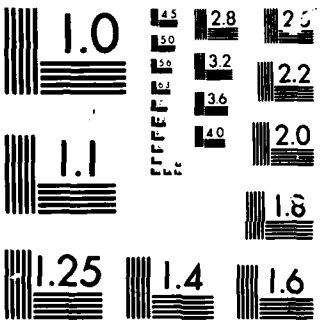
AD-A166 788 SYNTHESIS OF A SELF-TIMED CONTROLLER FOR A SUCCESSIVE-APPROXIMATION A/D C..(U) MASSACHUSETTS INST OF TECH CAMBRIDGE DEPT OF ELECTRICAL ENGIN.. 1/1

UNCLASSIFIED T CHU ET AL. OCT 85 VSLI-MEMO-85-274 F/G 9/5 NL



MICROCOPY CHART

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE, MASSACHUSETTS 02139

VLSI Memo No. 85-274

October 1985

Synthesis of a Self-timed Controller for a Successive-approximation A/D Converter*

Tam-Anh Chu and Lance A. Glasser**

ABSTRACT

This paper documents the procedure for designing a self-timed controller for a successive-approximation A/D converter. From the functional specification, a Signal Transition Graph is constructed to describe the operation of the control circuit. This graph is then modified into a well-formed graph. Such a graph can be transformed into a deadlock-free and hazard-free implementation directly. The structure of the control circuit and the logic equations are then derived directly from the graph.

*This research was supported in part by a Hughes Fellowship, by the National Science Foundation under grant number MCS-7915255, and by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622.

**Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139. Chu: Room NE43-237, (617) 253-8861; Glasser: Research Laboratory of Electronics, Room 36-880, (617) 253-4677.

Synthesis of a Self-timed Controller for a Successive-approximation A/D Converter

Tam-Anh Chu[1]
Department of EECS

Lance A. Glasser[2]
Department of EECS and the Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

This paper documents the procedure for designing a self-timed controller for a successive-approximation A/D converter. From the functional specification, a Signal Transition Graph is constructed to describe the operation of the control circuit. This graph is then modified into a well-formed graph. Such a graph can be transformed into a deadlock-free and hazard-free implementation directly. The structure of the control circuit and the logic equations are then derived directly from the graph.

1. Introduction

This paper describes the design of a control circuit for a successive-approximation A/D converter. This controller is an asynchronous self-timed circuit in which all control actions are carried out through the use of the Request/Acknowledge signaling protocol. From a functional specification of the control circuit, a Signal Transition Graph (STG) is constructed to describe the behavior of the circuit. It is then modified into a well-formed graph, which is one satisfying the liveness and persistency properties. This

[1]This research was supported in part by a Hughes Fellowship and the NSF under grant number MCS-7915255.

[2]This research was supported by the Defense Advanced Research Project Agency under contract number N00014-80-C-0622.

STG can be converted directly into a logic circuit through a number of synthesis steps. A theory of the Signal Transition Graph model is discussed in [1]. Sufficient details about the model will be given here to explain the synthesis process from STG.

The main advantage of this realization method is that it can produce a circuit directly from a well-formed specification, and the circuit is speed-independent, i.e., it operates correctly with any combination of delays of logic gates. This feature is important for VLSI applications, as it is inefficient and not always possible to fine-tune the delays of logic gates on chip to make an asynchronous system work. The STG model allows the specification of *concurrency*, and hence the control logic synthesized from this model supports concurrent operations. The traditional approach for designing asynchronous state machines can only model sequential control actions, and furthermore they are difficult to realize because of problems due to races and hazards. In contrast, the approach presented here produces hazard-free control circuits capable of handling parallel operations.

2. Behavior Specification of the Controller

The block diagram of the successive approximation A/D converter is shown in Figure 1. The input comparator compares the input voltage v_{in} and the reference voltage v_{ref} and produces a digital 1-bit result. The comparator has a control input Z_r which zeroes it when Z_r makes a low-to-high transition, and starts the comparison when Z_r makes a high-to-low transition. It also has a *mutual-exclusion* circuit whose output is active (=1) only when the comparator output is valid. This is required because the comparison time is a function of the difference between the input voltage and the reference voltage; the smaller the difference, the longer the time it takes for the comparator to decide. This is the familiar problem of metastability[2].

The latch and the combinational logic form a finite state machine performing the successive approximation algorithm. Note that this machine operates in *pulse mode* and is not the same as the self-timed controller we are synthesizing. Due to the fact that this machine performs many data-dependent operations, it is more economical and straight-forward to implement it in this form instead of a Huffman asynchronous state machine. Data are latched on the rising transition of signal L_r and held in registers

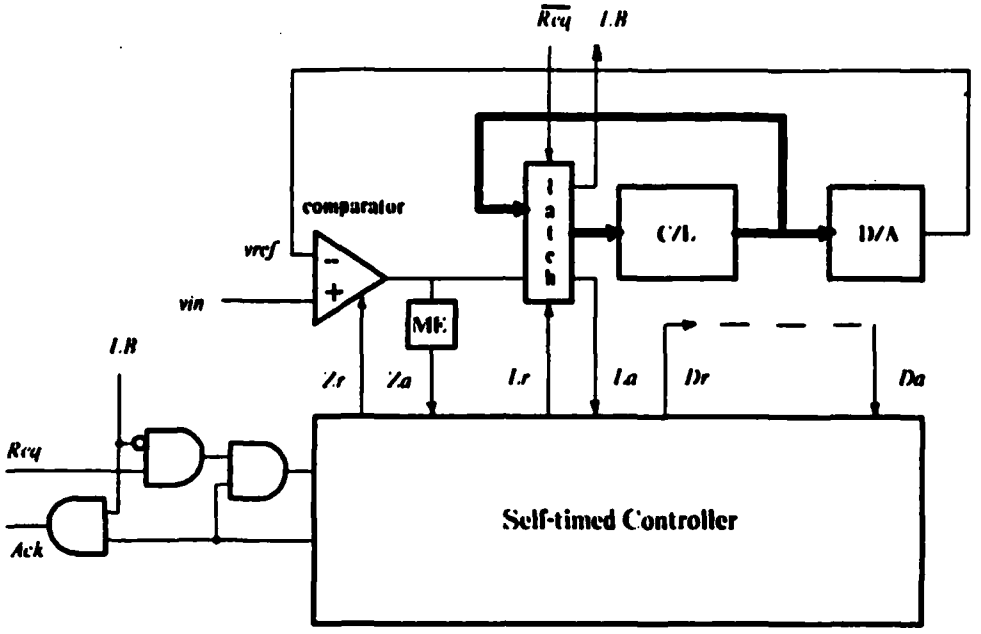


Figure 1: Block diagram of the successive-approximation A/D converter

after L_r goes low. Signal L_a goes high as soon as data are latched, and goes low shortly after L_r goes low. The reset input of the latch is controlled by signal $\overline{Req}$, so that when Req is low, its outputs are reset to the appropriate initial state. Signal LB is the *Last-Bit* signal which goes high when the converter has determined the last bit of the digital word. The D/A converter at the right of the diagram accepts the digital word produced by the state machine and generates the analog voltage v_{ref}. The combined delay of the combinational logic and the D/A converter is matched by some delay circuit as indicated by a dashed line from D_r to D_a. While it is possible to accomplish this timing constraint in a speed-independent manner using dual rail coding, a simple delay circuit is more justifiable from an engineering standpoint.

Initially, the state of the system is $Req = Ack = Z_r = Z_a = L_r = L_a = 0$ and $D_r = D_a = 1$. Since $Req = 0$, the latch is initialized with $LB = 0$. Thus, the and-gate whose input is Req is enabled and the and-gate whose output is Ack is disabled. When Req is raised, Z_r will go high and initiate a cycle of the successive-approximation algorithm. After each cycle, D_a will restart another cycle until LB becomes high during the last cycle of the

algorithm. In the last cycle, *Ack* is raised when D_a goes high. After that, *Req* drops, resetting *LB* and in turns *Ack* to low. At this point the circuit returns to its initial state, ready for the next conversion.

3. Constructing a STG from the Specification

A STG describing the operation of the self-timed controller is shown in Figure 2. In this STG, vertices represent control events corresponding directly to rising and falling transitions of signals in the controller; directed arcs between transitions are timing precedence constraints. Transitions of input signals to the circuit are underlined to distinguish them from those of *non-input* signals. Precendence constraints of the former are given by the specification and are assumed to be satisfied by the outside world, whereas precedence constraints of the latter are generated and satisfied by the circuit obtained from the STG. The dashed arcs in this figure are not derived from the behavior specification of the circuit, but are extra constraints to make the STG *persistent*, as will be explained later. The meaning of the transitions are also described in this figure. The notation $a+ \rightarrow b-$ indicates that the rising of signal a has to precede the falling of b, and that the transition $b-$ is directly *caused* by $a+$. The notation $a+ \rightarrow_p b-$ indicates that the occurrence of $a+$ precedes that of $b-$ but it may or may not *directly* precede $b-$. The And-fork relation $a\mathcal{R}_A(b,c)$ means that $a \rightarrow b$ *and* $a \rightarrow c$, and that the occurrence of a will cause the concurrent occurrences of b and c. The And-join relation $(a,b)\mathcal{R}_A c$ means that $a \rightarrow c$ and $b \rightarrow c$, and that c occurs only after both a and b have occurred.

There are two fundamental properties of STG concerning the synthesis of hazard-free and deadlock-free control circuits, those of *liveness* and *persistency*. A STG satisfying these properties is well-formed and possesses a hazard-free and deadlock-free realization. If a STG is strongly connected and for every transition of signal t, there exist at least a *directed path* from $t+$ to $t-$ and at least one from $t-$ to $t+$, then the corresponding circuit realization is *live*, i.e., free from deadlocks. The second property is *persistency*, which states that if a signal is *enabled* in some state of the system, only a transition of that signal can bring the system to another state in which it is no longer enabled. In terms of STG notations, Fig. 3a illustrates an

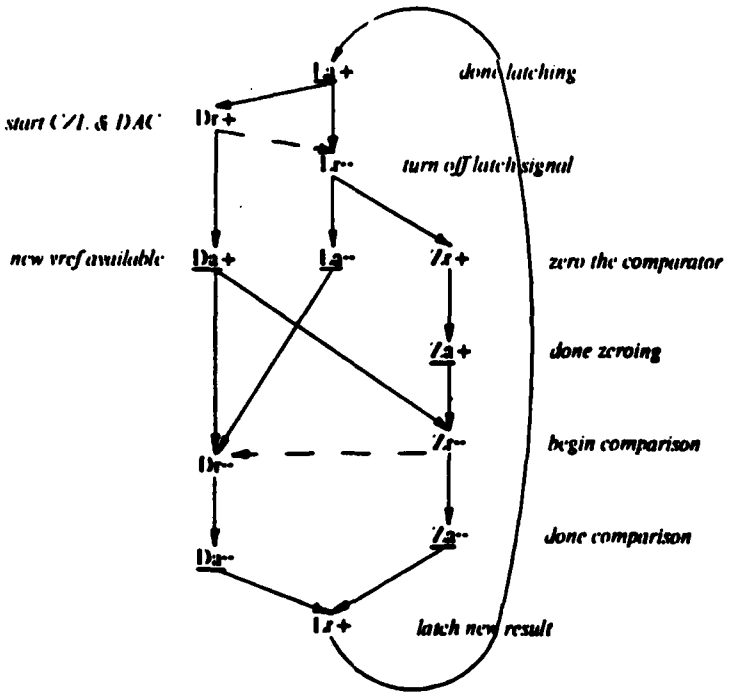


Figure 2: The STG description of the self-timed controller

example of a violation of persistency: transition $a+$ causes transition $b+$, and also, transition $a-$ is concurrent with $b+$. Thus, while b is going high, transition $a-$ may occur and remove the enabling condition of transition $b+$, resulting in a hazard of signal b. Fig. 3b shows that an additional constraint $b+ \rightarrow_p a-$ is required to satisfy the persistency property.

The STG in Fig. 2 is a strongly connected digraph which satisfies the liveness property, therefore it corresponding circuit is live. The transitions D_r-, D_a- and L_a- are merely reset transitions of the reset signaling protocol. Two constraints in this graph deserve careful attention: the constraint $D_a+ \rightarrow Z_r-$ is required because a comparison is not allowed to begin until after a new value of v_{ref} is available; the other constraint $L_r- \rightarrow Z_r+$ makes sure that the gating signal of the latch is turned off before the comparator changes its output value.

4. Making the STG persistent

In general, a STG constructed from the behavior specification of a control circuit alone may not be realizable. In order to have a deadlock-free and

Figure 3: (a) A violation of persistency, (b) elimination of the violation

hazard-free realization, it must satisfy two fundamental properties stated above. In the original specification in Figure 2 (without the dashed arcs), one can immediately detect two cases of violation of persistency. The fork $L_a + \mathcal{R}_A(D_r+, L_r-)$ indicates that L_a+ causes D_r+, but transition L_a- which directly follows L_r- is concurrent with D_r+. Thus, while D_r+ is occurring, L_a- may occur and remove the enabling condition of transition D_r+. The dashed arc $D_r+ \rightarrow L_r-$ eliminates this problem by adding the constraint $D_r+ \rightarrow_p L_a-$ to the graph. In this case a direct constraint $D_r+ \rightarrow L_a-$ is not allowed because L_a- is a transition of an input node of the corresponding circuit; constraints to input transitions are given by the specification and there can be exactly one arc incident to an input transition. Similarly, the violation at the fork $D_a + \mathcal{R}_A(D_r-, Z_r-)$ is eliminated by adding the dashed arc $Z_r- \rightarrow D_r-$ to the graph. After adding these two constraints to eliminate violations of persistency, one can further eliminate constraints $L_a+ \rightarrow L_r-$ and $D_a+ \rightarrow D_r-$ as they become redundant. At this stage, an STG as shown in Figure 4a is obtained.

It turns out that this modified graph is still not implementable due to the lack of internal state information. This is a subtle case of *violation of persistency* which will be discussed later. An internal state signal called x is introduced and the transitions $x+, x-$ are added as shown in Figure 4b. In order to understand how this node is introduced into the STG specification, the synthesis process from STG needs be discussed before this issue can be explored (Section 6.)

5. Synthesis of Circuit from STG

In the circuit realization of this graph, nodes L_a, D_a, Z_a are input nodes to the circuit, other nodes are non-input ones whose logic equations have to be

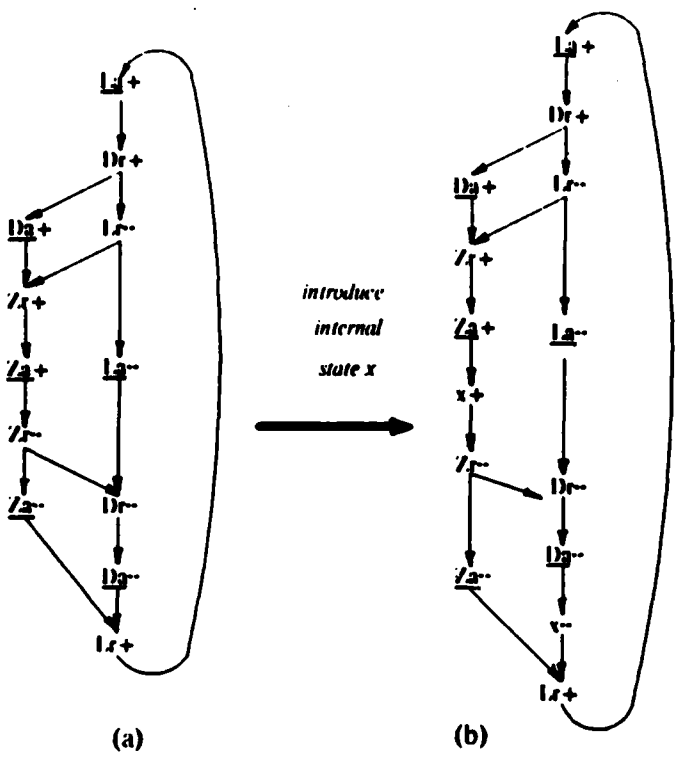
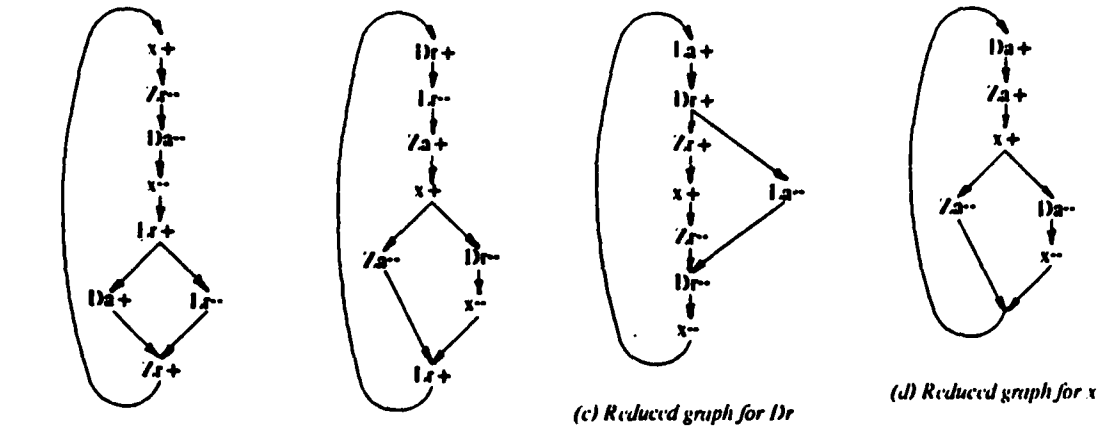


Figure 4: (a) The STG with two violations of persistency eliminated, and (b) the STG with the addition of the internal state x.

determined. The set of non-input nodes is $\{D_r, L_r, Z_r, x\}$. First, the STG in Fig. 4b is decomposed into number of *reduced graphs*, as described below. For each non-input node t_i in set $\{D_r, L_r, Z_r, x\}$, we find its *input set* $I(t_i)$ defined as the set of nodes whose transitions *directly precede* transition t_i, i.e.,

$$I(t_i) = \{t_j \mid t_j \rightarrow t_i\}.$$

Thus, in the STG in Fig. 4b, $I(L_r) = \{D_r, Z_a, x\}$, $I(Z_r) = \{L_r, D_a, x\}$, $I(x) = \{Z_a, D_a\}$ and $I(D_r) = \{L_a, Z_r, x\}$. The last insput set $I(D_r)$ is a special case because even though node x does not directly precede D_r, it is included in $I(D_r)$ to avoid a violation of persistency, as will be discussed in Section 6. A *reduced graph* $G_R(t_i)$ of node t_i is then obtained by removing all transitions in the STG that do not belong to the set $I(t_i) \cup \{t_i\}$, keeping all precedence constraints intact. The reduced graphs $G_R(D_r)$, $G_R(Z_r)$, $G_R(L_r)$ and $G_R(x)$ are shown in Fig. 5. A reduced graph for a non-input node, e.g. Z_r, contains transitions of nodes in the set $I(Z_r) \cup \{Z_r\}$, and the logic element that realizes node Z_r has one output terminal being Z_r and input terminals being those in the set $I(Z_r)$. This graph contains all the



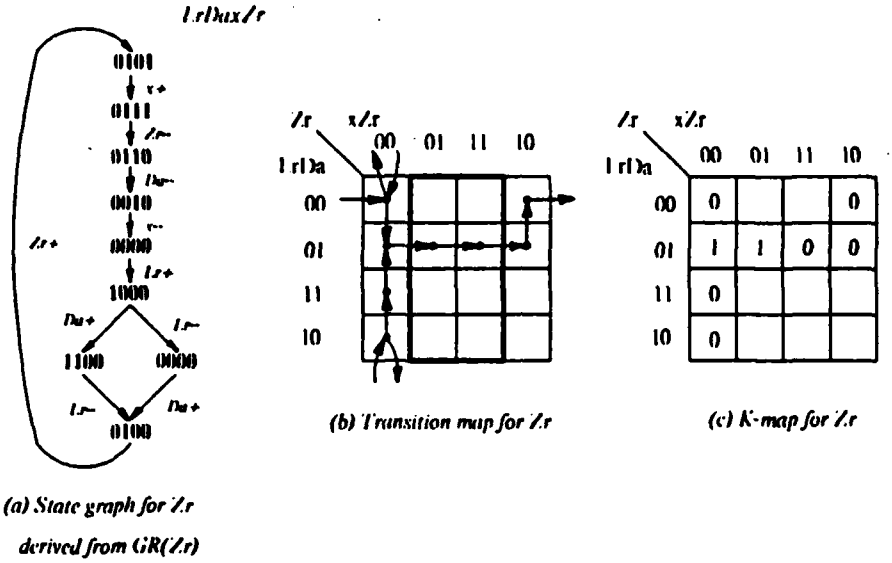
(a) Reduced graph for Z_r (b) Reduced graph for L_r (c) Reduced graph for D_r (d) Reduced graph for x

Figure 5: Reduced graphs for non-input nodes in the set $\{Z_r, L_r, D_r, x\}$

information about the timing behavior of logic element Z_r as specified by the precedence relations between its input and output terminals. A logic equation can be determined from this specification as shown next.

The final step in the synthesis process is to derive logic equations from reduced graphs. This step is illustrated for nodes Z_r and x. The equations for D_r and L_r will be given later and the readers can check them using the procedure described.

From the reduced graph $G_R(Z_r)$, one can derive a state graph (Fig. 6a) in which a state is a binary vector representing the state of terminals of logic element Z_r. This state is $L_r D_a x Z_r$. The transition from one state to another involves a single variable change, and it corresponds to a *signal-transition* in the reduced graph $G_R(Z_r)$ of Fig. 5a. For example, the state-transition 0101 $\rightarrow$ 0111 in Fig. 6a is caused by signal transition $x+$ in $G_R(Z_r)$. The concurrent transitions of D_a+ and L_r- is described by a 2-cube of states, containing 2 possible sequence of state changes 1000 $\rightarrow$ 1100 $\rightarrow$ 0100 and 1000 $\rightarrow$ 0000 $\rightarrow$ 0100. The first state sequence takes places if D_a+ occurs before L_r-, the second takes place if L_r- occurs before D_a+. Since the circuit behaves exactly the same no matter which transition occurs first, it is clear that these two cases also cover the case when L_r- and D_a+ occur at exactly the same time. This is how concurrency can be described in a state-based formulation.



(a) State graph for Z_r derived from GR(Z_r)

(b) Transition map for Z_r

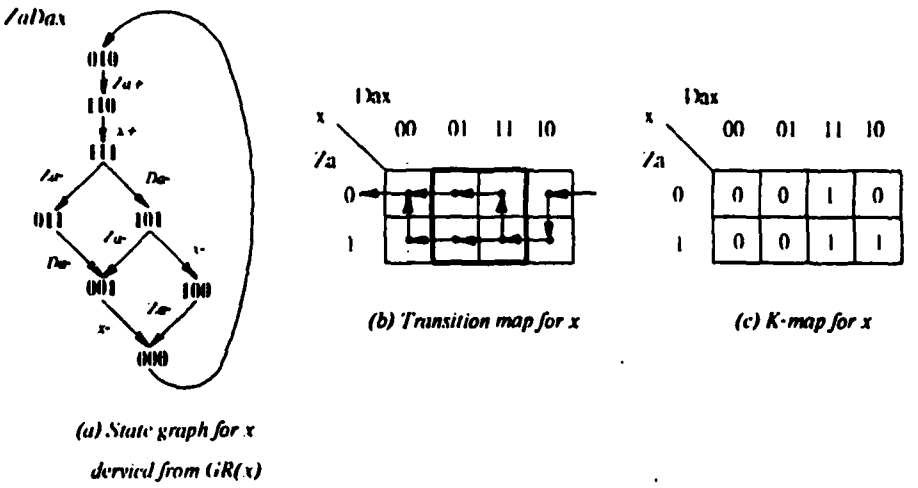
(c) K-map for Z_r

Figure 6: Steps in the transformation from a reduced graph to the logic equation for node Z_r.

An *output-conflict* exists if a state has at least two next-states in which values of the output variable are different. In this state graph, state 1000 has two next-states 1100 and 0000, and they both contain $Z_r = 0$, thus there is no output-conflict. This state graph can be programmed into a type of K-map called *transition map* shown in Fig. 6b. Each entry in this map corresponds to a binary representation of state $L_r D_a x Z_r$; arcs between entries are *walks* between adjacent neighbors and they are state transitions given by the state graph. In order to transform a transition-map into a K-map, each entry is replaced by its next-state value of Z_r. For example, in state 0111, the next-state value of Z_r is 0, thus this entry in the transition map is replaced by a 0. If there are more than one next-state and their values of Z_r are different, i.e., if there is an output-conflict, it may not be possible to determine the value of that entry in the K-map. The logic equation of Z_r can be found from this K-map to be

$$Z_r = \bar{L}_r D_a \bar{x}.$$

The derivation of the logic equation for node x is more interesting because this logic element not only has state information but also output-conflicts.



(a) State graph for x derived from $G_R(x)$

(b) Transition map for x

(c) K-map for x

Figure 7: Steps in the transformation from a reduced graph to the logic equation for node x.

The state diagram derived from the reduced graph $G_R(x)$ is shown in Fig. 7a. The concurrent transitions of Z_a- and the *chain* $D_a- \rightarrow x-$ are described by a structure consisting of two 2-cubes, with three possible allowed state sequences $\{(111,011,001,000),(111,101,001,000),(111,101,100,000)\}$. The *output conflict* exists in state $Z_aD_ax = 101$, as x are different in the next-states 001 and 100. However, this output-conflict is caused by concurrent transitions of an input signal Z_a and the output x. In determining the value of x for the K-map in state 101, transition due to $x-$ leading to state 100 must be chosen over transition $101 \rightarrow 001$ because the latter is caused by input transition Z_a-. The state graph in Fig. 7a shows that regardless of whether one is in state 101 or 001, transition $x-$ will always occur next and the circuit behaves exactly the same. The transition map in Fig. 7b doesnot contain the transition $101 \rightarrow 001$. The K-map derived from this state graph is shown in Fig. 7c, the logic equation is found to be

$$x = Z_aD_a + xD_a.$$

This equation has the general form $x = S + x\overline{R}$ with $S = Z_aD_a$ and $R = \overline{D}_a$, its implementation is a set-reset flipflop whose output is x, the set and reset inputs are Z_aD_a and $\overline{D}_a$, respectively. Note that in this implementation, it is required that $S.R = 0$ at all time.

Similarly, the same procedure can be applied to other reduced graphs

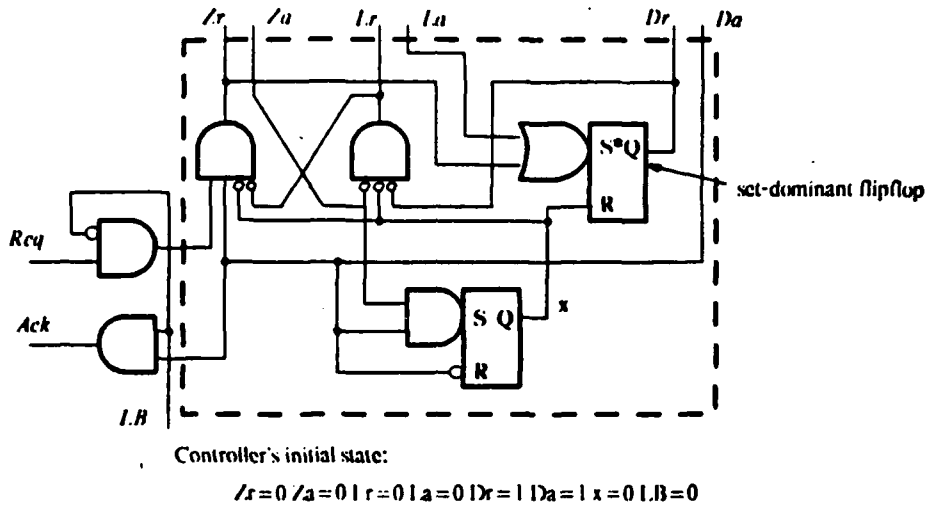


Figure 8: The final circuit realization of the self-timed controller.

to obtain the logic equation for L_r and D_r. They are $L_r = \overline{D}_r \bar{x} \overline{Z}_a$ and $D_r = Z_r + L_a + D_r \bar{x}$. The equation for D_r can be rewritten as $D_r = S + D_r \overline{R}$ with $S = Z_r + L_a$ and $R = x$, and it is implemented as a set-reset flipflop. The reduced graph of D_r in Fig. 5c shows that there is a time period during which both Z_r and x are high, causing both the set and reset inputs of the D_r flipflop to be active. However, it also indicates that output D_r is not to be reset until after both Z_r and L_a go low, and therefore, until after the set input goes low. The implementation of this flipflop is a *set-dominant* one, as indicated by an asterisk in Fig 8. On the other hand, one can choose to implement D_r directly from the equation given above instead of a set-reset flipflop and not worrying about this particular detail.

Finally, by putting all these elements together, one obtains the control circuit for the A/D converter as shown in the dashed box of Fig. 8. The readers should be able to verify that the self-timed control circuit shown is speed-independent, i.e., it operates correctly with any combination of delays of logic gates, assuming that the internal feedback delays of flipflops are negligible compared to other loop delays in the control circuit.

6. Introducing Internal Node x

The synthesis procedure of the controller from a *well-formed* STG specification have been described. We now return to the original STG specification in Fig. 4a and explain how the internal node x is inserted into this

graph to give the well-formed specification in Fig. 4b. The STG in Fig. 4a has non-input nodes D_r, L_r, Z_r, and their input sets are

$$I(D_r) = \{L_a, Z_r\}$$
$$I(L_r) = \{D_r, D_a, Z_a\}$$
$$I(Z_r) = \{L_r, D_a, Z_a\}.$$

Their reduced graphs derived from the above STG are shown in Fig. 9. The state graphs of $G_R(D_r)$ and $G_R(L_r)$ are Figs. 9d and e, respectively. The reduced graph of D_r (Fig. 9a) shows that there is an instance of consecutive transitions of node Z_r which directly precedes the output transition D_r ($Z_r+ \rightarrow Z_r- \rightarrow D_r-$). Generally, if an input signal to a logic element changes twice without any intervening transition which alters the state of the system, a hazard may result at the output of that logic element. The state graph of $G_R(D_r)$ (Fig. 9d) in which each state is of form $L_a Z_r D_r$ contains two instances of each of states 101 and 001. In state 101, the output D_r is not enabled and D_r does not make a transition from this state. However, state 001 has an output-conflict in the output variable D_r because the next-states of 001 are 011 and 000. This output-conflict causes hazard because whenever the logic element D_r gets into the upper state 001 both transitions D_r- and Z_r+ are enabled. If Z_r+ occurs first, the state-sequence 001 $\rightarrow$ 011 $\rightarrow$ 001 $\rightarrow$ 000 will take place and logic element D_r behaves correctly. However, if the delay of logic element D_r is smaller than that of Z_r, then transition D_r- will occur first in state 001, even before the occurrence of the chain $Z_r+ \rightarrow Z_r-$. This is a malfunction. In the case when both Z_r and D_r have approximately the same delay, both transitions D_r- and Z_r+ can occur simultaneously, resulting in a hazard at node D_r.

The reduced graph of L_r (Fig. 9b) contains a similar chain of transitions $Z_a+ \rightarrow Z_a-$ directly preceding an output transition L_r+. However, this case has no hazard because after transition Z_a+ occurs, transitions Z_a- and D_r- will occur concurrently. The occurrence of D_r- changes the state of the logic element before Z_a returns to low. Its state graph (Fig. 9e) shows that output transition L_r+ is enabled in the unique state0000, and there is no output-conflict in this case.

Thus, the hazard at node D_r is caused by the lack of internal state information to discern two instances of state 001. This problem shows

up in the reduced graph $G_R(D_r)$ as a pair of consecutive transitions of the same node Z_r. In order to remove this problem, an extra transition such as $x+$ is inserted between these two transitions. Now, transition $x-$ must be inserted into the graph to preserve its liveness. The reduced graph $G_R(D_r)$ shows that $x-$ cannot be inserted (i) between the pair (D_r+, Z_r+) or (Z_r-, D_r-) because this only produces the same problem but with *two* pairs of consecutive transitions of the same nodes; (ii) into the path $(D_r+, La-, D_r-)$ because $x+, x-$ become concurrent and this would violate both the liveness and persistency conditions. Thus $x-$ has to be inserted into the path (D_r-, L_a+, D_r+). Considering the original STG of Fig. 4a, this means that $x-$ must be inserted into the path contain transitions $(D_r-, D_a-, L_r+, L_a+, D_r+)$. Furthermore, transitions of input nodes D_a, L_a and Z_a can have only one incident arcs coming from transitions of their corresponding *request* signals D_r, L_r and Z_r, respectively. Thus $x-$ can be inserted between (D_a-, L_r+) as shown in Fig. 4b. In this final well-formed specification, transition $x+$ does not directly precede transitions of node D_r; however, as explained earlier, it is used as an input to logic element D_r to eliminate hazards at node D_r.

Finally, note that transition $x-$ can also be inserted between (L_a+, D_r+) in the STG in Fig. 4a. This results in another well-formed graph, from which a different implementation can be obtained using the synthesis steps described above. This fact indicates that the implementation is sensitive to the particular form of the STG. This is understandable as the state graphs extracted from STGs are unique state-based representations of the behavior of a circuit.

REFERENCES

[1] Chu, T.-A. "Signal Transition Graphs and the Modeling of Self-timed Circuits." Ph.D. thesis, MIT Dept. of EECS, expected December 1985.

[2] Glasser, L.A. "Synchronizer Failure in A/D Converter." MIT VLSI Memo, Dept. of EECS, July 1985.

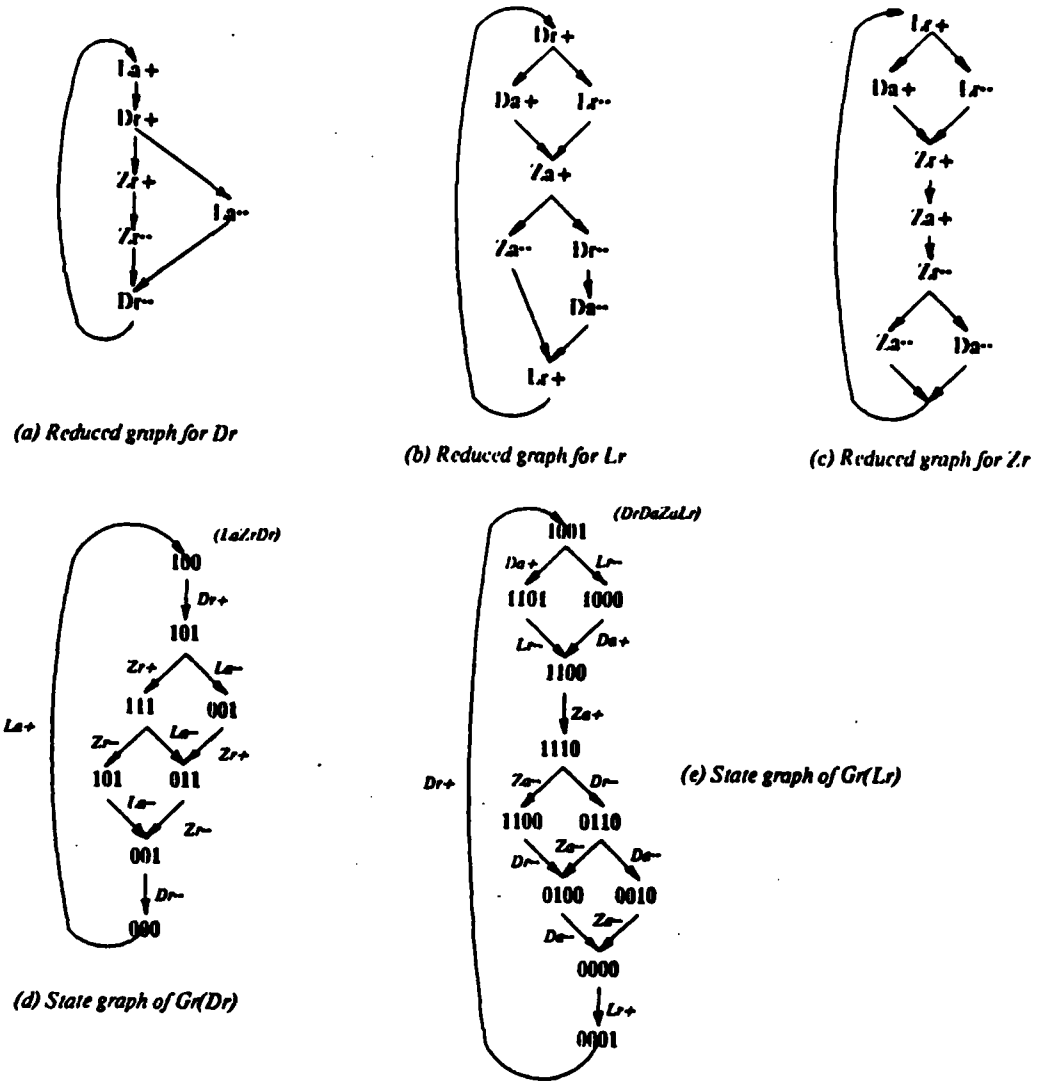


Figure 9: (a)-(c) Reduced graphs of nodes D_r, L_r and Z_r derived from the STG in Fig. 4a. (d) State graph of $G_R(D_r)$. (e) State graph of $G_R(L_r)$.

END

FILMED

5-86

DTIC