmatrix}; \quad \mathbf{S} \equiv \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{pmatrix};$$

matrix $\mathbf{A}$ is a diagonal matrix with I_p in the i^{th} row/column if the next state S_i is under I_p,

$$\mathbf{A} \equiv \begin{pmatrix} I_i & & & & 0 \\ & \cdot & & & \\ & & \cdot & & \\ & & & \cdot & \\ 0 & & & & I_j \end{pmatrix}$$

and matrix $\mathbf{G}$ is defined as

$$\mathbf{G} \equiv \begin{pmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & \ddots & \\ g_{n1} & & g_{nn} \end{pmatrix}$$

in which

$$g_{ik} = \begin{cases} 1 & \text{if } S_i \text{ is the next state of } S_k \\ 0 & \text{if } S_i \text{ is not the next state of } S_k \end{cases}$$

For example, the matrix expression for the flow table in Table 2 is:

$$\begin{pmatrix} \hat{S}_a \\ \hat{S}_{b1} \\ \hat{S}_{b2} \\ \hat{S}_c \\ \hat{S}_{d1} \\ \hat{S}_{d2} \end{pmatrix} = \begin{pmatrix} I_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & I_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & I_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & I_3 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} S_a \\ S_{b1} \\ S_{b2} \\ S_c \\ S_{d1} \\ S_{d2} \end{pmatrix} \quad (9)$$

The matrix $\mathbf{A}$ and $\mathbf{G}$ are directly related to hardware structure. As in the ILA realization, there will be a slice of the ILA circuit for each design equation, as shown in Figure 3. Now each element at the diagonal of the matrix $\mathbf{A}$ indicates the input state to the ILA slice. Each row of matrix $\mathbf{G}$ reveals the location of ILA cells in the slice. If the element g_{ik} is 1, then an ILA cell will be placed under the control of state S_k in the slice of the ILA circuit for next state $\hat{S}_i$. If g_{ik} is equal to 0, a wire will be placed in that position. An example of the ILA realization will be shown in next section.

6 Design Procedure

From the discussion in the previous section, the design of a synchronous machine can be completely automated by programming an ILA cell or a wire into a pre-interconnected layout floor. It allows the physical layout to be designed and stored in computer as a set of building blocks. Then for each instance of synchronous sequential logic, an ILA circuit can be implemented by placing ILA cells according to the corresponding **G** matrix. The A matrix will indicate the interconnection to input states.

From the layout point of view, a wire can be considered as a cell as well. Hence there will be two cell types in an ILA realization. Let the ILA cell which performs the multiplexer function be defined as a *MUX cell*. Let a wire be defined as an *ILA ZERO cell*. Now the computer program can search the **G** matrix and place a MUX cell once a "1" is encountered or a ZERO cell once a "0" is encountered. The schematic of a MUX cell and a ZERO cell are shown in Figure 4 (b) and (c) respectively.

Procedure 1 *Synchronous ILA network design procedure.*

Step 1. *For a synchronous machine specified by a state diagram, convert it into a synchronous flow table (state table).*

Step 2. *If a state appears as a next state under more than one input column, split the state and give a unique name to the state under each input column. Repeat this step until all states under one column are distinguished from states under other columns.*

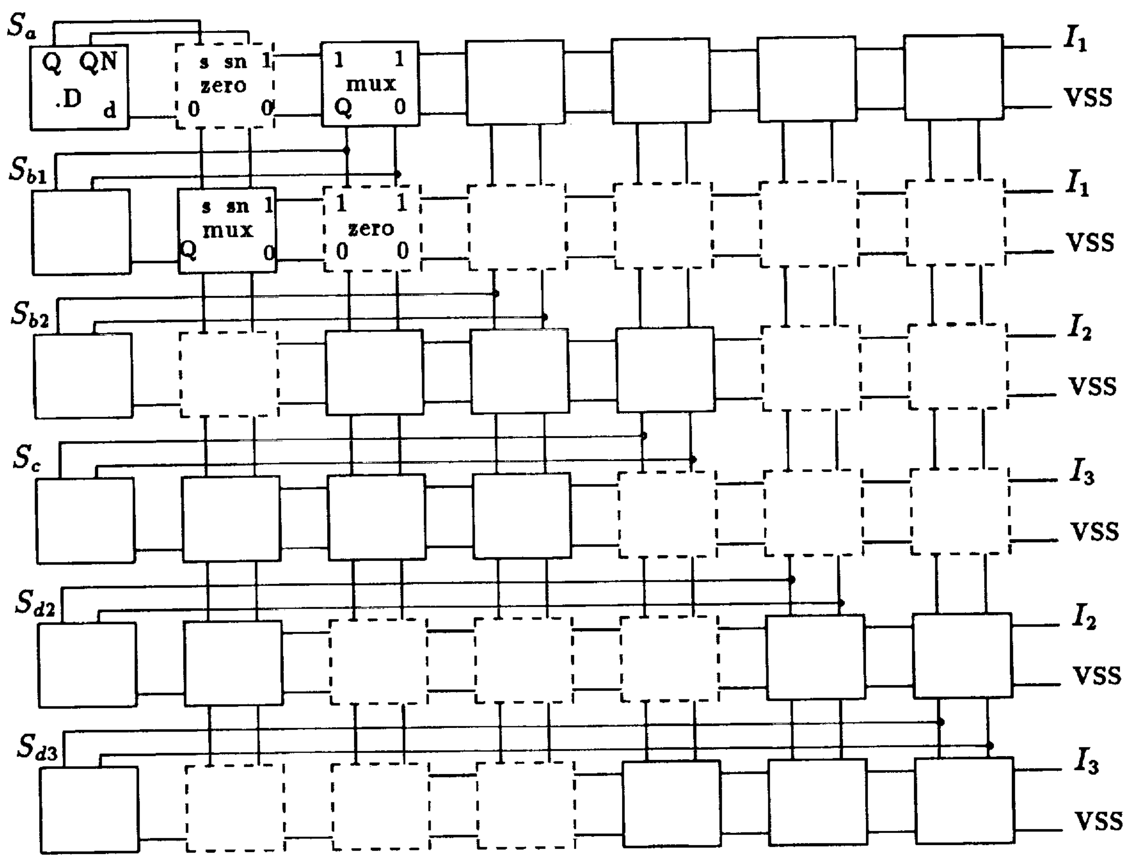
Step 3. *Generate the* **A** *matrix by setting the diagonal element in the* i^{th} *column to be* I_p *if state* S_i *appears as a next state in the flow table under* I_p.

Step 4. *Generate the* **G** *matrix such that* g_{ij} *is "1" if* S_i *is the next state of* S_j, $g_{ij} = 0$ *otherwise.*

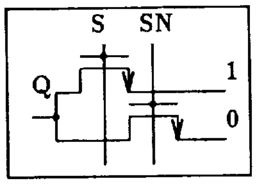
Step 5. *Map the matrices to the layout floor. Place a MUX cell under the control of* S_k *in the slice of the ILA circuit for the next state* $\hat{S}_i$ *if* $g_{ik} = 1$ *or place a ZERO cell if* $g_{ik} = 0$.

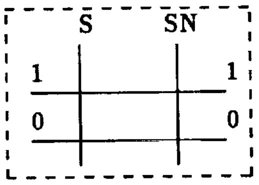
Step 6. *Connect Input-1 of the last ILA cell in the slice of the ILA circuit for* $\hat{S}_i$ *to* I_p *which is the diagonal element of matrix* A *in the* i^{th} *row. Connect Input-0 of the last ILA cell to the level of logic low (VSS).*

For example, for a synchronous machine specified a flow table shown in Table 1, it needs to find those states which are under more than one input state and to split them. The result of splitting is shown in Table 2. The matrices of the flow table can then be generated. For instance, S_a is a next state under I_1 of state $S_{b1}, S_{b2}, S_c, S_{d2}$ and S_{d3}. Then I_1 becomes the diagonal element a_{11} in matrix **A**; the 1^{st} row of matrix **G** will have a "0" in the first column since the next state of S_a under I_1 is not S_a, and have a "1" in the rest of columns. The **A** matrix and **G** matrix can then be mapped into an ILA network. The result is shown in Figure 4 (a) where each ILA cell is represented by a box. The boxes in dash line represent the ZERO cell ($g_{ij} = 0$) and boxes in solid line represent the MUX cell



(a) Synchronous ILA network





(b) ILA cell - mux

(c) ILA cell - zero

Figure 4: The ILA network for the example

($g_{ij} = 1$). As the first row of matrix **G** is "011111", the top slice of the ILA for $\hat{S}_a$ consists of one ZERO cell on the left and five MUX cells. Again, from matrix **A**, the input of the last ILA cell is tied to I_1 and VSS.

As mentioned before, a major advantage of the design approach in Procedure 1 is that it allows a hierarchical layout design. The high level layout, including interconnections, is identical for all synchronous flow tables. When the function of a flow table changes, the only thing one has to do is to instruct the computer to re-program the position of MUX cell and ZERO cell. Of course, the input state to each slice of the ILA may need to be changed as well.

7 Automated Synchronous ILA Design System

The ILA design procedure has been coded into computer programs and ported to Hewlett-Packard FUNGEN layout tool. The automatic synchronous ILA design system consists of an HP FUNGEN shell and three major subsystems:

- Sequential Logic Processor
- FUNGEN Configuration Code
- Library of Layout Building Blocks.

The Sequential Logic Processor is an ILA circuit topology generator which receives the specification of synchronous sequential machine and converts it into a form specified by FUNGEN Configuration Code. There are three phases in implementing the Sequential Logic Processor: *flow table revising, matrices generation* and *FGNRC formation*. The first two phases follows closely to the step 2, step 3 and step 4 in Procedure 1. The third phase is to generate parameters of device modules pre-defined by FUNGEN Configuration Code and write them into a FGNRC file. By modifying the last phase, the program can be ported to any other artwork generator systems.

The FUNGEN Configuration Code describes the artwork architecture and defines the modules in the FGNRC file. The FUNGEN Configuration Code is written in Fungen Configuration Language (FCL), a subset of C language with a number of functions for Hewlett-Packard TRANTOR database generation. The overall ILA architecture and a set of ILA configuration modules are specified in the FUNGEN Configuration Code.

When running the FUNGEN shell, the system invokes the FUNGEN Configuration Code, FGNRC file and Layout Library, and automatically generates a layout artwork by placing pre-designed ILA cells and peripheral buffers. It also labels all of blocks in accordance with the FUNGEN Configuration Code and FGNRC file. Figure 5 illustrates the block diagram of the ILA design system and the algorithm of Sequential Logic Processor implementation.

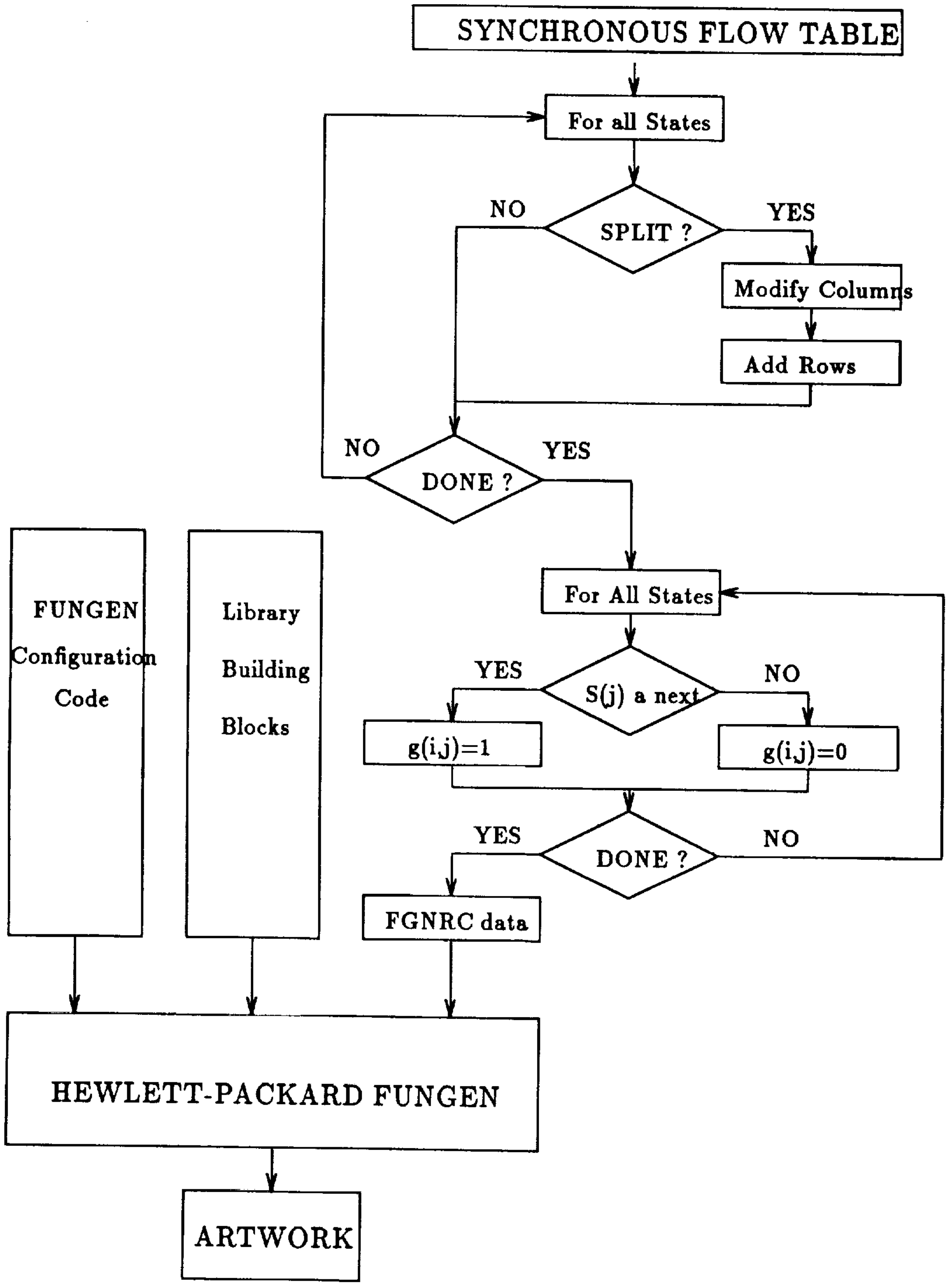


Figure 5: Block diagram of the automatic ILA design system

8 Summary

This paper presents an ILA architecture for synchronous sequential circuits. The design procedure is also proposed to realize synchronous sequential ILA circuits by programming the placement of two basic cells, a 2 to 1 multiplexer or a cell of metal wires. The interconnections between ILA cells is only a single route line in both the X and Y dimension. The simplicity and programmability of the procedure significantly reduce the effort in all stages of synchronous sequential circuit implementation, from logic design, circuit design, physical layout to verification.

The ILA design procedure utilizes matrices expression to represent design equations. One of the advantages of using matrices is that they directly indicate the placement of the ILA cells in the realization. An ILA design tool for synchronous sequential circuits has been implemented into a computer system which automatically generates layout artwork from a synchronous sequential machine specification.

References

[1] C. Roth, *Fundamentals of Logic Design*, 3rd Ed,. St. Paul, Minn., West Publishing, 1985.

[2] D. Givone, *Introduction to Switching Circuit Theory*, McGraw-Hill, Inc., 1970.

[3] S. Whitaker, "Design of Asynchronous Sequential Circuits Using Pass transistors," Ph.D Dissertation, University of Idaho, Feb. 1988.

[4] S. K. Gopalakrishnan and G. K. Maki, "VLSI Asynchronous Sequential Circuit Design", ICCD, Sept, 1990, pp. 238-242.

[5] S. Whitaker and G. Maki, "Pass-Transistor Asynchronous Sequential Circuits", IEEE JSSC, Vol.24, No.1, Feb. 1989, pp. 71-78.

[6] W. W. Stiehl, "A Mathematical Basis for the Optimal Synthesis of Finite State Machines." Master of Science Thesis, University of Idaho, Moscow, Idaho, June, 1986.