A PRACTICAL APPROACH TO ASYNCHRONOUS GATE NETWORKS

by

J.A. Brzozowski* and M. Yoeli

Technical Report No. 40
December 1974

* J.A. Brzozowski, University of Waterloo, Computer Science Dept.
Waterloo, Ontario, CANADA.
ABSTRACT

In many works, the analysis and synthesis of asynchronous gate networks is based on a model in which state variables are associated with feedback loops. This paper shows that such a model is incapable of properly representing race conditions and, consequently, may yield misleading results. A model in which a state variable is associated with each gate is described here, and is applied to the analysis of complex flip-flops.
1. INTRODUCTION

This paper is concerned with the analysis of gate networks of the type commonly used in integrated circuit modules such as complex flip-flops. Frequently, methods associating state variables with feedback loops are proposed for the analysis of such networks. Although a number of authors have noticed some shortcomings of this model, no alternative method applicable to practical networks seems to be available. In this paper we explain the difficulties with the conventional model and propose an alternative model, in which we associate a state variable with each gate. We demonstrate the feasibility of using this model by analyzing an edge-sensitive JK flip-flop.

2. CONVENTIONAL ANALYSIS AND SYNTHESIS

A large number of books and papers on the theory of asynchronous networks use the following analysis method. Given the logic diagram of a gate network, choose a certain set of signals to constitute the state variables: namely, choose a set of lines such that cutting each line removes all feedback loops in the network. We will call the set of corresponding variables a feedback set. For example, in Fig. 1(a), all the feedback sets are: \{v\}, \{w\}, and \{v,w\}. Usually, minimal sets are used; in this case one would choose either \{v\} or \{w\}. If the line corresponding to \(w\) is cut as in Fig. 1(b), there remains a combinational network. If we distinguish between the signals on the two ends of the cut line in
Fig. 1(b) by calling them W and w, we can think of w as a new input, W as a new output, and we find

\[ W = ((wx)')' = x + w. \]

The feedback variable w provides sufficient information to determine all the signals in the network of Fig.1(b) (provided the input is also known), and \{w\} is indeed an ''internal-state'' variable set for Fig.1(a).

Note that \( v = (wx)' \).

The behavior of a sequential network cannot be explained properly without taking into account the propagation delays of the gates. We assume here that line delays are negligible. In the conventional theory one delay is introduced for each line in the chosen feedback set. In our example we get Fig.1(c). This procedure is clearly equivalent to assuming that gate G_1 has no delay whereas gate G_2 has delay \( \Delta \). This assumption is not justified if we want a general analysis of Fig. 1(a). In reality every gate has a delay and it is likely that both gates of Fig. 1(a) will have delays of similar magnitude. We now proceed to show that the approach of Fig. 1(c) yields misleading results.

In the conventional theory the network behavior is explained with the aid of the excitation table, as in Fig. 2(a), where we assume that \( v \) and \( w \) are the outputs. We show the outputs only for stable total states, for simplicity.

From Fig. 2(a) we find that, if \( N_1 \) is started with \( x = 0 \), it is forced to state \( w = 1 \) with \( v = w = 1 \). Next, when \( x \) changes arbitrarily, \( w \) remains fixed at \( w = 1 \).
A similar analysis with \{v\} as the feedback set gives Fig. 2(b).

Here $x = 0$ forces $N_1$ to $v = 1$, again with $v = w = 1$. However, as $x$ changes subsequently, it is clear that $w = x'$.  

Now we started with one network $N_1$, analyzed it using two different sets of feedback variables, and obtained two different input/output behaviors. Obviously, the two analyses are not equivalent. In fact both are incorrect, in general. More realistically, we should associate a propagation delay with each gate, as mentioned earlier. When this is done we obtain Fig. 2(c). Obviously we get the same stable total states as before, but the transitions are quite different. In fact, when $x$ has been 0 and then changes to 1, there is a critical race, indicated by $\ast$.

The conventional synthesis method is, of course, based on the conventional analysis and would only be correct if one had available delay-free gates in addition to ordinary gates. Since this is not the case the conventional theory "overcomes" this difficulty by inserting additional delays into the selected feedback lines. This approach is clearly undesirable if fast networks are required. Although there does not yet exist a formal method for realizing flow tables by gate networks without additional delays, many practical flip-flop designs [BR-Y01, SIG, TI] constitute particular solutions to this problem.

Note that the conventional analysis and synthesis apply very well to relay networks. If we were to design the flow table of Fig. 2(a) (which coincides in this case with the excitation table) with relays we would get Fig. 3. Here there is a delay associated with relay $W$ but no delay is
associated with performing the function \( w + x' \), since these are performed by wiring. Alas relay networks are not very popular these days!

3. REALISTIC ANALYSIS OF GATE NETWORKS

From the previous section it follows that one can use the conventional analysis techniques, provided one assigns a state variable to each gate output. However, one encounters several difficulties due to the size of practical networks, as simple as flip-flops, which consists of 8 to 16 gates, typically [BR-YO1, SIG, TI]. Excitation table methods are completely out of the question for \( 2^{16} \) gate-states. However, the number of stable states is usually very small. Secondly, the number of inputs to such flip-flops is typically 5. Again \( 2^5 = 32 \) columns are undesirable. Often sets of equivalent columns can be treated all at once, as we will show below. Thirdly, the computation of transitions can be rather tedious. We can reduce the amount of computation by (a) computing only those transitions that are necessary, (b) looking for forcing inputs (to be explained), (c) taking advantage of symmetry in the given network, if present, and (d) using a suitable model for analyzing complex races.

We now illustrate some of these ideas by analyzing the flip-flop of Fig. 4. It has 8 gates, with outputs \( y_1, \ldots, y_8 \), two external outputs \( \bar{Q} \) and \( Q \), and 5 inputs as follows:

- O-CLEAR \( C_0 \)
- O-PRESET \( P_0 \)
- CLOCK \( t \)
- DATA INPUTS \( J \) and \( K \)
We view the network of Fig. 4 as a proposed design for a positive-edge-triggered J-K flip-flop and our task is to verify that the designed network works properly.

The gate-excitation equations are:

\[ Y_1 = (C_0 y_5 y_7)' \quad Y_2 = (P_0 y_6 y_8)' \]
\[ Y_3 = (y_1 y_7)' \quad Y_4 = (y_2 y_8)' \]
\[ Y_5 = (P_0 y_1 y_4 y_6)' \quad Y_6 = (C_0 y_2 y_3 y_5)' \]
\[ Y_7 = (C_0 y_5 y_8)' \quad Y_8 = (P_0 y_6 y_7)' \]

Note that these equations are listed in pairs, taking advantage of the symmetry.

Our first goal is to find all the stable states. We examine various inputs in turn. For example, if we set \( P_0 = C_0 = 0 \) and keep these inputs constant, we find

\[ y_1 = y_2 = y_5 = y_6 = y_7 = y_8 = 1. \]

This in turn implies \( y_3 = y_4 = 0 \). Thus, for any input combination with \( P_0 = C_0 = 0 \) (eight input columns altogether), there is only one stable state \( A \triangleq 11001111 \). Thus \( P_0 = C_0 = 0 \) is "forcing" to \( A = 11001111 \).

Similarly we find:

\[ P_0 = 0, \ C_0 = 1, \ \ell = 0 \] is forcing to \( B \triangleq 11101101 \),
\[ P_0 = 0, \ C_0 = 1, \ \ell = 1 \] is forcing to \( C \triangleq 11101001 \),
and
\[ P_0 = 1, \ C_0 = 0, \ \ell = 0 \] is forcing to \( D \triangleq 11011110 \),
\[ P_0 = 1, \ C_0 = 0, \ \ell = 1 \] is forcing to \( E \triangleq 11010110 \).

When \( C_0 = P_0 = 1 \), no gate output is forced. Specifying also \( \ell = 0 \)
forces \( y_5 = y_6 = 1 \), but the other variables are not determined. Further, if we add \( J = K = 0 \), we find \( y_1 = y_2 = 1 \), but the remaining variables are still not determined. At this point we may pick the value of one of the gate variables and see whether a stable state exists with that value. It is preferable to pick a value that forces as many gates as possible. For example, we can choose \( y_7 = 0 \). Then, in addition to \( y_1 = y_2 = y_5 = y_6 = 1 \), we find \( y_3 = y_8 = 1 \) and \( y_4 = 0 \). Thus, if there exists a stable state with \( y_7 = 0 \) for the input \( C_0 = P_0 = 1, \xi = J = K = 0 \), that stable state must be \( 11101101 = B \). Using the excitation equations we verify that this state is indeed stable.

We must determine next whether any stable state with \( y_7 = 1 \) can exist for the given input. Recall that \( y_1, y_2, y_5 \) and \( y_6 \) are all forced to 1. Thus \( y_8 = (P_0 y_6 y_7)' \) must be 0, forcing \( y_4 = 1 \) and \( y_3 = 0 \). Again we verify that this state \( 11011110 = D \) is stable. In summary, for the input just examined, there are two stable states B and D. Repeating this type of analysis for the remaining input columns, we find the "flow table" of Fig. 5, disregarding the unstable *-entries. Thus we have a reasonably manageable \( 7 \times 13 \) table instead of the \( 256 \times 32 \) excitation table. We remark here that analytical methods for solving sequential Boolean equations [EV-ME] are, of course, applicable to the problem of finding stable states; however, these methods are quite laborious. Also, we may not dismiss this problem as easily programmable. One can enumerate all total states and check whether each is stable, but this would involve \( 2^8 \times 2^5 \) computations for the network of Fig. 4. Consequently, this straightforward programming approach is not realistic. Of course, one can build
heuristic short-cuts into the program, in a way similar to what we are describing.

To complete the analysis we must now find the *-entries. In this particular case, we assume that the J and K inputs will not change while the clock input $\phi$ is changing. Thus we are not interested in the entries marked $-$ in Fig. 5. Also the $P_0$ and $C_0$ inputs are intended to be the "asynchronous 0-PRESET and 0-CLEAR inputs", namely, $P_0 = 0$, $C_0 = 1$ "presets" the flip-flop, forcing $Q = 0$, $\bar{Q} = 1$, and $P_0 = 1$, $C_0 = 0$ "clears" the flip-flop forcing $Q = 1$, $\bar{Q} = 0$. The values $P_0 = C_0 = 0$ need not be used; this leads to the don't care entries marked $-$.

There now remain 24 transitions to be evaluated. This problem will be discussed in the next section.

4. RACES

The evaluation of unstable entries in our flow table will involve the study of complex races. We have dealt with races in more detail elsewhere [BR-Y02], and will adopt here the easily understood General Single Winner (GSW) model introduced there. We now define the model.

Let the network have $n$ binary inputs $x_1, \ldots, x_n$, and $s$ gates $G_1, \ldots, G_s$. The ordered $s$-tuple $y \triangleq (y_1, \ldots, y_s)$ is the gate-state, the ordered $n$-tuple $x \triangleq (x_1, \ldots, x_n)$ is the input state, and the ordered $s$-tuple $Y \triangleq (Y_1, \ldots, Y_s)$ is the gate-excitation, where $Y_j$ is the value of the Boolean function $f_j$.
computed by $C_j$ at $(x,y)$, i.e.,

$$Y_j = f_j(x_1, \ldots, x_n, y_1, \ldots, y_s), \quad j = 1, \ldots, s.$$  

Our problem is to determine the possible $y$-states that can result when the network starts in total state $(x,y)$. We assume that $x$ will be fixed.

We use the following notation. For

$$y = (y_1, \ldots, y_{i-1}, y_i, y_{i+1}, \ldots, y_s),$$

$$y^{(i)} = (y_1, \ldots, y_{i-1}, y'_i, y_{i+1}, \ldots, y_s),$$

In other words, $y^{(i)}$ is $y$ with the $i$th variable changed.

For each input $x \in \{0,1\}^n$ define a binary relation $R_x$ on the set \{0,1\}^s of gate-states as follows. Given $(x,y)$ compute $Y$. If $y = Y$, then $yR_x y$. Otherwise, for each $i$ such that $y_i \neq Y_i$, we have $yR_x y^{(i)}$. For example, if $y = 110$, $Y = 011$ for some $x$, then $110 R_x 010$ and $110 R_x 111$.

We now consider the relation diagram for $R_x$. The nodes correspond to gate-states (elements of \{0,1\}^s), and a directed edge leads from node $y$ to node $\bar{y}$ iff $yR_x \bar{y}$. If $yR_x y$ then $(x,y)$ is a stable total state. Otherwise consider all directed paths in the relation diagram for $R_x$ that start at node $y$. Any such path reaches a cycle after a finite number of steps. If a cycle of length greater than 1 is found, there is an oscillation. In this paper we assume that oscillations do not occur. Thus every cycle reached must be of length 1. If exactly one cycle (of length one) can be reached, the state corresponding to the node in the cycle is the unique stable state reached from $y$ under input $x$. If more than one cycle can be reached from $y$, the situation represents a critical race. For further
details see [BR-Y01, BR-Y02]. The main point here is that for each (x,y) we have a unique entry in the flow table to be defined. This entry may be y if (x,y) is stable, \( \bar{y} \) if only state \( \bar{y} \) can be reached from y under x, or a critical race.

The flow-table of a gate network is a table with rows corresponding to those gate-states y that are stable for at least one input. The columns correspond to input n-tuples. The entries in the table are computed using the GSW model.

In Fig. 6 we illustrate the evaluation of unstable entries for two of the transitions. The unstable variables are underlined and edges are labeled with the subscript of the changing variable. The remaining unstable entries are obtained similarly.

The portion of the flow table in double lines indeed verifies "synchronous" behaviour of the flip-flop. When \( \ell = 0 \) and the flip-flop is originally set, it is in state B or F. When the clock arrives and \( K = 0 \), the network goes to state C and does not change its Q-value. If \( K = 1 \) when the clock arrives the network goes to E and Q changes. Once the clock goes on, the state becomes independent of the J and K inputs. (This feature is sometimes called "data lockout", and because the J and K values present during the rising edge of the clock \( \ell \) determine the future value of Q, the flip-flop is called "positive-edge-triggered".) When the clock falls, the new value of Q is remembered. Similar action takes place when the network was originally reset. Incidentally, the interested reader is referred to [SIG, TI] for a master-slave edge-sensitive
flip-flop somewhat similar to the one we described above, the TTL unit 74110 with 16 gates.

5. ANALYSIS WITH MINIMAL FEEDBACK SETS

As has been explained in Section 2, the stable states can be computed using a minimal set of feedback variables. We illustrate this now for the network of Fig. 4. One can verify that at least 3 variables are required to "break" all the feedback loops. A possible minimal set is \{y_1, y_6, y_7\}, for:

\[
\begin{align*}
y_3 &= (y_1y_7)' \quad y_8 = (P_0y_6y_7)'
y_2 &= (P_0Ky_6y_8)' = (P_0Ky_6)' + (P_0y_6y_7)'
y_4 &= (y_2y_8)' = (P_0Ky_6)(P_0y_6y_7)' + (P_0y_6y_7) \\
    &= P_0Ky_6 + P_0y_6y_7 \\
y_5 &= (P_0Ky_1y_4y_6)' = (P_0Ky_1y_6 + P_0Ky_1y_6y_7)'
\end{align*}
\]

Thus, under stable conditions, the variables \(y_2, y_3, y_4, y_5\) and \(y_8\) can be expressed in terms of \(y_1, y_6, y_7\). After substituting and considerable manipulation and simplification, we obtain the following set of three equations:

\[
\begin{align*}
y_1 &= (C_0y_7)' + P_0Ky_1y_6 \\
y_6 &= (C_0)' + P_0Ky_6' + y_1y_7 \\
y_7 &= C_0' + P_0y_6y_7 + P_0Ky_1y_6
\end{align*}
\]
We can use our shortcut techniques as before, to compute the stable states, or check which of the 8 triples \((y_1, y_6, y_7)\) is stable. Either way we find that the following states are stable:

\[
\begin{align*}
C_0 = 0 : & \quad a \triangleq 111 \\
C_0 = 1, P_0 = 0, & \quad t = 0 : b \triangleq 110 \\
& \quad t = 1 : c \triangleq 100 \\
C_0 = 1, P_0 = 1, & \quad \ell = 0, \quad J = 0 : b \text{ and } a \\
& \quad J = 1 : b \text{ and } d \triangleq 011 \\
& \quad \ell = 1 : a \text{ and } c
\end{align*}
\]

This of course agrees with the table of Fig. 5, except that states A, D, and E all correspond to state a, (since we are examining only a subset of the gate variables), B and F correspond to b, C corresponds to c and G to d.

Now, for each stable total state we can compute the values of \(y_2, y_3, y_4, y_5\) and \(y_8\), thus finding the complete gate-state. Then we can proceed to the analysis of races, as in the previous section, to complete the construction of the flow table.

A fallacy of conventional theory states that the minimal feedback set is sufficient to describe the entire behavior of the network, not just its stable states. Thus the equations given above are used in the following form:

\[
\begin{align*}
Y_1 &= (C_0 y_7)' + P_0 \delta y_1 y_6 \\
Y_6 &= (C_0 t)' + P_0 K y_6 y_7' + y_1 y_7 \\
Y_7 &= C_0' + P_0 y_6 y_7 + P_0 \delta K y_1 y_6
\end{align*}
\]

i.e. they are used as excitation equations. If we construct the flow table
from these equations, we find the table of Fig. 7. Now consider Fig. 5 assuming that all unspecified entries constitute don't cares. One easily checks that Fig. 7 is a reduced version of Fig. 5! Indeed, Fig. 7 constitutes a reduced description of the edge-sensitive JK flip-flop. Have we just shown, therefore, that the method of minimal feedback set also gives the proper transitions?

The answer to this question involves some subtle points as we proceed to explain, but the answer is definitely **NO**, in general. Here the method works "by accident".

Equations (*) do describe the behavior of the network of Fig. 4 correctly, **under the assumption that gates 2, 3, 4, 5 and 8 have zero delay.** This is **not the case** for the network of Fig. 4. However, we are now in a position to explain why the minimal feedback analysis (leading to Fig. 7) "works".

The simple fact is that the network of Fig. 4 is very well designed: It does have races, such as that of Fig. 6(b), but the outcome of these races is always uniquely determined, regardless of the magnitudes of the individual gate delays. (Note that we are referring only to the specified entries in the table of Fig. 5; the remaining entries are of no interest for the time being.) Thus our network will still work if we assume that some delays are zero!

On the other hand, the reader can verify that, when the network of Fig. 4 is started in state F with \( C_0 = P_0 = \ell = 1, J = K = 0 \), the gate-state model predicts a critical race to either C or E, but the minimal-
feedback-set model predicts that the corresponding transition from b will always go to c. This again shows that the minimal-feedback-set model does not always yield the correct transition.
Fig. 1 Conventional analysis
(a) Given network N1
(b) Cutting a feedback line
(c) Inserting a delay.
Fig. 2 Excitation tables for $N_1$ (a) Using $\{w\}$
(b) Using $\{v\}$
(c) Using $\{v, w\}$
Fig. 3 Relay network for table of Fig. 2(a).

Fig. 4 Flip-flop to be analyzed
### Present State

<table>
<thead>
<tr>
<th>Present State</th>
<th>( P_0 = 0 )</th>
<th>( P_0 = 0 )</th>
<th>( P_0 = 1 )</th>
<th>( P_0 = 0 )</th>
<th>( P_0 = 0 )</th>
<th>( P_0 = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_0 = 0 )</td>
<td>( C_0 = 0 )</td>
<td>( C_0 = 0 )</td>
<td>( C_0 = 0 )</td>
<td>( C_0 = 0 )</td>
<td>( C_0 = 0 )</td>
<td>( C_0 = 0 )</td>
</tr>
<tr>
<td>( d = 0 )</td>
<td>( d = 0 )</td>
<td>( d = 1 )</td>
<td>( d = 0 )</td>
<td>( d = 1 )</td>
<td>( d = 1 )</td>
<td>( d = 1 )</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>00 01 11 10</td>
<td>00 01 11 10</td>
<td>00 01 11 10</td>
<td>00 01 11 10</td>
<td>00 01 11 10</td>
<td>00 01 11 10</td>
</tr>
<tr>
<td>A 11 B C D E</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
</tr>
<tr>
<td>B 01 C D E</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
</tr>
<tr>
<td>C 01 D E</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
</tr>
<tr>
<td>D 01 C 10 E</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
</tr>
<tr>
<td>E 01 C 10 D</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
</tr>
<tr>
<td>F 01 C 10 B</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
</tr>
<tr>
<td>G 01 C 10</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
<td>* * * * * *</td>
</tr>
</tbody>
</table>

### Forcing Inputs

Next State, \( Q_0 Q_1 \)

Fig. 5  Flow table for network of Fig. 4
Fig. 6 Examples of evaluating unstable entries
<table>
<thead>
<tr>
<th>Present State</th>
<th>( P_0 = 0 )</th>
<th>( P_0 = 0 )</th>
<th>( P_0 = 1 )</th>
<th>( t = 0 )</th>
<th>( t = 1 )</th>
<th>JK</th>
<th>JK</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {A,D,E} ) 111 a</td>
<td>( a )</td>
<td>11</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>( \bar{a} )</td>
<td>10</td>
</tr>
<tr>
<td>( {B,F} ) 110 b</td>
<td>a</td>
<td>( \bar{b} )</td>
<td>01</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>( \bar{b} )</td>
</tr>
<tr>
<td>( {C} ) 100 c</td>
<td>a</td>
<td>b</td>
<td>( \bar{c} )</td>
<td>01</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>( {G} ) 011 d</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

Next State, \( \bar{Q}, Q \)

Fig. 7 Flow table using minimal feedback set \( \{y_1, y_6, y_7\} \)
REFERENCES


