A Tabular Approach to Transition Signaling

1. Introduction

Transition signaling, (or 2-phase handshaking) has obvious advantages over 4-phase handshaking in that, all other things being equal, it should result in faster operation and in less power dissipation. In a number of cases, it also leads to simpler circuits. But there are also many cases where a module based on transition signaling is more complex than is its 4-phase counterpart. Both 2-phase and 4-phase systems are being actively investigated, and, particularly because it is not difficult to translate between subsystems using different conventions, it may be useful to incorporate both approaches in the same overall system.

Among those who have been particularly productive in the transition signaling area are Sutherland [89], Rosenberger, Molnar, Fang, and Chaney [88], Brunvand [91], and Patra and Fussell [93]. Keller [74] did important basic work on unclocked systems with various protocols, including both transition signaling and 4-phase handshaking, as did Seitz [80].

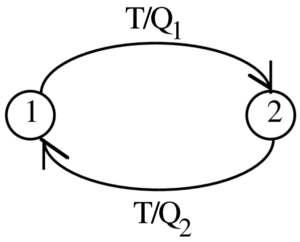
The above mentioned researchers, and others, have defined and designed various modules useful as building blocks for systems using transition signaling. In general, the functions implemented by these blocks are described either informally or in terms of Petri nets, trace-theory notation, or some other formal language. Implementations are then presented. These are often discussed informally to give the reader confidence that they are valid, and then it is shown how they may be used to realize more complex functions. What is generally missing is a design procedure linking the formal descriptions to the implementations, which would enable the reader to reproduce the design process for the given modules, to verify more rigorously the correctness of the given designs, or to design other modules in a systematic manner. The objective of this work is to sketch a procedure for filling in this gap. What is presented here does not constitute any startling new technique, but rather introduces (or perhaps of the literature in this field) a straightforward means for specifying the functions, and then points out that the resulting description can be algorithmically converted into a form that allows the use of all of the methods developed for the formal synthesis of proper asynchronous sequential circuits (e.g. Unger [69, 89]). The objective is to connect much of the recent work on unclocked systems to the existing body of logic circuits knowledge. An alternative synthesis process, based on the use of a few transition signaling building blocks, is also described that can be applied directly to the transition flow table.

2. The Transition State Table

A device appearing in a number of studies of asynchronous systems is often referred to as a *transition toggle*. It has one input T, and outputs Q_1 and Q_2. Following a transition on T, there is a transition on just one of the outputs, and this alternates for each T-transition. Thus a trace theory description of what the device does is: **pref**(T? Q_1! T? Q_2!)*. The same behavior can be succinctly described by the transition flow (or state) table shown in Fig. 1a. The heading of the column represents a transition of variable T. The cell entries indicate respectively the next state and the output on which a transition is to appear. As in the case of flow tables for synchronous or iterative circuits, the operating point for these tables is the row. Starting in some row (unless otherwise specified, the initial state corresponds to row-1) upon receipt of an input, a check is made of the cell in the corresponding column, the indicated output is produced and the operating point is moved to the row indicated as the next state. In general, such tables may have a multiplicity of columns representing different input variables, there may be several outputs, and, as is illustrated in subsequent examples, some of the input or output signals may be levels rather than transitions. An isomorphic description in the form of a state *graph* is shown in Fig. 1b. (Note that, computerized versions of these two forms would be identical. Which one a person chooses to use is generally a matter of taste, although there are situations in which one or the other may seem more suitable.)

	T
1	2,Q_1
2	1,Q_2

(a)



(b)

Fig. 1 State Table for Transition Toggle

Given such a table, a primitive Huffman type flow table can be generated in a routine manner, as illustrated in Fig. 2. This table has a column for each possible input state; in this simple example, 2 columns are required, corresponding to the two possible values of input T. (Next-state entries in stable states are emphasized by being placed in parentheses.) Table entries are generated one by one. In position 1-0 (row-1, column-0) a stable state is placed, with both outputs set at 0 (some other state for the outputs could be chosen if desired). The superscript 1 on the next-state entry is a comment that this total state corresponds to row-1 of the transition table. Starting from this total state, a transition on T corresponds to an input change to the 1-column. From Fig. 1, we see that a T-transition with the device initially in state-1 means that output Q_1 should change, and that the next state should be row-2. Therefore, in the primitive flow table, we insert in position 1-1 a next-state entry 2, and in row-2 of that column create a new stable state with Q_1 =1. The superscript here is a 2, indicating that this stable state corresponds to row-2 of the transition table.

	T	
	0	1
1	(1^1),00	2, 00
2	3,10	(2^2),10
3	(3^1),11	4, 11
4	1,01	(4^2),01

Fig. 2 Primitive flow table for transition toggle

A second change in T returns the operating point to column-1 of row-2. Since the corresponding entry in the transition table points to row-1 and indicates that Q_2 is to change, this means that the next state must have outputs 11. A new state is therefore necessary; designated as row-3, and its superscript is 1, since it maps into row-1 of the transition table. It should now be evident how the remaining entries were obtained. The outputs in the transient states were chosen so that output changes do not occur until the stable state is reached after each transition.

Note the substantial increase in the size of the primitive table as compared to the transition table. The number of columns increases to represent the number of input *states* as opposed to the number of inputs, and the number of rows grows as the number of output states. This case is unusual in that the primitive flow table is not reducible.

The synthesis process in this case is also unusually straightforward. The obvious STT state assignment is shown in Fig. 3.

	T		
	0	1	y_1y_2
1	(1^1),00	2, 00	0 0
2	3,10	(2^2),10	0 1
3	(3^1),11	4, 11	1 1
4	1,01	(4^2),01	1 0

Fig. 3 Primitive flow table for transition toggle

Using Karnaugh Maps, or any other convenient design tool, we can now generate the logic expressions shown below:

$Y_1=\bar{T}y_2+(T+y_2)y_1$, $Y_2=T\bar{y}_1+(\bar{T}+\bar{y}_1)y_2$, $Q_1=y_2$, $Q_2=y_1$

An examination of the expressions for the Ys reveals that these are the logic expressions for transparent latches. The latch generating Y_1 has T as its C-input and $\bar{y}_2$ as its D-input, while the latch generating Y_2 has $\bar{T}$ and y_1, respectively, as its C- and D-inputs. The resulting circuit, shown as Fig. 4, is precisely that shown in the Appendix of Brunvand's dissertation [].

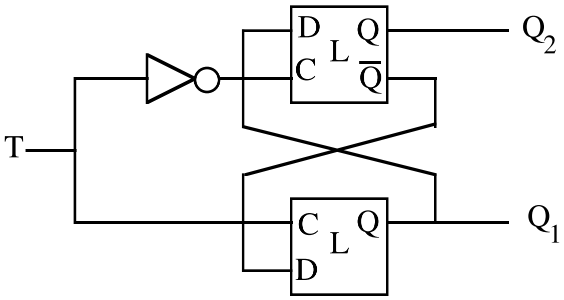


Fig. 4 Implementation of transition toggle

It is evident from the flow table that there are steady state essential hazards for every transition. But, unless there is an unusual distribution of wiring delays, this is unlikely to cause any trouble with this circuit. In any event the designer can check out the delays along the appropriate paths to ensure that all is well.

The next example introduces a new aspect in that there is a level input G as well as a transition input T. This what has been called a transition select module. It has a level input G, which steers a transition on input T to output Q_0 or Q_1 depending on whether G is) or 1 respectively. The transition flow table is in Fig. 5. It has two columns, the first where G=0 and T is changing, and the second where G=1 and T is changing. (It is assumed that only one of the two inputs changes at any time, and that the system is stable at the time of any change.) The description is quite simple. There is only one row of the table, so no next-state entries are necessary. If there is a transition on T while G=0, then a transition is to appear on Q_0. If G=1 when a transition on T appears, then a transition appears on Q_1. Because there are two input signals, the number of columns of the primitive flow table (Fig. 6) is 4, and, because of the fact that there are two outputs, the number of flow table rows is 8. Note that not all possible combinations of input and output states can occur starting from a given initial state. This is true of many transition signaling devices and saves us from even greater explosions of flow table size.

G	
T0	T1
Q_0	Q_1

Fig. 5 Transition table for transition select module

	TG			
	00	01	11	10
1	(1),00	2,00		3,~~0~~0
2	1,00	(2),00	8,0~~0~~	
3	1,~~1~~0		4,10	(3),10
4		5,1~~0~~	(4),10	3,10
5	6,11	(5),11	4,1~~1~~	
6	(6),11	5,11		7,~~1~~1
7	6,~~0~~1		8,01	(7),01
8		2,0~~1~~	(8),01	7,01

Fig. 6 Primitive flow table for transition select module

(Note that in certain transient states, an output may appear as ~~0~~ or ~~1~~. Such entries should be considered as 0s or 1s respectively if the rows containing them are not combined with other rows, but they may be treated as don't cares for the purpose of such merging of rows. These are used where we wish to delay output changes until the next stable state is reached.) As is often the case, there are many don't care entries in Fig. 6 and this means that there are many options for reducing the table. The choice made in such situations may have significant effects with respect to the circuit found. It is generally necessary to try a number of solutions in order to arrive at a near optimal circuit. In this case, good results follow from the reduction shown in Fig. 7, which is the result of combining the pairs of rows, 12, 34, 56, 78. There are no essential hazards associated with this table.

	TG				
	00	01	11	10	y_1y_2
1	(1),00	(1),00	4,00	2,00	0 0
2	1,10	3,10	(2),10	(2),10	0 1
3	(3),11	(3),11	2,11	4,11	1 1
4	3,01	1,01	(4),01	(4),01	1 0

Fig. 7 Reduced flow table for the transition select module

The logic expressions are:

$Y_1=\bar{G}y_1+G(\bar{T}y_2+T\bar{y}_2)=\bar{G}y_1+G(T\oplus y_2)$, $Y_2=Gy_2+\bar{G}(\bar{T}y_1+T\bar{y}_1)=Gy_2+\bar{G}(T\oplus y_1)$, $Q_1=y_2$, $Q_2=y_1$.

Here the Y_1 expression corresponds to a latch with $C=G$ and $D=T\oplus y_2$, and the Y_2 expression corresponds to a latch with $C=\bar{G}$ and $D=T\oplus y_1$. The resulting circuit, shown in Fig. 8, is the same as the one in Fig. A.5 of Brunvand's dissertation.

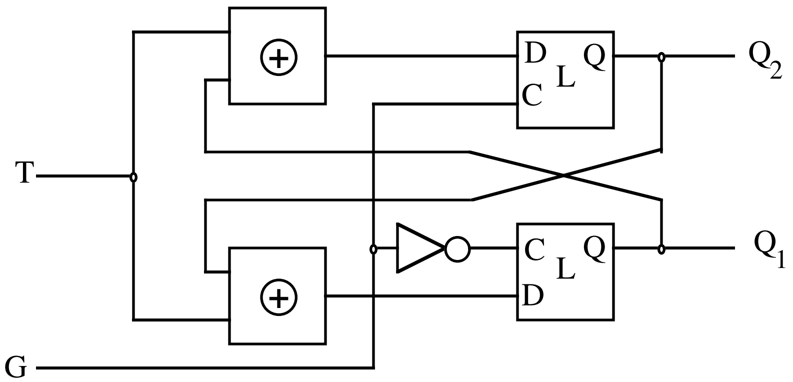


Fig. 8 Implementation of transition select module

The next example is the transition signaling version of the familiar RS flip-flop. It might be called an RS transition FF (Brunvand refers to it as a "Boolean variable"). The inputs are R and S, each being in the form of a transition signal. The output is a level signal Q, whose value indicates which of R or S last was active. If the most recent input was a transition on R, then Q=0, else if S was most recently active, Q=1. The transition flow table is shown in Fig. 8. Here the Q value is shown to be associated with the rows of the table. There is also a transition output A, which acknowledges transitions on R or S. (In Brunvand's version, there are distinct acknowledge signals for R and S. This difference has little consequence for the designs. It is a simple matter to modify the resulting circuit to get either type of acknowledgment.)

	R	S	Q
1	1,A	2,A	0
2	1,A	2,A	1

Fig. 9 Transition table for the RS transition FF

The resulting primitive flow table is shown in Fig. 10. (The first output is Q, the second is A.)

	RS			
	00	01	11	10
1	(1^1),00	3,-~~0~~	-	2,0~~0~~
2	1,0~~1~~	-	7,-~~1~~	(2^1),01
3	8,-~~1~~	(3^2),11	4,-~~1~~	-
4	-	5,0~~0~~	(4^1),00	6,-~~0~~
5	8,-~~1~~	(5^1),01	4,0~~1~~	-
6	1,-~~1~~	-	7,1~~1~~	(6^2),11
7	-	5,-~~0~~	(7^2),10	6,1~~0~~
8	(8^2),10	3,1~~0~~	-	2,-~~0~~

Fig. 10 Primitive flow table for RS transition FF

A reduced version of this primitive flow table is shown as Fig. 11. There are no essential hazards for this flow table.

	RS				
	00	01	11	10	y_1y_2
1	(1),00	2,-0	4,-1	(1),01	0 0
2	(2),10	(2),11	3,-1	1,-0	0 1
3	2,-1	(3),01	(3),00	4,-0	1 1
4	1,-1	3,-0	(4),10	(4),11	1 0
				QA	

Fig. 11 Reduced flow table for RS transition FF

The logic expressions for this table, with the given state assignment are:

$Y_1=RS+Sy_1+Ry_1=M(R,S,y_1)$ (Where M is the majority function, so Y_1 corresponds to a C-element)

$Y_2=\bar{R}S+Sy_2+\bar{R}y_2=M(\bar{R},S,y_2)$ (So Y_2 also corresponds to a C-element)

$A=R\bar{y}_1\bar{y}_2+S\bar{y}_1y_2+\bar{R}y_1y_2+\bar{S}y_1\bar{y}_2$.

A can be realized by a complex CMOS gate where the pullup network (setting A to 1) is:

$A^+=R\bar{S}\bar{y}_2+\bar{R}Sy_2$

and the pulldown network (setting A to 0) is:

$A^-=\bar{R}\bar{S}\bar{y}_1+RSy_1$

This requires a total of 12 transistors for A. (If we wished to generate separate acknowledgments for R and S, this could be done with 2 circuits of this type, each with 6 transistors. The resulting circuit is shown in Figure 12 (without the A-circuit). It is of about the same complexity as the rather different solution presented by Brunvand.

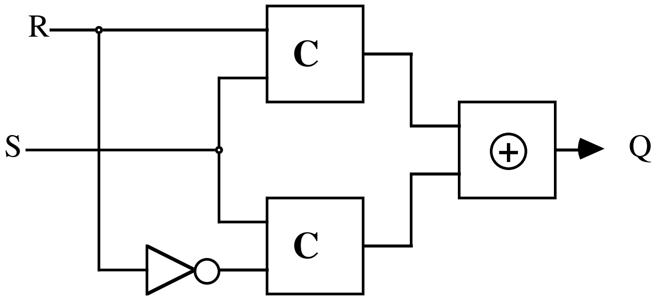


Fig. 12 RS transition FF (A-circuit not shown)

The call module is a device that seems to have been first described by Keller []. It has inputs R_1, R_1, A_s, and outputs R_s, A_1 and A_2. Following a signal on R_1 or R_2, an R_s-signal is generated. This eventually produces an A_s-input. (No other R_1- or R_2-input can occur before the A_s-input.) The module responds with an A_1-output if the process began with an R_1-signal or an A_2-output if the process began with an R_2-signal. In the present example, all input and output signals are transitions. A transition table description is in Fig. 13.

	R_1	R_2	A_s
1	2,A_s	3,A_s	-
2	-	-	1,A_1
3	-	-	1,A_2

Fig. 13 Transition table for call module

This expands to a 12-row, 8-column primitive flow table as shown in Fig. 14.

$R_1R_2A_s$

	000	001	011	010	110	111	101	100
1	(1^1),000	-	-	3,-00	-	-	-	2,-00
2	-	-	-	-	-	-	4,1-0	(2^2),100
3	-	-	7,10-	(3^3),100	-	-	-	-
4	-	5,-10	-	-	-	6,-10	(4^1),110	-
5	1,0-0	(5^2),010	-	-	-	-	-	-
6	-	-	-	-	10,01-	(6^3),010	-	-
7	-	9,-01	(7^1),101	-	-	8,-01	-	-
8	-	-	-	-	12,0-1	(8^2),001	-	-
9	1,00-	(9^3),001	-	-	-	-	-	-
10	-	-	-	11,-11	(10^1),011	-	-	12,-11
11	-	-	7,1-1	(11^2),111	-	-	-	-
12	-	-	-	-	-	-	4,11-	(12^3),111

$R_sA_1A_2$

Fig. 14 Primitive flow table for call module

There are many ways to reduce this table. One that produces a good result is shown in Fig. 15.

$R_1R_2A_s$

y_1y_2	000	001	011	010	110	111	101	100	
1	(1^1),000	-	4,10-	(1^3),100	-	-	2,1-0	(1^2),100	0 0
2	1,0-0	(2^2),010	-	-	3,01-	(2^3),010	(2^1),110	-	0 1
3	-	-	4,1-1	(3^2),111	(3^1),011	-	2,11-	(3^3),111	1 1
4	1,00-	(4^3),001	(4^1),101	-	3,0-1	(4^2),001	-	-	1 0

$R_sA_1A_2$

Fig. 15 A reduced flow table for the call module

There are no essential hazards for this flow table. Logic expressions realizing this flow matrix are:

$Y_1=X_2y_1+(X_2+y_1)(X_1 \oplus X_3)=C(X_2,(X_1 \oplus X_3))$,

$Y_2=X_1y_2+(X_1+y_2)(X_2 \oplus X_3)=C(X_1,(X_2 \oplus X_3))$,

$Z_1=X_1 \oplus X_2$, $Z_2=y_2$, $Z_3=y_1$

This leads to the circuit of Fig. 16, which is another circuit in Brunvand's dissertation.

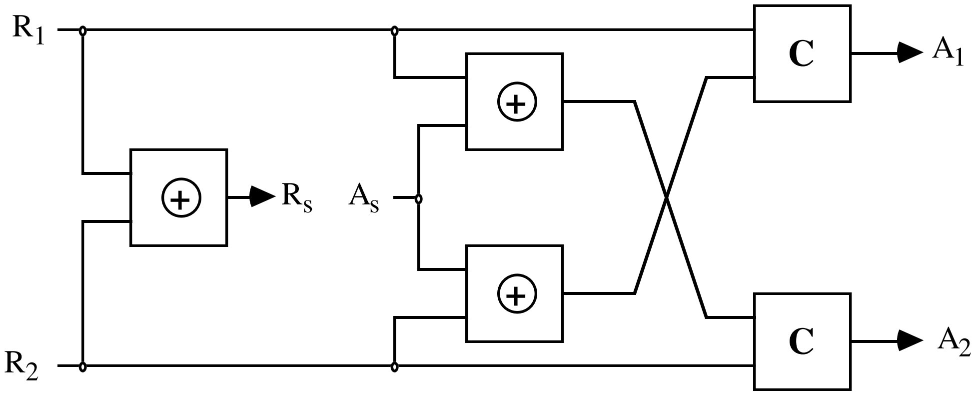


Fig. 16 An implementation of a transition call module

In some situations, it is appropriate to head a column of a transition flow table with a set of transition signals. The corresponding input is considered to be effective only after all of the variables in the set have undergone transitions. This constitutes a burst mode input, as discussed Dill and Nowick.

3. An Alternative Approach to Synthesis from Transition Tables

Since transition tables are generally very much more concise than are the primitive tables that they imply, it seems worthwhile to look for means for using them more directly to generate circuits. One approach along these lines (which has not yet been studied carefully) is to utilize as basic components the transition select module and the RS transition FF. The method would work as follows:

1. Assign a binary code to each row of the transition flow table.

2. Using the same technique employed when conventional RS-FFs are used as state devices, generate a logic expression from the table for each R and S input of a set of RS-transition FFs used to represent the states developed in step-1.

3. Where such an expression involves the Boolean sum of two transitions, use an XOR function instead.

4. Where a Boolean product of a transition variable and an ordinary Boolean expression is involved, use a transition select module with the G-input from the Boolean expression, and the T-input from the transition signal.

5. If a burst of transitions must be used, the corresponding signal is taken from the output of a C-element fed by these signals.

Although this procedure is relatively easy to implement, it does not seem to lead to economical logic for simple cases. The problem is that the basic elements are too costly. It is not clear whether or not the disadvantage would grow, diminish, or remain about the same for more complex subsystems. Finding a simpler set of basic components might make this approach more useful.

4. Conclusions

Using transition flow tables as a starting point facilitates the use of classical asynchronous sequential logic circuit methods for the design of unclocked systems. It is a simple matter to transform these tables into primitive flow tables. (A program to accomplish this has been written recently by John Cheng, a Columbia PhD student.)

Once this has been done, the designer has many options, since the primitive flow tables can be reduced in many ways, and even a particular reduced flow table may have many don't care entries that offer numerous alternatives. It is interesting that the 12-row primitive flow table for the call module (Fig. 14) can be reduced to a 2-row table as well as to any of several 4-row tables. The 2-row table in turn leads to a circuit somewhat different from the one presented here, although of similar complexity.

Because of many symmetries present in flow tables for transition signaling modules, it seems that decomposition techniques (Tan [69]) might usefully be re-examined. Perhaps decomposition efforts could be guided by information from the original transition tables.

References

Brunvand, Eric, "Translating Concurrent Communicating programs into Asynchronous Circuits", PhD Dissertation, School of Computer science, Carnegie Mellon U., September, 1991

Keller, Robert, "Towards a Theory of Universal Speed-Independent Modules", *IEEETC*, V. C-23, No. 1, Jan. 1974, 21-33.

Nowick, Steven M., and David Dill, "Synthesis of Asynchronous State Machines Using a Local Clock", *International Conference on Computer Design*, 1991.

Patra, Priyadarsan, and Donald Fussell, "Building Blocks for Designing DI Circuits", TR-93-23, CS Dept., U. Texas, Austin, November, 1993.

Rosenberger, Fred U., Charles E. Molnar, Thomas Chaney, Ting-Pien Fang, "Q-Modules: Internally Clocked Delay-Insensitive Modules", *IEEETC*, V 37, No. 9, September, 1988, 1005-1018.

Seitz, Charles, "System Timing", in *Introduction to VLSI Systems,* Mead, Carver, and Lynn Conway, Addison-Wesley, 1980, 218-262.

Sutherland, Ivan, "Micropipelines", *Communications of the ACM*, V. 32, No. 6, June 1989, 720-738.

Tan, C. J., P. R. Menon, A. D. Friedman, "Simplification and Decomposition of Asynchronous Sequential Switching Circuits", IEEE-TC, September, 1969.

Unger, Stephen H., *Asynchronous Sequential Switching Circuits*, Wiley, 1969 (re-issued by Robert E. Krieger, Malabar, 1983)

Unger, Stephen H., *Essence of Logic Circuits*, Prentice-Hall, 1989.