International Journal of Electrical and Power Engineering 2 (3): 134-146, 2008
ISSN: 1990-7958

Race-Free State Assignment Technique for Asynchronous Circuits

Abdelkrim Zitouni and Rached Tourki
Laboratory of Electronics and Micro-Electronics,
Faculty of Sciences of Monastir, 5019, Monastir, Tunisia

Abstract: One of the most growing areas in circuit design is asynchronous circuit's design. These circuits incorporate a number of hazard problems such as race. The multitude of the different techniques that have been proposed to avoid race problems in an asynchronous circuit can be classified into two major types: The techniques belonging to the first type yield large circuits with low power dissipation and the techniques belonging to the second type yield relatively small circuits with high power dissipation. The majority of these techniques are NP-complete and require extremely large computation times for large flow table. This study considers the state assignment problem for circuits operating in the normal fundamental mode and describes a new procedure especially suited to the automated synthesis of large circuits. The proposed technique constitutes a trade-off between silicon area and power dissipation.

Key words: Asynchronous circuits, race-free, technique, assignment, power dissipation

INTRODUCTION

Asynchronous sequential circuits are used extensively in digital integrated circuits and systems as Globally Asynchronous Locally Synchronous System on Chip (GALS-SoC) arbiters, wrappers and routers. During the last decade, there has been significant progress in developing methods and tools for asynchronous circuit synthesis (Sentovich *et al*., 1992; Chris, 1995; Steven, 1993; Chantal *et al*., 1994; Cortadella *et al*., 1997). This progress can be classified into two synthesis approaches, one based on the Huffman's state machine model (Huffman, 1954; Unger, 1996) and the other deriving from the Muller's concept of speed-independent circuits (David and Bartky, 1959). The former, also known as fundamental mode circuit design makes strong assumptions about the delay of the environment compared to that of the circuit. It requires that the environment to be slow enough in applying the new input values so as to allow the circuit to stabilize after responding to the previous input. The most well-known method associated with this approach is the one called Burst-Mode (BM) circuit design, developed by Steven (1993), Coates *et al*. (1993) and Yun (1994). The second approach on the contrary, makes no assumption about the delays of the environment, permitting some of the inputs to switch in response to changes in some of the circuit's outputs, without waiting for their complete stabilization. This model is called Input-Output (IO) mode. The recently developed design methods and software based on Signal Transition Graph (STG) (Cortadella *et al*., 1997; Moon *et al*., 1991) exemplify this approach and produce speed-independent circuits, whose behaviour are invariant to delays in gates but may be sensitive to wire delays.

The complexity of the state assignment procedure for asynchronous sequential circuits stems from the necessity of avoiding critical races, which may cause the machine to malfunction. A critical race occurs when, due to the asynchronous nature of state transitions, internal state variables may change values in an order which allows the circuit to reach a final stable state other than the intended one. It is well-known (Friedman *et al*., 1969; Liu, 1963) that critical races can be avoided by judicious choice of encoding.

The history of state assignment procedures dates back to the 1950's when relays were used as the main circuit components. In the Huffman's classical papers on sequential circuit design (Huffman, 1954, 1955), he first presented a mathematical model for sequential circuits (to become known as the "Huffman model") and then presented several state assignment procedures.

Starting from Huffman's works, asynchronous state assignment techniques have been widely studied by many researchers (Tracey, 1966; Tan, 1971; Liu, 1963; Maki *et al*., 1969; Friedman *et al*., 1969; Smith, 1974; Nanya and Tohma, 1978, 1979; Kuhl and Reddy, 1978; Hollaar, 1982; Maki and Tracey, 1971; Saucier, 1967, 1972; Unger, 1969; Kantabutra and Andreou, 1994).

Corresponding Author: Abdelkrim Zitouni, Laboratory of Electronics and Micro-Electronics, Faculty of Sciences of Monastir, 5019, Monastir, Tunisia

Most of the research relates to the Single Transition-Time (STT) state assignment technique for fundamental-mode. The fundamental mode assumption states that only one digit in the input may change at a time and no changes can occur in the input until the state machine stabilizes. STT state-assignment techniques were first presented by Liu (1963) and later extended by Tracey (1966). Other improvements or variations on STT state assignments were investigated by many researchers Tan (1971), Maki *et al.* (1969), Friedman *et al.* (1969), Smith (1974), Nanya and Tohma (1978, 1979) and Kuhl and Reddy (1978). Tracey's (1966) methods are very important representative approaches of STT state assignments. However, these methods require extremely large computation times for very large flow tables (Smith, 1974). Smith (1974) presented an extension of Tracey's techniques that produces near-minimum state assignments for very large tables, while requiring much less computational effort than previous methods. Systematic technique to solve the state assignment problem rather than the partition method (Tracey, 1966; Tan, 1971; Smith, 1974; Kantabutra and Andereou, 1994) used in the STT state assignment approach.

In this study, we present a new technique for resolving race problems of asynchronous circuits based on variable insertion scheme. The proposed technique begins by fetching the best state coding that respects the maximum of adjacency and try to create non-critical races by assigning racy state codes to the transitions that have the lowest transition probabilities. After that, an internal variable is inserted per race transition into the transition diagram.

The proposed technique differs fundamentally from the Huffman's. Huffman technique starts out by defining a large set of states to correspond to each flow table row. The proposed technique, on the contrary, start out small and attempt to successively refine the given numbered adjacency graph by inserting a set of internal variables according to the race transitions. The simplicity of the proposed technique allows him to be used even for large flow tables.

RACE-FREE STATE ASSIGNMENT TECHNIQUE

The first step of the proposed technique consists on numbering the states in the adjacency graph in such a way that the number of race transitions is minimized, or kept small. It may be possible to number the nodes so that every pair of adjacent nodes has numbers that differ at only 1 bit position. In such ideal cases, our state assignment algorithm does nothing; the attempt to use the adjacency graph as the state transition diagram is successful. In less ideal cases, the graph node numbering will leave some adjacent node pairs with numbers that differ from each other at 2 or more bit positions (race transition). The term "race transition" comes from the fact that if we were to build a circuit directly from the graph, then such a state transition would involve more than one state variable change, entailing a race condition. In this case, our state assignment algorithm inserts an internal variable for each race transition and modifies some conditions of transitions according to it. In all cases the total number of secondary state variables is n (2^n states), the minimum possible number of variables that we could use just to number all the internal states (rows of flow table) with. For adjacency graphs that don't have many race transitions, the logic that is added due to internal variables insertion will be small. We suspect that many circuits fall into this category. The total circuits will be also small relatively to the smallest circuits (Huffman, 1955) that are generated by Huffman technique ($2n = 2\log_2(N)$; where N is the number of rows of the flow table) and the Kantabutra circuits (Kantabutra and Andreou, 1994) which use (n+b) internal state variables; where b represents the race transition type, thus leading to $2^{(n+b)}$ internal states.

Case of one race circuits: To study the details of our state assignment algorithm in the case of 1 race transition, let us study the following examples:

Example 1: Consider the flow table in Fig. 1b, which corresponds to an asynchronous sequential circuit.

We create a race-free state transition diagram in a step-by-step manner: First, we number the adjacency graph in such a way that the number of race transition is minimized. Figure 1a presents the adjacency graph with a state coding that respects the maximum of adjacency. Note that the transition from the state (11) to the state (00) is a race transition. In fact, when the machine is in the stable state (11) and the input variables becomes $x_1x_0 = 11$, the machine can go to the intermediary state (01) if y_2 is fast than y_1 or to the intermediary state (10) if y_1 is fast than y_2 (Fig. 1b). Since the states (01) and (10) are stable in the column ($x_1x_0 = 11$), then the machine will stabilises in one of these state depending on the rates of y_1 and y_2 and never attain the target state (00). Our idea consists on leaving the race transition occurs and when the machine attempts to reach a wrong state depending on the state variable rates, we redirected the machine to its target state by: setting and resetting an inserted variable, add some transitions and modify some transition conditions. For example when the machine is in

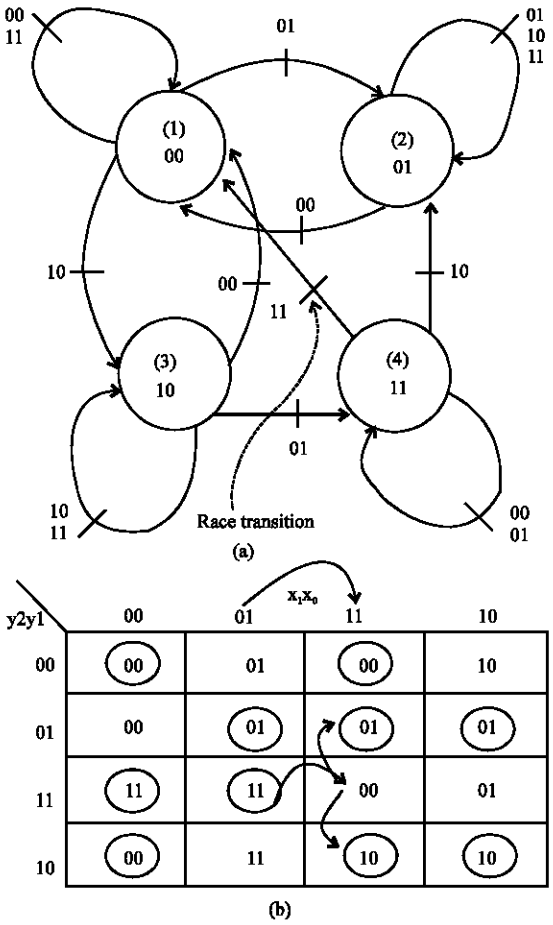

Fig. 1: Example of an adjacency graph (a) and its corresponding flow table (b)

the stable state (11) and the input vector changes from (01) to (11): raises the added variable V to 1; multiply logically the transition condition by $\overline{V}$ when the machine is in the state (01) or in the state (10) and the input vector is (11); add a transition from the state (01) to the state (00) and a transition from the state (10) to the state (00) both with the transition condition ($x_1x_0V = 111$) and reset the variable V when the machine attains the state (00). Thus, the machine can not stabilise in the state (01) or in the sate (10) when it starts from state (11) and the input vector is ($x_1x_0 = 11$).

The state transition diagram and the flow table are modified as presented in Fig. 2.

Notice that the inserted variable V is putted to 1 when ($y_2y_1x_1x_0 = 1111$) and is reset to 0 when ($y_2y_1x_1x_0 = 0011$). It remains unchanged in all other cases. The C-element based production rule of this variable is:

$$\begin{cases} y_1 y_2 x_1 x_0 \rightarrow V^+ \\ \overline{y}_1 \overline{y}_2 x_1 x_0 \rightarrow V^- \end{cases}$$

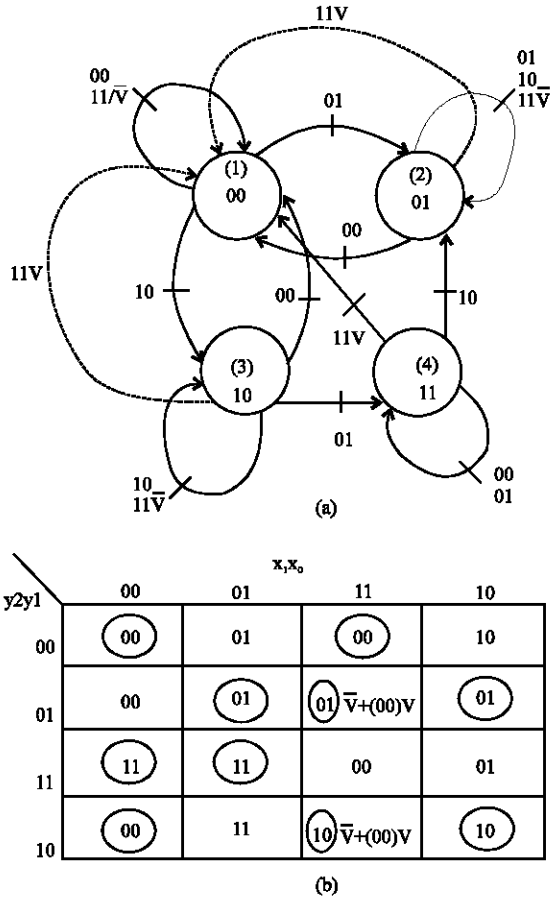

Fig. 2: The modified adjacency graph (a) and the modified flow table (b)

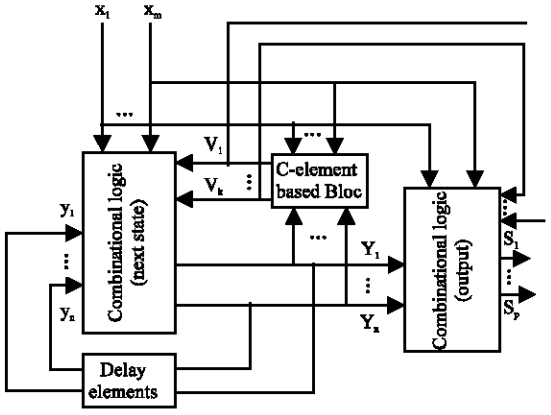

Fig. 3: Architecture of a Mealy machine allowed by our technique

The general architecture (generated by our technique) of a Mealy asynchronous circuit with m inputs, n state variables, p outputs and k race transitions is presented in the Fig. 3.

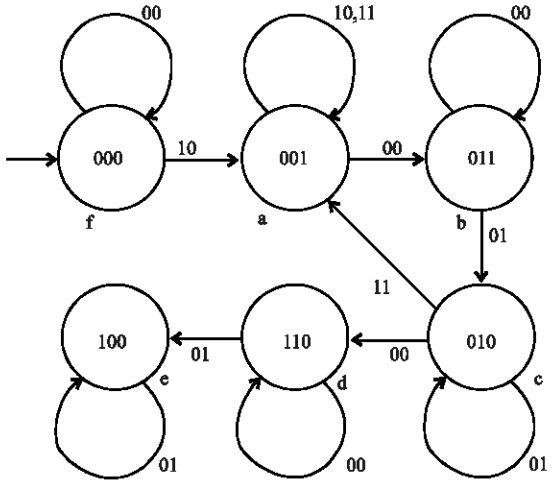

Fig. 4: The initial adjacency graph

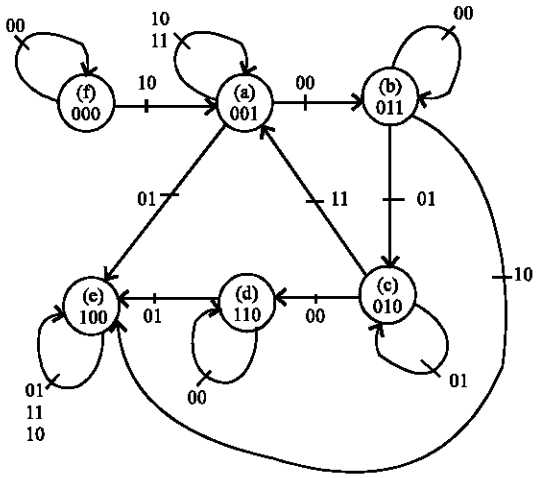

Fig. 6: The input adjacency graph of example 3

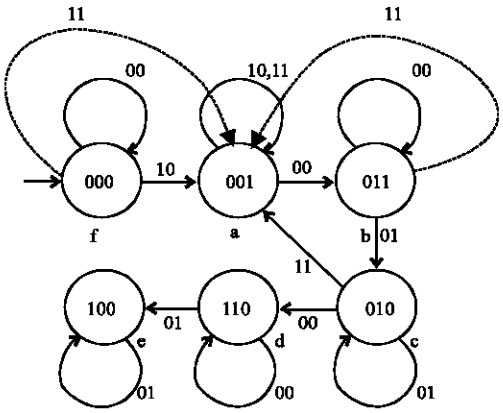

Fig. 5: The race free adjacency graph by our algorithm

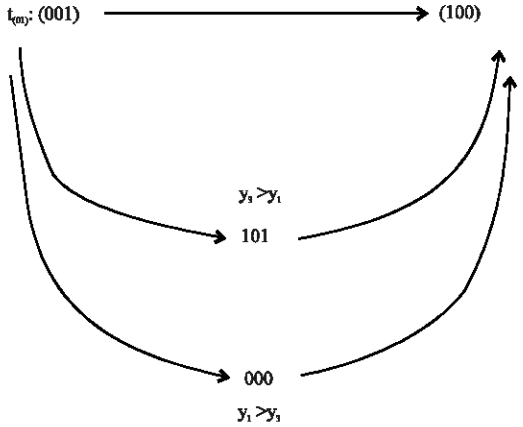

Fig. 7: The intermediate states for the transition $^{t}(01)$

Example 2: Consider the adjacency graph presented in Fig. 4 extracted from Kantabutra and Andreou (1994). Note that the state (010) and the state (001) differ by two state variables. When the machine is stable in the state (010) and the transition (11) is activated the machine can go to the intermediate state (000) or to the intermediate state (011). From the adjacency graph we notice that the transition (11) is not used when the machine is in the state (000) or in the state (011). Thus we can forces the machine to transit to the state (001) when it passes by the state (000) or by the state (011). Here we create a non critical race for the transition (11) without adding an internal variable. The modified adjacency graph contains only 3 state variables (the necessary variables to number the flow table row) and it is presented in Fig. 5. When we use the Kantabutra technique as presented by Kantabutra and Andreou (1994), the modified graph uses 4 state variables. Also if we had used the state assignment algorithm by Huffman (1955) that yields fast circuits, we would have obtained a 7-state variable circuit instead of the 3-state-variable circuit as we have just gotten.

Case of multi-race circuits: The proposed technique can be used in the case of multi-race circuits even when the state codes of adjacent nodes differ by more than two bits positions. Once again, we will study an example to get an idea of the general algorithm in which there may be more than one race transition.

Example 3: Consider the example of Fig 6 as input of our algorithm. Let $^{t}(xy)$; (A) → (n) be a transition from state (A) to state (B) when the transition condition on the inputs variables is (yx). In this example we have 3 race transitions:
$^{t}(11)$; (010) → (001), $^{t}(10)$; (011) → (100) and $^{t}(01)$; (001) → (100). In the case of the transition $^{t}(11)$ and the transition

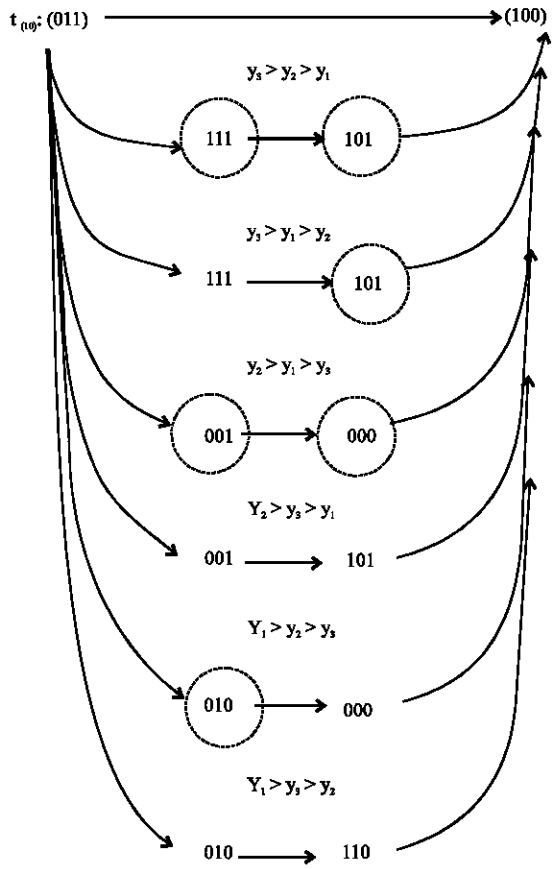

Fig. 8: The intermediate states for the transition $^t(10)$

$^t(01)$ the codes of the source and the target states differ only by 2 state variables. The machine can go to 2 intermediate states in each case depending on the rates of y1 and y2 in the case of the transition $^t(11)$ (y3 steel unchanged) and depending on the rates of y1 and y3 (y2 steel unchanged) in the case of the transition $^t(01)$. The inequality (y_i>y_j) implies that the rate of y_i is higher than the rate of y_j. The resolution of the race problem in the case of the transition $^t(11)$ is resolved in the same way as performed in the case of example 2. In fact, the transition (11) is not used when the machine is in the intermediate states (000) and (011). Concerning the transition $^t(01)$ the intermediate states are presented in Fig. 7.

In the case of the race transition $^t(01)$ the codes of the source and the target states differ by 3 state variables. The possible intermediate states that the machine can reaches are encircled in Fig. 8.

Let V_1 and V_2 be, respectively the internal variables to be inserted for the race transitions $^t(01)$ and $^t(10)$. For each transition in the race transition set S($^t(01)$, $^t(10)$) the algorithm puts the variable V_i to 1 when the machine is in the source state of the corresponding race transition and

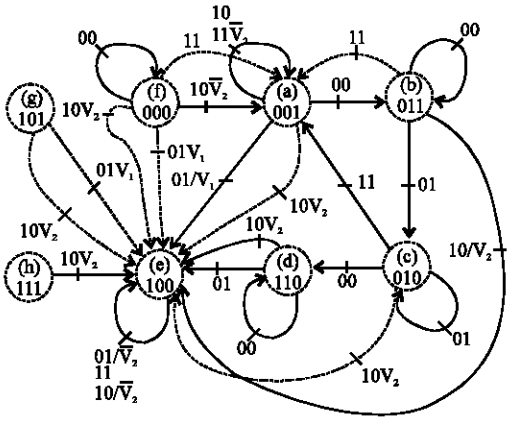

Fig. 9: The race free adjacency graph by our algorithm

the transition condition is activated. For example in the case of the transition $^t(10)$, the setting of the variable V_2 ($10/V_2$) is performed when the machine is in the source state (011) and the transition condition (10) is true. The resetting of this variable ($10/\bar{V}$) is performed when the machine is in the target state (100) and the transition condition is true.

In the second step, we create a transition from each state of the intermediate states set to the target state of the corresponding race transition. The transition condition that is associated to these transitions is ($10V_2$). This allows the machine to go to the target state as soon as it reaches a given intermediate state due to the race problem. We modify all the existents $^t(10)$ transition condition by forming a logic and of the transition condition with $\bar{V}_2$. For example the transition condition from the intermediate state (000) to the state (001) becomes ($10\bar{V}_2$). This implies that if the machine is stable in the state (000) and its precedent state is not the source state of the race transition (V_2 is low) it can transit to its intended target state (001). Note that if a given intermediate state does not exist in the original adjacency graph we will add it (eg. state (101) and (111)). This situation may be encountered when the adjacency graph contains a number of states that is less than the possible states number that can be generated by a code with n bits (2^n). Note that in all cases our technique uses just the number of states that is necessary to code the flow table. The final race free adjacency graph generated by our algorithm is presented in Fig. 9.

Adjacency graph with output: In this example we present how our technique can study the case of a finite state machine with outputs. Both Mealy (output are associated to the transition) and Moore (output are associated to the state) machines are considered.

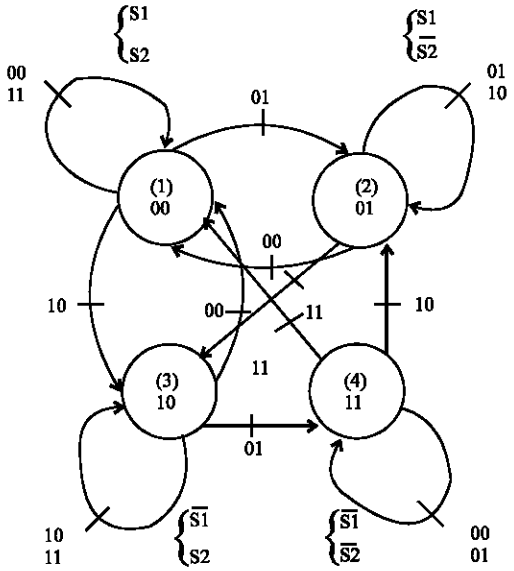

Fig. 10: Adjacency graph with output (Moore machine)

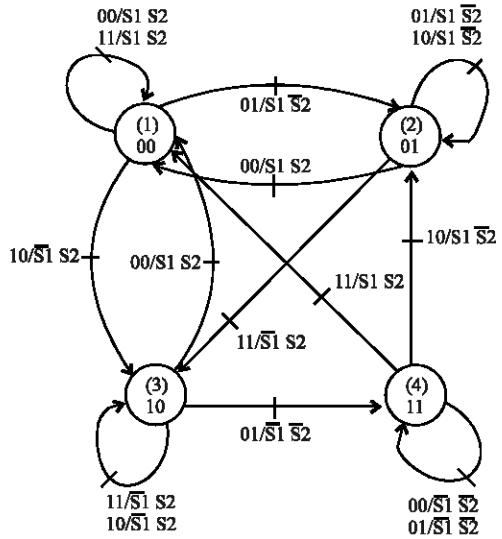

Fig. 11: Adjacency graph with output (Mealy machine)

Example 4: Consider the adjacency graph presented in Fig. 1.b, where we add a race transition from the state 01 to the state 10 (^{t}(11): (01) → (10)). We also add two outputs S_1 and S_2 that are affected in each state of the adjacency graph as presented in Fig. 10 (in the case of a Moore machine) and Fig. 11 (in the case of a Mealy machine).

The modification of the adjacency graph corresponding to the race transition 10 (^{t}(11): (01) → (10)) and 10 (^{t}(11): (11) → (00)) are performed as presented in the precedent sections by inserting two variables V_1 and V_2. We notice that it is necessary that the modification according to internal variables insertion don't affect the

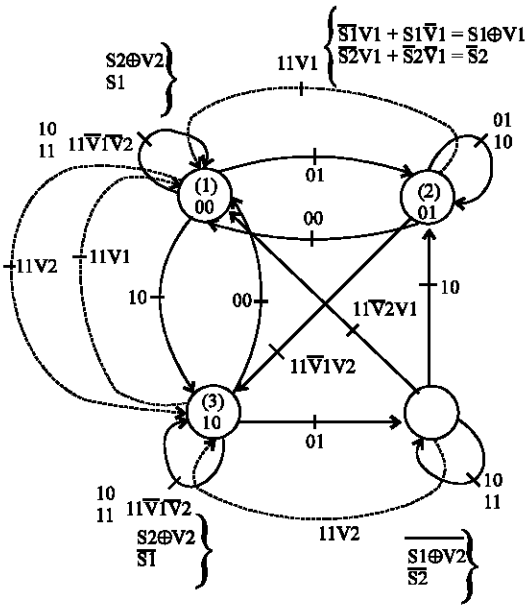

Fig. 12: Output adjacency graph with outputs affectation by our algorithm (Moore machine)

outputs values of the machine. For this reason we modify the outputs values depending to the internal variables insertion. For example in the case of the race transition 10 (^{t}(11): (11) → (00)), the outputs will change from the values (S_1S_2 = 00) to the values (S_1S_2 =11). In order to not change the values of these outputs when the machine reaches the intermediate states (01) and (10), these outputs will conserve their values in the initial state of the race transition (when V_1 is active). Also these outputs will conserve the outputs values of the state (01) or (10) when these states are reached from states other than the source state of the race transition (V_1 is not active). In the case of a Moore machine (Fig. 12) the logic expression:

$$\overline{S}_1 V_1 + S_1 \overline{V}_1 = S_1 \oplus V_1$$

for the output S_1 and the logic expression

$$\overline{S}_2 V_1 + \overline{S}_2 \overline{V}_1 = \overline{S}_2$$

will ensure these conditions. We notice that since the output S_2 will be (S_2=0) when the intermediate state is a stable state or a transitory state of the race transition ^{t}(11): (11) → (00) its value before the modification ($\overline{S}_2$) of the adjacency graph will be conserved

$$(\overline{S}_2 V_1 + \overline{S}2\overline{V}_1 = \overline{S}_2).$$

In the case of a Mealy machine (Fig. 13), the outputs associated to an added transition from an intermediate state of a race transition will be affected by the output of the target state of the race transition. For example, the

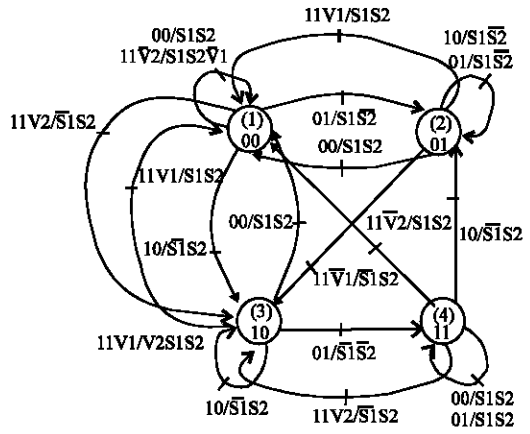

Fig. 13: Output adjacency graph with outputs affectation by our algorithm (Mealy machine)

outputs of the added transition from the state (01) to the state (00) will have the same values as the outputs of the state (00) (11V1/S1S2). The outputs of the transitions other than the added transitions are not affected.

State assignment algorithm: The following sections describe the steps allowed by the proposed algorithm (Fig. 14) that takes the state transition diagram as input.

Internal variables generation: Let the transition $t^k s_i \rightarrow s_j$ from a state S_i to a state S_j, when there is at least two bits change, be a race transition and $C_t{}^k{}_{Si} \rightarrow s_j$ be the transition condition associated to this $t^k s_i \rightarrow s_j$ transition. For each transition, the algorithm generates an internal variable V_k. Thus, a set V $\{v_1,...v_K,...v_m\}$ of internal variables to be added to the adjacency graph for all race transitions will be created.

Set/reset of state variables: For each race transition $t^k s_i \rightarrow s_j$ from a state S_i to a state S_j, set the variable V_k that is associated to the $C_t{}^k{}_{Si} \rightarrow s_j$ transition condition and add this setting assignment as a Mealy notation to the $C_t{}^k{}_{Si} \rightarrow s_j$ condition ($C_t{}^k{}_{Si} \rightarrow s_j$/vk). The others state transitions conditions from state S_i will not be modified.

For the same transition condition $C_t{}^k s_i \rightarrow s_j$ from the target state S_j to the same state S_j or to another state, the algorithm resets the variable V_k and add this resetting assignment to the $C_t{}^k s_i \rightarrow s_j$ condition

$$(C_t{}^k s_i \rightarrow s_j)$$

Generate the intermediate states: For each race transition $t^k s_i \rightarrow s_j$ generate all the intermediate states that can be reached between the state S_i and the state S_j depending on all inequality combinations of the next state variables

```
Algorithm State_Assignment;
Begin
For each t^K s_i →s_j ∈ T do
      If S_i and S_j are not adjacent then
      Add t^K s_i →s_j to T_R;
      Add V_k to V;
      End if;
End for;
For each t^K s_i →s_j ∈ T_R do
      Replace C^t Ks_i →s_j from state S_j by C^t Ks_i →s_j/vk;
      Replace C^t Ks_i →s_j  from state S_j by C^t Ks_i →s_j/vk;
End for;
For each state S_p ∈ [S_i; S_j] depending on the state variables
     rates combinations do
     If S_p ∉ R_k then
        Add S_p to R_k;
     End if;
End for;
For each S_p ∈ R_k do
      If there exist a t^K s_i →s_j from the state S_p then
   If this transition don't goes to S_j then
            Add a transition from S_p don't goes to S_j
     with the transition condition (C^t Ks_i →s_j vk);
       Replace the exists transition condition by C^t Ks_i →s_j  v̄k ;
      End if;
     Else
            Add a transition from S_p don't goes to S_j
     with the transition condition (C^t Ks_i →s_j vk);
      End if;
End for;
Minimize the inserted variables;
End State_Assignment;
```

Fig. 14: Race-free state assignment algorithm

rates. Add these states to the adjacency graph. If a given state is added to the adjacency graph according to another race transition do not duplicate it.

Add transitions/modify transition conditions: For each state Sq from the intermediate state set of the race transition, $t^k s_i \rightarrow s_j$ add an edge from the state Sq to the state S_j with the transition condition $vkCt^k s_i \rightarrow s_j$ (a logic and between the variable V_k and the race transition condition). Do not perform this task if the existent transition condition ($C_t{}^k s_i \rightarrow s_j$) allows to the machine to reaches the state S_j.

Modify the existent or the generated $Ct^k s_i \rightarrow s_j$ transition condition from each state Sq among the intermediate state set by forming a logic and between the $\bar{v}k$ variable and the $Ct^k s_i \rightarrow s_j$ condition

$$(\bar{v}kC_t{}^k s_i \rightarrow s_j)$$

Optimization: As indicated in the above sections, it is not necessary to add a V_k variable if the race transition condition $C_t{}^k s_i \rightarrow s_j$ does not already exist for all intermediate states. In the first step, our algorithm inserts all possible variables and only in the final step it

eliminates the unnecessary variables and their associated transition conditions modifications. In certain situation, the algorithm eliminates only some transition conditions from some intermediate states corresponding to the race transition of a given variable without eliminating the variable itself. In fact the insertion of an internal variable according to a given race transition can create a transition condition from an intermediary state of another race transition that have the same transition condition which don't exist in the graph. This step constitutes the variable optimization step.

CORRECTNESS VERIFICATION

The functionality correctness of our algorithm has been verified by simulating a set of industrial circuit's examples. In each example, the input test vectors have been chosen in a manner that let the machines passes by all race transitions. The first simulation is performed on the initial transition diagram, without race problem resolution. All race transitions have been shown based on the state variable rate combinations. In each case we slow a state variable than the others by retarding it with some delay. Different delays are also used in each case. The second simulation is performed on the state diagram after the race problems have been resolved by our algorithm.

Race transition simulation: Consider the example of the Moore machine of Fig. 12. The outputs and the next state equations are extracted from the Karnaugh maps and described by a data-flow VHDL description. The next state variable Y_2 and the output S_2 after the race problem resolution are extracted as follow (Fig. 15).

The simulation results presented in Fig. 16 shows the race transition from the state (11) to the state (00) by retarding the y_2 state variable than the y_1 state variable by 2 ns. The machine goes to the state (01) instead of the state (00). The values of the output S_1 are wrong duo to this race. Figure 17 shows the simulation results of the circuit without race with the same test vector after applying our technique. Note that if we slow the y1 state variable than the y_2 state variable, the machine enter in an infinite cycle between the state (10) and the state (11). In fact, if the machine enters into the state (10) and the input transition is steel active (x_1x_0 = 11) the machine return to the state (11) and so on. This situation is also resolved by our technique showing thus the same simulation results as those of Fig. 17. Notice that the inserted variables V_1 and V_2 are presented in order to shown how they are activated and deactivated when the race transition is activated.

Hazard freeness verification: The next state equation of the variables V_1 and V_2 of the Moore machine example of

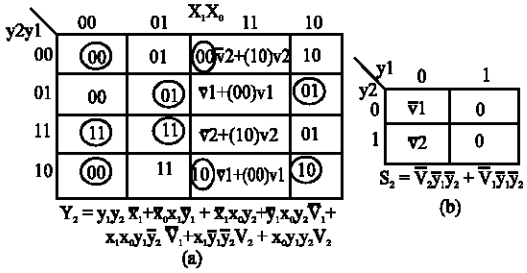

Fig. 15: Equations of the next state variable Y_2 (a) and the output variable S_2 (b), after race resolution

Fig. 12 are generated from the C-element production rules as follow.

$$\begin{cases} y_1 y_2 x_1 x_0 \overline{V}_2 \rightarrow V_1^+ \\ \overline{y}_1 \overline{y}_2 x_1 x_0 \overline{V}_2 \rightarrow V_1^- \end{cases}$$

$$\Rightarrow V_1 = y_1 y_2 x_1 x_0 \overline{V}_2 + v_1 \overline{(\overline{y}_1 \overline{y}_2 x_1 x_0 \overline{V}_2)}$$

Figure 18 presents the two level logic implementation of the variable V_1.

Notice that the presented logic structure can be generalized for each state machine. For example, in the case of a machine with n internal state variables, m input variables and p race transitions; we have p state variables to be added to the state graph. Let

$$(y_n^s \ldots y_1^s) \text{ and } (y_n^t \ldots y_1^t)$$

and be, respectively the binary values of the source and the target states variables codes of the i^{th} race transition. Let also $(x_m \ldots x_1)$ be the values of the variables of the race transition condition and $(V_p, \ldots, V_1)$ the variables to be added to the state graph according to the p races transitions. The production rule for a given V_i variable is presented as follow:

$$\begin{cases} (y_n^{s^*} \ldots y_1^{s^*})(\overset{*}{x}_m \ldots \overset{*}{x}_1)(\overline{V}_p \ldots \overline{V}_{i+1}\, \overline{V}_{i-1} \ldots \overline{V}_1) \rightarrow V_i^+ \\ (y_n^{t^*} \ldots y_1^{t^*})(\overset{*}{x}_m \ldots \overset{*}{x}_1)(\overline{V}_p \ldots \overline{V}_{i+1}\, \overline{V}_{i-1} \ldots \overline{V}_1) \rightarrow V_i^- \end{cases}$$

Notice that if x_m = 0 thus $x_m^* = x_m$ and if x_m = 1 thus $x_m^* = \overline{x}_m$. The two level logic equation extracted from the above production rule is shown by the following equation.

$$V_i = (y_n^{s^*} \ldots y_1^s)(\overset{*}{x}_m \ldots \overset{*}{x}_1)(V_p \ldots \overline{V}_{i+1}\, \overline{V}_{i-1} \ldots \overline{V}_{i-1})$$
$$v_i y_n^{-t^*} + \ldots + v_i y_n^{-t^*} + v_i \overline{x}m^* + \ldots + v_i \overline{x}_1^* +$$
$$v_i V_p + \ldots + v_i V_{i+1} + v_i V_{i-1} + \ldots + v_i V_1$$

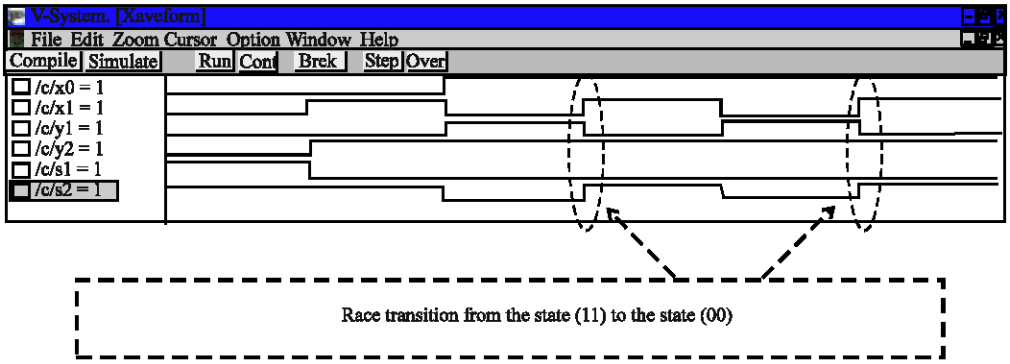

Fig. 16: Simulation that show the race transition problems

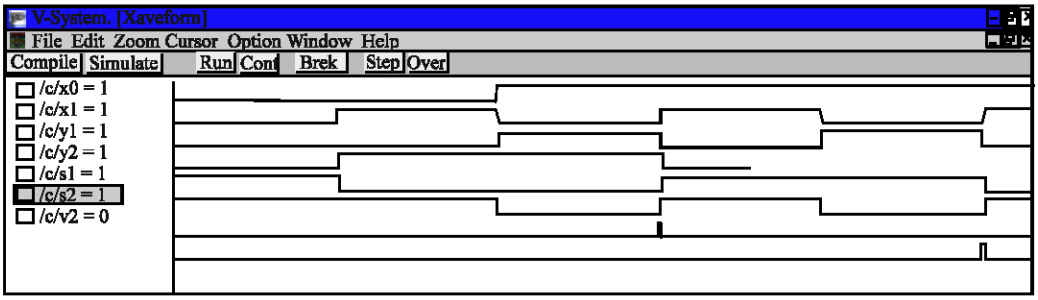

Fig. 17: Simulation results after applying our algorithm

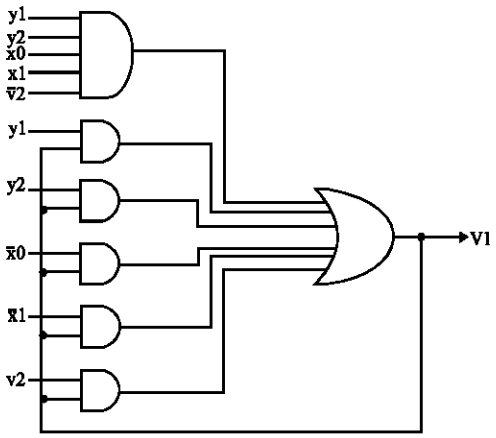

Fig. 18: Two level logic implementation of the variable V_1

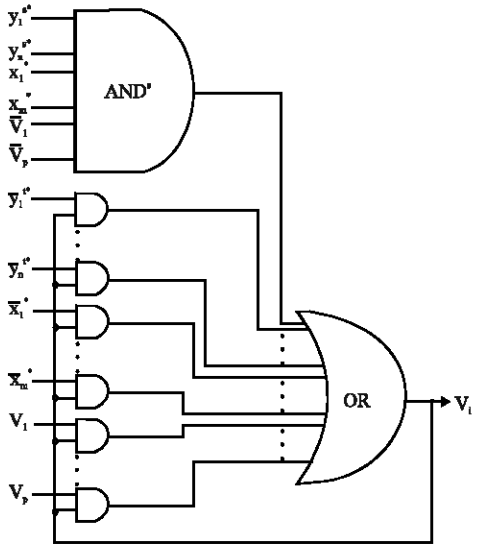

Fig. 19: The general gate level structure of an inserted variable

The two level gate structure that implements this general equation is given by Fig. 19.

Notice that all variables to be inserted according to the race transitions are initially putted at logics 0 (V_p...V_1 = 0...0). Thus, when the machine reaches the source state of a race transition and if this transition occurs, the output of the AND^S gate that is initially putted at logic 0 will rises to logic 1. This in turn put the output of the OR gate (initially at logic 0) to logic 1. The output of the AND^S gate is putted at logic 0 since the machine reaches an intermediate state. Since it is necessary that at least 2 states bits of the source and the target state are different (race transition), we have at least two AND gates (among the two inputs AND gates which have as inputs the next

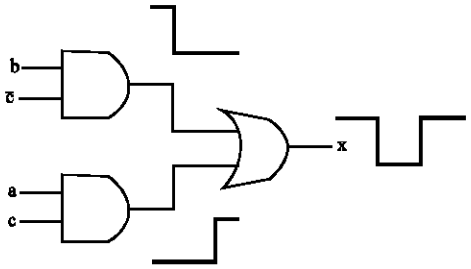

Fig. 20: Example of a circuit with a static 1 hazard

state variables bits $\bar{y}_i^{t^*}$) where their outputs are at logic 1. When the machine reaches an intermediate state there is only one output among these outputs which is putted to logic 0. Only when the machine reaches the target state we have all the output of these and will be at logic 0. This in turn put the output of the V_i variable to logic 0. Thus when the machine reaches an intermediate state or another depending on the next state variables rates combination, we have at least an and output that maintain the value of the or gate to logic 1. Notice that we have not consider the and gate that are associated to the $\bar{x}_m^*$ variables that can also contain some outputs that have logic 1. Thus in all cases, no hazard will be entered to the output variable V_i. The corresponding circuit is hazard free. Particularly, the static 1 hazard that can occur on the OR gate output in similar architecture (Fig. 20) can not be encountered.

The presented variable generation architecture will be the same for all circuits' cases. Only the input number of the two inputs AND^s gate can be changed depending on the state variables and the added internal variable numbers. From Fig. 19 we notice that the output of a variable V_i is activated after a Δt time (propagation time of an AND gate cascaded with an OR gate) starting from their inputs change. Thus, the only limitation of our approach is that if the delay associated to the most fast state variable path is low than the Δt time, the race problem will not be solved. Thus our technique is not applicable to very small circuits where the state variable path is longer than a path formed by an AND gate cascaded with an OR gate.

SWITCHING POWER CONSIDERATION

In CMOS circuit's dynamic power is the dominant contribution of the power consumption. This contribution is composed of power consumed in sequential logic and combinational logic. Power dissipated in the combinational logic mainly depends on the complexity of the Boolean logic functions and their implementation. Power dissipation in sequential logic is due to capacitance charging and discharging caused by the state bit switching. Thus the sequential power dissipation is proportional to the switching activity in the state graph of an asynchronous circuit. We consider the switching activity of a state graph, the indicator of power efficiency of the designed asynchronous circuit.

We use the same formula as Benini and De Micheli (1995), for computing the transition activities. By considering the input transition diagram as a weighted directed graph the transition activities is given as follow:

$$A = \sum_{(i,j)}^{n} P_{ij} H(s_i, s_j)$$

The weight P_{ij} for each (i, j) represents transition probability from state s_i to state s_j. $H(s_i, s_j)$is the Hamming distance between the codes, (two bitstreams C_i and C_j), of states s_i and s_j under the state encoding scheme.

The state transition probability measures how frequently each transition occurs and the Hamming distance gives the amount that each transition contributes to the total switching activity.

The input probability distributions can be obtained by simulating the state graph at a higher level of abstraction in the context of its environment, or by direct knowledge from the designer (Benini and Micheli, 1995). This gives usP_{ij}, the conditional probability that the next state is s_j if the current state is s_i. Then we build a Markov chain based on these conditional probabilities to model the state graph. The Markov chain is a stochastic process whose dynamic behaviour depends only on the present state and not on how the present state is reached (Hachtel *et al*., 1994). We now can obtain the steady state probability P_i of each state corresponding to the stationary distribution of the Markov chain. The state transition probability P_{ij} for the transition from state s_i to state s_j is given by: $P_{ij} = P_{ij} P_i$. This equation implies that, in order to have high total transition probability both the state probability and the conditional transition probability must be high. Using only the conditional transition probability can lead to incorrect estimates.

After applying the Markov chain theory, a weighted state graph is generated and used as input to our state assignment technique. In general a directed graph is transformed into undirected graph by collapsing all multiple directed edges between two states into a single undirected weighted edge with a weight equal to the sum of the directed edges probabilities. Thus in the weighted undirected graph, the weights are proportional to the total probability of a transition between the two states connected by the edge. More details concerning

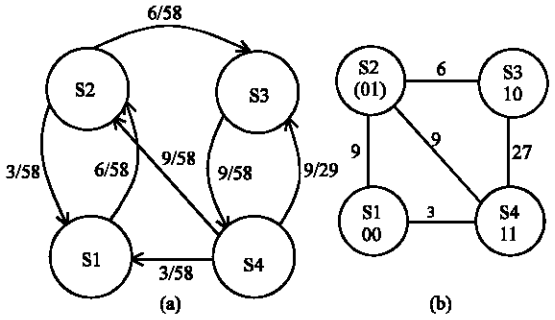

Fig. 21: Example of transforming a directed weighed graph (a) to an undirected weighted graph (b)

the estimation of transition probability technique can be found by Benini and Micheli (1995). For example in the case of the directed weighted graph of Fig. 21a, the transformation into an undirected graph is given in Fig. 21b.

The optimal switching activity corresponds to the state coding with a Hamming distance H (s_i, s_j) = 1. The expression of the optimal transition activity is then reduced to the following formula:

$$A = \sum_{(i,j)}^{n} P_{ij}$$

For a state code with m bits change such as the Tracey technique, the transition activity and thus the dissipated power are very larges when compared to the techniques that allow an optimal transition activity such as the Kantabutra and Huffman techniques.

For our technique only the races transitions will have bits change depending on the number of bits positions that change for each race transition. The rest of states transitions which represent the majority have only one bit change.

Starting from a weighted undirected graph, we associate state codes of the race transitions to the transitions that have the lowest probability. This means that states codes are associated to the states in a manner that minimises the bits changes in the case of the transitions with higher probabilities. For example in the case of Fig. 21a, the states codes that are associated to the highest probability transition (27) have only one bit change (S3:10 and S4: 11). Also the states that are associated to the transition that have the lowest probability (3) are assigned by a racy code with two bits change (S1: 00 and S4: 11). The expression used to evaluate the transition activity of our technique is as follow:

$$A = \sum_{(i,j)-(i',j')}^{n-m} P_{ij} + \sum_{(i',j')}^{m} P_{i'j'}[H(s_{i'},s_{j'}) + 2]$$

The first term

$$\sum_{(i,j)-(i',j')}^{n-m} P_{ij}$$

denotes the transition activities of the transition that have one bit change (H(s_i, s_j) = 1). The second term

$$(\sum_{(i',j')}^{m} P_{i'j'}[H(s_{i'},s_{j'}) + 2])$$

denotes the transition activities of race transitions (H(s_i, s_j) > 1). Notice that the number 2 is added for each race transition since such transition include a transition from 0 to 1 of the added variable in the source state Si' and a transition from 1 to 0 of this variable in the target state Sj'. In all other states the added variable is steel unchanged and not contributes to the transition activity. The least probability Pi'j' is associated to the race transition that has the highest bits changed and so on. For example, in the case of the example of Fig. 21a, the transition activity computed by this formula in the case of our technique is: A = (9 + 27 + 9) + 6×4+3×4 = 81. This transition activity value is 1.60 time higher than the optimal switching activity as given by the Kantabutra and Huffman techniques (P = 3 + 9 + 9 + 27 + 6 = 51) and it is 1.33 time lower than those of Liu/Tracey technique (3+92+9+27+6)×2=108. In fact, the Tracey's state assignment is: (S1) = 000, (S2) = 011, (S3) = 110, (S4) = 101 and two bit change is associated to each state transition. The Liu/Tracey switching activity is 2.12 time higher than those of Kantabutra and Huffman techniques.

Notice that for a state graph with a large number of states (n>>m) and a small number of race transitions with low probabilities, the expression of the transition activity tend to the optimal transition activity

$$(A = \sum_{(i,j)}^{m} P_{ij})$$

The power dissipation is also dependent on the structure of the combinational part of the final synthesised graph. Thus, to obtain low power dissipation in the final circuit, area must sometimes be taken into account. A trade-offs in terms of the relative importance between area and power may be performed.

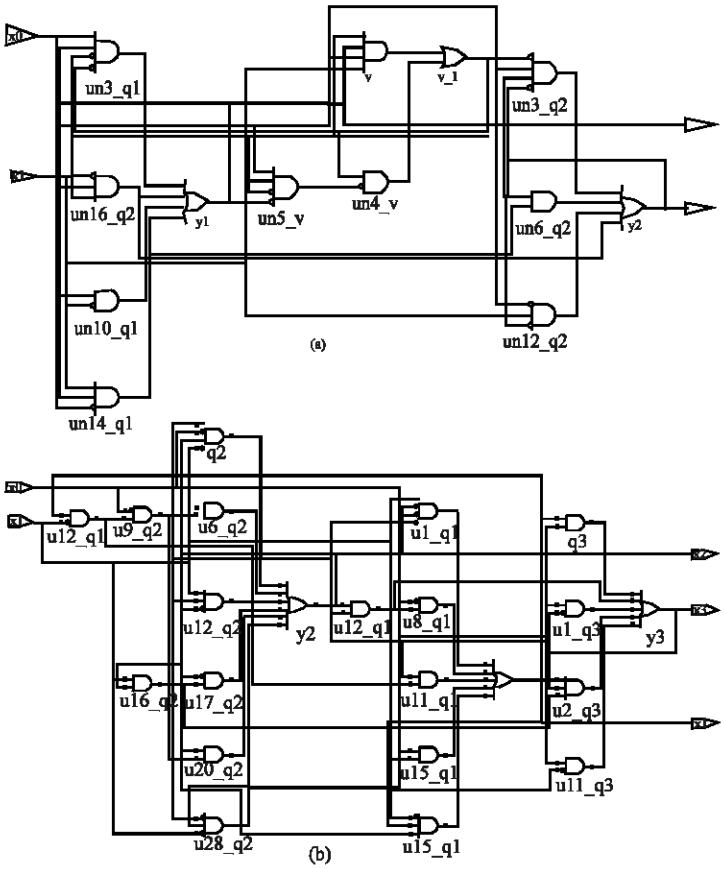

Fig. 22: Logic synthesis result of a circuit example by our technique (a) and by the Kantabutra technique

Since our technique allows the lowest state variable code length it generates generally a smallest final circuit than the other techniques (Huffman, Kantabutra, Liu/ Tracey). Thus the dissipation power relatively to the combinational part of our circuits is lower than those techniques. For example, the logic synthesis of the simple example of Fig. 1 is given by Fig. 22a. The same circuit synthesized by the Kantabutra technique as presented in (Kantabutra and Andreou, 1994), generate a transition graph with 8 states. The logic synthesis of this circuit generate a circuit that is (1.70) large than our circuit (Fig. 22b).

CONCLUSION

The race-free state-assignment method presented herein reduces the complexity of solving the race-free state-assignment problem by decomposing the problem into three principal stages. In the first stage, the flow table or the state transition diagram are coded in a manner that minimizes the number of race transition by respecting the maximum of adjacency. In the second stage, transition probabilities are calculated and each racy transition that has the maximal hamming distance code is weighted by the lowest probability. In the third stage, internal variables for the race transitions are inserted.

Actually we have under implementing the state assignment algorithm and applying it to some benchmarks circuits. The primary results show that the proposed technique constitute a trade-off between silicon area and power dissipation when compared to the state of the art techniques.

REFERENCES

Benini, L. and G. De Micheli, 1995. State Assignment for Low Power Dissipation. IEEE. J. Solide State Circuits, 30: 3.

Chantal, Y.C., L. Bill and M.D. Hugo, 1994. ASSASIN: A synthesis system for asynchronous control circuits. Tech. Rep., IMEC.

Chris, J.M., 1995. Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits. Ph.D. dissertation. Department Electrical Engineering, Stanford Univeristy.

Coates, B., A. Davis and K. Stevens, 1993. The post office experience: Designing a large asynchronous chip. Integration, VLSI. J., 15: 341-366.

Cortadella, J., M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, 1997. Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. IEICE. Trans. Inform. Sys., 80: 315-325.

David, E.M. and W.S. Bartky, 1959. A theory of asynchronous circuits. In: Proc. Int. Symp. Theory of Switching. Cambridge, MA: Harvard Univ. Press, pp: 204-243.

Friedman, A.D., R. L. Graham and J.D. Ullman, 1969. Universal single transition time asynchronous state assignments. IEEE. Trans. Comn. Dput., 18: 541-547.

Friedman, A.D., R.L. Graham and J .D. Ullman, 1969. Universal single transition time asynchronous state assignments. IEEE Trans. Comput., 18: 541-547.

Hachtel, G.D., B. Macii, A. Pardo and F. Somoza, 1994. Probabilistic analysis of large finite state machines" Proceeding of the ACM Design Automation Conferences, San Diego, CA.

Hollaar, L.A., 1982. Direct implementation of asynchronous control units. IEEE. Trans. Comput., 31: 1133-1141.

Huffman, D.A., 1954. The synthesis of sequential switching circuits. J. Franklin Inst., 257: 161-190, 275-303.

Huffman, D.A., 1995. A study of the memory requirements of sequential switching circuits, Tech. Rep. 293, MIT Research Laboratory For Electron., Cambridge, MA.

Kantabutra, V., A.A. Andreou, 1994. A state assignment approach to asynchronous CMOS circuit design. IEEE. Trans. Computer, 43: 460-469.

Kuhl, J.G. and S.M. Reddy, 1978. A multicode single transition-time state assignment for asynchronous sequential machines. IEEE. Trans. Comput., 27: 927-934.

Liu, C.N., A state variable assignment for asynchronous sequential circuits. J. Ass. Comput. Mach., 10: 209-216.

Liu, C.N., 1963. A state variable assignment method for asynchronous sequential switching circuits. J. ACM, 10: 209-216.

Maki, G.K. and J.H. Tracey, 1971. A state assignment procedure for asynchronous sequential circuits. IEEE. Trans. Comput., 20: 666-668.

Maki, G.K., J.H. Tracey and R.J. Smith, 11, 1969. Generation of design equations in asynchronous sequential circuits. IEEE. Trans. Comput., 18: 467472.

Moon, C.W., P.R. Stephan and R.K. Brayton, 1991. Synthesis of Hazard-free asynchronous circuits graphical specifications. In Proc. Int. Conf. Comput. Aided Design (ICCAD)., pp: 322-325.

Nanya, T. and Y. Tohma, 1978. On universal single transition time asynchronous state assignments. IEEE. Trans. Comput., 27: 781-782.

Nanya, T. and Y. Tohma, 1979. Universal multicode SlT state assignments for asynchronous sequential machines. IEEE. Trans. Comput., 28: 811-818.

Saucier, G., 1972. State assignment of asynchronous sequential machines using graph techniques. IEEE Trans. Comput., 21: 282-288.

Saucier, G., Encoding of asynchronous sequential networks. IEEE. Trans. Electron. Comput., 16: 365-369.

Sentovich, E.M. *et al*., 1992. SIS: A system for sequential circuit synthesis, Tech. Rep. UCB/ERL M92/41, U.C. Berkeley, (Ed.).

Smith, R.J., 1974. Generation of internal state assignments for large asynchronous sequential machines. IEEE. Trans. Comput., 23: 924-932.

Steven, M.N., 1993. Automatic Synthesis of Burst-Mode Asynchronous Controllers, Ph.D. dissertation, Stanford University, Department of Computer Science.

Tan, C.J., 1971. State assignments for asynchronous sequential machines. IEEE. Trans. Comput., 20: 382-391.

Tracey, J.H., 1966. Internal state assignments for asynchronous sequential machines. IEEE. Trans. Elec. Comput., 15: 551-560.

Unger, S.H., 1969. Asynchronous Sequential Circuits. New York: Wiley Interscience.

Unger, S.H., 1996. Asynchronous Sequential Switching Circuits, New York: Wiley.

Yun, K.Y., Synthesis of Asynchronous Controllers for Heterogeneous Systems, Ph. D. dissertation, Stanford University.