SEQUENTIAL LOGIC DESIGN

USING COUNTERS AS MEMORY ELEMENTS

by

ARTHUR DAVID SCHRANK

Thesis submitted to the Graduate Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

R. A. Thompson, Chairman

A. W. Bennett

F. G. Gray

October, 1974

Blacksburg, Virginia

ACKNOWLEDGEMENTS

My deepest and most sincere thanks must go to my advisor, . Without his guidance, counsel and encouragement, this thesis might never have been written. Whatever benefit is to be found from this thesis must be attributed to him.

I also wish to thank NSF for their financial support during the Electronic Telephone System project from which the inspiration for this thesis emerged.

Thanks are also due to for her efforts in typing the thesis. The encouragement, understanding, and prayers of my family and friends must also be mentioned. Phil. 4:13.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

1.1 Background

Consider the canonical form of sequential machines (3) as shown in figure 1.1.1. The sequential machine consists of combinational logic and unit delay elements. The combinational logic, which is a function of input variables and the present value of the state variables, determines both the output variables and the next-state transition variables. The unit delay elements, or binary storage devices (BSD), are elements whose output y(t) is related to its input Y(t) by

$$y(t) = Y(t-1) \qquad (1.1.1)$$

(6). Typically these devices are either D or JK flip flops.

Now, suppose these BSD's are removed and replaced with another type of memory device, which we will call memory function devices (MFD's). These MFD's each have within them one or more BSD's plus additional logic to provide certain controls over the operation of these BSD's. The MFD's then, have in addition to the normal BSD inputs (some MFD's do not use these inputs), special control inputs. See figure 1.1.2.

How can an increase in the number of inputs to memory devices be beneficial to logic design? The question here is similar to, and in a sense an extension of, the design considerations encountered when deciding whether to use D or

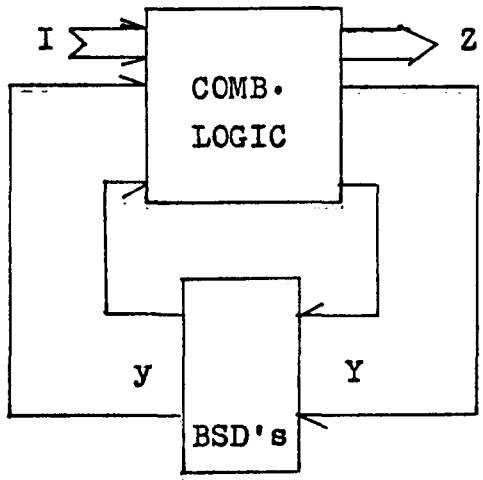


Figure 1.1.1. Canonic Sequential Machine

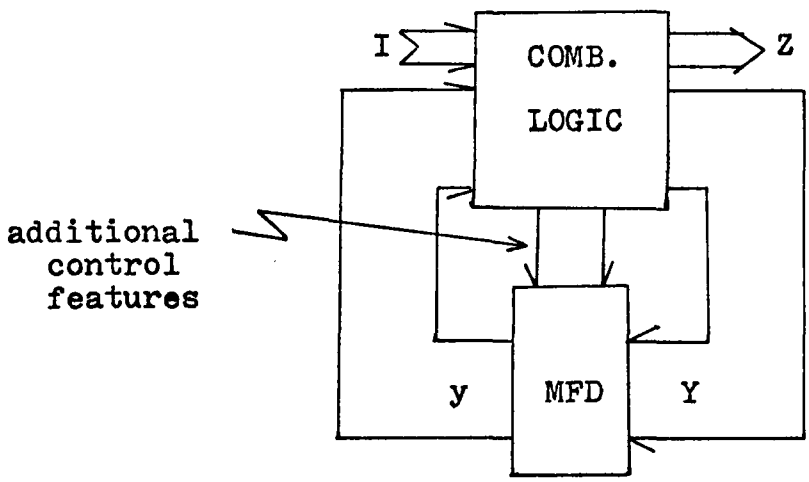


Figure 1.1.2. Modified Sequential Machine

JK flip flops. JK flip flops have twice as many inputs as D flip flops, yet in many machine realizations less logic is needed when JK rather than D flip flops are used. Why? Although there are twice as many inputs, and therefore twice as many Karnaugh maps for JK designs, there are also more "don't-care" entries on each map than there are for D flip flop designs. This usually leads to less logic needed to realize a machine. In the same way, the additional control inputs increase the total number of memory inputs but also add many "don't-care" entries on the various Karnaugh maps of the MFD's. In some designs this will require less logic than either JK or D designs. To show that this logic reduction is possible in certain machines is one objective of this thesis. A second point to be made is that some MFD's can be used to realize any finite sequential machine.

With the continuing advances in integrated circuit technology, more memory functions are becoming available on single chips, adding even more reason to consider their use in the design of sequential machines. The three most prominent types of memory functions are: shift registers, decoders, and counters. Much work has already been done in the design of machines using one or more shift registers (8, 10). The design of sequential machines using decoders is the topic of a thesis now being prepared by another graduate student here at VPISU. The use of count-

ers to realize a given machine will be the primary topic under discussion in this thesis.

1.2 Types of Counters

Counters can be grouped according to several characteristics (4). One division is asynchronous and synchronous counters. An asynchronous counter has an arrangement of BSD's in which the clock does not simultaneously activate all BSD's. An example of an asynchronous counter is the ripple counter since each BSD (following the first) in the counter is clocked by the preceding BSD. Note that this property will cause false state transitions during the time the BSD's are changing. This problem can be corrected by adding logic to permit the counter outputs to appear only after the BSD's have reached a stable state. This is usually done by adding a strobe pulse input to the counter. In synchronous counters all the BSD's are clocked simultaneously. Synchronous counters do not have the transition problem of asynchronous counters. The speed of synchronous counters is limited by the delay in the slowest BSD plus the delays in control logic.

Another distinguishing feature of counters is the type of output code used (4). Most manufactured counters are of two types: binary - those counters that with n BSD's have 2^n distinct output codes; and decimal - those counters that with n BSD's have 10^m distinct output codes

where $2^{n-1} < 10^m \leq 2^n$. Counters of modulus other than 2^n are possible and can be constructed from binary counters and appropriate control logic.

This paper will consider synchronous binary counters, although the general results should apply equally well to synchronous decimal counters. The types of counters under consideration in this paper can be roughly ordered in terms of their internal complexity: UP counters, UP/RESET counters, UP/DN counters, UP/DN/RESET counters, UP/DN/PRESET counters and finally UP/DN/RESET/PRESET counters. A symbolic and truth table representation for each of these counters appears in figures 1.2.1 and 1.2.2. We assume that all counter features are synchronous (i.e. all inputs affect the counter only at the normal clock time).

The first counter type, a modulus 2^n UP counter has only the count UP feature (note that DOWN could be used in place of UP without changing the results of this discussion), n present state outputs and no next-state transition inputs.

The UP/RESET counter has the UP counter features with the additional feature of being able to reset to zero (00---0) from any state. Some decision must be made (by the manufacturer of the chip) concerning the 11 condition on the truth table. We will consider RESET to have priority over the UP function and therefore assume that 11 is a RESET condition for the UP/RESET counter.

Figure 1.2.1 shows two UP/DN counters. With the exception of the particular input combinations to perform each function, these counters are essentially the same. The counters are capable of counting up or down from each of the n states. Figure 1.2.1(d)does have one advantage over figure 1.2.1(c)in the introduction of a "don't-care" situation for the UP/DN function when ENABLE is "0". For this reason figure 1.2.1(d)will be considered. We will continue to refer to it as an UP/DN counter rather than an UP-DN/ENABLE counter.

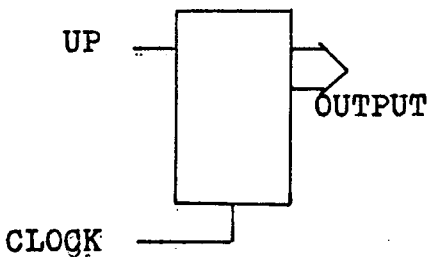
Note that there is still another type of UP/DN counter - one that has no ENABLE input and must therefore count either up or down at every clock pulse. This counter has been investigated and design procedures for using it are available (8, 5).

The UP/DN/RESET counter in figure 1.2.2 has all the UP/DN counter features with the additional feature of being resetable to zero state (00---0) from any state. Assuming that RESET has priority over ENABLE causes "don't-cares" in ENABLE and UP/DN when RESET is "1".

The UP/DN/PRESET counter has, in addition to the UP/DN feature, n input lines which,with the PRESET,control the inputs to the BSD's within the counter. This allows a transition from each state to any other state. We will assume PRESET has priority over ENABLE. This results in a truth table very similar to UP/DN/RESET except that "dd1"

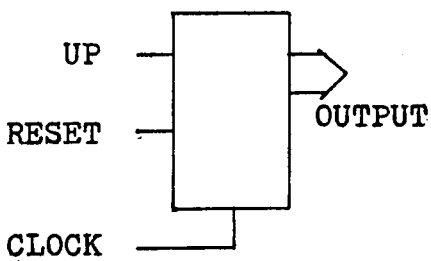
causes the outputs to change to whatever value is on the input lines.

The last counter shown in figure 1.2.2 is an UP/DN/PRESET counter with the addition of the RESET feature. Assume that PRESET has highest priority, RESET next, and then ENABLE. This counter and the UP/DN/PRESET counter are almost identical in their capabilities since PRESET can be used as RESET. In some cases the addition of the RESET feature may reduce the PRESET logic enough to justify its use.



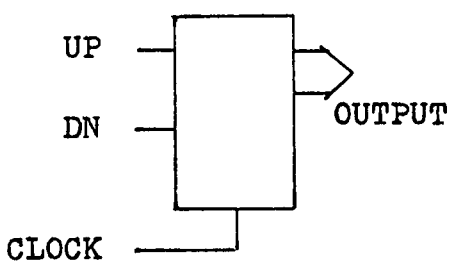
UP	OUTPUT
0	No Change
1	Count Up 1

(a) UP COUNTER



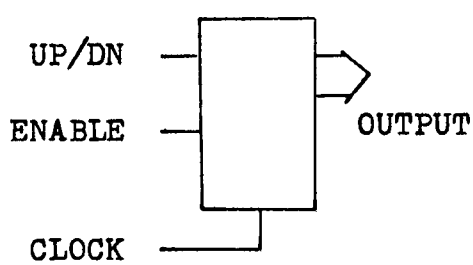
UP	RESET	OUTPUT
0	0	No Change
0	1	Reset
1	0	Count Up 1
1	1	Reset

(b) UP/RESET COUNTER



UP	DN	OUTPUT
0	0	No Change
0	1	Count Dn 1
1	0	Count Up 1
1	1	No Change

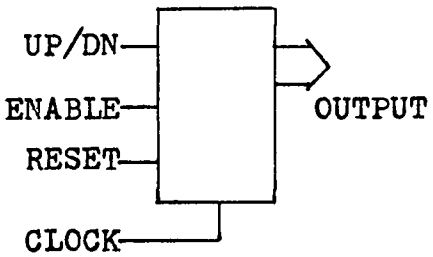
(c) UP/DN COUNTER



UP/DN	ENABLE	OUTPUT
d	0	No Change
0	1	Count Dn 1
1	1	Count Up 1

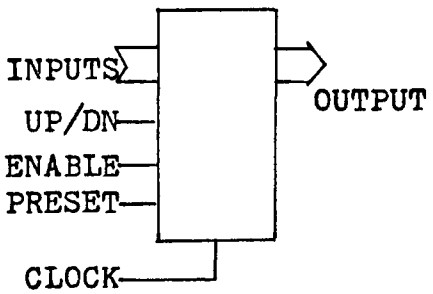
(d) UP/DN COUNTER

Figure 1.2.1. UP, UP/RESET, and UP/DN Counter Features



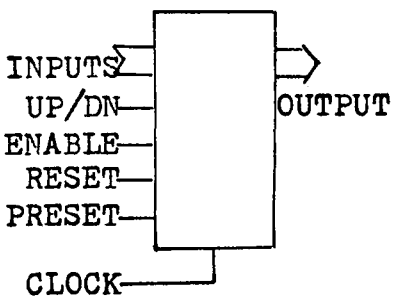
UP/DN	EN.	RE.	OUTPUT
d	d	1	Reset to 00
d	0	0	No Change
0	1	0	Count Dn 1
1	1	0	Count Up 1

(a) UP/DN/RESET COUNTER



UP/DN	EN.	PRE.	OUTPUT
d	d	1	Input Value
d	0	0	No Change
0	1	0	Count Dn 1
1	1	0	Count Up 1

(b) UP/DN/PRESET COUNTER



UP/DN	EN.	RE.	PRE.	OUTPUT
1	1	0	0	Count Up 1
0	1	0	0	Count Dn 1
d	0	0	0	No Change
d	d	1	0	Reset to 0
d	d	d	1	Input Value

(c) UP/DN/RESET/PRESET COUNTER

Figure 1.2.2. UP/DN/RESET, UP/DN/PRESET and UP/DN/RESET/PRESET Counter Features

2. NON-PRESETTABLE COUNTERS

2.1 Introduction

This chapter considers those counters discussed in Chapter 1 which do not have the PRESET feature. Section 2.2 introduces the concept of "sort of cares" used in this thesis. Sections 2.3 - 2.6 discuss the types of machines in which each of the counters can be used as memory devices. Some design procedures to follow when using these counters are also presented. Section 2.7 is a practical design example to illustrate the methods discussed in the preceding sections.

The determination of whether a machine can be realized using a certain counter can be made from an inspection of the state diagram of that machine. One requirement common to all the counters in this chapter is that there exist one or more transition sequences from the initial state to every other state in the machine.

Some definitions will be useful before starting the next sections. A successor state is the resulting state after a transition from the present state. A predecessor state is a state from which there is a transition to the present state. A self loop is a transition from a state to itself. A loop is a transition sequence that will start and end at the initial state and pass once through all other states. A machine is hamiltonian cyclic, if ignoring

"arrows" on the state diagram, there is a path that will start and end at the initial state and pass once through all other states (2).

The following design procedures do not consider the possibility of state splitting. Modifying the state diagram through the appropriate use of state splitting techniques may, in some cases, enable an otherwise unrealizable machine to be realized by a non-presettable counter. However, the restrictions that non-presettable counters place on the number of successor and predecessor states per state will severely limit the application of this technique to a few special cases.

2.2 Sort-of-Cares

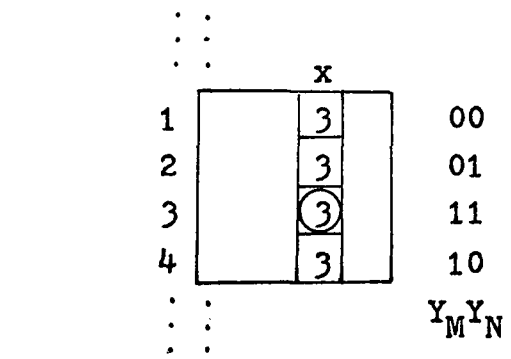
In the sections on design procedures which follow, a set of special conditions occur which we will refer to as "sort-of-cares". These "sort-of-cares" are somewhat unusual in sequential logic design, but are not entirely unique. A similar situation exists in asynchronous, (no clock) design problems in cases where cycles occur. We shall take a brief look at the asynchronous design problem.

Consider figure 2.2.1(a)in which a portion of a state table is shown. For the particular input combination x, there are transitions from states 1, 2 and 4 to stable state 3. Suppose the state codes of these four states differ only by the two state variables Y_M and Y_N as shown to the right of the table. Since this is an asyn-

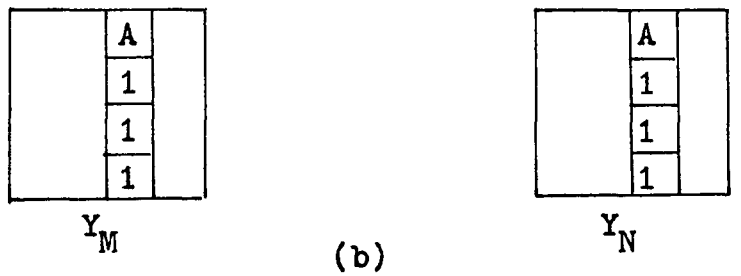
chronous machine, there are several possible transition combinations which could occur. These combinations are illustrated in figure 2.2.1 (b), (c) and (d). In figure 2.2.1 (b), suppose states 2 and 4 make transitions directly to state 3 as shown by "1"'s in both the Y_M and Y_N maps for states 2 and 4. Since state 3 is the stable state it obviously must have "1"'s in both Y_M and Y_N. With these conditions, the possible transitions from state 1 could be to states 2, or 4, or directly to state 3 (a non-critical race). The A is a "sort-of-care" condition in which one of the A's is a "1" if the other A is "0" and "don't care" if the other A is "1". In other words, one of the two A's must be "1", the other being "don't care".

In figure 2.2.1 (c), suppose the transition from state 2 is directly to state 3. Next the transition from state 1 is determined. Depending on the transition from state 1, there are several possible transitions from state 4. If the transition from state 1 is to state 2 or 3, then possible transitions from state 4 are to states 1, 2 or 3. If the transition from state 1 is to state 4, then the possible transitions from state 4 are to states 2 or 3. The B indicates this type of "sort-of-care" which depends on the values assigned to other "sort-of-cares".

In a similar manner, if the transition from state 4 is directly to state 3, then there are various transitions from state 2 depending on the transition from state 1. How



(a)



(b)

(c)

(d)

Figure 2.2.1 Sort-of-Care Situation

can these variables be assigned so as to minimize logic costs? The efficient solution, other than exhaustion, is an open problem.

2.3 UP Counter

An UP counter can be used as the memory in a machine which has the following features: (1.) all states must have at most one successor state; (2.) only the initial state must have a successor state; (3.) the initial state must have at most one predecessor state; (4.) all other states must have one and only one predecessor state; (5.) any state may have self-loops; (6.) if there is a loop of n states, then a counter of modulo n must be used. Note that the UP counting function restricts the next state transition from any state to one specific next state (the state whose state variable's binary value is +1 that of the present state's value, modulo n). Likewise, each state may have only one specific predecessor state (the state whose state variable's binary value is -1 that of the present state's value, modulo n). Self-loops occur when the UP function is inhibited. When the above conditions are met, the machine can be designed using an UP counter as the memory device. For an m state machine, an n bit binary counter ($m \leq 2^n$) must be used. If the machine consists of a loop of m states the counter must be modulo m. We will assume that there is some method of starting the

counter at the initial state (0----0) when the machine is "turned on". There is only one possible state assignment for this type of design. Each successor state starting from the initial state must have a state assignment value +1 that of its predecessor state, (modulo n). Since the only input to this counter other than the clock is the UP function, there is only one K map to determine. The UP function is determined by system inputs and the present state, therefore the K map variables will be the system inputs and state variables. Assuming a "1" corresponds to UP on the counter, insert "1"s in the K map wherever an UP condition occurs. Also add "don't-cares" for any unused states. The K map can now be minimized by any of the usual minimization techniques.

2.4 UP/RESET Counter

This section considers UP/RESET counters in which the RESET function causes a transition to the initial state. This design procedure, with appropriate changes, could be extended to counters whose reset state is different from the initial state.

A sequential machine can be designed using an UP/RESET counter as the memory element if it has the following characteristics: (1.) the initial state must have one and only one successor state; (2.) all other states may have as successor states the initial state and/or one and

only one other state; (3.) except for the initial state, all states must have one and only one predecessor state; (4.) any state may have self-loops. Any machine meeting these conditions may be designed with an UP/RESET counter. Note that a machine consisting of a loop of m states can be realized by any UP/RESET counter of modulo m or greater.

The state assignment, assuming the initial state is (0----0), requires that, except for the transitions to the initial (reset) state, each successor state assignment is +1 that of its predecessor state. Since this counter has two functions, there are two K maps to complete, one for RESET and the other for UP. Recall from Chapter 1 that we assume that RESET is synchronous and has priority over UP. On the RESET K map, each minterm corresponding to a condition where RESET must be used should be made a "1". The corresponding minterms in the UP K-map become "don't-cares". Add "don't cares" in both maps for unused states. Where RESET or UP could be used, place a "*" in both the RESET and UP K maps. This indicates that the minterm in one map must be a "1", and in the other map it becomes a "don't care". For the UP/RESET counter this condition can occur from only one state. However, there may be as many "*"s as there are input combinations. If there are n "*"s, then there are 2^n possible combinations of "*"s as "1"s to try in order to determine the minimal solution.

2.5 UP/DN Counter

The counter being considered is the UP/DN/ENABLE type, with ENABLE having priority. Machines that can be designed using this counter as memory must have the following features: (1.) all states must have no more than two predecessor states; (2.) if a state has two predecessor states, then transitions from this present state must only go to those two states; (3.) if a state has only one predecessor state, then transitions may go to that predecessor state and/or one other state; (4.) only the initial state need not have any predecessor state, if this condition exists, the initial state may have one or two successor states; (5.) all states may have self loops; (6.) if the machine is hamiltonian cyclic, then a modulo m counter is necessary. The preceding restrictions on machine configuration are due to the UP-DN counting function. From any state there are only two possible "counter adjacent" states - those states which have state assignments whose binary value is ±1 that of the present state, modulo n. The ENABLE function permits self-loops to occur in the machine.

A machine that meets the above criteria can be designed using an UP/DN counter as memory by following the design procedure given. Assuming the initial state is (0----0), there are two possible state assignments. If there are two successor states from the initial state, one

must be given the (0----01) state assignment and the other must be given the state assignment whose binary value is one less than the modulo of the counter. From these two states each new state will have a binary value +1 or -1, respectively, of its predecessor state. The other state assignment occurs when the state assignments for the first two states are reversed and the other state assignments determined from them as given above. If there is only one successor state from the initial state then two state assignments result from giving that state either the state assignment (0----01) or one less than the modulo of the counter. For a given state assignment there are two K maps to determine, the UP-DN and the ENABLE. Assume that "1" corresponds to ENABLE, and "1" corresponds to UP. Determine where the counting sequence should be inhibited (at all self-loops). For each minterm with this condition insert a "0" in the ENABLE map and a "don't-care" in the UP-DN map. Add "don't-cares" in both maps for all unused states. The remaining minterms in the ENABLE map must have "1" and the minterms in the UP-DN map must be either "0" or "1" depending on whether the count DN or count UP condition occurs. Logic can now be minimized by the usual methods.

2.6 UP/DN/RESET Counter

This section considers UP/DN/RESET counters in which the RESET function causes a transition to the initial

state. As mentioned for the UP/RESET counter, this design procedure could be extended to counters which reset to states other than the initial state.

The determination of whether a machine can be designed using an UP/DN/RESET counter follows the UP/DN procedure of section 2.5 with these modifications: (1.) remove all transitions into the initial state except for those of the initial states successor states, (2.) ignore condition (6.). If the machine meets these conditions then it may be designed using an UP/DN/RESET counter with this restriction: if the modified machine is hamiltonian cyclic and the initial state has two successor states, then the counter must be of modulo m, where m is the number of states. All other machines may be designed using UP/DN/RESET counters of modulo $\geq$ m.

The state assignment problem is identical to that of the UP/DN counter of section 2.5, assuming again that the initial state variables are (0----0).

For a given state assignment, there are three K maps to determine, one each for RESET, ENABLE, and UP-DN. Start by inserting "1"s in the minterms of the RESET map wherever RESET must be used. The corresponding minterms in the ENABLE and UP-DN maps become "don't-cares". Next, add "$_*$" in each RESET minterm where RESET could be used (this is only possible from each of the initial state's predecessor states). The corresponding minterms in the ENABLE map

become "1*" and the UP-DN terms become either "1*" or "0*" depending on whether a count UP or count DN function is required. A "1*" (or "0*") indicates a condition in which the minterm is either a "1" (or "0") if the "*" in the RESET map is not included in a cube, or a "don't-care" if the "*" becomes a "1" in the RESET map. Add "don't-cares" in all three maps for all unused states. Determine where the counting sequence should be inhibited. For each appropriate minterm in the ENABLE map insert a "0" and in the UP-DN map insert a "don't-care". The remaining minterms in the ENABLE map must have "1" and the minterms in the UP-DN map must be either "0" or "1" depending on whether the count DN or count UP condition occurs. If there are n "*"s, then there are 2^n possible combinations of "*"s as "1"s to observe to determine the minimal logic solution.

2.7 Practical Design Example Using UP/DN/RESET Counter

The following design example was the starting point for the author's study of the use of counters as memory elements.

Figure 2.7.1 is the state diagram and state table for a portion of a line circuit used in an electronic telephone system. The line circuit acts as an interface between the phone and the originating and terminating sides of the switching network. It uses input information from several sources to control the operating sequence of the phone.

Present State	INPUTS					
	00	01	11	10	H	P
0	0	1	1	7	-	-
1	0	1	2	2	-	-
2	0	0	3	3	-	-
3	0	0	3	4	-	-
4	0	0	4	4	-	5
5	0	0	5	5	-	6
6	0	0	6	6	-	-
7	0	0	7	7	6	-

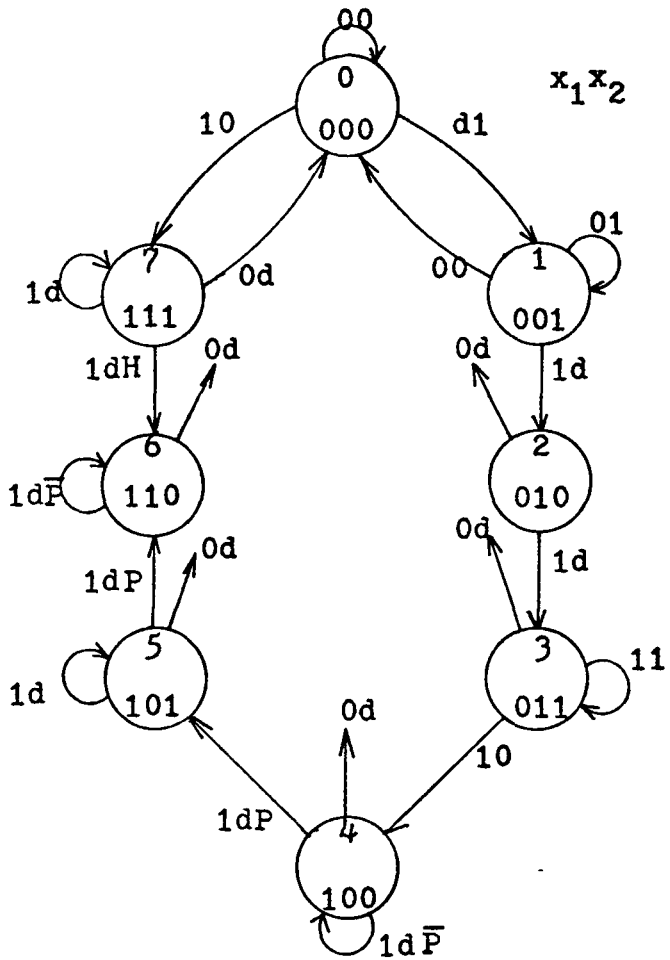


Figure 2.7.1. Line Circuit State Table and State Diagram

Four input variables are shown on the state diagram: x_1, x_2, H and P. The only states from which P causes a transition are 4 and 5. State 7 is the only state in which H can cause a transition. The K maps in figure 2.7.2 include H and P only where they have an effect on the state transitions. It is hoped that the significance of this actual design example will outweigh the slight confusion added by the inclusion of these two extra inputs.

The counter used in this design is a 74191 UP/DN/PRESET counter (11). The PRESET is only used as a RESET in this example. The PRESET on the 74191 is asynchronous, but was made synchronous by ANDING the PRESET input with the clock. A "1" on ENABLE inhibits the counter, therefore the $\overline{\text{ENABLE}}$ K map was determined. A "0" on PRESET (RESET) will reset the counter, therefore the $\overline{\text{RESET}}$ K map was determined. Also, for this counter, a "1" on the UP-DN function causes a down count, therefore where section 2.6 relates "1" to UP and "0" to DN this example does just the opposite. With these changes in mind, the design procedure of section 2.6 can be used to form the K maps of figure 2.7.2.

Note that all of the "* sort-of-cares" in $\overline{\text{RESET}}$ were set to "1" to obtain a minimal logic realization. Note also that this caused the "1*"s and "0*"s in the $\overline{\text{ENABLE}}$ and UP-DN maps to become "don't-cares".

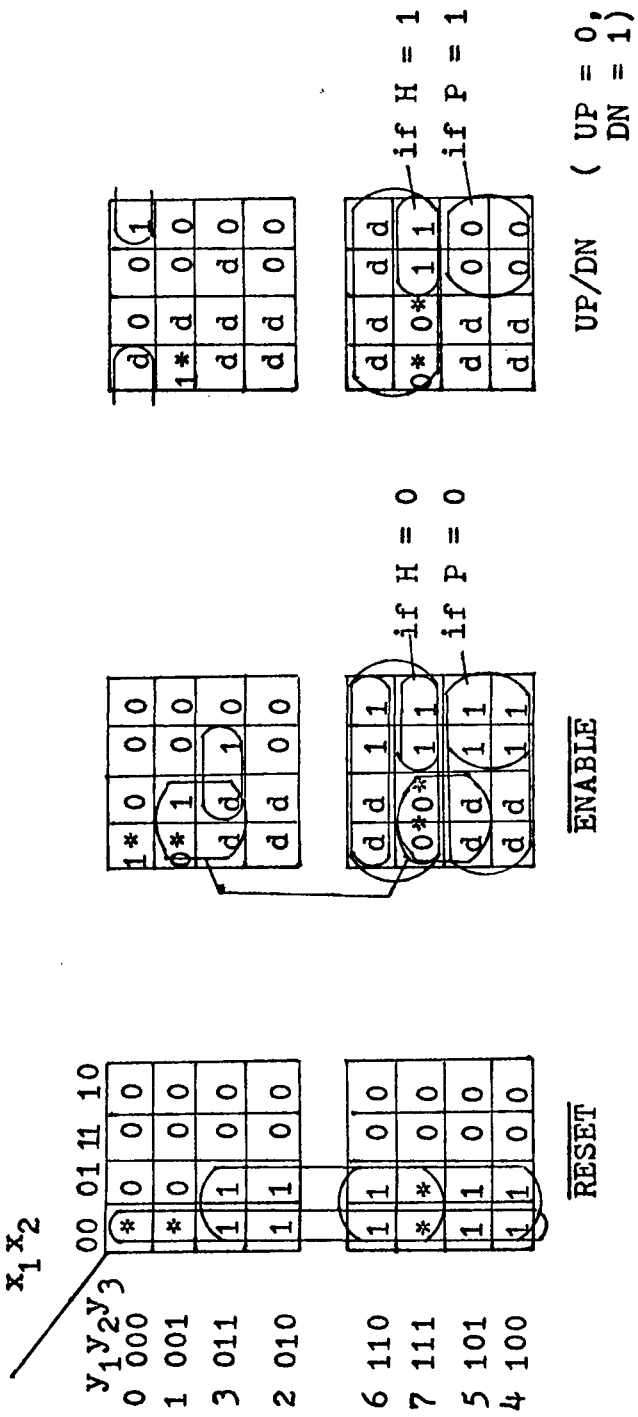


Figure 2.7.2. Line Circuit K Maps

The resulting equations are:

$$\overline{\text{RESET}} = \bar{x}_1 y_1 + \bar{x}_1 y_2 + \bar{x}_1 \bar{x}_2$$

$$\overline{\text{ENABLE}} = \bar{x}_1 y_3 + x_2 \bar{y}_1 y_2 y_3 + y_1 y_2 \bar{y}_3 + \bar{H} y_1 y_2 + \bar{P} y_1 \bar{y}_2$$

$$\text{UP-DN} = \bar{x}_2 \bar{y}_1 \bar{y}_2 \bar{y}_3 + y_1 y_2 .$$

In order to compare this realization with various D and JK realizations we will use "chip count" as a relative measurement. The function value is based on the number of functions per chip in the 74 series of TTL logic as defined in table 2.7.1.

The minimal cost using the counter was 6 3/4 chips. For the given state assignment the JK realization had a chip cost of 8 2/3 while the D realization cost 8 chips.

This example demonstrates not only the ability of an UP/DN/RESET counter to realize this machine, but also, for the given state assignment, the ability of the counter to achieve a better total cost than either JK or D flip flops.

There is no guarantee that this is the lowest cost realization. In fact, if output functions are ignored, this example has a JK realization which for another state assignment has a lower logic cost than the three realizations considered in the example.

Table 2.7.1

Chip Cost

FUNCTION	COST
7 Input and Gate	1
6 Input and Gate	1/2
5 Input and Gate	1/2
4 Input and Gate	1/2
3 Input and Gate	1/3
2 Input and Gate	1/4
7 Input or Gate	1
6 Input or Gate	1/2
5 Input or Gate	1/2
4 Input or Gate	1/2
3 Input or Gate	1/3
2 Input or Gate	1/4
Inverter	1/6
D Flip Flop (Q Output only)	1/4
JK Flip Flop	1/2
Counter	1

The cost indicates "real estate" or space costs only. We assume that space cost outweighs all other cost factors.

3. PRESETTABLE COUNTERS

3.1 Introduction

The previous chapter has shown that non-presettable counters are able to realize only a limited class of sequential machines. By adding the PRESET function to any one of the previously discussed counters, it is possible to effect a transition from any one of the counter's states to any other state. PRESET, therefore, will allow us to realize any finite sequential machine (given an n bit counter, where $2^n \geq$ the number of states of the machine) using only the counter for memory. While this fact may or may not appear obvious to the reader it is in fact the heart of this thesis.

Many switching theory textbooks discuss the design of counters from BSD's and control logic to realize certain machines (6, 7, 9), O'Keefe (8) has demonstrated the use of UP/DN counters as memory modules for certain machines, but nowhere (to the author's knowledge) has a counter been proposed as a universal memory module, needing only sufficient control logic to realize any given machine.

Note that to realize any machine, the counting functions could be omitted and only the PRESET function used. When only PRESET is used the counter becomes equivalent to n unconnected D flip flops. An upper bound on the logic cost using a counter is therefore the same as

that of D flip flops having only Q outputs.

3.2 State Assignment

The problem of assigning a unique combination of state variables to each state in a machine in such a way that the resulting logic cost is minimal is finite, but one for which the only known solution is exhaustion. With

$$\frac{2^r!}{(2^r-m)!} \qquad (2^{r-1}<m\leq 2^r) \tag{3.2.1}$$

possible ways to assign the 2^r combinations of r state variables to the m states (3), the practicality of this method is limited.

One method of finding "good" state assignments for D flip flops (and also JK flip flops) is Armstrong's method, which attempts to maximize the clustering of 1-points and 0-points on a K map (1). In this method, an attempt is made to give adjacent codes (codes which differ by only one state variable) to two states which for a particular input combination: have either common successor states or common predecessor states; or are each the predecessor of the other; or both have self loops. While this method would find state assignments which could be implemented using a presettable counter, the application of this method is questioned. For instance, if for a particular input combination, two states have the same successor state, then Armstrong's method would tend to give these states adjacent

codes. To make better use of the counting feature, however, it would be desirable to give the states codes in which the two predecessor states have codes with binary values ±1 of the successor state's code. In general, Armstrong's method tends to limit the potential of the counter.

Another method of determining "good" state assignments when designing with counters is proposed here. This method will assign to the states those codes which will maximize the number of next state transitions where the counting feature could be used. Maximizing the number of counting transitions will also maximize the number of "don't-cares" and the various "sort-of-cares" in the K maps. Note from section 3.3 that for every minterm where a possible counting transition exists, there are "sort-of-cares" in every map except the output maps. It is certainly possible to contrive examples where this method will yield poor results, even when disregarding the output functions. However, it is felt that in cases where a machine is well suited to design with a counter this method is likely to find good state assignments.

A determination of likely counting chains can be made from an inspection of the state diagram. A counting chain is a transition sequence in which none of the states are repeated except possibly for the same first and last state.

Keep in mind that the longest chain does not necessarily have the largest number of possible counting transitions. Several of these longer chains should then be examined in detail in order to determine the best of these solutions.

3.3 Design Procedure For UP/DN/PRESET

For a given state assignment the design procedure is as follows. The K maps to be determined are PRESET, Y_1-Y_N input variables, ENABLE, UP-DN, and Z_1-Z_M output functions. Each map uses the inputs and state variables as its variables.

Begin the design by determining where PRESET must be used. Appropriate PRESET minterms become "1"s, corresponding minterms in the Y K maps take on the values of the next state variables, and corresponding minterms in ENABLE and UP-DN become "don't-cares". Next, add "don't-cares" in all maps for all unused states. The remaining minterms in the PRESET map become "*"s. The corresponding minterms in the Y K maps become either "⓪" or "①" depending on whether the next state variable's value is "0" or "1", respectively. "⓪" (or "①") indicates that the minterm is a "don't-care" if the "*" in the PRESET map is not included in a cube and a "0" (or "1") if the "*" becomes a "1". The corresponding minterms in the ENABLE map become either "0*" or "1*" depending on whether or not the counting function must be inhibited. The corresponding

minterms in the UP-DN map become "don't cares" where the ENABLE minterms are "0*"s and either "0*"s or "1*"s for DN count or Up count. "0*" (or "1*") indicates that a minterm is a "don't-care" if the "$_*$" in the PRESET map is included in a cube and a "0" (or "1") otherwise. As in the design procedures of Chapter 2, if there are n "$_*$"s in the PRESET map, then there are 2^n possible combinations of these "*"s as "1"s. In order to determine the minimal logic for a given state assignment, each one of these would have to be investigated. This is a finite problem and therefore solvable. A brute force approach, however, would involve a large amount of time and work even for a relatively small number of "$_*$"s. Unfortunately, this author has no better solution at this time other than heuristicly choosing a "$_*$", making it a "1" and then determining whether the net effect on all the K maps is a lower or higher logic cost.

Note that once the "$_*$"s in the PRESET map have been set to either "0" or "1" all the "sort-of-cares" ("0*", "⓪", "1*", "①") become either "0", "1", or "don't-care" and the maps can be minimized using standard techniques.

Keeping the following relationships in mind may be useful in determining a least total cost solution:

RULE 1 The most "don't-cares" occur in the Y maps when all "*"s in PRESET are "0"s. Therefore logic cost

for the Y inputs can only rise as more and more "$_*$"s are set to "1"s.

RULE 2 The most "don't-cares" occur in the ENABLE and UP-DN maps when all "$_*$"s in PRESET are "1"s. Therefore logic cost for ENABLE and UP-DN functions can only decrease as more "$_*$"s are set to "1"s.

RULE 3 Depending on the particular PRESET K map, setting a "$_*$" to a "1" may increase or decrease the logic cost of the PRESET FUNCTION.

RULE 4 Setting the PRESET "$_*$"s to "1"s has no effect on the output functions nor on their cost except for the possibility of gate sharing with other K maps.

RULE 5 Gate sharing between all K maps should also be considered in minimizing total cost.

The PRESET K map is the pivot point in this minimization problem. Minimal logic cost may occur by using PRESET as little as possible, as much as possible (see section 3.6), or somewhere in between. As already stated, the efficient determination of this point is an open and challenging problem.

3.4 Design Procedures for UP/PRESET UP/RESET/PRESET and UP/DN/RESET/PRESET

The design procedures discussed in the previous section for UP/DN/PRESET counters can be modified for UP/PRESET, UP/RESET/PRESET and UP/DN/RESET/PRESET. The

same "sort-of-care" conditions occur in each of these counters and therefore we will not go into the same detail as the previous section. Symbols used in section 3.3 have the same meaning in this section.

The UP/PRESET design does not have either the ENABLE or UP-DN maps of the UP/DN/PRESET design. Instead it has an UP map. The PRESET map is formed with "1"s where PRESET must be used and "*"s where the UP function (including inhibit) could be used. The Y maps are formed exactly as in section 3.3. "Don't-cares" are inserted in the UP map wherever PRESET must be used and either "1*"s or "0*"s elsewhere corresponding to UP and $\overline{UP}$ (inhibit) respectively.

UP/RESET/PRESET design has the PRESET, and Y_{1-N} input maps of the UP/DN/PRESET design. In addition it has a RESET map and an UP map. "1"s are inserted in the PRESET map where PRESET must be used, and "*"s where RESET or UP might be used. Y maps are formed as in section 3.3. "Don't-cares" are inserted in the RESET and UP maps wherever PRESET must be used. Insert "1*"s in the RESET map and "don't-cares" in the UP map for conditions where RESET could be used but not UP. The minterms for which UP but not RESET could be used are set to "0*" for RESET and either "1*" or or "0*" for UP. The remaining minterms represent situations where either RESET or UP could be used. Insert "1*" in

those RESET minterms and "1**" in those UP minterms. "1**" indicates a "don't-care" condition if either the PRESET or RESET minterms are set to "1" and a "don't-care" otherwise. The UP/DN/RESET/PRESET design adds the RESET K map to the UP/DN/PRESET design procedure. The PRESET minterms are set to "1"s where PRESET must be used and "$_{*}$"s where either RESET, ENABLE or UP-DN could be used. Y maps are formed as in section 3.3. Where RESET but not UP-DN could be used, insert "1*"s in the RESET map and "don't-cares" in the ENABLE and UP-DN maps. Where ENABLE and UP-DN but not RESET could be used, insert "0*"s in the RESET map and the appropriate symbols in the Enable and UP-DN maps as described in section 3.3. Where either RESET or UP-DN could be used, insert "1*"s in the RESET map, "1**"s in the ENABLE map and either "0**"s or "1**"s in the UP-DN map corresponding to possible DN or UP counts respectively. "0**" is defined as "1**"with "1" replaced by "0".

3.5 Example Using An UP/DN/PRESET Counter

The intent of this example is to illustrate the design procedure in section 3.3 and also to show, for a given state assignment, the ability of the counter to achieve a better logic cost than either JK or D flip flop designs.

Figure 3.5.1 shows the state diagram of the machine under study. For this example the output functions will be neglected. The state assignment chosen uses as many count-

ing transitions as possible. Note that it is rather unlikely that this state assignment would be chosen for a JK or D design.

Figure 3.5.2 shows the K maps necessary to realize this machine design with the counter. The entries in the maps were determined by using the design procedures of section 3.3. Logic minimization was performed as outlined in section 3.3. Observe that in this example PRESET was used only where necessary with the UP-DN feature involved in most of the transitions. Also note the use of POS terms. The resulting equations, listed below, had a chip cost of 2 2/3 chips as determined from table 2.7.1. Including the cost of the counter, the total logic cost is 3 2/3.

$$\begin{aligned}
\text{PRESET} &= \bar{y}_1\bar{y}_3x + y_1y_3x \\
Y_1 &= \bar{y}_3 \\
Y_2 &= \bar{y}_3 \\
Y_3 &= y_3 \\
\text{ENABLE} &= (y_1 + y_2 + \bar{x}) \cdot (y_1 + y_2 + x) \\
\text{UP-DN} &= (y_1 + y_3 + x)
\end{aligned} \qquad (3.5.1)$$

Using the same state assignment, the total logic costs for JK realization were 4 11/12 and for D realization the cost was 6 5/12 chips. The precise chip value is unimportant. There may very well be JK realizations using other state assignments which would realize this machine at a lower total cost.

Present State	I	
	0	1
0	1	6
1	2	1
2	3	6
3	4	4
4	3	5
5	6	1
6	5	1
7	x	x

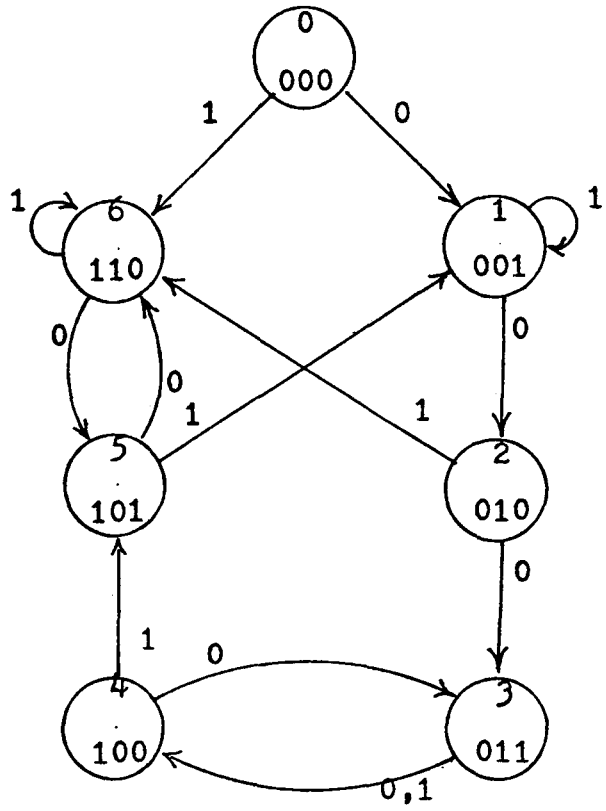


Figure 3.5.1. State Table and State Diagram For Example 3.5

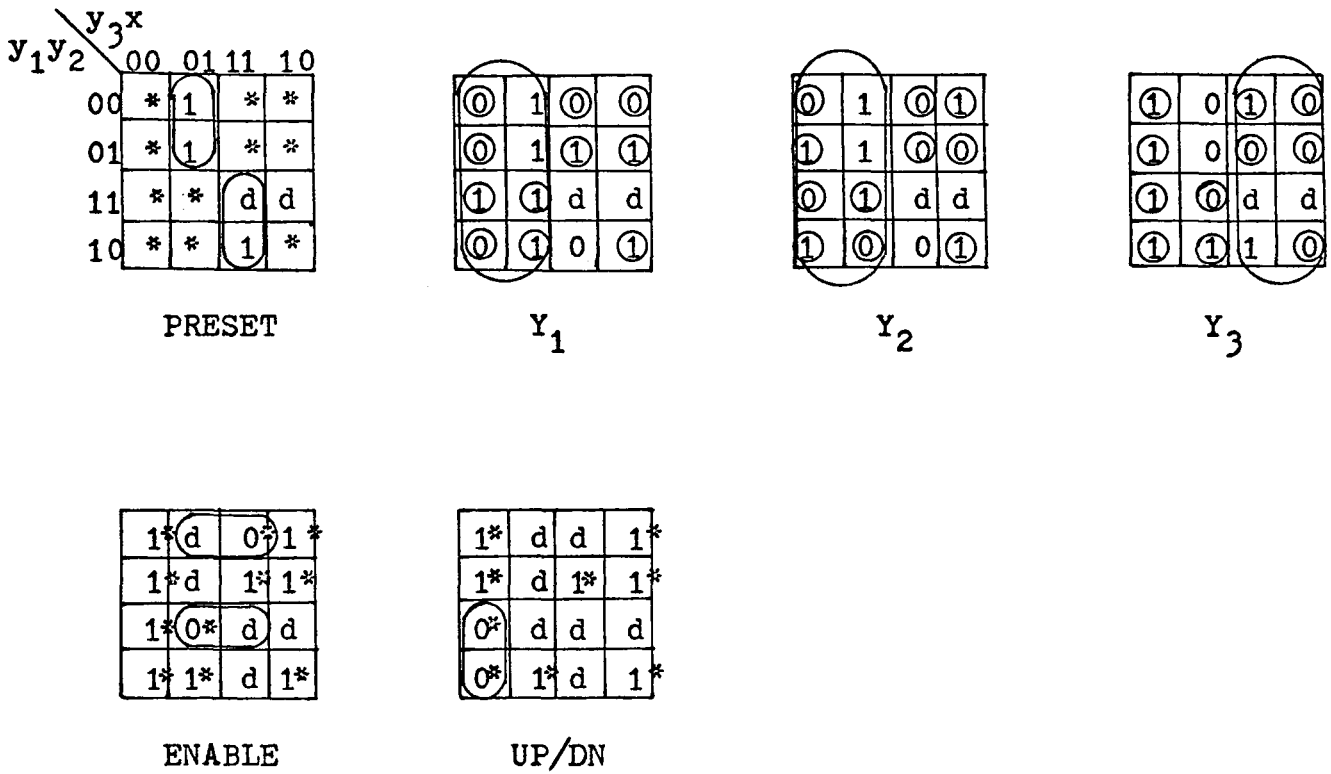


Figure 3.5.2. K Maps For Example 3.5

3.6 Shift Register Example

In this example we will construct a shift register from an UP/DN/PRESET counter. The state table and state diagram are shown in figure 3.6.1. Note that in this example the state assignment has been chosen to match the binary value in the shift register. That this was a wise choice will become apparent when the K maps are determined.

Observe the K maps in figure 3.6.2. An important observation to make is that the minimal logic costs for all of the Y input functions are independent of the PRESET terms. Therefore we choose to set all "*"s and "don't-cares" to "1"s in the PRESET map. The resulting equations are:

PRESET = 1

$Y_1 = y_2$

$Y_2 = y_3$

$Y_3 = x$

ENABLE = d

UP-DN = d

$Z = y_1$

This design amounts to using the counter as a set of unconnected D flip flops. It is obvious that this realization is minimal.

Present State	I				
	0	1	Y_1	Y_2	Y_3
0	0	1	0	0	0
1	2	3	0	0	1
2	4	5	0	1	0
3	6	7	0	1	1
4	0	1	1	0	0
5	2	3	1	0	1
6	4	5	1	1	0
7	6	7	1	1	1

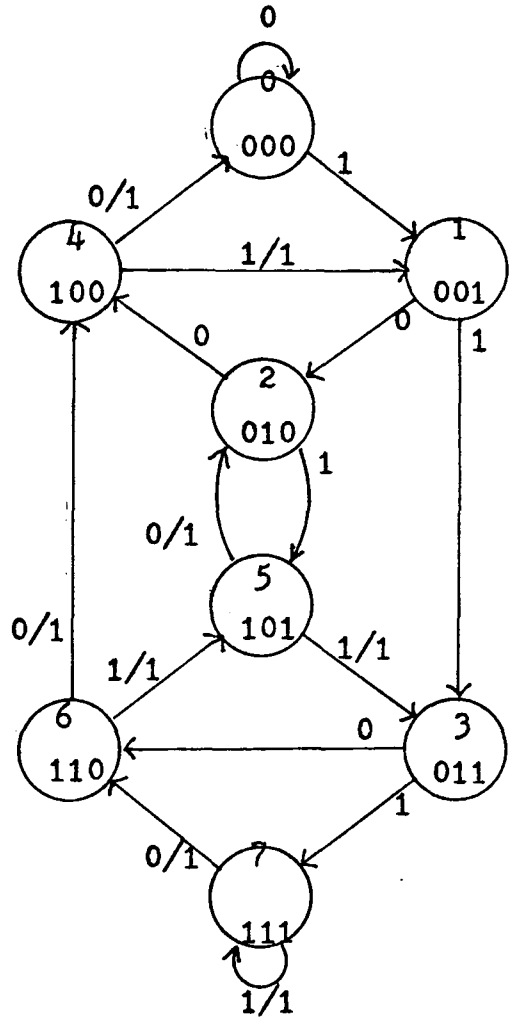


Figure 3.6.1. Shift Register State Table and State Diagram

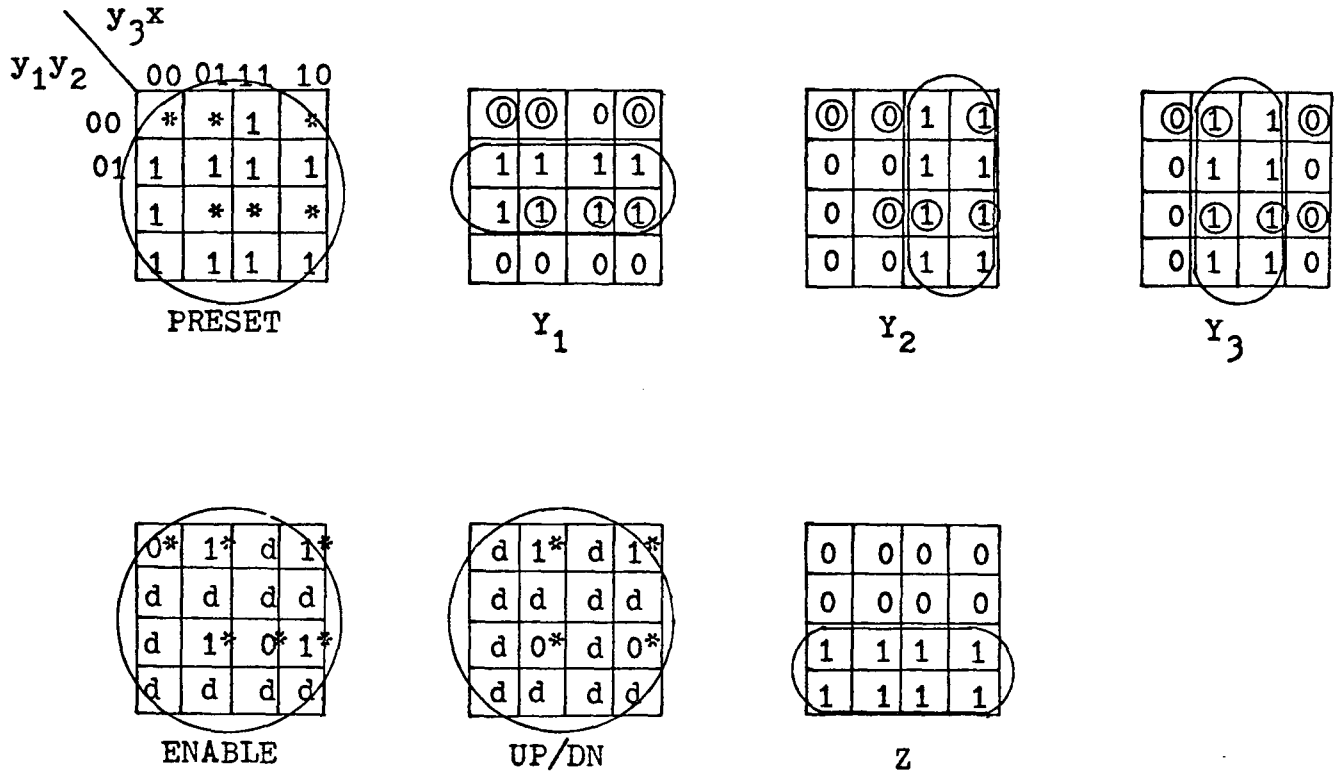


Figure 3.6.2. Shift Register K Maps

4. EXTENSIONS AND CONCLUSIONS

4.1 Extensions

The preceding chapters have described in some detail the abilities and limitations involved when using any one of the several counters in realizing a sequential machine. No mention has been made in this thesis of the possibility of using a counter and flip flops or a counter and shift registers or several counters as memory elements of a sequential machine. Some work has been done in this area such as Roome's multiple shift register realizations (10) and the use of several UP/DN counters as discussed by O'Keefe (8). The extent of possibilities in this area remains relatively unexplored, however. The intention of this section is merely to point out this large area of unsolved design problems.

Two machine structures in particular seem to stand out as being particularly suited to design with two or more counters. We shall describe these briefly and in very general terms and without attempting to make any formal or rigorous definitions. Consider first a machine consisting of a number of groups of states where for each group of states there are many transitions to other states in that group but very few transitions to states in other groups. If the transitions from group to group resemble those of one of the counters described in Chapter 2, the

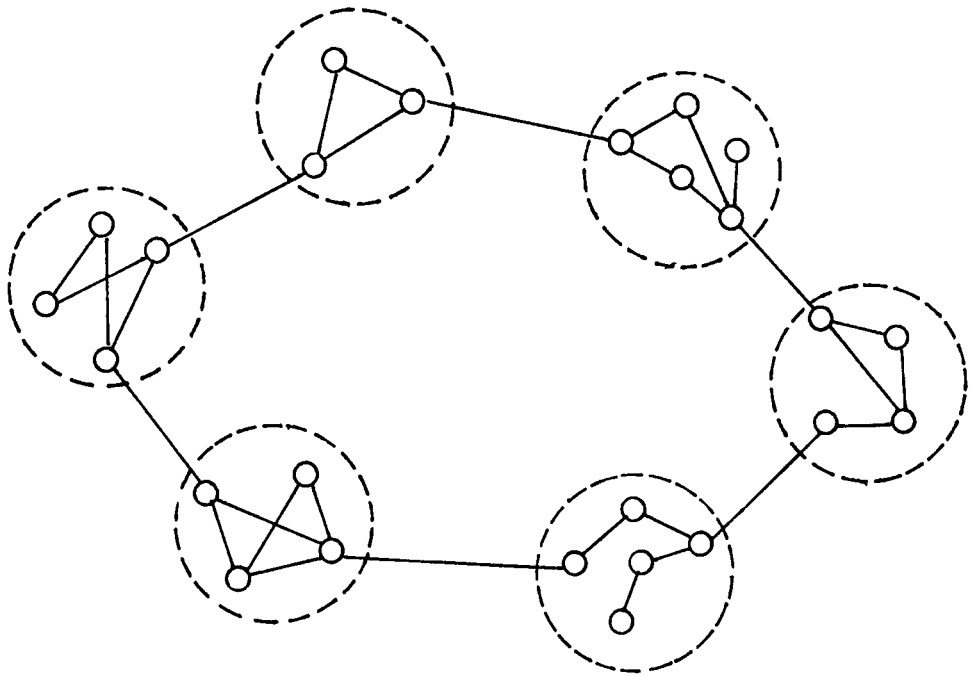
machine could be realized by using a nonpresettable counter to control the transitions from group to group and a presettable counter to control transitions within each group of states. Note that the term group here is not used as defined in abstract algebra. In further distorting the language we might choose to call the above condition a "ring of groups".

The second machine to consider is one in which there are collections of states which have highly counter oriented structures. From each of these collections or "rings" there are transitions to one or more of the other "rings". This situation could be realized using a presettable counter to control transitions from "ring" to "ring" and a nonpresettable counter to control transitions within each "ring". This machine configuration might be called a "group of rings". Figure 4.1.1 depicts both this and the "ring of groups" situation. From these two classes of machines one might consider the possibilities of "rings of rings", "groups of groups" and the many other possible structures.

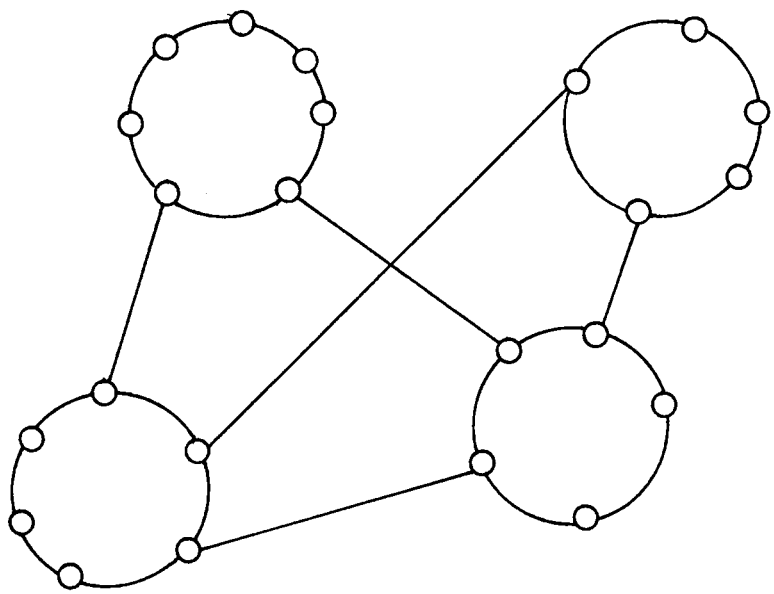
Hopefully these examples give some idea of the directions further study might take.

4.2 Conclusions

This thesis has demonstrated the ability of presettable counters to be used as the memory elements in any sequential machine. It has also demonstrated the useful-



a) Example of a "Ring of Groups" Machine



b) Example of a "Group of Rings" Machine

Figure 4.1.1. Complex Machines

ness of counters in reducing logic costs for some machines and state assignments.

In addition to these two primary objectives, several other contributions, in the form of new problems or extensions of existing problems, have been made. These problems center around the efficient minimization of logic costs when designing machines using counters. State assignment, a problem no matter what design method is used, has additional considerations with the addition of "sort-of-cares" and "counter adjacent" states. Minimization of logic cost for a given machine and state assignment when there are "sort-of-care" entries in Karnaugh maps is still lacking an efficient solution. Beyond these problems involving the use of a single counter there is still the entire area of multiple memory device realization of sequential machines.

BIBLIOGRAPHY

1. D. B. Armstrong, "On the Efficient Assignment of Internal Codes to Sequential Machines," IRE Transactions on Electrical Computers, Vol. EC-11, No. 5, Oct. 1962.

2. M. Behzad and G. Chartrand, Introduction to the Theory of Graphs, Allyn and Bacon, 1971.

3. F. J. Hill, and G. R. Peterson, Introduction to Switching Theory and Logical Design, Wiley, 1968.

4. E. R. Hnatek, A User's Handbook of Integrated Circuits, Wiley, 1973.

5. R. Y. Kain, "Synthesis of Up-Down Counters," IEEE Transactions on Computers, Vol. EC-16, No. 2, April 1967.

6. Z. Kohavi, Switching and Finite Automata Theory, McGraw Hill, 1970.

7. D. Lewin, Logical Design of Switching Circuits, American Elsevier, 1969.

8. K. H. O'Keefe, "Modularity in Design: Shift Registers and Counters Used as System Building Blocks," IEEE Transactions on Computers, Vol. C-22, No. 2, Feb. 1973.

9. V. T. Rhyne, Fundamentals of Digital Systems Design, Prentice-Hall, 1973.

10. W. D. Roome, and H. C. Torng, "Algorithms for Multiple Shift Register Realizations of Sequential Machines," IEE Transactions on Computers, Vol. C-22, No. 10, Oct. 1973.

11. Digital Integrated Circuits, National Semiconductor, 1973.

The vita has been removed from the scanned document

SEQUENTIAL LOGIC DESIGN USING COUNTERS AS MEMORY ELEMENTS

by

Arthur David Schrank

(ABSTRACT)

This thesis is concerned with the use of memory function devices in place of binary storage devices in sequential machines. In particular, various counters are considered as memory elements. Design limitations and design procedures for each type of counter are determined, with emphasis placed on UP/DN/PRESET type counters. It is shown that a presettable counter is capable of realizing any sequential machine.

Special considerations involved in state assignment and minimization in designs using counters are investigated. Finally, extensions and areas of possible further study are discussed.