A CAD SYSTEM FOR AUTOMATIC SYNTHESIS OF GENERALIZED ASYNCHRONOUS CIRCUITS

Ming-Der Shieh, Jyh-Ming Horng, Ming-Hwa Sheu and Yu-Chin Hsu

Department of Electronic Engineering
National Yunlin Institute of Technology
Touliu, Yunlin, Taiwan, R.O.C.
E-mail: shiehm@cad.el.yuntech.edu.tw

ABSTRACT

This paper presents a unified synthesis method for dealing with generalized asynchronous circuits which comprise specifications of both Asynchronous Finite State Machines (AFSMs) and Signal Transition Graphs (STGs). By comparing the similarities and differences of two different synthesis methods, we describe algorithms and techniques underlying our CAD system for automating the design of the generalized asynchronous circuits based on the well-developed synthesis procedures of Huffman-modeled asynchronous circuits. Experimental results show that the developed system, implemented in C, can handle concurrence and sequencing of STGs and fundamental mode operation of AFSMs at the same time, therefore, it allows for a more natural and compact specification of asynchronous behavior.

1. INTRODUCTION

Asynchronous sequential logic circuits (ASLCs) have received considerable attention in recent years due to their potential for higher speed, lower average power dissipation and as clock skew problems in clocked sequential circuits continue to persist. Traditional techniques for synthesis and analysis of asynchronous circuits [1] have not been very successful because of the complex processes for ensuring the critical race-free state assignment and hazard-free operation. Recently, the synthesis of asynchronous circuits from signal transition graphs (STGs) has become an active research area since its first introduced by Chu [2]. From this event-based specification, it is suitable for synthesizing asynchronous control circuits. However, the verification and rectification for a realizable STG are non-trivial tasks [3,4,5].

Up to date, significant advances have been made in automating the design and synthesis of asynchronous circuits. However, the existing techniques use different models to synthesize ASLCs separately [5]. On the other hand, from the analysis of STG-modeled ASLCs and Huffman-modeled ASLCs, which is also referred to as asynchronous finite state machines (AFSMs), it can be shown that these two different synthesis procedures are not totally independent [6,7]. By combining both approaches, a synthesis procedure for the generalized asynchronous circuits which comprise both data inputs and control signals becomes achievable. Therefore, it allows for a more natural and compact specification of asynchronous behavior. Based on the similarities and differences of two different synthesis methods, in this paper we present algorithms and techniques underlying our CAD system, implemented in C, for automating the design of generalized asynchronous circuits based on the well-developed synthesis procedures of Huffman-modeled AFSMs from high-level specifications.

This paper is organized as follows: Section 2 defines the terminology and describes previous results related to this work. Section 3 gives an overview of the developed system. Section 4 presents the details of our algorithms and techniques for the generation of the concurrent flow table, the encoding of introduced internal signals and critical race-free state assignment. Section 5 gives some experimental results. Section 6 gives our conclusions.

2. STG- AND HUFFMAN-MODELED ASLC

2.1 Definition and Terminology

A Huffman-modeled ASLC, also referred to as an AFSM, is generally described by a 5-tuple (S,I,O,δ,ω), where S, I, O, δ and ω represent a set of internal states, inputs, outputs, next-state function, and output function, respectively. It is generally defined in terms of a *flow table* in which each column of the flow table represents an input state and each row represents an internal state. And, it is assumed that circuits are operated in the fundamental mode operation, i.e., the input cannot change until the circuit stabilizes with only one input allowed to change at a time. Moreover, critical races are eliminated by using state assignment techniques. Among existing algorithms, the totally sequential state assignment [8], which is also referred to as a unit-distance code (UDC) state assignment, and the single transition time (STT) state assignment are commonly employed [9].

The STG-modeled ASLCs can be viewed as interpreted free-choice Petri-nets [2-5], where transitions in nets are interpreted as the value changes on input/output signals of a specified circuit, and places in nets are specified by causal relations. A place can be marked with one or more tokens, meaning that the corresponding condition holds in the circuit. The causal relations joining pairs of transitions represent how the circuit and its environment can react to signal transitions. A realizable STG [2,3] must first satisfy the conditions of *liveness*, *safeness* and *unique state coding* (USC). Since a state in the state graph is a bit-vector, if two distinct state s_i, s_j have the same bit-vector assignment, a violation of the unique state coding (USC) property results. If, further, there is a transition t of a non-input signal enabled in s_i, but not in s_j (or vice versa), then a violation of the CSC property results.

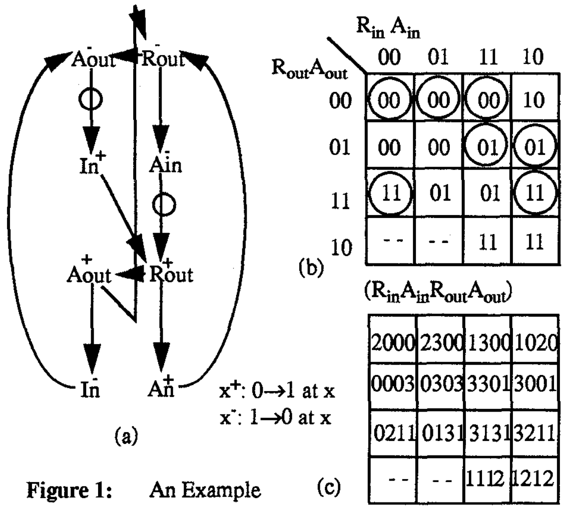
2.2 Comparisons between AFSMs and STGs

The STG without input/output concurrency can be viewed as a Moore-type AFSM with a *unity function*, i.e., the output variables are the same as state variables [6]. In addition, the signal transition graph is equivalent to the flow table with concurrency, referred to as a *concurrent flow table* (CFT). Table I concludes the results:

Table I: Comparisons between AFSMs and STGs

	STGs	AFSMs
Similarities	Input Concurrency Output Concurrency Sequential Output Change	Multiple Input Change STT State Assignment UDC State Assignment
Differences	Input/Output Concurrency Signal Protocol Graphic Specification	Fundamental Mode Fundamental Mode Flow Table

Consider the four-phase handshaking circuit in Figure 1(a). Figure 1(b) show the equivalent CFT, and an alternative representation of this CFT is represented in Figure 1(c), where the symbols 2 and 3 are used to represent the 1-to-0 and 0-to-1 transitions, respectively. In this paper, we are concentrated on the Moore AFSMs. Mealy AFSMs can also be directly translated into Moore AFSMs. And, we use the 'state' to refer to either states in AFSMs or total states in STGs, depending on the context.



$\overset{-}{\text{Aout}}$
$\overset{-}{\text{Rout}}$
In^{+}
$\overset{-}{\text{Ain}}$
$\overset{+}{\text{Aout}}$
$\overset{+}{\text{Rout}}$
$\overset{-}{\text{In}}$
$\overset{+}{\text{An}}$
x^{+}: 0→1 at x
x^{-}: 1→0 at x
(a)
(b)
(c)

$R_{in}A_{in}$

$R_{out}A_{out}$	00	01	11	10
00	00	00	00	10
01	00	00	01	01
11	11	01	01	11
10	- -	- -	11	11

$(R_{in}A_{in}R_{out}A_{out})$

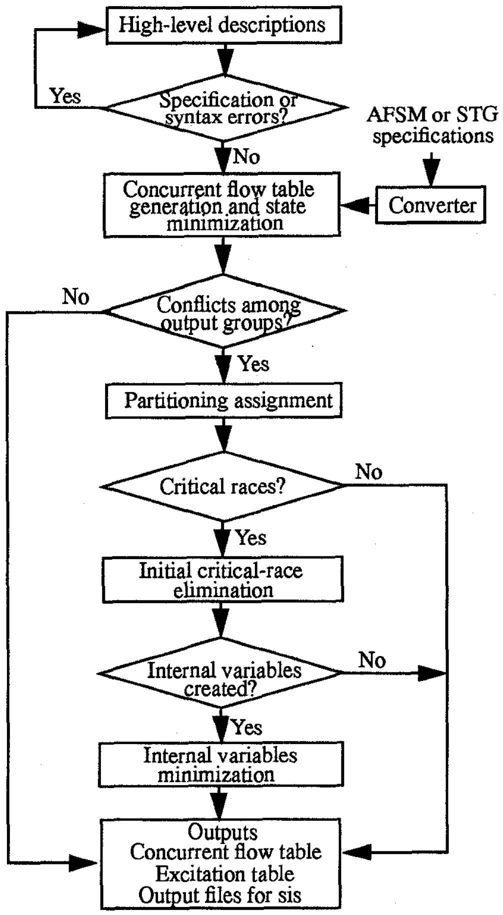
2000	2300	1300	1020
0003	0303	3301	3001
0211	0131	3131	3211
- -	- -	1112	1212

Figure 1: An Example

3. OVERVIEW OF THE DEVELOPED SYSTEM

Figure 2 shows an overview of the developed system. It provides the high-level description of a circuit's specification. The simplified description consists of input specification, output specification and functional specification. The functional specification is a collection of *if-then rules* used in the expert system. And, the general form of a rule is '*if* condition(s) *then* action(s)'. The conditions, referred to as the production rules, are used to determine what kinds of actions should be taken when the conditions are satisfied. For a deterministic circuit's behavior, three properties must be satisfied: (1) Each rule should be unique; (2) Rules should not conflict each other; and (3) One rule should not cover another rules. If any of the properties is violated, it is treated as the specification mistakes. Note that both input and output signals can be used as the triggering signals in the production rules. Four symbols 0, 1, 2, and 3 are used to represent the static 0, static 1, 0-to-1 transition, and 1-to-0 transition of the signals, respectively, to incorporate both AFSM and STG specifications. If the condition in a production rule consists of either symbol 2 or symbol 3, it is treated as the STG behavior. Otherwise, it is interpreted as fundamental mode AFSM specifications, which can be also viewed as the free-choice input condition in STG specifications.

The synthesis procedures in this system are shown in Figure 2. Techniques used in each step will be discussed in next section.



High-level descriptions
Yes
Specification or syntax errors?
No
AFSM or STG specifications
Concurrent flow table generation and state minimization
Converter
No
Conflicts among output groups?
Yes
Partitioning assignment
Critical races?
No
Yes
Initial critical-race elimination
Internal variables created?
No
Yes
Internal variables minimization
Outputs
Concurrent flow table
Excitation table
Output files for sis

Figure 2: System overview

4. SYNTHESIS OF THE GENERALIZED ASLC

4.1 Generation of the CFT

The generating strategy of a CFT is to control state transitions in the current CFT entry and generate the next CFT entry. It can be accomplished by determining which production rules should be fired by matching each pattern in the condition part of function specification with current state. If the firing condition is satisfied, then the CFT generator looks for its corresponding next-state entry in the CFT based on the actions taken. More specifically, the generation of each entry in the CFT can be achieved under the following rules: (1) If the next-state entry does not exist, a new entry is created in response to the corresponding actions executed; (2) If the next-state entry has already existed and the actions taken can be merged without conflicts, no new entry is created; and (3) If the next-state entry has already existed and the violation of CSC property is detected, a new entry is created and a different group number is appended to eliminate the conflicts.

For example, assume that a group number 0 is assigned to the initial state from which every entry in CFT is generated. If the next entry, say $(S_N, 0)$, of the current state $(S_C,0)$ resides in Rule 3, then a new entry $(S_N,1)$ is created as the next entry of $(S_C,0)$, where the notation (S,G) is denoted as a state S with a group number G. It should be noted that the process of state minimization is implicitly incorporated in Rule 2. In other words, for a current state (S_C,G_1), the CFT generator will try to search for a proper next state (S_N,G_2) to perform the state minimization, where either $G_1=G_2$ or $G_1\neq G_2$. The whole procedure is repeated until all the generated entries have been filled with the assigned values and no more firing of production rules and actions will cause new entries created in the current CFT.

As mentioned above, the production rules may consists of free choice of single-input change, sequential or concurrent changes of the input and output signals. In the former case, it is referred to as the AFSM behavior, while, the latter case is treated as the STG behavior. Therefore, the CFT generator will take the correct actions for the activated entry being checked based on the production rules. Let U be the set of incompleted entry in the CFT and MakeFsm ( ) and MakeStg ( ) be the procedures for dealing with the AFSM and STG behavior, respectively. The algorithm for generating the CFT can be stated as follows:

```
1. Find a reachable initial state for generating the CFT
2. while $U \neq \phi$
       select an entry $e \in U$
           match the current state with the production rules
           if the current status is AFSM behavior then {
               MakeFsm( ) /* firing of the single-input change */
               update the elements in U}
           else {        /* the current status is STG behavior */
               MakeStg( ) /* concurrent or sequencing change*/
               update the elements in U}
   end
```

4.2 Partitioning Assignment

As described in the generation of a CFT, whenever a violation of the CSC property is detected, a different group number is assigned to its next state to solve this problem. Since the introduced group numbers, denoted as GNs, are only symbols for eliminating conflicts, therefore, the goal for partitioning assignment is to encode GNs such that potential critical races introduced may be kept as little as possible. Note that all the critical races will be eliminated during the critical race-free state assignment discussed later. Procedures to achieve this goal can be divided into two main steps: Step 1, since the GNs are generated based on local informations, a global adjustment is performed to make most of state transitions occurred within the same group, i.e., to avoid the change of GNs during a state transition. Thus, potential critical races resulted from the change of GNs are not likely to occur. The adjustment is achieved by counting the number of transitions, defined as the transition counts TCs, occurring within different GNs in the CFT and reassign the numbers such that the total number of changes within different GNs is minimized. Note that some modifications for the existing entries may be needed to make the CFT consistent.

The next step is to encode GNs by using minimum number of bits. Before describing the technique for this assignment, we first define the adjacent relationship as follows: Two state, s_i and s_j, are adjacent if there exists at least a state transition from s_i to s_j in a CFT, or vice versus. However, since we only consider the relationship among different GNs, a simplified adjacent diagram for the partitioning assignment can be constructed with each node representing a discrete GN and each edge is weighted based on the TCs derived from the CFT. Then, the encoding process can be formulated as mapping the given adjacent diagram into the n-cube structure, where $n=\lceil \log_2 m \rceil$ for m different GNs in a CFT, with the criterion that the GNs with larger TCs are arranged to be adjacent to each other. For example, Figure 3(a) show a weighted adjacent diagram and its corresponding mapped n-cube structure is shown in Figure 3(b).

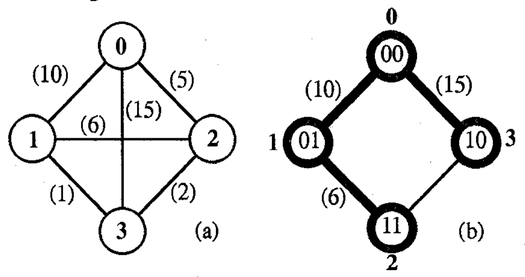


Figure 3: An example for partitioning assignment

4.3 Critical-Race-Free State Assignment

When critical races occur in a circuit, the circuits may fail to get the correct response. As a result, all critical races should be eliminated after partitioning assignment. Note that a state is now represented by the combination of input, output and internal signals after partitioning assignment. By combining the concept of STT and UDC state assignments, we present a simple but efficient method to reduce the complexity of critical race eliminations. First, we restate the definition of a transition pair in STT as follows. A transition pair TP_{ij}, denoted as $[S_i,S_j]$, consists of one total state S_i and its corresponding next total state S_j after firing of specified signals. Due to the unequal transmission delays in a circuit, some intermediate states may be involved during a concurrent change of signals in TP_{ij}. Therefore, the process for detecting critical races can be stated as follows: Expand all the TP's so that the implied intermediate states become visible. For unavoidable conflicts, it implies the existence of critical races.

In order to reduce the complexity, we accomplish the task of critical-race elimination by going through a two-pass process. Pass 1, if a critical race is detected after the expansion of TP's, an extra internal variable is added to solve the conflict, i.e., we force one of the state transition to go through a different plane to skip the conflicts. In order words, the internal variable is used to distinguish these two different TP's. By using this concept, we can guarantee that no more critical races will result from this modification. Note that the same internal variable can still be used to eliminate the conflicts among other TP's. For example, as shown in Figure 4(a), assume that there exists two TP's [000,010] and [100,001]. After the expansion of the TP's, it can be found that critical races may occur at the state (000), which is an intermediate state in [100,001], Therefore, one extra internal variable is needed to distinguish these two TP's. By using this created internal variable, the modified state transition is illustrated in Figure 4(b). Therefore, the critical race can be eliminated. In other words, the state transition for [000,010] is resided on plane 0 and the state transition [100,001] is forced to go through plane 1, where the plane b ∈ {0,1} means that the value of the internal variable is set to b.

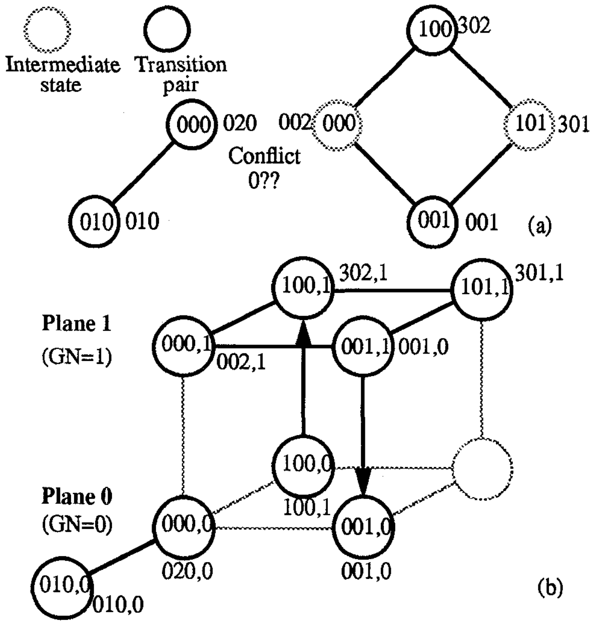


Figure 4: Process for critical-race elimination

After eliminating all the critical races, the second pass can be used to reduce the number of added internal variables. The approach is trying to project all the state transitions from plane i to plane j, where i > j. If it can be achieved, then its corresponding internal variable can be eliminated. Figure 5 shows the derived state transitions in Figure 4(b).

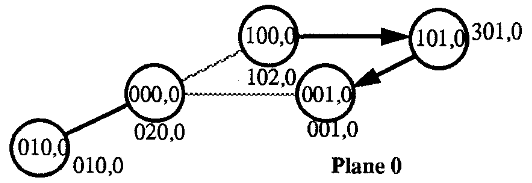


Figure 5: A result after internal variable reduction

5. EXPERIMENTAL RESULTS

The synthesis procedures discussed in this paper have been implemented in C on Sun workstation and applied to numerous practical examples. Experimental results show that it can efficiently handle STG specification for most of the examples and reduce the number of internal variables in AFSM example. Some of the experimental results are shown in Table II.

Table II: Experimental results

Examples	No. of states in CFT	No. of inputs	No. of outputs	No. of internal variables
full.g	8	2	3	0
sbuf_read_ctl.g	10	3	5	0
nowick.g	8	3	3	0
vb24a.g	8	3	3	0
ebergen.g	8	2	3	0
fsm1.hl	8	2	2	3
fsm.hl	6	2	2	1
fsm3.hl	2	2	2	0
fsm4.hl	9	2	3	2
fsm5.hl	20	3	3	5

6. CONCLUSIONS

In this paper, a CAD system is developed to simplify the design of generalized asynchronous circuits. Although some of the techniques described in paper are heuristic, experimental results show that the developed system can efficiently handle a large number of practical examples. In conclusion, the major contributions of this paper can be summarized as follows: (1) provide a unified approach for synthesizing generalized ASLCs which can handle concurrency, free-choice, and sequencing of STG and fundamental mode AFSM at the same time; (2) provide a simple but efficient technique for critical-race elimination; and (3) develop a CAD system for automating the design of asynchronous circuits.

REFERENCES

[1] Unger, S.H., *Asynchronous Sequential Switching Circuits*, wiley interscience, 1969.
[2] Chu, T.A., "Synthesis of self-timed VLSI circuits from graph theoretic specifications," ph.D. dissertation, MIT, June 1987.
[3] Moon, C.W., Stephan, P.R. and Brayton, R.K., "Synthesis of hazard-free asynchronous circuits from graphical specifications," ICCD, pp.322-325, Nov. 1991.
[4] Lin, K.-J. and Lin, C.-S., "On the verification of state-coding in STGs," ICCAD, pp.118-122, 1992.
[5] Hauck, S., "Asynchronous design methodologies: An overview," Proceedings of the IEEE, vol.83, no. 1, pp. 69-93, Jan. 1995.
[6] Wey, C.-L., Shieh, M.D. and Fisher, P.D., "ASLCScan: A scan design technique for asynchronous sequential logic circuits," ICCD, pp.159-162, 1993.
[7] Lavagno, L., Moon, C.W., Brayton, R.K. and Sangiovanni-Vincentelli, A.L., "An efficient heuristic procedure for solving the state assignment problem for event-based specifications," IEEE Trans. on Computer-Aided Design, vol. 14, no. 1, pp. 45-60, Jan. 1995.
[8] Hazeltine, B., "Encoding of asynchronous sequential circuits," IEEE Trans. on Computers, pp. 727-729, 1965
[9] Tracey, J.H., "Internal state assignments for asynchronous sequential machines," IEEE Trans. Electronic Computers, EC-15, pp.551-560, Aug. 1966.