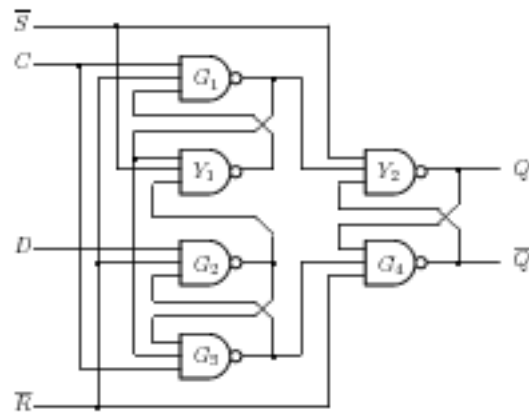


Advanced Logic Design Techniques
in
Asynchronous Sequential Circuit Synthesis

Charles R. Bond
<http://www.crbond.com>



©1990 – 2008, All rights reserved.

Contents

I	Synthesis Methods	4
1	Development of Methods and Techniques	7
1.1	Derivation and Construction of a 'T' Flip-Flop	7
1.1.1	The Flow Table	7
1.1.2	The Secondary Equations	9
1.1.3	Implementing the 'T' Flip-Flop	10
1.2	'D' Flip-Flop	12
1.2.1	The Flow Table	12
1.2.2	The Circuit Equations	14
1.2.3	Location and Identification of Signals	20
1.2.4	Asynchronous Set and Clear	21
1.3	Negative Pulse Generator	23
1.3.1	The Flow Table	23
1.3.2	Equations and Implementation	24
1.3.3	Analysis of Implementation	24
1.4	Up-Down Counter Controller	25
1.4.1	The Flow Table	25
1.4.2	Equations and Implementation	27
1.5	A Phase/Frequency Detector	28
1.5.1	The Flow Table	29
1.5.2	The Equations	32
2	The Complete Synthesis Method	37
3	Miscellaneous Problems and Solutions	40
3.1	Clock Stream Switch	40
3.2	Two-Phase Clock Generator	42
3.3	Glitch Suppressor	44
3.4	Digital Single Shot	45
3.5	Master/Slave J-K Flip-Flop	46
3.5.1	The Flow Table	46
3.5.2	The J-K Flip-Flop Circuit Equations	47
3.6	Edge-Triggered J-K Flip-Flop	53
3.6.1	The Flow Table	53

3.6.2	Edge-Triggered J-K Flip-Flop Flow Table	53
3.6.3	The Circuit Equations	54
3.6.4	The Implementation	55

II Analysis Methods 58

4	Sequential Circuit Analysis	59
4.1	Fundamental Properties of Secondary Variables	59
4.2	An Outline of the Analysis Method	61
4.3	A Simple Example	61
4.3.1	Labelling the Circuit Elements	61
4.3.2	Writing the Raw Circuit Equations	61
4.3.3	Reducing the Equations	62
4.3.4	Constructing the Flow Table	63
4.3.5	The Circuit Description	64
4.4	Another Example	64
4.4.1	The Circuit Equations	64
4.4.2	The Flow Table	65
4.5	Caveats and Cautions	66

List of Figures

1.1	NAND Implementation of Y_1	10
1.2	Partial Construction of “T” Flip-Flop	11
1.3	Construction of ‘T’ Flip-Flop	12
1.4	Implementation of ‘T’ Flip-Flop with Hazard Suppression	13
1.5	Implementation of ‘D’ Flip-Flop	15
1.6	Combining a Signal with its Inverse	16
1.7	Suppressing a Hazard by Cancellation	16
1.8	Combinational Circuit with Undesired Transient	17
1.9	Combinational Circuit with Subscripted Signal	18
1.10	Circuit Solution Providing Hazard Suppression	18
1.11	Hazard-free Implementation of ‘D’ Flip-Flop	20
1.12	Complete Implementation of ‘D’ Flip-Flop with Set and Clear	22
1.13	Negative Pulse Generator	24
1.14	Up-Down Counter Controller	28
1.15	Partial Implementation of Phase/Frequency Detector	34
1.16	Complete Implementation of Phase/Frequency Detector	35
3.1	Clock Stream Switch	42
3.2	Timing Diagram for Two-Phase Clock Generator	42
3.3	Two-Phase Clock Generator	44
3.4	Positive Glitch Suppressor	45
3.5	Timing Diagram for Digital Single Shot	45
3.6	Digital Single-Shot	46
3.7	Implementation of Y_1	49
3.8	Box Model of Logic Circuit	49
3.9	Box Model of J-K Flip-Flop	50
3.10	Partial Implementation of Y_2	52
3.11	Master/Slave J-K Flip-Flop	52
3.12	1st Partial Implementation of Y_2	55
3.13	2nd Partial Implementation of Y_2	55
3.14	Edge-Triggered J-K Flip-Flop	56
4.1	Cross-Coupled Flip-Flop (Latch)	60
4.2	Unknown Circuit with Labels	62
4.3	2nd Example Circuit	65

Part I

Synthesis Methods

Introductory Notes

In this paper a number of advanced techniques for solving sequential logic circuit design problems are developed. Special methods are presented for taking a problem from its initial statement to a fully implemented solution. The objective is to find practical solutions for a variety of typical sequential circuit problems. Each problem begins with a problem statement and proceeds with a flow table description of the desired behavior, the derivation of the circuit equations, and the implementation of those equations using standard logic devices. A typical solution will be constructed from some family of logic gates and inverters. See [Huff 54, Cald 58] for complete information on the flow table method of synthesis. Many of the problems posed here yield solutions which will be familiar to the reader. The emphasis is more on ‘how to get there’ than on whether anyone has been there before. Some complex problems with novel solutions are also treated, however, and others can be readily developed.

It is assumed that the reader is already familiar with combinational logic problems and solutions using Boolean algebra and with the use of Karnaugh maps. See [Karn 53] for details. A thorough understanding of DeMorgan’s theorem and its application to logic devices is also assumed.

Some experience with state machines would be helpful, but not necessary, as we will exclusively use flow tables to specify sequential circuit behavior. Thus, any previous experience in generating flow table descriptions of logic blocks will aid materially in getting through the various exercises. Other synthesis methods can be found in [Moore 54, Maley 63, Mealey 55].

The reader should be advised that this paper deals with design at the lowest *logic* level; *i.e.*, we will be designing logic circuits using NAND gates, NOR gates, etc. We will not address the higher level problem of building counters from flip-flops or state machines from counters. Nor will we deal with the lower level (analog) problem of designing *NAND* or *NOR* gates using discrete components. Typical problems will be the design of flip-flops, arbiters, synchronous switches and a variety of other sequential circuit blocks which are used in complex digital systems. Treatments of synthesis using higher level logic blocks can be found in many digital design texts and in [Maley 63, Marc 62, Cald 58].

The terms *synchronous* and *asynchronous* are used in a context sensitive manner. In general, the terms are used to distinguish between logic circuits which only change external states following changes in a particular input (clocked or ‘synchronous’ behavior) from those whose external states may change following changes in any or all inputs (‘asynchronous’ behavior). The circuits we will design are ‘input driven’, which means that the internal states will change following changes in inputs, and output states may or may not change. An external clock will not be required to trigger state changes, although there is nothing to prevent us from designating one of the circuit inputs as a clock. When designing circuits intended for use in synchronous (clocked) systems, we will often take one of the inputs as the ‘synchronizer’ (clock) and develop the behavioral description accordingly.

For example, the ‘D’ flip-flop is a standard logic storage element in synchronous systems where one input is designated as a clock input. Even though the internals of the flip-flop are asynchronous, the outputs are synchronous with the clock. However, in the discussions of the flip-flop *set* and *clear* signals, we will refer to those inputs as asynchronous, since they drive the output directly, independent of the clock. In reality, these inputs are neither more nor less asynchronous than any other part of the circuit. It is purely a matter of convenience at a higher level that we create these distinctions.¹

One caution should be mentioned. There are no detailed discussions in this paper about device timing requirements. Specifications for gate delays, rise and fall times, setup and hold time, etc. are beyond the scope of this paper. In fact, such timing issues are crucial in any sequential circuit and it is not always possible to guarantee performance when multiple input changes occur at or about the same time. Specific timing requirements are entirely implementation dependent and are best determined on completion of the design. No functional problems should arise if all gates have approximately the same delay or, as a minimum condition, if the longest gate delay through a single gate is less than the shortest delay through two gates. For the most part, the only restriction on rise and fall times is that they should be less than the shortest gate delay.

Plan to review the circuits after they are finished to determine the minimum *setup* and *hold* time requirements for each input. Optimization techniques for logic minimization and layer reduction should be considered, as well.

Most of the designs have been implemented with NAND gates. There is no requirement for this, except that it often simplifies the descriptions of the implementation techniques. In some cases, there will be good reasons to make use of other logic blocks, and in those cases we will not hesitate to do so.

Although the primary objective of this paper is to develop design methods, in Part II a method for analyzing existing circuits is presented. The techniques used to derive circuit equations from existing sequential circuits are not generally covered in existing texts and appear to be unknown to many designers. For some readers this section will provide a useful complement to the core material.

¹Perhaps the term *non-sequential* would be better than *asynchronous*.

Chapter 1

Development of Methods and Techniques

1.1 Derivation and Construction of a ‘T’ Flip-Flop

The ‘T’ (toggle) flip-flop has a single input, T , and a single output, Q . It serves as a divide-by-two circuit in counter and control applications and can be constructed from simple gates, as will be shown in the following paragraphs.

1.1.1 The Flow Table

For this device, the output changes state, or toggles, each time the input goes positive. A flow table, which provides a concise description of its behavior, is in Table 1.1. The flow table is organized with the input (externally controlled)

T		Q
0	1	
(1)	2	0
3	(2)	1
(3)	4	1
1	(4)	0

Table 1.1: Flow Table for ‘T’ Flip-Flop

signals labelling the columns. Responses are indicated by providing rows associated with critical output signals.

To see that this table provides a complete circuit description, begin by considering state (1). Stable states are represented by bold numbers surrounded

by parentheses, and unstable or transitional states are represented by ordinary numbers. Notice that in state **(1)**, the input, T , is zero and the output, Q , is also zero. When T changes state to a one, the circuit enters the transitional state indicated with a ‘2’ and then drops into stable state **2** in the next row. The required value of Q is now one. When T changes back to a zero, the circuit responds by moving to state **3**, with no change in Q . The next change of T (to a one) causes Q to go to zero with the circuit entering state **(4)**. Finally, another change of T returns the circuit to its original state. Note that changes of the input variable cause movement from column to column, and internal state changes cause movement along the rows. See [Huff 54].

For most flow tables the next step would be to attempt a state reduction, but in this case no reduction is possible because none of the rows in the table can be ‘merged’ with any of the others. (Other examples in this paper will deal with state reduction.)

After constructing the flow table, we associate each row with an internal state which can be identified with some combination of internal variables. Since there are four rows in the table, two binary symbols will be required.¹ There is considerable freedom in assigning the internal (secondary) variables, and it is customary to do so in a way which will simplify the derived equations. We will set the values of the secondaries in the first row to zero, and use a gray code for subsequent rows to assure that only one internal variable will change when the circuit moves from any row to an adjacent row. See Table 1.2.

		T		
y_1	y_2	0	1	Q
00	(1)	2		0
01	3	(2)		1
11	(3)	4		1
10	1	(4)		0

Table 1.2: Flow Table with Secondary Variable Assignments

Note that with the secondary variable assignments we have chosen, the value of y_2 follows the output, Q . When we construct the gate which generates y_2 , which will be labelled Y_2 , we can equate it with Q .

A stable state is defined as a state in which the secondary variables have settled; *i.e.*, no transitions are occurring internally. To make use of the table, we replace the state numbers with their Boolean equivalent row assignments. For example, the stable state cell in row 1, with secondary assignment 00 will contain 00. The unstable state in row 1, which leads to row 2, will contain the assignment 01. Continuing in this manner, we arrive at Figure 1.3. For the table, we are not concerned with which cells represent stable states and which

¹Since the symbols are binary, we use: $\text{ceil}(\log_2 \text{number_of_states}) = \text{number_of_variables}$.

do not. We are only concerned with deriving a truth table from which we can extract the equations for the secondary (internal) circuit elements.

$y_1 y_2$	T		Q
	0	1	
00	00	01	0
01	11	01	1
11	11	10	1
10	00	10	0

Table 1.3: Flow Table with Boolean State Entries

The leftmost entries of each cell in Figure 1.3 represent the truth table for y_1 and the rightmost entries represent y_2 . To simplify the derivation of the corresponding equations, we will split the table into two separate tables, one for each element. This done in Table 1.4.

$y_1 y_2$	T		$y_1 y_2$	T	
	0	1		0	1
00	0	0	00	0	1
01	1	0	01	1	1
11	1	1	11	1	0
10	0	1	10	0	0

Y_1
 Y_2

Table 1.4: Split Internal State Tables

1.1.2 The Secondary Equations

There is now a truth table for each internal element. These elements are capable of distinguishing between internal states by providing unique codes for each row in the state table. We are now able to write the circuit equations, which can be taken directly from the tables.²

$$Y_1 = \bar{T}y_2 + y_1y_2 + Ty_1 \quad (1.1)$$

$$Y_2 = \bar{T}y_2 + \bar{y}_1y_2 + T\bar{y}_1 \quad (1.2)$$

A note about the symbols used in forming circuit equations is in order. It is customary to label the logic elements used to implement the equations with

²The redundancy in these expressions will be taken up later.

upper case letters, *e.g.* Y_1 , Y_2 . These elements (gates) can also be regarded as synonymous with their output function or signal, *i.e.* y_1 , y_2 , since there is a one-to-one correspondence here. The standard practice is to use the upper case label on the left side of circuit equations to emphasize the fact that we are connecting devices together to implement our solutions. Some caution is in order in adopting these conventions, because the equations are then not strictly reciprocal mathematical relations, but are more like process flow statements or cause and effect relations. In general, no problems should arise from mixing device labels with circuit function, but keep in mind that Y_1 is *not* strictly identical with y_1 . Where circuit feedback exists, be sure to verify the signal flow.

Part of the challenge in the implementation phase of the design is in carefully selecting and including redundant terms. Generally, it is advisable to include the terms which tie row elements together even if they are already included in vertical tie sets. We have done this with the y_1y_2 term in the equation for Y_1 and the \bar{y}_1y_2 term in the equation for Y_2 . The intent is to eliminate critical hazards associated with column movement in the resulting circuit.³

The next challenge is to group the terms in each equation for easy assignment to standard logic devices. We will choose NAND gates and group the minterms with them in mind. By grouping the terms of the equation as the sum of two products, an application of DeMorgan's theorem will produce the input terms required for a two input gate.

$$Y_1 = y_2(\bar{T} + y_1) + Ty_1 \quad (1.3)$$

$$Y_2 = y_2(\bar{T} + \bar{y}_1) + T\bar{y}_1 \quad (1.4)$$

1.1.3 Implementing the 'T' Flip-Flop

Now we are in a position to construct the circuit. Figure 1.1 shows Y_1 with its output and two input equations labelled.

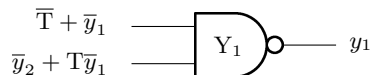


Figure 1.1: NAND Implementation of Y_1

The upper input is a part of the memory (feedback) loop for Y_1 and is satisfied by the configuration in Figure 1.1, in which we have also begun the construction of the Y_2 element. The memory loop for this variable is completely specified by the first term on the right side of Equation 1.4. We can assign Y_2

³There are situations in which this redundancy can be avoided without causing problems, so each design should be treated as unique and the implemented circuit should be analyzed carefully.

to a two-input NAND gate and generate the feedback term, $y_2(\bar{T} + \bar{y}_2)$, with another two-input NAND.

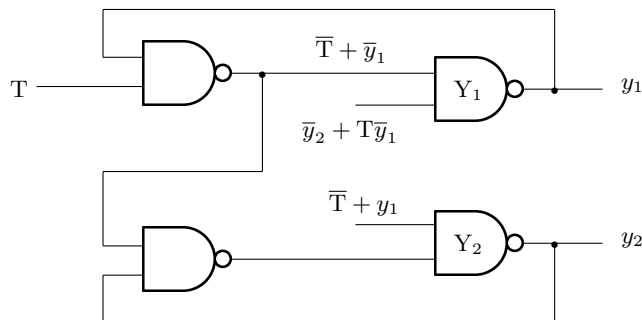


Figure 1.2: Partial Construction of “T” Flip-Flop

Before proceeding with the rest of the construction, some potential problems with our design approach should be mentioned. First, there is some risk in assuming that signal inverted twice can be treated the same as the original signal. The inversions generate delays which may (and often will) affect the circuit behavior. All assumed equivalences should be carefully verified. Second, some of the signals we create will be formed by the cancellation of literals, *e.g.* when we combine a signal with its inverse. This can produce glitches (races, hazards, etc.) caused by timing differences between switching of the signal and the corresponding change of state of its inverse. This subject will be taken up in detail later.

With these cautions in mind, we proceed as shown in Figure 1.2. Here we find that the upper input to Y_2 can be obtained by combining the existing signals $T(\bar{T} + \bar{y}_1)$ to form $T\bar{y}_1$ and inverting the result.

To confirm the circuit behavior we must consider the problems cited previously with the implementation method and examine any potentially troublesome details. Note that the gate marked with ‘*’ in Figure 1.2 is responsible for a static hazard. The hazard is generated whenever y_1 is high and the input T goes positive. But, from the flow table in Table 1.2, the circuit must then be in state 3 with y_2 also positive. Under these conditions, there will be a transition to state 4, whether the hazard occurs or not. A detailed timing analysis will show that the flip-flop does behave as expected, given reasonable assumptions about the elements used.⁴

It may be a small comfort that in this case the hazard does not cause a malfunction, so for this and more serious cases we will provide a method for eliminating the hazard entirely. Although the method is not discussed until later, a version of the ‘T’ flip-flop with hazard suppression is shown in Figure 1.3.

⁴The usual assumptions are that the longest gate delay is less than twice the shortest gate delay, and that rise and fall times are less than a gate delay.

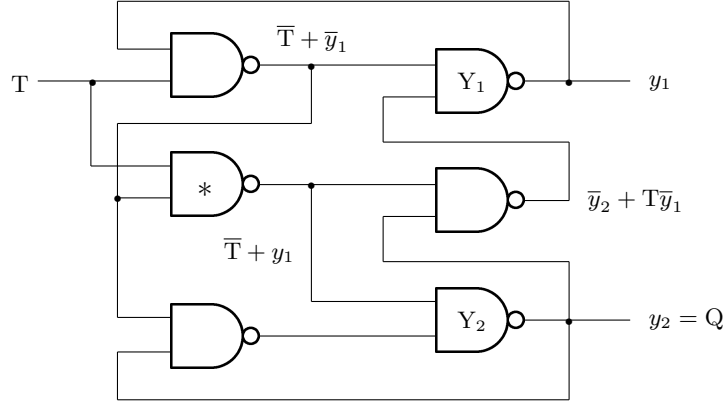


Figure 1.3: Construction of ‘T’ Flip-Flop

1.2 ‘D’ Flip-Flop

In this section we will derive equations for an edge-triggered ‘D’ type flip-flop. We will also develop the method mentioned earlier for eliminating hazards caused by cancellation of literals. To complete the section, we will show how to implement asynchronous set and clear lines in an otherwise synchronous device.

1.2.1 The Flow Table

We start with a flow table representing the desired circuit behavior. Now, there are some more or less standard conventions used in constructing the table which deserve comment. First, we will adopt the rule that only one input to a circuit may change at a time. This rule is not always necessary or desirable, but it can result in simplification by permitting the use of *don’t care* entries (designated by hyphens) in the table. Second, we will attempt to ‘merge’ rows in the resulting table to reduce the number of secondary state assignments required to specify each unique internal state. Both of these rules will be invoked in the treatment of the ‘D’ flip-flop. Note that there are situations in which these simplifying assumptions are not appropriate and we will forego them when necessary.

The following two tables show the initial (primitive) and merged versions of the flow table for a ‘D’ flip-flop, Tables 1.5 and 1.6, respectively.

Merging rows is facilitated using the method described by Mealey,[Mealey 55]. Briefly, we can merge two internal states (rows) if there are no conflicting states in the cells, if they have the same output value, and if all possible next states for each of them have the same output value. *Don’t cares* (hyphens) will be overridden by any numbered entry after merging. Merging can be repeated until it is no longer possible. Using this method, the original (primitive) flow table

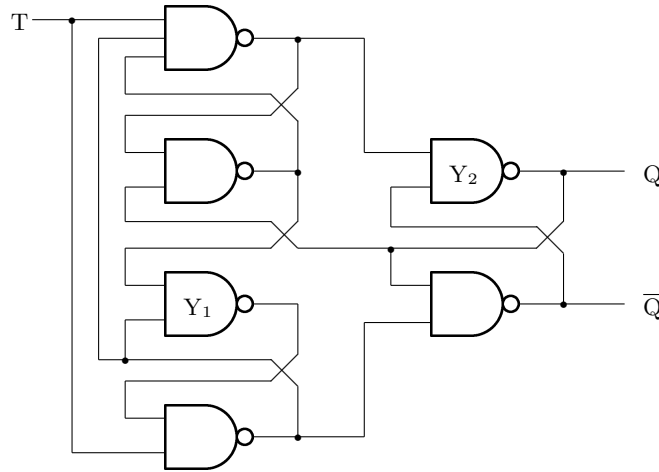


Figure 1.4: Implementation of ‘T’ Flip-Flop with Hazard Suppression

can be reduced from eight rows to four. Consequently, the number of internal secondary variables is reduced from three to two.

Once the table has been reduced, the row assignments can be made. For this table the gray-coded secondary assignments in Table 1.6 are suitable. Note that our assignment selection allows us to identify the secondary labelled y_1 with the required Q output. This identification will simplify the circuit implementation.

Using Table 1.6 we now can develop the state maps for the required secondaries. Table 1.7 shows the combined map with all secondary variables entered.

Table 1.8 shows the same information in two individual maps — one for each secondary.

DC				Q
00	01	11	10	
(1)	2	–	4	0
1	(2)	3	–	0
–	2	(3)	4	0
1	–	7	(4)	0
(5)	2	–	8	1
5	(6)	7	–	1
–	6	(7)	8	1
5	–	7	(8)	1

Table 1.5: Primitive Flow Table for ‘D’ Flip-Flop

DC					
y_1y_2	00	01	11	10	Q
00	(1)	(2)	(3)	4	0
01	1	-	7	(4)	0
11	5	(6)	(7)	(8)	1
10	(5)	2	-	8	1

Table 1.6: Merged Flow Table with Secondary Assignments

DC					
y_1y_2	00	01	11	10	Q
00	00	00	00	01	0
01	00	-	11	01	0
11	10	11	11	11	1
10	10	00	-	11	1

Table 1.7: Completed Composite Flow Table for ‘D’ Flip-Flop

1.2.2 The Circuit Equations

The equations corresponding to the maps in Table 1.8 are:

$$Y_1 = y_1y_2 + y_2C + y_1\bar{C} \quad (1.5)$$

$$Y_2 = y_2C + y_2D + D\bar{C} \quad (1.6)$$

The following regrouping eases the implementation problem:

$$Y_1 = y_1(y_2 + \bar{C}) + y_2C \quad (1.7)$$

DC					
y_1y_2	00	01	11	10	Q
00	0	0	0	0	0
01	0	-	1	0	0
11	1	1	1	1	1
10	1	0	-	1	1

Y_1

DC					
y_1y_2	00	01	11	10	Q
00	0	0	0	1	0
01	0	-	1	1	0
11	0	1	1	1	1
10	0	0	-	1	1

Y_2

Table 1.8: Completed Maps for ‘D’ Flip-Flop Secondaries

$$Y_2 = D(y_2 + \bar{C}) + y_2C \quad (1.8)$$

As in the earlier treatment of the ‘T’ flip-flop, we will construct our circuit starting from the explicit secondary gates. Since we will be using this example to develop methods for the elimination of hazards as well as for implementing asynchronous *set* and *clear* control modes, the step by step creation of the ‘D’ flip-flop is left to the reader. No new techniques beyond those shown already are needed. Some ingenuity, however, will always be required. Our result is shown in Figure 1.5 with the internal gates numbered for reference.

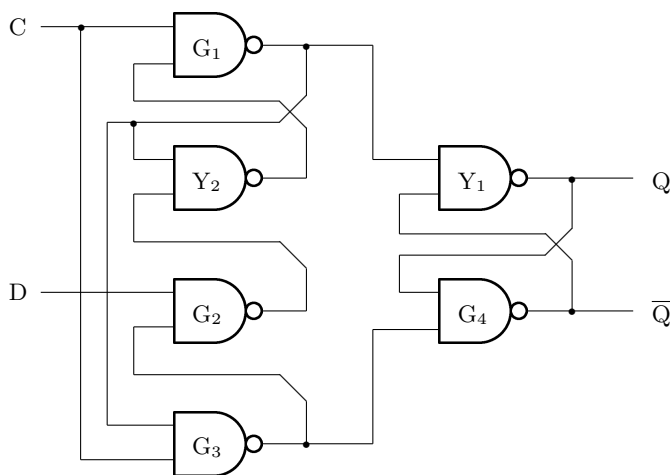


Figure 1.5: Implementation of ‘D’ Flip-Flop

A Digression on Hazards

The circuit implementation in Figure 1.5 harbors a serious hazard which needs to be eliminated. The hazard, as in the previous ‘T’ flip-flop implementation, is caused by attempting to cancel terms which do not occur simultaneously. The consequence is incomplete cancellation and spurious transient excursions of the affected signal line. See [Huff 55, Ung 59] for a full treatment of static and dynamic hazards.

A logic diagram and timeline representation of the problem is shown in Figure 1.6. In this figure, a signal and its (derived) inverse are combined at the input of an AND gate. Note that the result expected from the raw equations⁵ does not suggest the glitch which actually occurs. Part of the reason for this discrepancy is that the raw equations do not have provisions for keeping track of time delays, *i.e.* they are intrinsically static. Rather than developing a more elaborate algebra to include time explicitly we will simply make use of heuristic methods to eliminate the problem.

⁵In Boolean algebra, $a + \bar{a} = 1$ and $a \cdot \bar{a} = 0$.

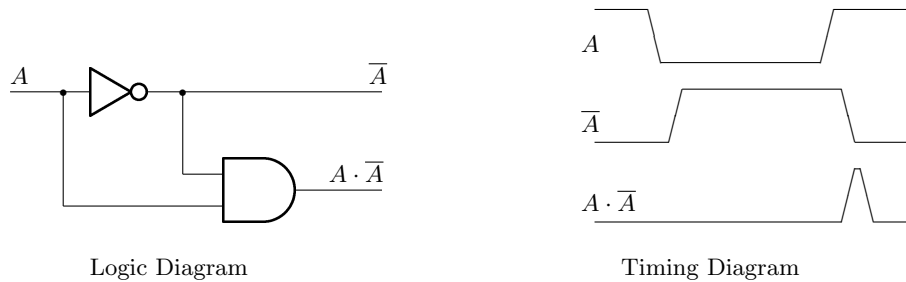


Figure 1.6: Combining a Signal with its Inverse

The method uses the fact that the undesired glitch noted above can be suppressed by combining a further delayed version of the original signal (or its equivalent). We can illustrate the concepts in detail by using subscripted variables to show time placement, as in Figure 1.7.

All this may seem like an exercise in futility, since in this example we have only succeeded in creating a signal which never changes. But the point of this discussion is not to find ways to cancel the *only* term in an expression, it is to find some way to completely cancel *one* of the unwanted terms in an SOP (Sum Of Products) expression, without disrupting the others.

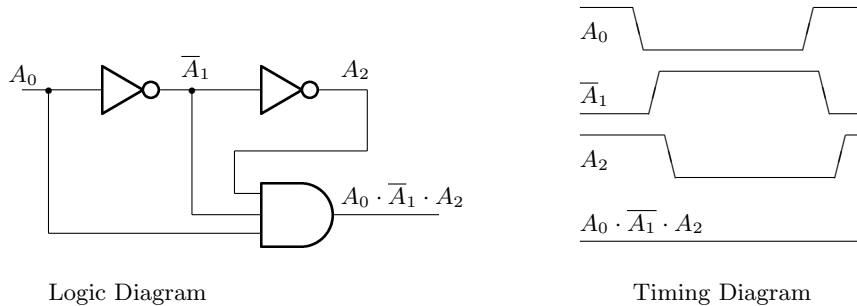


Figure 1.7: Suppressing a Hazard by Cancellation

In the next section, these concepts will be applied to a more complex circuit, which illustrates their use in real, non-trivial problems.

Example of Hazard Suppression

Suppose we are given three input signals, A , B , and C . We require a signal $\overline{C} + A \cdot \overline{B}$, which must be free of transients when A changes state. As stated, this example is basically a combinational problem with dynamic behavior constraints.

Consequently, we take the following approach. First, note that the behavior of the signal of interest is only important during the time that A switches. This

means that the inputs B and C can be considered static during the analysis. Second, if we keep track of propagating signals by using subscripts for time intervals, only the signal A needs to be subscripted.

Here is one candidate solution to the combinational part of the problem with the desired signal appearing at the output of Q_3 .

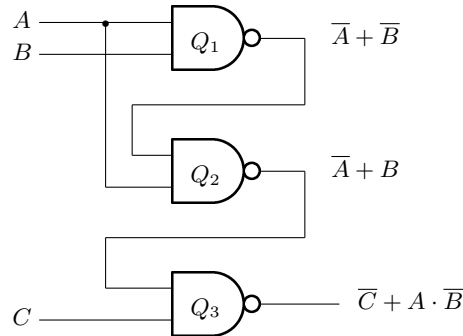


Figure 1.8: Combinational Circuit with Undesired Transient

Of course, the signal $\overline{C} + A \cdot \overline{B}$, could be generated by other choices of gates without creating the transient problem we are called upon to solve. But we can suppose that the other signals from Q_1 and Q_2 are already needed for other purposes and are therefore available for use without additional hardware.

The glitch appears at the output of Q_2 when B is high and A goes from low to high. It is caused by the attempt to produce $A \cdot \overline{B}$ at the input side of Q_2 by cancelling the \overline{A} term in $\overline{A} + \overline{B}$. It propagates through Q_3 when C is high and disturbs the target signal.

We now use subscripts to identify the time periods (delays) as a change of state of A propagates through the circuit. Since B and C are static during the time interval under consideration, there is neither any benefit nor motive for subscripting them.

Since the transient begins at the input of Q_2 , we turn our attention to eliminating it here. We hoped to create a servicable version of $A \cdot \overline{B}$ at this point. Instead, we created $A_0 \overline{A}_1 + A_0 \cdot \overline{B}$. From the previous discussion, we are tempted to look for another signal approximating $A \overline{B}$, but with later delays. The output of Q_3 contains such a signal and might serve to eliminate the spurious $A_0 \cdot \overline{A}_1$. If a third input is added to Q_2 the expression for the combined input then becomes,

$$A_0 \cdot (\overline{C} + A_2 \cdot \overline{A}_3 + A_2 \cdot \overline{B}) \cdot (\overline{A}_1 + \overline{B}).$$

This expands to,

$$A_0 \overline{A}_1 \overline{C} + A_0 \overline{A}_1 A_2 \overline{A}_3 + A_0 \overline{A}_1 A_2 \overline{B} + A_0 \overline{B} \overline{C} + A_0 A_2 \overline{A}_3 \overline{B} + A_0 A_2 \overline{B}$$

where we have omitted the *AND* dot multipliers for compactness.

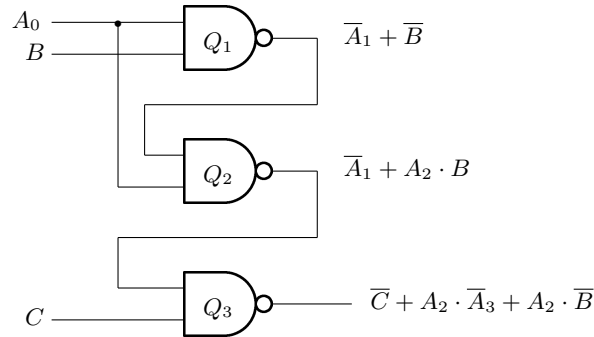


Figure 1.9: Combinational Circuit with Subscripted Signal

Examination of the above expressions reveals that, 1) the second and third terms, are completely eliminated by the previous results, 2) the last two terms can be combined into a single term.⁶ Rearranging the surviving terms yields,

$$\overline{C}(A_0\overline{A_1} + A_0\overline{B}) + A_0A_2\overline{B}.$$

We have now arrived at the following result: When the signal C is low, no transient can propagate through Q_3 because the gate is disabled. When C is high (true), \overline{C} is low, so the only term which survives to propagate through Q_2 is $A_0 \cdot A_2 \cdot \overline{B}$. This signal is a slightly trimmed version of our intended $A \cdot \overline{B}$ and solves the problem posed. The circuit is shown below.

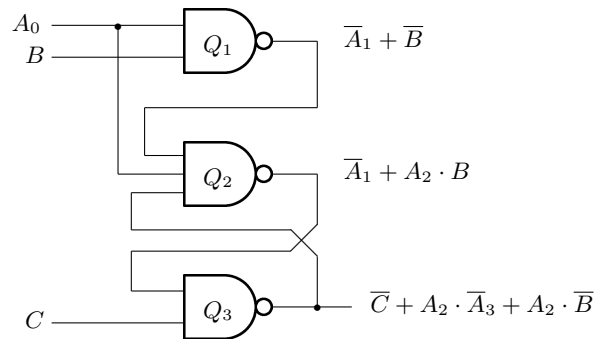


Figure 1.10: Circuit Solution Providing Hazard Suppression

Having presented the method, we must report that analysis by subscripting is error-prone and often prohibitively difficult. It is best suited for automated or computer driven solvers. Fortunately, there are simpler techniques for solving

⁶ $A_0A_2\overline{A_3}\overline{B} + A_0A_2\overline{B} = A_0A_2\overline{B}(1 + \overline{A_3}) = A_0A_2\overline{B}.$

similar problems with pencil and paper. These will be developed in following paragraphs.

Suppressing the ‘D’ Flip-Flop Hazard

Let’s examine our implementation of the ‘D’ flip-flop more closely. The hazard occurs at the output of gate G_3 when Y_2 is positive and the C input rises. The gate is implementing the term $C\bar{y}_2$ on the input side to produce $\bar{C} + y_2$ at its output. $C\bar{y}_2$ is constructed from the combination $C(\bar{y}_2 + \bar{C})$, and the cancellation of C is the cause of the problem.

We need to locate (or generate) another signal which contains the term $C\bar{y}_2$ where the C term is delayed. That signal may contain other terms summed with this term if certain cautions are observed. On searching through the existing signal set we find, at the output of G_2 the signal $\bar{D} + C\bar{y}_2$. This signal satisfies those requirements which have been articulated so far.⁷

The requirement is that the candidate signal must be false at the time the hazard would occur and must be true when a legitimate $C\bar{y}_2$ signal appears at the input of G_3 . Our choice meets this requirement. There are several ways to verify our modification. One way is to construct a detailed timing chart for the circuit, or submit it to a computerized simulation. Another is to examine the circuit Karnaugh maps.

How do we use the maps to analyzed hazards? We start with the following:

1. Each stable state represents a specific combination of Primary and Secondary variables,
2. No circuit action takes place until an input (Primary) variable changes,
3. The stable state entries in the merged flow table represent the defined circuit behavior; the other (unstable) entries define the transitory behavior from the time a Primary changes until the circuit settles into a stable state.

From the maps we can identify those stable states for which C is false and Y_2 is true. Since y_2 is true,⁸ the term $C\bar{y}_2$ should remain false during changes of the Primary, C . The skeletal flow tables in Table 1.9 are maps of the indicated expressions using the form of the merged flow table and show that there are only two stable states where C is false and y_2 is true — in the rightmost column of the table. It is here that we need to maintain the zero state of $C\bar{y}_2$ when the C input rises.

It is clear that our chosen hazard suppression signal will be zero at this time, and that it will prevent any other signals from activating the gate G_3 . What may not be clear is that the gate will still be properly functional at other critical times. To prove this, we only need to establish that when the term $C\bar{y}_2$ goes positive, the auxiliary signal $\bar{D} + C\bar{y}_2$ is true. Why is this sufficient? Because

⁷Note that $C\bar{y}_2 \cdot (\bar{D} + C\bar{y}_2) = C\bar{y}_2$.

⁸The potential ambiguity between Y_2 and y_2 doesn’t occur in the stable states.

y_1y_2	DC				Q
	00	01	11	10	
00	(0)	(1)	(1)		0
01		-		(0)	0
11		(0)	(0)	(0)	1
10	(0)		-		1

$C\bar{y}_2$

y_1y_2	DC				Q
	00	01	11	10	
00	(1)	(1)	(1)		0
01		-		(0)	0
11		(1)	(0)	(0)	1
10	(1)		-		1

$\bar{D} + C\bar{y}_2$

Table 1.9: Skeletal Flow Tables

we only need the effect of the delayed $C\bar{y}_2$ at the time of the hazard; we need to disable that line when the driving version of C sets $C\bar{y}_2$ to one. It happens that $C\bar{y}_2$ is driven true only from the two stable states in the leftmost column of the table. In these states, our chosen signal is true and no inhibition of the required signals will take place.

The conclusion? To complete the properly functions ‘D’ flip-flop we simply need to connect a wire from G_2 to G_3 . That’s it! The result is shown in Figure 1.11

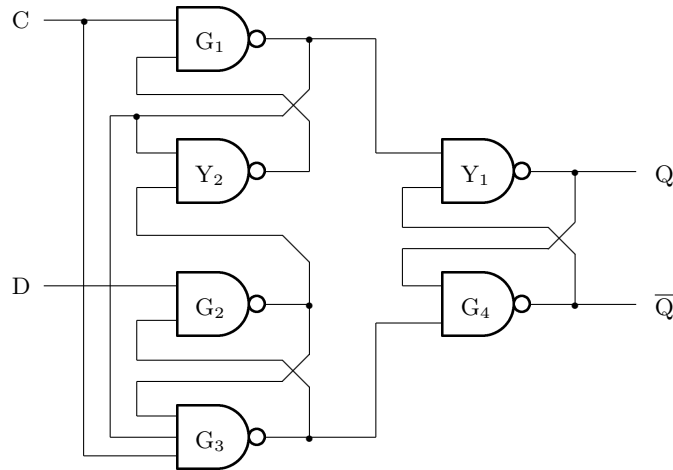


Figure 1.11: Hazard-free Implementation of ‘D’ Flip-Flop

1.2.3 Location and Identification of Signals

We will now make further use of skeletal flow tables to show how we were able to identify the \bar{Q} signal labelled in the schematic of Figure 1.11. The technique

is generally useful for locating or generating a signal with predetermined properties. To develop \bar{Q} populate the stable cells of our circuit flow table with the required signal values. This is done in Table 1.10.

y_1y_2	DC				Q
	00	01	11	10	
00	(1)	(1)	(1)		0
01		-		(1)	0
11		(0)	(0)	(0)	1
10	(0)		-		1

Table 1.10: Skeletal Map for \bar{Q}

Recall that only the stable states are required to define the external behavior of the circuit. Any signal with the stable state values in Table 1.10 matches our requirement. The gate labelled G_4 , for example, implements the Boolean function $\bar{y}_1 + C\bar{y}_2$, but its truth values in the stable states are the same as \bar{Q} . We are thus justified in labelling it as we did. The only differences between the desired and actual signals occur during transitional states, and since those transitions are gray-coded the net effect is simply that some edges of our derived \bar{Q} occur earlier or later than those of an 'ideal' \bar{Q} . Incidentally, it is a characteristic of memory devices (latches, R-S flip-flops, etc.) that the circuit equations for \bar{Q} do not represent strict Boolean inverses.

1.2.4 Asynchronous Set and Clear

The completed 'D' flip-flop can be seen to consist of an output latch and some control logic. The state of the output latch does not affect the control logic in any way, and may be in any initial state at power up. It is desirable to have some way to force the output into a pre-determined state independent of the current state of the output and input variables; *i.e.*, to introduce *set* (or *preset*) and *clear* (or *reset*) signal lines into the circuit.

This is best accomplished by referring to the equations for the circuit secondary variables Y_1 and Y_2 . Since our requirement is to force a known output state regardless of the input combination, we must alter the states of some of the secondaries. The equations for these are repeated below.

$$Y_1 = y_1(y_2 + \bar{C}) + y_2C \quad (1.9)$$

$$Y_2 = D(y_2 + \bar{C}) + y_2C \quad (1.10)$$

We also need the merged flow table to identify the stable state entries for our altered functions. We begin by examining the flow table.

According to the table, if we can force both y_1 and y_2 to the true state momentarily (third row of table), Q will go true and remain that way until

y_1y_2	DC				Q
	00	01	11	10	
00	(1)	(2)	(3)	4	0
01	1	-	7	(4)	0
11	5	(6)	(7)	(8)	1
10	(5)	2	-	8	1

Table 1.11: Flow Table for ‘D’ Flip-Flop

some other signal occurs. The circuit will be in state 5, 6, 7, or 8, This is the action needed for an asynchronous *set*. Similarly, an asynchronous *clear* can be accomplished by forcing both y_1 and y_2 low (false). The resulting *set* or *clear* is independent of the current states of Q or any of the inputs.

Another, possibly better, way to derive the *set* and *clear* conditions is by inspecting the right sides of (1.9) and (1.10). From (1.9) we can see that to set Y_1 (on the left) true, we must force both y_1 and y_2 (on the right) true. This does not guarantee that Y_2 will remain true, but it does guarantee that Q will be set. Why? Because Y_2 can only revert to false if C is false. But in this case, Y_1 (Q) will remain set because of the presence of the term $y_2\bar{C}$. By the same token, forcing y_1 and y_2 low will clear Q . Here, the state of Y_2 is also indeterminate, but it can only revert to true if C is false, in which case Y_1 will stay clear.

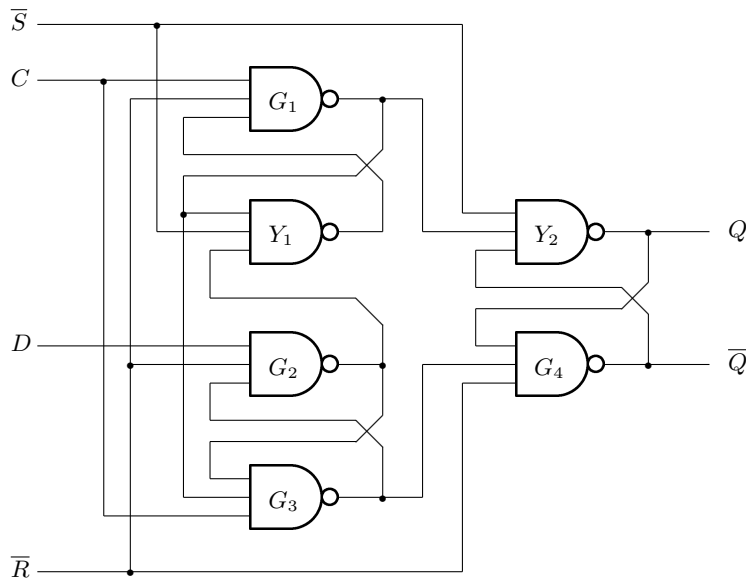


Figure 1.12: Complete Implementation of ‘D’ Flip-Flop with Set and Clear

The complete ‘D’ flip-flop solution is shown in Figure 1.12. To implement the enhancements we simply added additional inputs to the gates for Y_1 and Y_2 to handle the asynchronous *set* function. These are tied together and brought out as \bar{S} . To create an asynchronous *clear* we need to provide similar inputs to G_1 and G_4 to force Y_1 low, and an input to G_2 to force Y_2 low. These are tied together and brought out as \bar{R} .

1.3 Negative Pulse Generator

1.3.1 The Flow Table

This exercise will illustrate some of the power of the flow table as a circuit description tool. We will be using the table in unusual ways and developing some special design techniques which will be used later.

The problem posed in this section is to devise a circuit with a single input, P , and a single output, Q , which produces a short negative pulse whenever the input goes positive. This sort of circuit is encountered frequently, and many logic designers have met it and solved it one way or another — usually by using a single-shot or some sort of logic delay line. Our solution is clearly not the only one possible, but it does have a certain elegance, and offers a vehicle for introducing new design methods.

Our circuit description only provides for a single stable output value — with Q positive. The negative pulse is generated on the rising edge of the input, and will automatically complete before stability returns. So how do we describe this behavior with a flow table? Table 1.12 shows a technique for doing this.

P		Q
0	1	
(1)	2	1
–	3	0
–	4	0
1	(4)	1

Table 1.12: Flow Table for Negative Pulse Generator

Note that we have identified four states, but only two of them are stable. In the table, we have explicitly provided transitional rows to control the circuit behavior during unstable operation. We will decode the secondary combinations which correspond to the unstable rows and use the decoded signal to satisfy the problem statement. In a sense, we have *expanded* the flow table by the addition of these rows. Contrast this with the previous problem where we found it convenient to merge rows. Also note that we have placed *don't cares* in those places where the circuit behavior is not specified. Generally, this practice can

simplify the design, but always confirm the operation of the completed circuit to assure that no unwanted behavior occurs.

The selection of secondary assignments is closely linked with the problem statement. We have two secondary variables, one of which we would like to equate with Q . Our choice is shown in Table 1.13.

$y_1 y_2$		P		Q
		0	1	
10	00	10	00	1
00	01	-	01	0
01	11	-	11	0
11	10	10	11	1

Table 1.13: Secondary Assignments for Negative Pulse Generator

1.3.2 Equations and Implementation

The equations for the circuit are:

$$Y_1 = \bar{P} + y_2 \quad (1.11)$$

$$Y_2 = \bar{y}_1 + Py_2 \quad (1.12)$$

These equations are implemented in the rather surprising configuration in Figure 1.13.

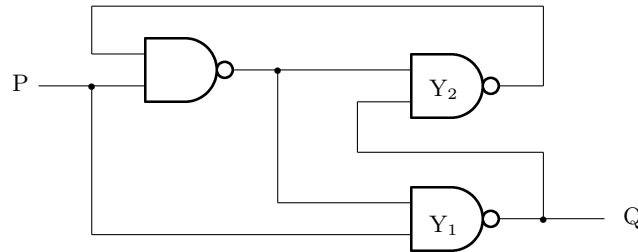


Figure 1.13: Negative Pulse Generator

1.3.3 Analysis of Implementation

If this paper carries any consistent message, it is that every circuit implementation should be reviewed carefully. The assumptions and simplifications made during the design may cause unwanted behavior.

In the current design, we placed *don't care* entries in some of the cells in column one. As expected, this decision simplified the logic equations by accepting a default transition to state (1). The practical consequence is that if the input goes positive and then returns to negative *before* the output pulse completes, the output pulse will be forced to complete early.

Engineering judgment comes into play here. It may be acceptable to place a *hold time* restriction on the input to the circuit such that the input must remain positive until the output completes automatically. It is also possible to latch the input so that no such requirement is needed, with the penalty that the circuit implementation will be more complex. Review the individual circuit requirements carefully.

1.4 Up-Down Counter Controller

1.4.1 The Flow Table

The next circuit solves another common problem — that of controlling the action of an up-down counter which has separate count-up and count-down inputs. The controller is needed to deal with the case in which the count-up and count-down signals arrive simultaneously. When this occurs, such counters often drop a count (or add a count) because one the inputs gets lost. In applications where up-down counters are used to control a FIFO or cyclic buffer, this fault is fatal. Because the controller assures that count-up and count-down signals occurring at arbitrary times are properly ordered, it is sometimes called a *derandomizer*.

Our objective is to devise a control circuit which is able to properly arbitrate when the commands occur at or near the same instant.

Development of the flow table follows from these requirements:

1. The up-down counter has separate count-up and count-down pins,
2. The count action occurs on the leading edge of a negative going pulse at the appropriate pin,
3. Pulses on the count-up and count-down pins must be separated by at least one gate delay to guarantee proper operation of the counter.

For this application, we will assume external count-up and count-down commands consisting of positive-going signals of arbitrary duration; *i.e.*, the signals may coincide or overlap in time.

This will be our first exposure to incompletely specified functions, and we will develop our solution in carefully contained stages. The first step will be to create a partial flow table documenting what we know from the problem statement. This table is shown in Figure 1.14.

In this table we have only provided rows for the stable states. Those state transitions which do not involve output signals are shown, but the other transitions are left blank for the moment. What can we learn from this partial table?

$C_u C_d$				$Q_u Q_d$
00	01	11	10	
(1)		-		11
1	(2)		-	11
-	2	(3)	4	11
1	-		(4)	11

Table 1.14: Partial Flow Table for Counter Controller

1. We will have expand the table to provide unstable rows, such as we did for the pulse generator,
2. If the output variables are equated with specific secondaries, two additional secondaries will be required to guarantee unique row assignments.

A fully expanded, partially filled, table is reproduced in Table 1.15. In this table we have replaced the stable state numbers with their secondary assignment equivalents and have added the transition requirements for the unstable states. We have also retained the stable state indicators (parentheses, bold type) for easy comparison with Table 1.14. The secondaries y_1 and y_2 have been identified with Q_u and Q_d , respectively.

$y_1 \cdots y_4$	$C_u C_d$				$Q_u Q_d$
	00	01	11	10	
0000					
0001					
0011					
0010					
0110			[1110]	1110	01
0111			1111		01
0101			0111		01
0100			[0110]	0110	01
1100	(1100)	1000	--00	0100	11
1101	1100	(1101)	0101		11
1111		1101	(1111)	1110	11
1110	1100		1010	(1110)	11
1010			1011		10
1011			1111		10
1001		1101	[1101]		10
1000		1001	[1001]		10

Table 1.15: Partially Filled Flow Table

Note the use of blanks instead of hyphens in some table entries. In our method there is an important distinction between the two. The blank means that the circuit behavior has not been specified. It doesn't necessarily mean the action can't happen or won't matter, it only means we have not chosen to restrict it at this time. If we truly do not care, we will place hyphens in the cell. Otherwise, we may specify some or all of the variables in the cell at a later time.

A case in point concerns the blank cells in the rows where y_1 and y_2 equal zero (first four rows). From the circuit description, this is a forbidden combination; *i.e.*, at least one of these secondaries should be true at all times. Hence, the cells in these rows can be used to simplify the equations for Y_1 and Y_2 .

A more problematic situation occurs in the row labelled 1100 in column 11. Here we have the opportunity to specify what happens when both count commands occur at the same time. But we expect the circuit to 'decide' which count pulse to output first, so we choose not to select the next state identifiers for y_1 and y_2 .⁹ On the other hand, we also expect the circuit to immediately output one or the other pulse so we need to assure that the next state is outside the four stable rows. We can do this by specifying the secondaries y_3 and y_4 as shown.

1.4.2 Equations and Implementation

The circuit equations are:

$$Y_1 = \bar{C}_u + y_3 + \bar{y}_2 \quad (1.13)$$

$$Y_2 = \bar{C}_d + y_4 + \bar{y}_1 \quad (1.14)$$

$$Y_3 = \bar{y}_1 + y_3 C_u \quad (1.15)$$

$$Y_4 = \bar{y}_2 + y_4 C_d \quad (1.16)$$

The first two equations borrow heavily from the unspecified portions of the table for simplifications and redundancy. In this implementation Y_3 and Y_4 were constructed first, and the others followed. See Figure 1.14.

This circuit has several aspects which deserve comment. First, the latch consisting of Y_1 and Y_2 operates as a 'turnstile' which passes a signal through one side or the other, but which prohibits simultaneous passage of both. It arbitrates between coincident requests by latching in one of two states. Whichever signal is honored first will produce an appropriate output pulse, after which the other will be allowed to pass. Second, with 'real-world' devices the possibility of metastable behavior exists. That is, there may be a short period time during which the latching process produces uncertain outputs on Y_1 and Y_2 . This can occur if the input signals arrive within a very small time interval; *e.g.*, less than a gate delay. But even if this does occur, the duration of the uncertainty will be very short and its effects can be controlled with glitch suppression logic

⁹These secondaries are identified with Q_u and Q_d , respectively.

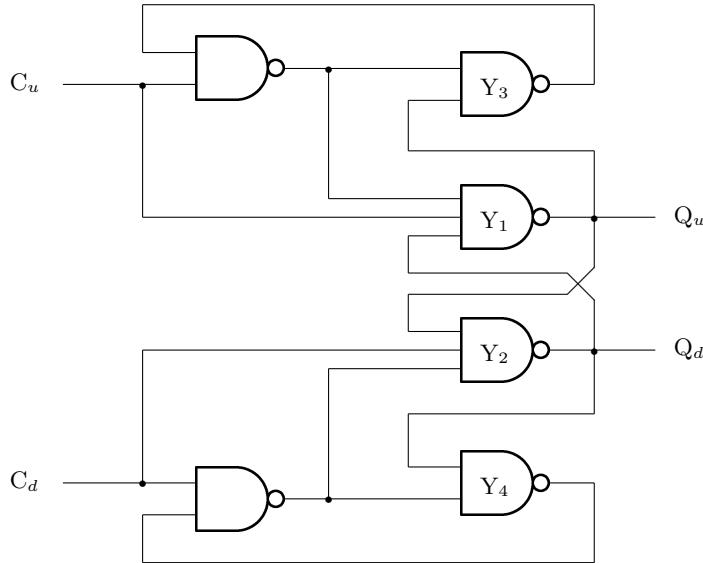


Figure 1.14: Up-Down Counter Controller

(described later), or an ‘exclusive-latch’ which consists of an input latch driving an output latch.

It is worth recalling that metastability can be managed and, in general, the potential for its appearance does not necessarily suggest circuit failure.¹⁰ It is a constant companion in digital environments and even lurks inside the common ‘D’ flip-flop. Since it can only occur during changes of state, such as when inputs change, it can be managed by imposing proper *setup* and *hold* time requirements. In situations where the inputs are unrestricted, such as here, additional logic may be considered. But the author has implemented this configuration in TTL logic and driven it with a reference clock consisting of a 1 MHz sine-wave modulated reference clock over a period of several days without any miscounts.

1.5 A Phase/Frequency Detector

This section calls upon techniques already developed and introduces the concept of ‘strong’ and ‘weak’ secondaries. It also provides a last example before we complete the development method and summarize it in its entirety.

We are to implement a phase/frequency detector which generates negative pulses to drive the charge pumps associated with a phase-locked loop (PLL).

¹⁰Metastability has been the subject of many analytic studies using statistical models and various plausible device timing distributions. But the likelihood of an extended metastable conditions is vanishingly small.

The circuit action is controlled by two inputs — one is the variable clock signal derived from the PLL oscillator, ϕ_{pll} , and the other is the incoming reference signal, ϕ_{ref} , with which phase lock is desired. In the following exposition, these signals will be designated as C and R , respectively. The input logic is driven by the negative edges of these two input signals and will activate (or deactivate) one of two pump signals, *pump up* or *pump down*, according as the reference signal is ahead of or behind the PLL clock. The active level of the pump signals is negative, and their normal state is high.

Phase lock is achieved when both signals are coincident at their negative transitions. In this condition, both pump signals remain high, although we will accept small glitches on both outputs on the assumption that they will have no net effect on the total charge driving the phase locked oscillator (PLO). By activating the *pump up* output when the reference signal is early, and clearing all pumps when the PLL clock arrives, we can develop a driving signal that is proportional to the time difference between the two. Similarly, if the clock signal is early, we activate the *pump down* signal and clear all pumps when the reference arrives.

The circuit is insensitive to duty cycle because of the edge triggering. It is also immune to spurious lock-up on harmonics because it forces a one-to-one correspondence between clock edges and reference signal edges. Hence, it is a phase/frequency detector as opposed to a phase detector only.

1.5.1 The Flow Table

A suitable flow table description of the desired behavior is shown in Table; 1.16, where R is the reference signal and C is the PLL clock. This table can be constructed easily from the observation that the circuit can only have three different output states, and can have any combination of inputs for each possible output. For this problem, it will be expedient to merge the table and then expand it. The reason for this unusual approach is that the output requirements cannot be met with any direct secondary variables. Decoding of secondaries would be required to produce the outputs unless the table is expanded to include a greater number of secondaries, as was done in the previous example. But an expansion at this point would create an unnecessarily, if not prohibitively, large flow table. Our solution is to first reduce the table by merging, and then expand the merged table to equate some secondaries to the output variables.

Merging presents no unusual problems and the merged table is shown in Table 1.17.

Before expanding the table, we will reorder the rows. Why? Because our table construction process has left us with a scrambled list of output states and we would like to gray-code them before proceeding. Recall that the reordering of flow table rows is essentially dealing with the row assignment problem, and by taking some action to order the outputs at this time, we may be able to simplify the row assignments later. The sorted table is shown in Table 1.18.

A glance at the output column shows why an expansion of the table is called for. If we reserve a pair of secondary variables to provide the P_u and P_d signals,

RC				$P_U P_D$
00	01	11	10	
(1)	4	-	10	11
(2)	5	-	11	01
(3)	6	-	12	10
3	(4)	7	-	11
1	(5)	8	-	01
3	(6)	9	-	10
-	5	(7)	12	11
-	5	(8)	10	01
-	4	(9)	12	10
2	-	7	(10)	11
2	-	8	(11)	01
1	-	9	(12)	10

Table 1.16: Raw Flow Table

RC				$P_U P_D$
00	01	11	10	
(1)	4	-	10	11
(2)	5	8	(11)	01
(3)	(6)	9	12	10
3	(4)	7	-	11
1	(5)	(8)	10	01
-	5	(7)	12	11
1	4	(9)	(12)	10
2	-	7	(10)	11

Table 1.17: Merged Flow Table

RC				$P_U P_D$
00	01	11	10	
(2)	5	8	(11)	01
1	(5)	(8)	10	01
(1)	4	–	10	11
3	(4)	7	–	11
–	5	(7)	12	11
2	–	7	(10)	11
1	4	(9)	(12)	10
(3)	(6)	9	12	10

Table 1.18: Sorted Flow Table

we need two more to distinguish between the four rows whose outputs are 11. Because of the symmetry in the circuit description and the table as developed so far, we will try and preserve symmetry in the additional secondaries. Our expanded table is shown in Table 1.19 with the state entries already replaced by their corresponding row designations.

A few words are in order concerning constructing the expanded flow table. The strategy used to create a satisfactory table is intended to deal with some of the ‘real world’ problems that complicate some designs. For one thing, we are at all times attempting to produce a flow table which permits easy movement between states without critical races. This means we try to assure that movement from row to row will involve the change of one variable only. We normally gray-code the secondaries to facilitate this task and will sometimes need to expand the table to increase the number of state transition options available.

If often happens that there is no way to move from one unstable state to a stable one without multiple secondary changes. In those cases, we will often take advantage of any available *don't care* cells as ‘stepping stones’; *i.e.*, we will make a gray coded move to an auxiliary cell and then another gray coded move to the final state. We may even have to make several such steps to get to the target state. If there are any blank cells in the table, we can use those, too.

Another way to approach this problem is to avoid ‘overspecifying’ the secondaries. If we only really care about a few of the secondaries, we can specify them and leave the others in *don't care* conditions wherever possible. We have done this in the previous example, and do it again here in Table 1.19 in two cells in row 0111, columns 00 and 10, as well as in row 1011, columns 00 and 01. Each of these cells would require multiple steps to get to the target state. We have chosen to only specify those secondaries whose values concern us, and not the others.

$y_1 \cdots y_4$	RC				$P_U P_D$
	00	01	11	10	
0000					
0001					
0011					
0010					
0110	(0110)	0111	0111	(0110)	01
0111	11--	(0111)	(0111)	11--	01
0101					
0100					
1100	(1100)	1101	----	1110	11
1101	1001	(1101)	1111	----	11
1111	----	0111	(1111)	1011	11
1110	0110	----	1111	(1110)	11
1010					
1011	11--	11--	(1011)	(1011)	10
1001	(1001)	(1001)	1011	1011	10
1000					

Table 1.19: Expanded Flow Table

1.5.2 The Equations

Before deriving the equations for this circuit we will introduce the concepts of *external* and *internal* secondaries. Heretofore we have regarded all secondaries as ‘internal’ in the sense that they were implicit in the problem statement. We now find it convenient to qualify certain secondaries depending on whether they satisfy any of the given signal requirements. External secondaries are those which can be immediately identified with a required output signal, and whose values may be partly or completely fixed by the problem statement. Internal secondaries are those which are merely required to complete the internal row identification requirements, and whose implementation details are unconstrained beyond that.

The reason for making such a distinction follows from the many degrees of freedom available in implementing a set of Boolean equations. We are, of course, always interested in economy. We can solve the implementation problem in more than one way, but an optimal or quasi-optimal solution is not easily pulled from the raw equations without imposing external constraints. In fact, more than one set of equations can represent the same circuit.¹¹

With this in mind, we begin the equation development by taking a first cut at the equations for the external secondaries, Y_1 and Y_2 .

¹¹Consider that fact that some boolean terms are technically redundant, and do not alter the truth values of the equations. We add or remove redundant terms when it serves our purpose to do so — particularly to eliminate hazards.

$$Y_1 = y_1R + \bar{y}_3 + y_3y_4(\bar{y}_2 \cdot \bar{R} + \bar{y}_1 \cdot \bar{C}) \quad (1.17)$$

$$Y_2 = y_2C + \bar{y}_4 + y_3y_4(\bar{y}_2 \cdot \bar{R} + \bar{y}_1 \cdot \bar{C}) \quad (1.18)$$

With all the blank map space we have used, there is a real danger of incompletely specifying the circuit equations. Indeed, an example of such an error is revealed in equations (1.19) and (1.20) where we have partially completed the equations for the internal secondaries, Y_3 and Y_4 . Note that if either of these terms ever becomes *true*, the secondary will ‘latch up’ because there is no way to set it *false*.¹²

$$Y_3 = y_1R + y_3 \quad (1.19)$$

$$Y_4 = y_2C + y_4 \quad (1.20)$$

Clearly there should be some zeros in the y_3 positions of cells in rows of the map where the y_3 row identifier is one. Similarly for y_4 . Why? Because those zeros would prevent us from collecting all the rows where y_3 is true into our equation. The problem has occurred because of careless and over-optimistic treatment of blanks and *don't cares*. Some of the y_3 terms need to be qualified by some other variable which, when false, will unlatch Y_3 . We intend to correct this oversight in the course of completing the design, but for the moment we will simply note the problem and put it aside to concentrate on the external secondaries.

When confronted with potentially conflicting design objectives, such as the negotiable desire for economy with the absolute need for correctness, we may find it necessary to re-examine the design space in which we are working. Here, for example, we have ‘opened up’ the flow table to make the external secondaries reflect the required circuit outputs. Now we need to place restrictions on the secondaries to assure that they are not underspecified. We hope to do this in a way that reduces the number of auxiliary signals we need to generate. This will help assure that the final implementation is, in some sense, minimized.

To begin, note that the last term in the equations for Y_1 and Y_2 are the same. Since this term, or an equivalent, will be implemented in our design, it is worth examining it closely to determine whether it can be constructed from terms which are usable in Y_3 and Y_4 . This heuristic part of the design process simply tries to minimize the total number of logic terms required by making best use of common terms, if possible.

Relabelling the term under consideration as F_1 and applying DeMorgan’s theorem to obtain its inverse, \bar{F}_1 , we have:

$$F_1 = y_3y_4(\bar{y}_2 \cdot \bar{R} + \bar{y}_1 \cdot \bar{C}) \quad (1.21)$$

$$\bar{F}_1 = \bar{y}_2 + \bar{y}_4 + y_1y_2 + y_1R + y_2C + RC \quad (1.22)$$

¹²Recall Einstein’s admonition to *make everything as simple as possible, but no simpler*. Here, in our zeal to maximize blank space, we have ‘overshot’ the mark.

It can be shown, by the use of skeletal flow tables, for example, that the Boolean function in (1.22) can be ‘AND’ed with the y_3 in (1.19) to eliminate the problem of underspecification.¹³ In other words, replacing the simple y_3 with a qualified version give by the term: $\bar{F}_1 \cdot y_3$, manages to capture all the required *true* states in the flow table without the risk of latch-up. We can use the same function to correct (1.20). Thus, to complete the specification for Y_3 and Y_4 we need only ‘AND’ the term \bar{F}_1 with y_3 and y_4 , respectively.

Now, we still may be able to simplify the logic if we can further reduce (1.22). However, reductions in the expression for \bar{F}_1 must be done with careful regard for its impact on Y_1 and Y_2 , as there are no obvious constraints available from Y_3 and Y_4 .

A partial implementation of the circuit is shown in Figure 1.15.

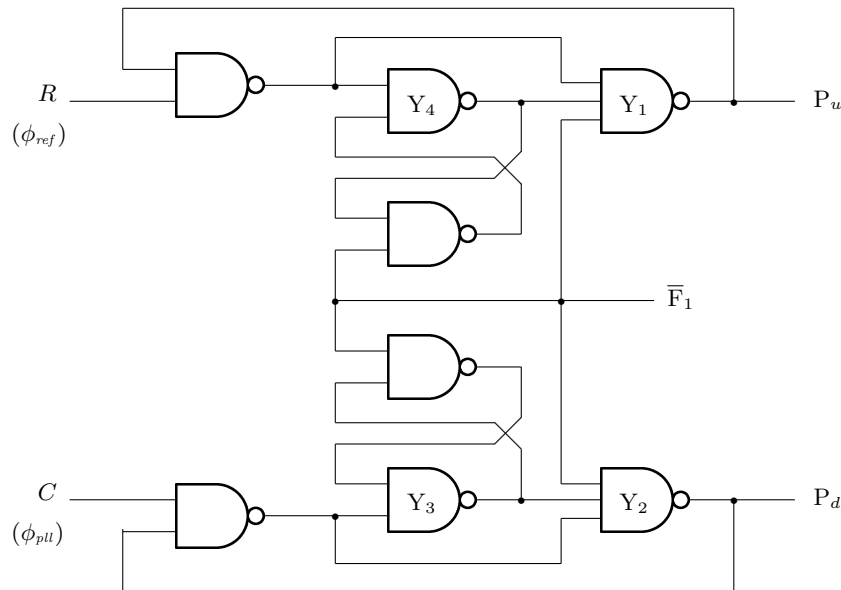


Figure 1.15: Partial Implementation of Phase/Frequency Detector

Note that we have already generated terms for Ry_1 and Cy_2 and we would like to know if we really need the terms y_1y_2 and RC . After all, these entered into our equations for Y_1 and Y_2 following a very loose grouping of their maps. We will always strive to eliminate unnecessary terms. To determine whether we can simplify \bar{F}_1 we can withhold the questionable terms, apply DeMorgan’s theorem, and inspect the maps to see if we have introduced any damaging zeros into the *true* equations.

The new values for \bar{F}_1 and its inverse, now called \bar{F}_2 and F_2 , are shown in

¹³This isn’t too surprising since y_3 covered *true*, blank or unspecified cells only.

(1.23) and (1.24).

$$\bar{F}_2 = \bar{y}_3 + \bar{y}_4 + y_1R + y_2C \quad (1.23)$$

$$F_2 = y_3y_4(\bar{y}_1\bar{y}_2 + \bar{y}_1\bar{C} + \bar{y}_2\bar{R} + \bar{R}\bar{C}) \quad (1.24)$$

We can now examine the changes to F_1 embodied in F_2 to verify them. A careful comparison of the skeletal maps with the Expanded Flow Table of Table 1.19 reveals that they do not invalidate our earlier implementation of Y_1 and Y_2 , so we can accept them and proceed.

The revised circuit equations are listed in (1.25) through (1.28) for reference and the completed circuit diagram is shown in Figure 1.16.

$$Y_1 = y_1R + \bar{y}_3 + y_3y_4(\bar{y}_1 + \bar{R})(\bar{y}_2 + \bar{C}) \quad (1.25)$$

$$Y_2 = y_2C + \bar{y}_4 + y_3y_4(\bar{y}_1 + \bar{R})(\bar{y}_2 + \bar{C}) \quad (1.26)$$

$$Y_3 = y_1R + y_3(\bar{y}_4 + y_2C) \quad (1.27)$$

$$Y_4 = y_2C + y_4(\bar{y}_3 + y_1R) \quad (1.28)$$

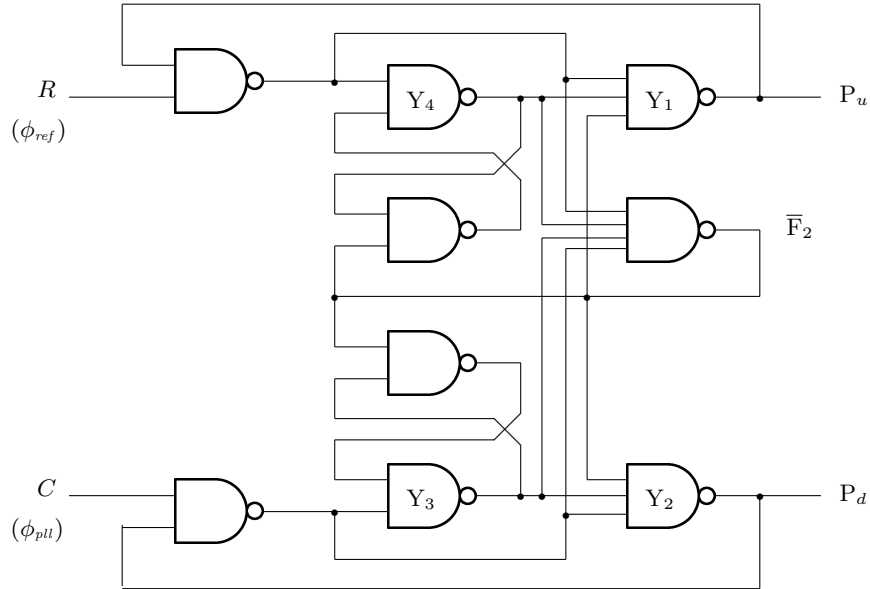


Figure 1.16: Complete Implementation of Phase/Frequency Detector

Dealing with incompletely specified functions can be very confusing, and each case must be examined closely. In this case we have expanded the raw flow table to allow us to identify certain secondary variables (external secondaries)

with the circuit outputs. In doing so we introduced new rows in the table which serve as ‘wild cards’ in gathering truth values from the resulting Karnaugh maps. Handling the entries in these rows is part of the specification and design problem. We may insert values in specific cells when it is necessary or desirable to do so, or we may determine that we can leave blank and cells which represent *don't care* conditions. Hence, we should be prepared to place 1s or 0s in empty cells representing transient states if there a reason for it.

We have accepted one slight departure from the strict original circuit specification in this exercise. When both inputs go negative simultaneously, *e.g.* when the PLL is in lock, there should be no corrective pump signals from P_u or P_d . But in our circuit there will be coincident short glitches on both pump outputs. These glitches can be eliminated with glitch suppression logic (discussed later) or by redesigning the circuit to control the transient behavior under these conditions. We have not bothered with either of these steps because, by the nature of the problem, such glitches will not adversely affect the performance of the PLL. This is because the PLL responds to the total time the pump up or down outputs are active, and short glitches or spikes will have negligible impact.

Chapter 2

The Complete Synthesis Method

We are now in a position to pull together the techniques developed in the previous sections and generalize the entire process.

A summary is shown in the outline below.

1. **Develop Circuit Description** This should completely define the behavior of the identified output signals in terms of the inputs.
2. **Derive the Raw Flow Table** There are many ways to do this depending on the ‘fine print’ in the circuit description. Different raw flow tables can describe the same circuit. If you have problems here, there may be problems with the specification itself.
3. **Expand or Merge Flow Table** This step may not be necessary or desirable in many cases. But if there is some problem in equations outputs to unique rows in the raw flow table, we can assume additional secondaries and expand the table accordingly.
4. **Sort Flow Table** If there is a desire to equate output signals with specific secondary variables, the rows should be sorted so that the outputs are in gray coded order.
5. **Make Secondary Assignments** Having created a suitable flow table, we can now assign a unique combination of secondary variables to each row. The most direct way of doing this is to simply number the rows in binary counting order. But because we are concerned with controlling movement from row to row, we will use gray-coding exclusively. Note that it is not necessary to start the assignments with the first row equal to zero. The starting count is a matter of convenience and consistency with the output.

6. **Make the Cell Assignments** This is a multi-step process. Use the following order:
 - (a) Fill those entries which represent stable states with their corresponding row assignment numbers. This is a mandatory step.
 - (b) Fill the cells whose Hamming distance from their stable rows equals one. In other words, if a change of input (column movement) places the circuit on a transient (unstable) cell, and if there is only one bit which is different in the present and next states, then fill the cell with the next stable state value.
 - (c) Use stepping stone rows if the Hamming distance between the present unstable state and target state exceeds one. Be advised that sometimes this will not be possible, and sometimes it won't be worth the effort. If there is only one stable state in the column, we may not care how the secondaries get there. It is often possible to specify a few of the secondaries to avoid undesired movement and leave the rest unspecified. If more than one stable state exists in the column, great care should be taken to assure that the circuit always ends up in the right one.
 - (d) Fill in any remaining necessary or desirable constraints. This is where any blank cells are given 1s or 0s to force secondary movement into or out of certain rows or blocks of rows in the table. We also use this step to identify or eliminate potential problems associated with underspecified secondaries.
7. **Split the Completed Flow Table** Note that this step is purely mechanical and does not involve any decision making. We have not shown the split tables for all our examples, because they can be easily deduced from the complete table.
8. **Derive First Cut Secondary Equations** In this step we consider the simplest forms for each of the secondaries. Decisions about whether to incorporate redundant terms in an equation are made at this point. Problems with underspecification in expanded tables will also appear here, hence we refer to this step as a 'first cut'.
9. **Partially Implement the External Secondaries** If the circuit equations permit a complete solution at this point, we can implement all the secondaries and complete the design process. Otherwise, we should move ahead interactively constructing portions of the circuit as conditions permit. While we have not yet discussed the exploitation of symmetry in the defining equations, there are opportunities here to do so. (See the J-K flip-flops later in this paper.) Complex terms which include blank areas on the maps are targets for reduction and should be identified at this point.
10. **Partially Implement the Internal Secondaries** If these secondaries allow a simple, unambiguous solution we can inspect the terms generated

in the solution process to see if any are useful in completing the external secondaries. Otherwise consider the implementation as tentative.

11. **Apply DeMorgan's Theorem to External Secondaries** Actually, we only need to look at those terms which are candidates for reduction. Some of the terms in the secondary equations will be straightforward and easily constructed. The other terms need to be inverted so that the input requirements to the selected gate can be examined.¹ We are looking for existing signals or easily obtained signals to satisfy the requirements for this term. We are also looking for ways to simplify or reduce the term without invalidating the secondary equations.
12. **Perform any Reductions** If it is possible to simplify the input equations to the secondary under consideration, do so at this point.
13. **Refine the Secondary Equations** Having developed a new set of equations which satisfy the derived maps and which are simpler to implement, apply them to the affected secondaries.
14. **Loop over Previous Five Steps** This interactive reduction should continue until the equations and implementation are satisfactory.

It should be emphasized that logic synthesis is as much an art and a developed skill as it is a science. No completely mechanical process for deriving circuit solutions will render the conscientious, experienced designer obsolete. Do not think of these procedures as substitutes for thought, but rather as tools. They will not work if used carelessly. Many of the most important constraints in logic design are imposed by considerations which are external to the problem specification. Make sure you understand the problem, the context, the constraints, and the design space before attempting a solution.

In the hands of a thoughtful engineer, the methods presented here can help develop solutions to many complex and difficult synthesis problems. They can also aid in finding simpler solutions to problems normally solved at higher system levels. This is important when available power or silicon real estate is at a premium or when constructing custom cell libraries.

¹In all the designs here we use an inverting gate as the source of the secondary signal. This isn't absolutely necessary, but this choice carries a great deal of simplification power with it.

Chapter 3

Miscellaneous Problems and Solutions

Here is a collection of sequential logic synthesis problems with solutions. Other solutions to these same problems are possible, so these shouldn't be taken as definitive or necessarily optimal. Nevertheless, it is believed that they represent good solutions which are in some sense minimal.

The solution process follows the outline in the previous section. No fundamentally new issues will arise, but each problem may have some unique aspect which may require explanation. Considerations of symmetry in the implementation phase of the development will be given in some cases, and the handling of exceptions to the general procedure will also be treated.

3.1 Clock Stream Switch

The circuit required here is to act as a clock stream multiplexer. It will have a clock input C , an enable E , and two outputs C_1 and C_2 . The clock input is to be routed through the circuit to one of the two outputs, depending on the state of the enable line. The enable input, E is asynchronous with the clock stream and may switch at any time. The outputs, however, should complete the clock cycle in process before they switch so that only full clock periods are presented to the driven circuits. The output lines should be low when they are 'off' (no clock stream is being passed through) and should follow the clock in the same polarity when they are 'on'.¹

A suitable flow table is shown in Table 3.1. After merging and sorting the rows we have Table 3.2. Note that this table has one completely unspecified row already. This is because we were able to merge the original table into three rows, but three rows require *two* secondary variables. Thus an additional unique combination of secondary variables is unused and available.

¹A fixed delay from the source clock to the output clock is acceptable.

CE				$C_1 C_2$
00	01	11	10	
(1)	3	–	2	00
1	–	6	(2)	01
1	(3)	4	–	00
–	3	(4)	5	10
1	–	4	(5)	10
–	3	(6)	2	01

Table 3.1: Flow Table for Clock Stream Switch

CE				$C_1 C_2$
00	01	11	10	
(1)	(3)	4	2	00
1	3	(6)	(2)	01
–	–	–	–	–
1	3	(4)	(5)	10

Table 3.2: Merged, Sorted Flow Table for Clock Stream Switch

Having sorted the rows, the secondary assignments follow easily. C_1 and C_2 are identified with Y_1 and Y_2 , respectively. Secondary assignments are indicated in Table 3.3 with the composite map entries.

The equations for the circuit are shown (3.1) and (3.2), and are implemented in the final circuit shown in Figure 3.1.

$$Y_1 = y_1 C + C E \bar{y}_2 \quad (3.1)$$

$$Y_2 = y_2 C + C E \bar{y}_1 \quad (3.2)$$

$y_1 y_2$	CE				$C_1 C_2$
	00	01	11	10	
00	00	00	10	01	00
01	00	00	01	01	01
11	–	–	–	–	–
10	00	00	10	10	10

Table 3.3: Secondary Assignments for Clock Stream Switch

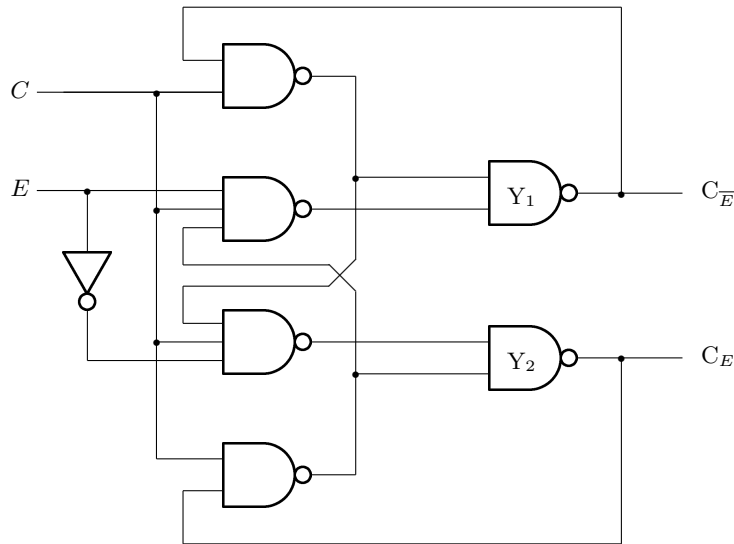


Figure 3.1: Clock Stream Switch

3.2 Two-Phase Clock Generator

This circuit can best be described by referring to Figure 3.2. It has a single input, C , and two outputs, ϕ_1 and ϕ_2 . The input is an externally provided clock signal which is to be separated into a phase-one and phase-two portion. For this reason, the circuit is sometimes referred to as a clock separator or an ‘underlapped’ clock generator.

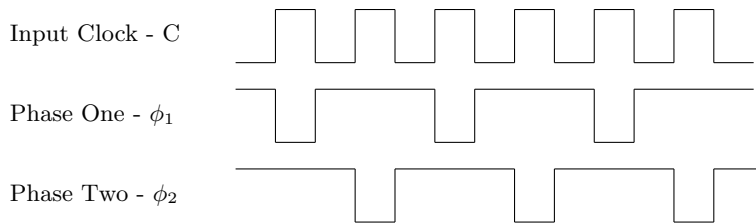


Figure 3.2: Timing Diagram for Two-Phase Clock Generator

The flow table is shown in Table 3.4. Note that this flow table is the same as the table for the ‘T’ flip-flop except for the output assignments. Because the outputs are not unique for two of the rows (row 1 and row 3), we cannot directly assign secondary variables to them.

From our previous experience with the designs in this paper, we might be tempted to try expanding the table. This would, of course, lead us to a solution. But we have a different approach in mind. Because of the similarity

C		$\phi_1 \phi_2$
0	1	
(1)	2	11
2	(3)	01
(3)	4	11
1	(4)	10

Table 3.4: Flow Table for Two-Phase Clock Generator

of the flow table with that of the ‘T’ flip-flop, we choose to examine our previous implementation of the ‘T’ flip-flop to look for already decoded variables which might correspond to ϕ_1 and/or ϕ_2 . Remember that we typically assign secondary variables to outputs so that decoding is not necessary, but such assignment decisions are purely for convenience and may not always be possible or yield the simplest circuit.

The effort required to examine the ‘T’ flip-flop is easily worth the cost. Skeletal flow tables from that device but with stable state entries corresponding to the truth table values for ϕ_1 and ϕ_2 are shown in Table 3.5. We can derive the equations for those outputs from this table, assuming the same row assignments as were used for the ‘T’ flip-flop (not shown in these maps).

C		$\phi_1 \phi_2$
0	1	
(1)		11
	(0)	01
(1)		11
	(1)	10

ϕ_1

C		$\phi_1 \phi_2$
0	1	
(1)		11
	(1)	01
(1)		11
	(0)	10

ϕ_2

Table 3.5: Skeletal Maps for ϕ_1 and ϕ_2

On these assumptions, the decode values for ϕ_1 and ϕ_2 are:

$$\phi_1 = \bar{C} + y_1 \tag{3.3}$$

$$\phi_2 = \bar{C} + \bar{y}_1 \tag{3.4}$$

Except for the change of input variable from T to C these same signals are available already. (See Figure 1.2 for the locations of these pre-existing signals.) Therefore, we do not need to separately implement this circuit — we merely redraw our implementation of the ‘T’ flip-flop to identify the required outputs. This is done in Figure 3.3.

circuit will return to state 1 with not change in the output. If, on the other hand, the input remains high long enough for the circuit to reach state 4, the output will go low.

We could have increased the suppression span by demanding that the output be high in row three, but by accepting the shorter value we can identify the output with \bar{y}_2 or its equivalent. In the implemented circuit (see Figure 3.4) we have found that $\bar{X} + \bar{y}_2$ meets our requirements.

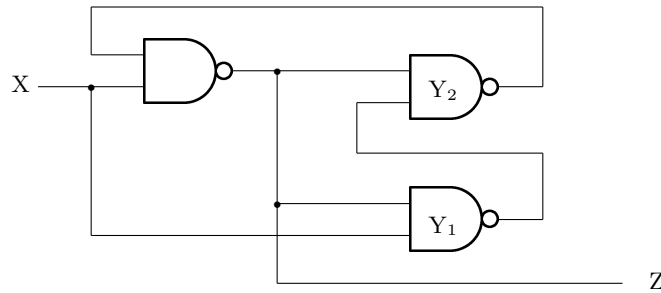


Figure 3.4: Positive Glitch Suppressor

3.4 Digital Single Shot

The digital single-shot is a circuit which picks out a single, full-sized clock pulse from a regular clock signal stream for transmission to the output. The clock input, C , and the control line, E , are the only inputs. Q is the only output. We will require that the output pulse be negative for positive clock pulses and that the control (enable) line be active true. A timing diagram is shown in Figure 3.5.

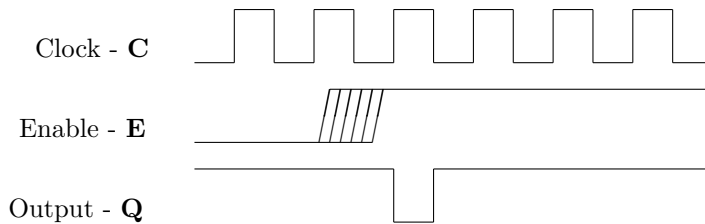


Figure 3.5: Timing Diagram for Digital Single Shot

The merged flow table is shown in Table 3.7 and the equations derived from it are in (3.5) and (3.6).

$$Y_1 = Cy_2 + Ey_1 \quad (3.5)$$

EC				Q
00	01	11	10	1
(1)	(2)	(8)	3	1
1	-	4	(3)	1
1	(5)	(4)	6	0
1	2	(7)	(6)	1

Table 3.7: Merged Flow Table for Digital Single Shot

$$Y_2 = Cy_2 + E\bar{C}\bar{y}_1 \quad (3.6)$$

The completed circuit is shown in Figure 3.6.

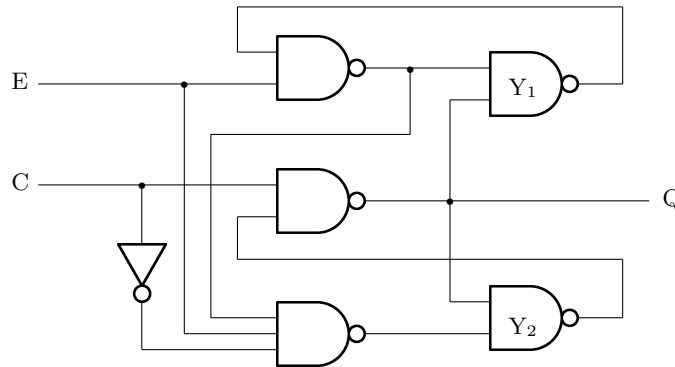


Figure 3.6: Digital Single-Shot

3.5 Master/Slave J-K Flip-Flop

3.5.1 The Flow Table

This device illustrates an interesting aspect of logic design — the compromise of performance requirements vs. circuit complexity. For the ‘D’ flip-flop the problem statement implied edge triggering and the resulting circuit is more complex than, say, a simple ‘R-S’ flip-flop or latch. In the problem statement for the J-K flip-flop, we will accept a constraint on the inputs which simplifies the flow table and the corresponding circuit. The limitation is that the ‘J’ and ‘K’ inputs must not change when the clock is high. This concession allows us to increase the number of *don't care* entries in the flow table, which will reduce the number of terms required in the circuit equations. The circuit outputs changes when the clock goes low.

Most of the popular Master/Slave flip-flops place this constraint on changing the ‘J’ and ‘K’ inputs at the positive level of the clock, leading to the descriptive name ‘ones catcher’ for this device. The derivation presented here will show why this is so.² As will be seen later, the inverted sense of the clock is just as easy to implement.

Before deriving the flow table, we should look at the *truth table* description of the desired circuit behavior. This is shown in Table 3.8.

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Table 3.8: Truth Table for Master/Slave J-K Flip-Flop

This type of description is valuable for specifying the external behavior of the circuit and in constructing the flow table, but it does not have any means of identifying input constraints or distinguishing between edge-triggered and level-sensitive operation. Nor does it indicate the clock polarity.

It is instructive to examine the full, unmerged flow table for this flip-flop. Such a table is shown in Table 3.9.

The *don't care* entries are due to the external restrictions on the ‘J’ and ‘K’ input changes and the general restriction on multiple simultaneous input changes we have used before. We have used $C = 1$ on the right half of the table where the restriction on input changes is in force.

Merging will reduce the number of rows from sixteen to four, as shown in Table 3.10. Because of the large number of initial *don't care* entries, we still have some present in the merged table.

The secondary assignments derived from the merged flow table are shown in Table 3.11.

3.5.2 The J-K Flip-Flop Circuit Equations

The equations for the secondaries are:

$$Y_1 = y_1 C + y_1 y_2 + y_2 \overline{C} \quad (3.7)$$

$$Y_2 = y_1 \overline{K} + \overline{y_1} y_2 + y_2 \overline{C} + C J \overline{y_1} \quad (3.8)$$

Implementing Y_1 is straightforward, but the redundant term $y_1 y_2$ must be included to prevent hazards from occurring during transitions between stable

²The choice of logic levels for the clock is largely arbitrary, but we have chosen the negative clock to comply with standard convention.

CJK								Q
000	001	011	010	110	111	101	100	Q
(1)	2	–	4	–	–	–	8	0
1	(2)	3	–	–	–	7	–	0
–	2	(3)	4	–	6	–	–	0
1	–	3	(4)	5	–	–	–	0
–	–	–	12	(5)	–	–	–	0
–	–	11	–	–	(6)	–	–	0
–	2	–	–	–	–	(7)	–	0
1	–	–	–	–	–	–	(8)	0
(9)	10	–	12	–	–	–	16	1
9	(10)	11	–	–	–	15	–	1
–	10	(11)	12	–	14	–	–	1
9	–	11	(12)	13	–	–	–	1
–	–	–	12	(13)	–	–	–	1
–	–	3	–	–	(14)	–	–	1
–	2	–	–	–	–	(15)	–	1
9	–	–	–	–	–	–	(16)	1

Table 3.9: Flow Table for Master/Slave J-K Flip-Flop

states **12** and **13** in the merged flow table. Our grouping of the three terms is intended to exploit the obvious symmetry in the problem statement.

Y_1 is, by our definition, an external secondary. It is directly identified with the output, Q , and is easily implemented as shown in Figure 3.7.

Considerations of Circuit Symmetry

Suppose that some logic circuit can be represented by a box, as in Figure 3.8. The inputs are on the left, and the outputs are on the right. Internal feedback is concealed within the box, so that external inputs and outputs are all that are visible.

CJK								Q
000	001	011	010	110	111	101	100	Q
(1)	(2)	(3)	(4)	5	6	(7)	(8)	0
–	–	11	12	(5)	(6)	–	–	0
(9)	(10)	(11)	(12)	(13)	14	15	(16)	1
–	2	3	–	–	(14)	(15)	–	1

Table 3.10: Merged Flow Table for J-K Flip-Flop

		CJK							
y_1	y_2	000	001	011	010	110	111	101	100
00		00	00	00	00	01	01	00	00
01		-	-	11	11	01	01	-	-
11		11	11	11	11	11	10	10	11
10		-	00	00	-	-	10	10	-

Table 3.11: Secondary Assignments for J-K Flip-Flop

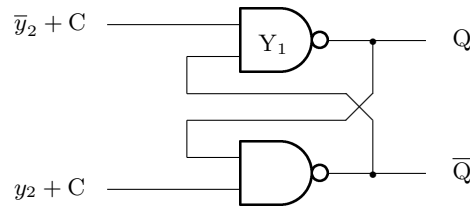


Figure 3.7: Implementation of Y_1

There may be an arbitrary number of inputs and outputs, but in order for the circuit to qualify as a candidate for symmetric treatment the following rules must apply:

- Output lines will occur in related pairs; *e.g.*, each Y will have a corresponding \bar{Y} . It isn't necessary that the problem statement require such pairs, but only the circuit can be put in this form.
- Input lines will be of two types:
 1. Clock or Trigger inputs, which will usually be single-ended in the problem statement,

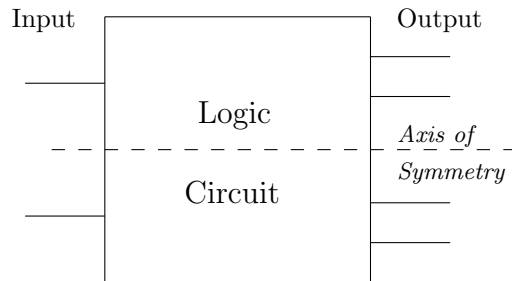


Figure 3.8: Box Model of Logic Circuit

2. Other inputs which can be arranged in pairs with corresponding functions.

If the circuit specification is such that the box representation rules can be applied, we should consider a fully symmetrical implementation.

There are several benefits in exploiting symmetry in a design. For one thing, it assures that corresponding inputs will have similar delay paths through the network. Propagation delays which are equal for symmetric inputs have obvious advantages to the user. For another, the logic implementation process has so many degrees of freedom that it may be largely undirected unless we adopt 'meta' rules to restrict choices. Many different circuit arrangements can meet a given requirement, but some of the designs decisions will be more or less arbitrary. Imposing external constraints allows us to meet other objectives at the price of reducing the number of arbitrary decisions. Symmetry, like minimization, is an aid in directing the design process.

The J-K flip-flop can be put in symmetric form, as shown in Figure 3.9.

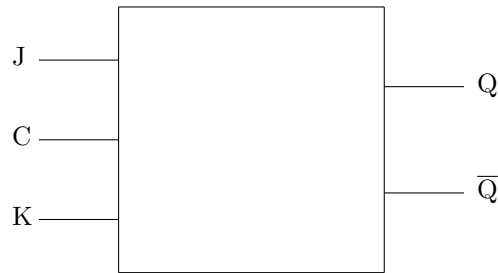


Figure 3.9: Box Model of J-K Flip-Flop

We have already implemented the external secondary, Y_1 , in a symmetrical arrangement. Now we want to look more closely at the internals of the logic box to see what rules are applicable to the rest of the circuit. We find the following:

- For every gate that exists on one side of the axis of symmetry, there will be an identical gate in the mirror image position on the other side axis.
- All secondaries will exist in true and complemented forms (this follows from the previous item).
- Each gate input which comes from a secondary will have a mirror image input which uses the *complemented* secondary.
- Each gate which uses a single-ended or clock signal will have a corresponding mirror image input which uses the *same* signal.
- Each gate input which uses one of the paired input lines will have a mirror image counterpart using the *other* member of the pair.

A close examination of Figure 3.7 in light of these rules will help to clarify their usage. First, notice that Y_1 does exist in true and complemented form already as an R-S latch. Again, this was not part of the original problem statement, but the circuit readily fits the symmetric template and the two signals arose naturally in the implementation. But note that a different grouping of the terms in the equation for Y_1 might have obscured the potential for symmetry rather than bringing it to the surface. Second, note that the input terms are symmetric by our definition. The clock input is single-ended. Logic expressions which include the secondary, y_2 , will be implemented so that one of the mirror image inputs uses the true version and the other uses the complement.

Implementing Y_2 provides a good opportunity to employ a design technique which can reduce the effort in deriving gate input requirements from the output equations. It is particularly useful in designs using NOR gates and AND-OR-INVERT blocks, but is not restricted to those devices. Instead of picking up the 1s from a Karnaugh map to determine the locally true expression for a variable, we can pick up the 0s and invert the entire expression. This approach directly gives a Sum Of Products form for the inputs to the gate and relieves us from having to apply DeMorgan's theorem to determine what the inputs should be. On the other hand, it gives the input equation in a form which is not well-suited for NAND gate usage. NAND gate input requirements are better expressed in a Product Of Sums form where each input line corresponds to one of the product terms.

The equation for Y_2 using this technique is:

$$Y_2 = \overline{y_1\bar{y}_2 + \bar{C}\bar{y}_2 + \bar{y}_2\bar{J} + y_1CK} \quad (3.9)$$

Factoring out y_2 so that only two inputs are required gives (on the input side of the gate):

$$\bar{Y}_2 = \bar{y}_2(y_1 + \bar{C} + \bar{J}) + y_1CK \quad (3.10)$$

The question, "Can this expression be implemented symmetrically?" is answered in the affirmative. To see this, consider the terms in (3.10). First, the equations defines, as many secondaries do, a cross-coupled latch or memory element. Such equations, which are expressed in terms of one of the two latch outputs (or its input), contain terms for both of the gates forming the latch. Thus, (3.10) consists of two major terms. One, which is written: y_1CK , represents a direct input line to the gate Y_2 , and the other, which is written: $\bar{y}_2(y_1 + \bar{C} + \bar{J})$, represents the output of the other gate of the pair. We see that the direct term defines one input to Y_2 . The other term defines the other input to Y_2 and, since this signal comes from the other member of the latch, its inverse defines the inputs required for the other gate. Thus, if we invert the second term, the resulting unimplemented inputs should satisfy our symmetry rules. A partial implementation of Y_2 is shown in Figure 3.10, where the symmetry can be easily seen.

The completed circuit is shown in Figure 3.11.

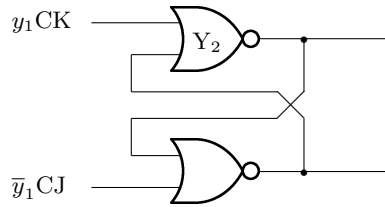


Figure 3.10: Partial Implementation of Y_2

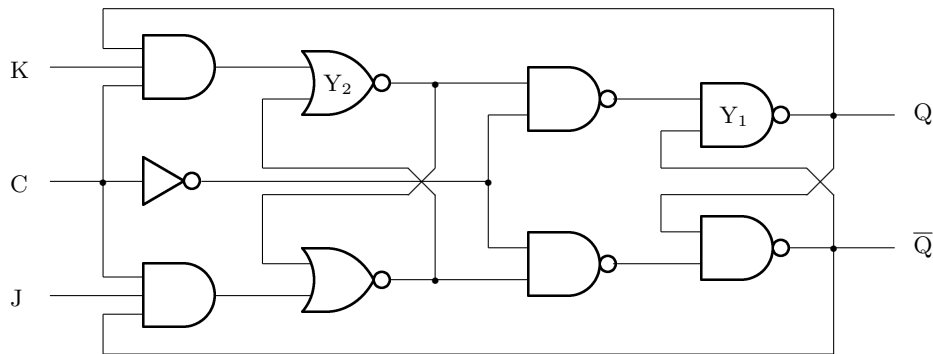


Figure 3.11: Master/Slave J-K Flip-Flop

Before leaving this design a few comments are in order. First, the circuit as shown expects C as an clock input, rather than \bar{C} . In our original problem statement we made the decision to use negative clocking and built the flow table to reflect that choice. But, since the circuit uses both the input clock and its inverse, we could accept either as an input and provide the other with an inverter. If the logic devices have equivalent delays, no critical races should occur.

Second, our implementation of the external secondary, Y_1 , could have used AND and NOR gates rather than NANDs. This is a ‘dealer’s choice’ issue. Other implementation choices could have been made as well.

Third, the Master/Slave J-K flip-flop represents a simple problem in designing by symmetry rules. The Edge-Triggered J-K flip-flop in the next section is more complex, and some of the subtle difficulties in dealing with symmetry are taken up there.

Finally, this is a good example of a circuit type which calls out for asynchronous *set* and *clear* inputs. Because of their symmetric role, they can also be retrofitted symmetrically. The solution to this design challenge is left to the reader.

3.6 Edge-Triggered J-K Flip-Flop

The best way to avoid having to impose restrictions on changing the inputs to a J-K flip-flop when the clock is high (or low) is to require that the flip-flop be edge triggered. The design will be more involved in this case, but the added complexity may be worth the added difficulty.

3.6.1 The Flow Table

For the edge-triggered J-K flip-flop the flow table must account for input changes when the clock is in either state. Only at the time of a clock change will the values of the ‘J’ and ‘K’ inputs matter. We will build the flow table for positive edge triggering, but otherwise the circuit behavior is completely described by a truth table which is similar to the previous Master/Slave J-K version, except that here we will use positive edge-triggering rather than negative level triggering.

3.6.2 Edge-Triggered J-K Flip-Flop Flow Table

The full flow table is shown in Table 3.12.

CJK								Q
000	001	011	010	110	111	101	100	Q
(1)	2	–	4	–	–	–	8	0
1	(2)	3	–	–	–	7	–	0
–	2	(3)	4	–	14	–	–	0
1	–	3	(4)	13	–	–	–	0
–	–	–	4	(5)	6	–	8	0
–	–	3	–	5	(6)	7	–	0
–	2	–	–	–	6	(7)	8	0
1	–	–	–	5	–	7	(8)	0
(9)	10	–	12	–	–	–	16	1
9	(10)	11	–	–	–	7	–	1
–	10	(11)	12	–	6	–	–	1
9	–	11	(12)	13	–	–	–	1
–	–	–	12	(13)	14	–	16	1
–	–	11	–	13	(14)	15	–	1
–	10	–	–	–	14	(15)	16	1
9	–	–	–	13	–	15	(16)	1

Table 3.12: Flow Table for Positive Edge-Triggered J-K Flip-Flop

Merging is straightforward, except that there are fewer *don't cares* than there were in the Master/Slave flip-flop. The merged table is shown in Table 3.13

CJK								Q
000	001	011	010	110	111	101	100	0
(1)	(2)	3	4	(5)	(6)	(7)	(8)	0
1	2	(3)	(4)	13	14	–	–	0
(9)	10	11	(12)	(13)	(14)	(15)	(16)	1
9	(10)	(11)	12	–	6	7	–	1

Table 3.13: Merged Flow Table for Edge-Triggered J-K Flip-Flop

Secondary assignments present no difficulties, largely because of the care we have taken in constructing the flow table. The result is shown in Table 3.14.

CJK								
$y_1 y_2$	000	001	011	010	110	111	101	100
00	00	00	01	01	00	00	00	00
01	00	00	01	01	11	11	–	–
11	11	10	10	11	11	11	11	11
10	11	10	10	11	–	00	00	–

Table 3.14: Secondary Assignments for Edge-Triggered J-K Flip-Flop

3.6.3 The Circuit Equations

The equation for Y_1 is:

$$Y_1 = y_1\bar{C} + y_1y_2 + y_2C \quad (3.11)$$

Note that this is the same as the equation for Y_1 of the Master/Slave device. Rather than dwell on the implementation, we can just take it from the previous design. See Figure 3.7.

The implementation of Y_2 is more of a challenge. We begin by writing the equation for \bar{Y}_2 in order to deal directly with the input side of a NOR latch. Equation (3.12) contains all the zeros and only the zeros from the Karnaugh map for Y_2 . We have included redundant terms or *tie sets* covering those states between which transitions can occur.

$$\bar{Y}_2 = \bar{C}\bar{J}\bar{y}_1 + \bar{C}K y_1 + C\bar{y}_1\bar{y}_2 + K y_1\bar{y}_2 + \bar{J}\bar{y}_1\bar{y}_2 \quad (3.12)$$

Since we wish to avail ourselves of readily available signals we will group the terms of (3.12) as in Equation 3.13. This provides us with a starting point for our design by symmetry.

$$\bar{Y}_2 = K y_1(\bar{C} + \bar{y}_2) + \bar{y}_1(\bar{C}\bar{J} + C\bar{y}_2 + \bar{J}y_2) \quad (3.13)$$

3.6.4 The Implementation

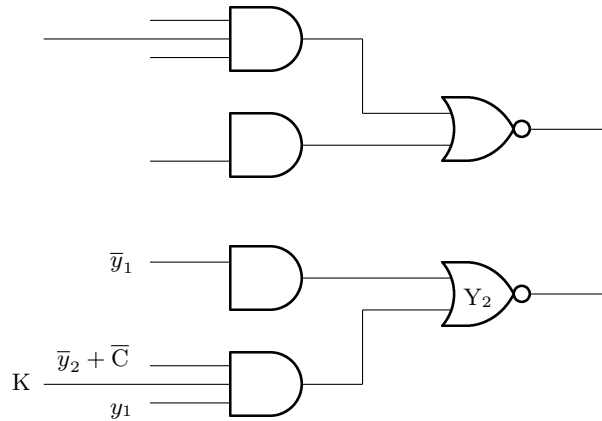


Figure 3.12: 1st Partial Implementation of Y_2

A partial implementation of Y_2 provides clues which will carry us forward to the complete circuit. See Figure 3.12. We can see from the terms in (3.13) that two AND gates will be required at the input of Y_2 , hence the mirror image gate, which we expect to identify with \bar{y}_2 , also has two AND gates. The symmetric signal requirements are spelled out in Figure 3.13, in accordance with the rules given previously.

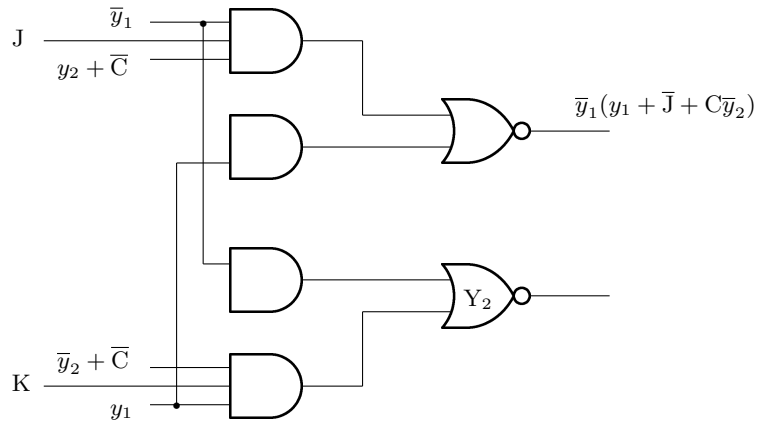


Figure 3.13: 2nd Partial Implementation of Y_2

The schematic diagram in Figure 3.13 can be derived from the partial schematic in Figure 3.12 by appealing to the rules of symmetry. Now we should assess our progress by looking at the partially developed output of the mirror image

gate, and comparing it with the remaining requirements for the input to Y_2 . Naturally, we would like to cross couple the gates as soon as we are certain that the equation for Y_2 is satisfied.

We see from (3.13) that we need to provide $\overline{C}J + \overline{y}_2C + \overline{J}\overline{y}_2$ at the unused input to the upper AND gate of Y_2 . The output of the mirror image gate will produce this if it is ANDed with the existing signal $\overline{C} + \overline{y}_2$. We must verify, however, that connecting the lines suggested by symmetry does not invalidate the equation. This validation and the assurance that cancellation of literals does not introduce new problems is left as exercises. The complete circuit is shown in Figure 3.14.

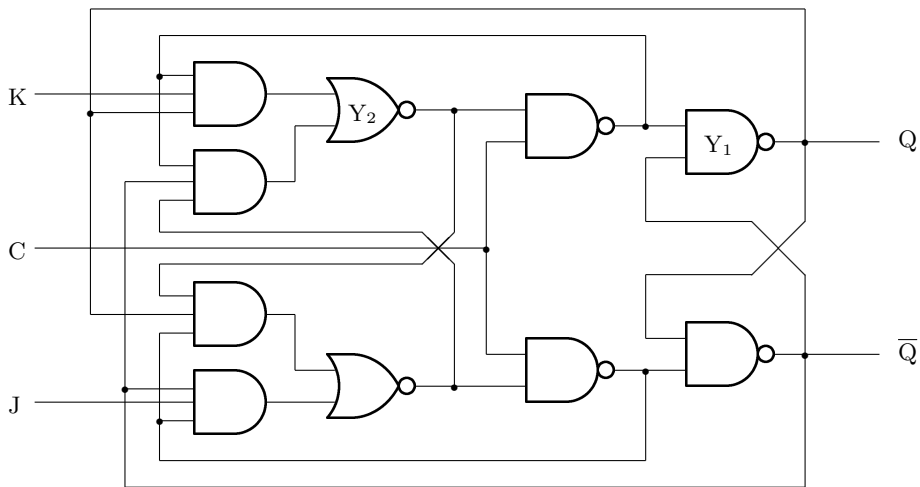


Figure 3.14: Edge-Triggered J-K Flip-Flop

A Comment on the Concept of Edge-Triggering

The term ‘edge-triggering’ is used rather loosely in logic hardware literature. Its meaning must be determined from the context, because there are two distinct senses in which the term is used. One, to mean *event driven* and two, to mean *edge sensitive* (as opposed to *level sensitive*).

An event driven device³ may be designed to perform its function on the rising or falling edge of some input, but may not be immune to changing input conditions at other times. On the other hand, an edge sensitive device is, presumably, unaffected by input changes at any time except near the rising or falling edge of some specific input. The term *pulse triggered* is also sometimes used to refer to clocked devices which are designed to respond to a relatively narrow input clock.

³We may consider state transitions or edges to represent *events*. When all transients have completed, we are in a steady state.

Data sheets for many J-K flip-flop implementations refer to them as edge-triggered when, by our definition, they are not. Specifically, we define an edge-triggered device to be one which is driven by the stated edge of its input line, and whose next state is determined solely by the conditions that exist at or near the time of the triggering event. Input changes that occur at other times have no effect on the operation of the device. Of course, practical circuit implementation considerations will affect the specific timing limits for any circuit. These should be identified in the *set up* and *hold* time specifications.

For example, the Master/Slave J-K flip-flop is intended to perform its function when the appropriate clock occurs and it is the leading edge of the clock which drives the output changes. Nevertheless, when the clock is high, changes in the ‘J’ or ‘K’ inputs can change the intended future state.⁴

It is important to read the fine print in the data sheets to determine whether there are any constraints on the inputs during either state of the clock for a specific device. Don’t depend on ambiguous terminology in selecting a device for your application.

⁴This property leads to the ‘ones-catching’ behavior sometimes noted in the literature.

Part II

Analysis Methods

Chapter 4

Sequential Circuit Analysis

Synthesis is the major problem in digital circuit design. Analysis, although it enjoys wide coverage in the analog domain, and even in digital signal processing, is poorly represented in the digital logic domain. Nevertheless, analytic tools are powerful aids in solving the synthesis problem and in understanding or verifying the performance of existing digital circuits. We need a good analytic method if we are to be able to examine the schematic for a given logic circuit and answer the questions: What does it do? How can we use it? Will it meet certain requirements? In other words, the analysis problem we are addressing is behavior analysis, not timing analysis.

In this chapter, we present a method for analyzing sequential circuits which complements the synthesis procedures given earlier. The process is almost exactly the reverse of that for circuit synthesis. With few exceptions,¹ it can be used to derive the governing Boolean equations for a given circuit, and to reconstruct the flow tables and high level behavioral descriptions.

4.1 Fundamental Properties of Secondary Variables

Before presenting the analysis method, we should gain a clear understanding of what a secondary (internal) variable is, and what its role is in satisfying the overall sequential circuit requirements.

In the literature it is shown that secondary variables are associated with internal states of a sequential state machine, *i.e.*, a machine whose current output depends on previous outputs as well as present inputs.² These internal states can be numbered and identified, for example, with some kind of binary

¹Some logic circuits are designed to exploit special properties of the devices used to implement them, and their behavior cannot be completely specified by a formal method.

²For purely combinational circuits, no secondary variables are required because the output values are completely determined by the current input values.

counter which has a sufficient number of stages to uniquely determine them. Thus, we expect approximately $\log_2(n)$ latches for every n states.

We further expect that in examining the equations for a sequential machine we will find equations for memory or storage elements which cannot be reduced to purely combinational form. The difference between combinational equations and sequential equations is of fundamental importance, leading us to consider the question of what that difference is, and how to identify it.

There is no better way to understand the algebra of memory than to examine the simplest memory element: the cross-coupled latch. In Figure 4.1 we show an R-S flip-flop whose boolean equation can be written:³

$$Q = \bar{S} + qR \tag{4.1}$$

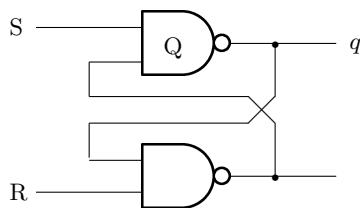


Figure 4.1: Cross-Coupled Flip-Flop (Latch)

The memory feature of this circuit is found in the presence of the Q term in both the input and output of the circuit equation. From an analog perspective, this is feedback with greater than unity gain. If the S and R lines are set to 1, then the circuit equation reduces to $Q = q$, which is an identity. Thus, if q was 1 then Q will be 1. Similarly, if q was 0 then Q will be 0. When we speak of memory in a digital circuit context, this behavior is what we mean.

Notice that the only way to prevent the latch from remaining forever in the same state is to active S or R . (Of course, we are assuming that power to the circuit is always available.) S is the means to force a 1 state and R is the means to force 0. In general we will find that memory elements, or secondaries, will be defined by equations of the form,

$$Y = f(x_i, y_j) + y \cdot g(x_m, y_n) \tag{4.2}$$

where $f(x_i, y_j)$ and $g(x_m, y_n)$ are functions of the inputs, X_p and outputs, Y_q . The variable y can be forced to either state by appropriate manipulation of the primaries and secondaries, and will serve as a memory element when $f(x_i, y_j)$ is false and $g(x_m, y_n)$ is true. Corresponding equations for NOR latches or combinations of gates can be developed as long as the elements have gain, delay, and can be connected so as to satisfy the memory requirements.

³**R** and **S** stand for Reset and Set, respectively.

4.2 An Outline of the Analysis Method

Our analysis strategy will be to develop the circuit equations for the given circuit from the schematic diagram and to construct the flow table from the equations. We will then determine the stable state behavior from the flow table, giving us the information needed to generate a high level description.

The procedure will follow a simple set of elementary steps:

- Label each of the input signals,
- Label all gates with arbitrary names or symbols,
- Write the equations for each gate in terms of the labelled gates and the input signals,
- Reduce the equation set by the elimination of unnecessary internal variables,
- Construct the flow table from the remaining equations,
- Identify the stable states,
- Formulate a circuit description.

We will step through the solution of several example problems to illustrate and clarify the procedure.

4.3 A Simple Example

Although we have chosen the Negative Pulse Generator for this example, we will assume no prior knowledge of the circuit behavior. The circuit design is provided to us with no description and only the unlabelled schematic.

4.3.1 Labelling the Circuit Elements

First, we must label the inputs and gates. This is shown in Figure 4.2.

We are using upper case alphabetic characters to identify the gates, and lower case letters to identify the output signals from those gates. This is essentially a mnemonic device and could be replaced by some other labelling scheme.

4.3.2 Writing the Raw Circuit Equations

We may write the equations directly from the labelled schematic.

$$A = \bar{X} + \bar{c} \quad (4.3)$$

$$B = \bar{X} + \bar{a} \quad (4.4)$$

$$C = \bar{a} + \bar{b} \quad (4.5)$$

$$(4.6)$$

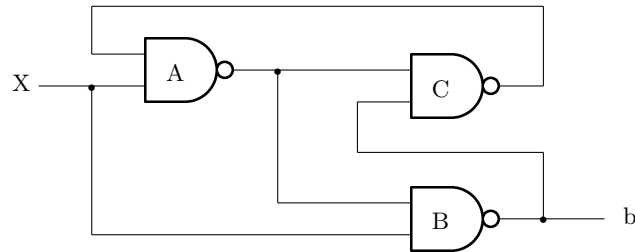


Figure 4.2: Unknown Circuit with Labels

We have placed the gate label, possibly representing a secondary variable, on the left side of each equation and the terms of the required equation on the right. This is also a convenient strategy, but it reminds us that the Boolean equations in a sequential circuit are not true equalities — they are flow statements. We understand them to mean that the conditions on the right side of the equation (true or false) determine the condition on the left.⁴

Clearly the memory capability (if any) of the circuit is hidden among the equations which describe the relations between the variables, and some steps must be taken to uncover memory elements. This is the heart of the analysis procedure.

4.3.3 Reducing the Equations

For a combinational circuit we can label gates and write equations as we have done for the circuit we are presently examining. But we can also describe each output solely in terms of the inputs. Thus, for the output of each gate, we can substitute the controlling expression (possible inverted) for its input, and eliminate the unnecessary label it used. By successive eliminations we can eventually represent all outputs in terms of the inputs.

For a sequential circuit, the substitution process will terminate differently. We will eventually arrive at a minimum set of equations involving inputs and memory elements beyond which no further reduction is possible. In some cases we will have irreducible equations which indicate essential secondaries, and reducible equations which represent circuit outputs. We will retain the latter, even if they are reducible, but our objective is not to eliminate gates from the equations set, but only to eliminate unnecessary labels. The remaining labels will be used in the expanded flow table.

The process of reduction begins by examining the circuit to identify any output signals which, for convenience, we wish to identify with secondary variables. Clearly *B*, in the present example, is an obvious choice for a secondary and we

⁴It actually would have been better to use arrows instead of equal signs, but we prefer to submit to convention rather than unnecessarily disturbing it.

will proceed with this in mind. Turning to the equation for A , we see that all instances of a in the other equations can be replaced by $\bar{X} + \bar{c}$. By making this replacement, we find that no secondary called A is actually needed. Hence we can reduce the equation set to,

$$B = \bar{X} + Xc \quad (4.7)$$

$$C = \bar{b} + Xc. \quad (4.8)$$

Now, the equation for C is clearly that of a memory element, and cannot be eliminated because it contains itself as a term. We could attempt to eliminate B , but that would be absurd because it is the generator for the circuit output signal. At this point we have a set of equations which describe a sequential circuit constructed from an expanded flow table.

One further simplification of the equations is possible. The X term in the equation for B is redundant. We can therefore simplify B to,

$$B = \bar{X} + c. \quad (4.9)$$

4.3.4 Constructing the Flow Table

Having derived equations for the circuit, we can now construct the truth table from which the flow table is derived. This truth table has the same structure as the Karnaugh map for the circuit variables. Because of this identification between the map and flow table, we need not be too rigorous in our terminology. The main difference between the map and flow table is that the table is concerned with state changes and the map facilitates term grouping.

First, we determine the number of rows and columns from the number of inputs and secondaries. An appropriate skeletal table is shown in Table 4.1.

		X		
b c	0	1	B	
00			0	
01			0	
11			1	
10			1	

Table 4.1: Skeletal Flow Table (map) for Example Circuit

Now we simply fill in the truth values derived from the equations. That is, B is true whenever X is false or C is true. We put the ones in the table for all these cells, and put zeros in the remaining cells. Since we have two secondaries to consider, we can double up and put the truth values for both B and C in the map at once. We have done this in Table 4.2. On completion of the map, we

can identify those cells which represent stable states as those cells whose entries agree with their row assignments. These are already identified in the table.

bc	X		B
	0	1	
00	11	01	0
01	11	11	0
11	10	(11)	1
10	(10)	00	1

Table 4.2: Karnaugh Map for Example Circuit

Although it is not necessary, we can convert this to the same form as was used in the Negative Pulse Generator. The conversion simply consists of relabelling the input and secondaries, coupled with re-ordering of the rows and decimal numbering of the stable and unstable states. None of these operations alters the circuit behavior.

4.3.5 The Circuit Description

The stable state change that occurs when the input X falls does not produce any output response. This is seen by simply tracing the behavior from the stable state cell labelled '11' to the adjacent cell and then into the row with the corresponding entry. On the other hand, when X rises, the circuit produces the negative pulse for which it was originally designed. Our analysis has successfully retained and identified the performance feature of interest and provided us with a complete description.

4.4 Another Example

Given the circuit in Figure 4.3, derive the circuit equations, generate a flow table, and provide a description of its function.

4.4.1 The Circuit Equations

Using the labels we have placed on the gates, we write the following equations,

$$A = \bar{X}_1 + X_2 \quad (4.10)$$

$$B = \bar{X}_1 + \bar{d} \quad (4.11)$$

$$C = \bar{X}_2 + \bar{d} \quad (4.12)$$

$$D = \bar{a} + \bar{b} + \bar{c}. \quad (4.13)$$

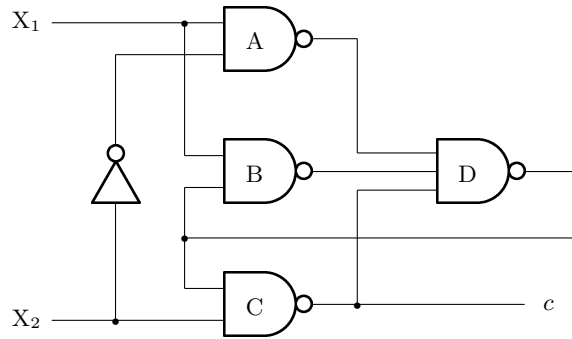


Figure 4.3: 2nd Example Circuit

We will retain C because it is a required output signal, and retain D because we can see that it is part of an internal latch. B or D could be retained, as either one could be identified with the memory element.

Eliminating A and B by substitution, we have,

$$C = \bar{X}_2 + \bar{d} \tag{4.14}$$

$$D = X_1\bar{X}_2 + X_1d + \bar{c}. \tag{4.15}$$

4.4.2 The Flow Table

We can now construct the truth table. There are two inputs and two secondaries of interest. A table with the truth values derived from the equations is shown in Table 4.3. We have also identified the stable states in the table.

		$X_1 X_2$				
$c d$		00	01	11	10	C
00		11	11	11	11	0
01		11	(01)	(01)	11	0
11		10	00	01	(11)	1
10		(10)	(10)	(10)	11	1

Table 4.3: Karnaugh Map for 2nd Example

At this point we will move on to the construction of a flow table. We do this by numbering the stable states as shown in Table 4.4, where the unstable states with a Hamming distance of unity are also shown.

To create the primitive flow table we need only build a table with one row for each stable state and fill in the unstable cells from the merged table in Table 4.4. Note that stable states **3** and **4** are the only stable states in their respective

		X ₁ X ₂		
00	01	11	10	C
				–
	(1)	(2)	3	0
4	–	2	(3)	1
(4)	(5)	(6)	3	1

Table 4.4: Partial Flow Table for 2nd Example

columns, so it is reasonable to fill the remaining cells in those columns with the same number, although we could also leave them initially blank.

In fact, we do not really need the expanded table to determine the circuit behavior. From inspection of the merged table we can see that there are two possible output levels. The output *zero* level is associated with stable states **1** and **2**, and can only be entered following state **3**. The only way to return the output to the *one* level is by moving to state **3** or **4**.

States **2** and **3** represent a cycle or loop. When in one of these states the circuit moves to the other following a change in X_2 . Thus, X_2 is passed inverted to C in this mode. Before entry to this mode, or after exit via state **4**, the output, C , stays high.

The circuit is a synchronous clock enable. Inspection of the flow table shows that it is designed to pass (inverted) the first full length positive pulse which occurs on X_2 (clock) after X_1 (enable) is raised, and to pass all subsequent pulses until X_1 is lowered, permitting the last pulse to complete before raising the output line. It is similar to our earlier Clock Stream Switch, and was taken from a partial schematic for a 74120 pulse synchronizer.

4.5 Caveats and Cautions

As mentioned in an earlier footnote, this analysis procedure will not necessarily reverse the synthesis procedure used to create every device. Some designs depend for their operation on special properties of the components used in the implementation. For those cases, part of the logic description is implicit in the elements used, and may not be amenable to the analytic procedure described here.

Finally, since the synthesis procedure often involves arbitrary decisions and creative adjustments, the analysis procedure may not be entirely straightforward. Be prepared to consider any analysis problem from a fresh point of view.

In spite of these limitations, the analysis procedure will work on all the designs presented in this paper, and most of the designs one is likely to encounter in practice.

Bibliography

- [Cald 58] S. H. Caldwell, *Switching Circuits and Logical Design*, John Wiley & Sons, 1958.
- [Huff 54] D. A. Huffman, *The Synthesis of Sequential Switching Circuit*, J. of Franklin Institute, pp. 275–303, March and April 1954.
- [Huff 55] D. A. Huffman, *The Design of Hazard-Free Switching Networks*, M.I.T. Press, Cambridge, Mass., 1955.
- [Karn 53] M. Karnaugh, *The Map Method for the Synthesis of Combinatorial Logic Circuits*, Comm. and Electronics, No. 9, 1953.
- [Maley 63] G. A. Maley, *The Logic Design of Transistor Digital Circuits*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1963.
- [Marc 62] Mitchell P. Marcus, *Switching Circuits for Engineers*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1962.
- [Mealey 55] George H. Mealey, *A Method for Synthesizing Sequential Circuits*, Bell System Technical Journal, September 1955.
- [Moore 54] E. F. Moore, *Gedanken-Experiments in Sequential Machines*, Automata Studies, Princeton University Press, 1954.
- [Ung 59] S. H. Unger, *Hazards and Delays in Asynchronous Sequential Switching Circuits*, IRE Trans. Circuit Theory, Vol. CT-6, pp. 12–25, March 1959.

Index

- A**
 - analysis
 - method, 61
 - asynchronous, **5**
- B**
 - Boolean, 8, 21
 - algebra, 5
 - equation, 32, 59, 60, 62
 - function, 21
- C**
 - charge pump, 29
 - circuit
 - analysis, 59
 - reduction, 62
 - sequential, 5, 6, 40, 59, 62, 63
 - synthesis, 40
 - circuit symmetry, 48
 - clear, 6, 21, 22
 - clock stream switch, 40–41
 - counter controller, 25–28
- D**
 - DeMorgan's theorem, 5, 10, 33, 34, 51
 - don't care*, 25, 31
- E**
 - edge-triggered, 12
 - edge-triggering, **56**
 - equation
 - circuit, 5, 9, 10, 14, 21, 27, 33, 35, 47, 54, 60, 61, 64
 - secondary, 9
- F**
 - flip-flop
 - D, 6, 12–23
 - edge-triggered, 53–56
 - J-K, 46–56
 - master/slave, 46–52
 - R-S, 60
 - T, 7–11, 15, 43
 - flow table, 5, 7, 12, 21, 23, 25, 29, 40, 42, 44, 45, 47, 53, 63, 65
 - expanded, 31
 - merged, 13, 19, 29, 54
 - skeletal, 19, 34, 63
- G**
 - glitch suppressor, 44–45
- H**
 - hazard
 - critical, 10, 11, 15, 48
 - dynamic, 15
 - static, 15
 - suppression, 17
 - suppression, 11, 15, 16, 19
- K**
 - Karnaugh map, 5, 19, 65
- L**
 - latch, 21, 27, 60
 - logic
 - combinational, 5
- M**
 - merge, 12, **12**
 - metastable, 27
- P**
 - phase-locked loop, 29
 - phase/frequency detector, 28–36
 - pulse generator, 23–25

R

race condition, 11

S

secondary

 strong, 28

 weak, 28

set, 6, 21, 22

signal

 clear, 21

 identification, 21

 inversion, 11

 location, 21

 preset, 21

single shot, 45–46

stable state, 21

state

 stable, 7, 8, 19

 transitional, 7, 23

state machine, 5, 59

synchronous, 5

synthesis, 37

T

truth table, 9, 63

two-phase clock, 42–43

U

up-down counter, 25

V

variable

 primary, 19

 secondary, 8, 13, 19, 21, 31, 40, 59