State Reduction and State Assignment Techniques

Derek Hildreth and Timothy Price
Brigham Young University - Idaho

Abstract

This article will demonstrate several methods and techniques which will reduce the complexity and cost of designing, modeling, and building finite state machines. State reduction is the elimination of states which are equivalent within the state machine and state assignment is the method of assigning a binary value to a state name that will create a reduced logic equation. The three methods examined for state reduction are (1) row matching method, (2) implication chart method, and (3) successive partitioning method. The four methods examined for state assignment are (1) binary, (2) gray code, (3) one hot, and (4) enumerated.

State Reduction and State Assignment Techniques

Introduction

When faced with a design which requires only a small number of inputs and states, an engineer won't necessarily concern themselves with cost and labor. Here, the simplest of applications of state machine design can be used. However, when faced with a complex system with many inputs and states, an engineer will be required to keep the costs and labor down to a minimum as well as produce a design which is accurate and stable. In the absence of state reduction and state assignment techniques, the engineer would easily be overwhelmed with the complexity of designing and building such a system. In order to cut down on costs and labor, there are several methods they can use in both areas of state reduction and state assignment.

State reduction is the elimination of states which are equivalent within the state machine and state assignment is the method of assigning a binary value to a state name that will create a reduced logic equation. The time most spent in these areas are within the state transition table. When the state transition table has been reduced as much as possible, the next step will be to decided on how to implement the circuit which is most commonly by way of flip-flops. The next step would be to find the logic equations necessary and then finally, construct the circuit. It's within the last few steps where the true fruit of the state reduction and state assignment techniques start to bloom. Generally, state reduction will occur first, and then state assignment following.

The optimal situation would be for example, if a finite state machine drops from 7 states to 4 states and compact state assignments are used, the design drops from three flip-flops to two flip-flops. The sub-optimal situation is when the number of states is reduced, but the number of flip-flops is not (Grover, 2007).

State Reduction

The three main methods of state reduction include: row matching, implication charts, and successive partitioning. Row matching, which is the easiest of the three, works well for state transition tables which have an obvious next state and output equivalences for each of the present states. This method will generally not give the most simplified state machine available, but its ease of use and consistently fair results is a good reason to pursue the method. The implication chart uses a graphical grid to help find any implications or equivalences and is a great systematic approach to reducing state machines. Successive partitioning is almost a cross between row matching and implication chart where both a graphical table and equivalent matching is used. Each of these methods will, in most cases, reduce a state machine into a smaller number of states. Keep in mind that one method may result in a simpler state machine than another. Experience will help in understanding and determining the best method to use in any particular situation.

Row Matching

The row matching method, as previously described, is the quite possibly the simplest of the three methods. It uses the state equivalence theorem: Si = Sj if and only if for every single input X, the outputs are the same and the next states are equivalent (Grover, 2007). All input sequences must be considered, but any information about the internal state of the system can be ignored. When using the theorem, both the output and next state must be considered. However, only single inputs rather than input sequences are considered. The typical recipe for reducing in the row reduction method is (Katz, 1993):

1. Start with the state transition table
2. Identify states with same output behavior
3. If such states transition to the same next state, they are equivalent
4. Combine into a single new renamed state
5. Repeat until no new states are combined

An example can be seen by observing Table 1 below:

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S7	S8	0	0
01	S4	S9	S10	0	0
10	S5	S11	S12	0	0
11	S6	S13	S14	0	0
000	S7	S0	S0	0	0
001	S8	S0	S0	0	0
010	S9	S0	S0	0	0
011	S10	S0	S0	1	0
100	S11	S0	S0	0	0
101	S12	S0	S0	1	0
110	S13	S0	S0	0	0
111	S14	S0	S0	0	0

Table 1: Initial state transition table before any reduction.

First, notice that S10 and S12 have the same next states as well as the same outputs. These two states can be combined and renamed into S10' (the tick mark ' signifies that this state has been merged with another and is considered a new, reduced state). Because S12 no longer exists, all instances of S12 will be replaced with S10'. This will result in Table 2 below:

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S7	S8	0	0
01	S4	S9	S10'	0	0
10	S5	S11	S10'	0	0
11	S6	S13	S14	0	0
000	S7	S0	S0	0	0
001	S8	S0	S0	0	0
010	S9	S0	S0	0	0
011 OR 101	S10'	S0	S0	1	0
100	S11	S0	S0	0	0
110	S13	S0	S0	0	0
111	S14	S0	S0	0	0

Table 2: S10 and S12 were combined into a single state S10'.

This process is continued until no new states can be combined. The resulting table can be seen below in Table 3:

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3'	S4'	0	0
1	S2	S4'	S3'	0	0
00 or 11	S3'	S7'	S7'	0	0
01 or 10	S4'	S7'	S7'	0	0
Not (011 or 101)	S7'	S0	S0	0	0
011 OR 101	S10'	S0	S0	1	0

Table 3: The final reduced state transition table.

In this particular example, the number of states was reduced from fifteen (15) to seven (7) states. This literally amounts to a reduction of eight (8) states and the elimination of one flip-flop for the final design.

Implication Chart Method

The implication chart method uses a graphical grid of sorts to systematically find equivalences among the states. The implication chart assists in keeping track of any implications such as, for example, c-d and e-f are implied pairs for a-b in Table 4 below:

Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
a	c	f	0	0
b	d	e	0	0
c	h	g	0	0
d	b	g	0	0
e	e	b	0	1
f	f	a	0	1
g	c	g	0	1
h	e	f	0	0

Table 4: A state transition table to demonstrate implications.

Focusing attention towards a different example, the implication chart method will be demonstrated in more detail. The steps that need to be taken in this method can be summed up into a recipe (Grover, 2007):

1. Construct implication chart, one square for each combination of states taken two at a time
2. Square labeled Si, Sj, if outputs differ then the square gets an 'X'. Otherwise, write down implied state pairs for all input combinations
3. Advance through chart top-to-bottom and left-to-right. If square Si, Sj contains next state pair Sm, Sn and that pair labels a square already labeled 'X' then Si, Sj is labeled 'X'
4. Continue executing Step 3 until no new squares are marked with 'X'
5. For each remaining unmarked square Si, Sj, then Si and Sj are equivalent

Consider the following transition table:

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

Table 5: Initial state transition talbe for implication chart method reduction.

The first step is to construct an implication chart which can be seen in Table 6 below:

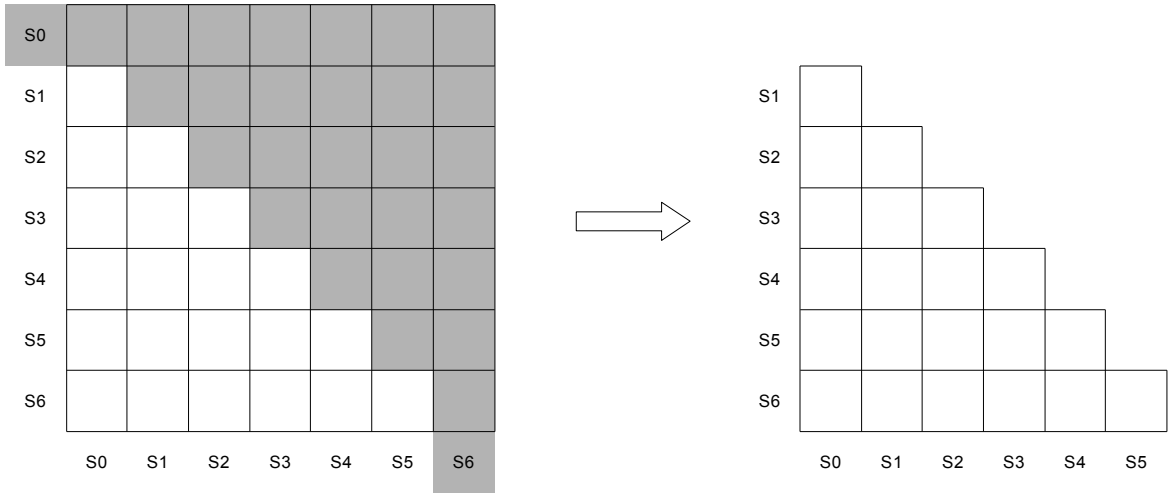


Table 6: Building the implication chart for state transition table.

Construction of the implication chart also includes filling in the values of the state transition table as seen in Table 7 below. Notice the 'X's where the outputs of the compared states are different, this is done in step 2. In step 3, which is demonstrated in Table 8, will further eliminate any states which cannot be implied.

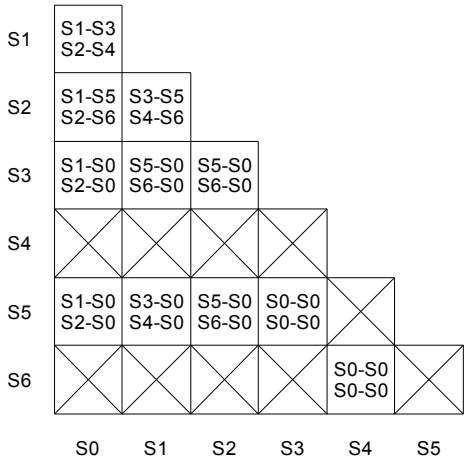


Table 7: Constructing and completing the implication chart.

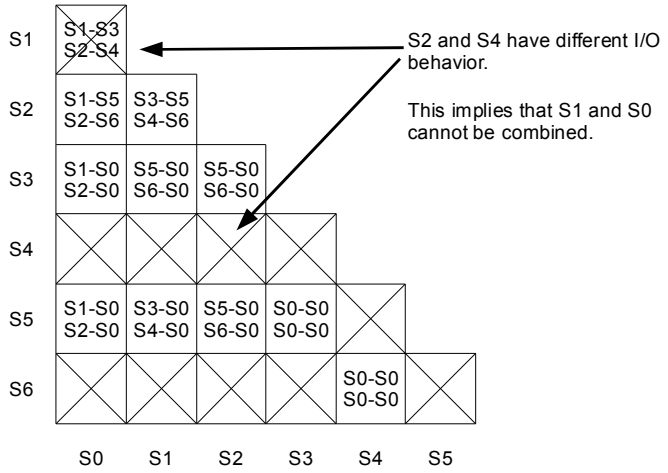


Table 8: A second pass which eliminates any additional states which cannot be implied.

Step 4 is to repeat the processes of step 3 until no more comparisons can be made. The final result is what is left over after this step is completed. The final implication chart and transition table can be seen below in Tables 9 and 10.

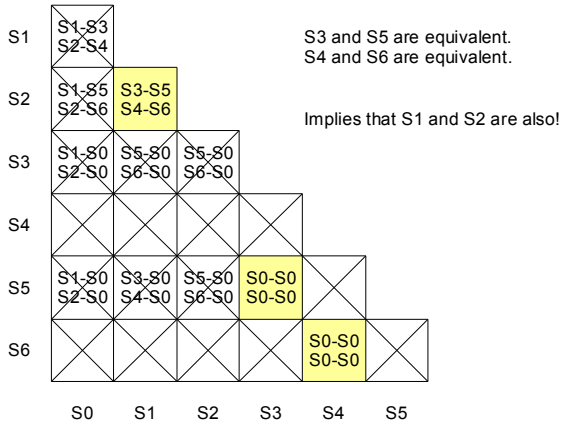


Table 9: Final implication table after all steps have been completed.

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1'	S1'	0	0
0 or 1	S1'	S3'	S4'	0	0
00 or 10	S3'	S0	S0	0	0
01 or 11	S4'	S0	S0	1	0

Table 10: Final state transition table.

In this particular example, the implication chart method simplified the state transition table from seven (7) states down to five (5) states.

Partitioning Method

As mentioned before, the partitioning method is a sort of hybrid between row matching and implication chart in that it uses a visual detection for equivalences as well as a chart to organize the process. Partitioning provides a straightforward procedure for determining equivalency for any amount of complexity. Successive partitioning steps produce smaller partitions. If the next step does not yield any smaller partitions no further steps will yield any smaller partitions and, hence, the partitioning process is then complete. All states that are in the same partition after k steps are k equivalent. All states that are in the same partition when no further partitioning can be accomplished are equivalent. States that are not in the same final partition are not equivalent (Whitaker, 2005).

The following state transition table example will be used to demonstrate the partitioning method:

	NS				Z			
PS	00	01	10	11	00	01	10	11
A	A	H	G	G	1	0	1	0
B	C	H	G	F	1	0	1	0
C	A	H	G	A	1	0	1	0
D	D	D	F	F	0	0	0	0
E	H	H	F	E	1	0	0	0
F	H	H	E	F	0	0	0	0
G	A	H	A	A	1	0	1	0
H	H	G	H	G	1	0	1	0

Table 11: Sample transition table

	00	01	10	11
A	A/1	H/0	G/1	G/0
B	C/1	H/0	G/1	F/0
C	A/1	H/0	G/1	A/0
D	D/0	D/0	F/0	F/0
E	H/1	H/0	F/0	E/0
F	H/0	H/0	E/0	F/0
G	A/1	H/0	A/1	A/0
H	H/1	G/0	H/1	G/0

Table 12: Machine table.

In order to start this reduction method, it is beneficial to convert the transition table into a machine table which will make it easier to transfer to the partition table. This is an optional step, but it will save a little time in the long run. Refer to Table 12. The rows of this machine table are the state names from the transition table. The columns are the next state conditions. The values within the cells are in the form next state/output. The next step is to transfer the information into the partitioning table as seen in Table 13 below:

P0	A	B	C	D	E	F	G	H
x1,x2	OUTPUTS							
00	1	1	1	0	1	0	1	1
01	0	0	0	0	0	0	0	0
10	1	1	1	0	0	0	1	1
11	0	1	1	0	0	0	1	1
P1								

Table 13: Partitioning table.

As illustrated in Table 15, the next step is to find any of the outputs which are the same, and then partition them off (or group them into a partition) and record these in the P1 row (which basically stands for partition pass 1). The next step is to replace the outputs with the next state names and then determine if any of the states need to be partitioned out. In order to tell this, if states within the column being observed are not within the partition, then it should be partitioned off (as seen Table 14). After repeating this step for all columns and partitions, the final partition table can be seen in Table 17. The composite table with all of these steps combined can be seen in Table 16.

P0	A	B	C	D	E	F	G	H
x1,x2	OUTPUTS							
00	1	1	1	0	1	0	1	1
01	0	0	0	0	0	0	0	0
10	1	1	1	0	0	0	1	1
11	0	0	0	0	0	0	0	0
P1	A	B	C	G	H	D	F	E

Table 15: Partition 1.

P1	A	B	C	G	H	D	F	E
x1,x2								
00	A	C	A	A	H	D	H	
01	H	H	H	H	G	D	H	
10	G	G	G	A	H	F	E	
11	G	F	A	A	G	F	F	
P2	A	C	G	H	B	D	F	E

Table 16: Partition 2.

P1	A	C	G	H	B	D	E	F
x1,x2								
00	A	A	A	H				
01	H	H	H	G				
10	G	G	A	H				
11	G	A	A	G				
P3	A	C	G	H	B	D	E	F

Table 14: Partition 3 (Final).

P0	A	B	C	D	E	F	G	H
x1,x2	OUTPUTS							
00	1	1	1	0	1	0	1	1
01	0	0	0	0	0	0	0	0
10	1	1	1	0	0	0	1	1
11	0	1	1	0	0	0	1	1
P1	A	B	C	G	H	D	F	E
00	A	C	A	A	H	D	H	
01	H	H	H	H	G	D	H	
10	G	G	G	A	H	F	E	
11	G	H	A	A	G	F	F	
P2	A	C	G	H	B	D	E	F
00	A	A	A	H				
01	H	H	H	G				
10	G	G	A	H				
11	G	A	A	G				
P2	A	C	G	H	B	D	E	F

Table 17: Composite partition table.

From the final step in the partition table, the states left grouped in the partitions are all equivalent. In this case, the equivalences are as follows:

A' = A = C = G = H F
B' = B
C' = D
D' = E
E' = F

The final step is to put the results of the successive partitioning method back into either the state transition table or the state machine table as seen in Tables 18 and 19 below:

	00	01	10	11
A'	A'/1	A'/0	A'/1	A'/0
B'	A'/1	A'/0	A'/1	E'/0
C'	C'/0	C'/0	E'/0	E'/0
D'	A'/1	A'/0	E'/0	D'/0
E'	A'/0	A'/0	D'/0	E'/0

Table 19: Final state machine table.

	NS				Z			
PS	**00**	**01**	**10**	**11**	**00**	**01**	**10**	**11**
A'	A'	A'	A'	A'	1	0	1	0
B'	A'	A'	A'	E'	1	0	1	0
C'	C'	C'	E'	E'	0	0	0	0
D'	A'	A'	E'	D'	1	0	0	0
E'	A'	A'	D'	E'	0	0	0	0

Table 18: Final state transition table.

In this particular example, the number of states was reduced from seven (7) states to five (5) states.

Between these three states (row matching, implication charts, and successive partitioning), an engineer faced with a complex and challenging state machine can more easily and affordably accomplish a circuit without being overwhelmed. Some methods will result in a smaller number of states than others (particularly in the case of row matching verses implication chart) so it is beneficial to the engineer to try several methods before deciding upon a single one. In the end, using any of the state reduction methods will result in a more cost effective design than not using any simplification at all. The next overall step for an engineer after reducing the number of states, is to find a state assignment which will help simplify the finite state machine even more.

State Assignment

Binary

The Binary Method for assigning states is a common method to use. It counts up in binary starting from 0 and going up assigning each state the next number. It will use Log_2 bits to assign the states. Binary is a common method to use because it is easy to think of what the bit code will be for each state. Students often learn binary coding as their first encoding sequence because of this reason. However, it is very inefficient. There are a minimum number of bits used to encode the machine, but the encoding equations derived for each bit are often large and complex.

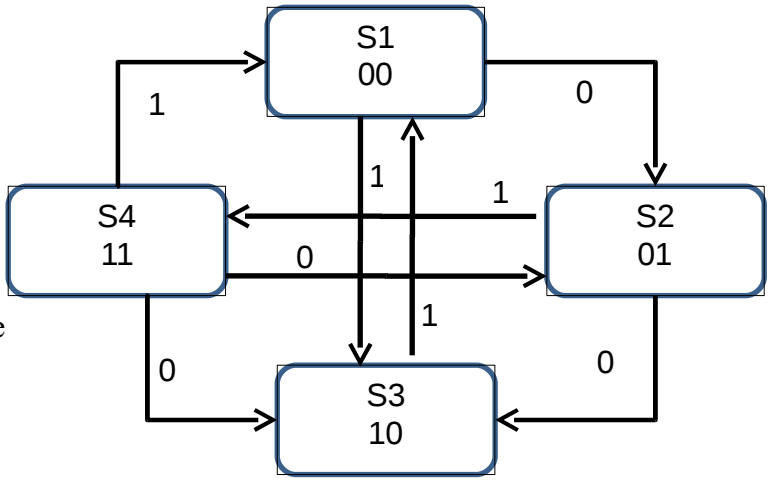


Table 20: A state representation of binary code

The equations derived for table 20 is:

Q0+ = (Q0' * Q1) + (X * Q0') + (Q0 * Q1' * X')
Q1+ = (X' * Q1') + (X' * Q0) + (X * Q0' * Q1)

Because of this reason, different methods have been created that shorten the glue logic needed between the different gates. The

Gray Code

Gray code was named after Frank Gray. In gray code, each successive state differs from the previous state by only one bit. With only one bit changing from each state to the next, power consumption is reduced from binary code where multiple states can change at the same time. In addition, This allows for a minimum number of bits used and an also a small equation for each bit. However, the encoding only works if the machine transfers from one state to the next in order. If the states do not travel in order, then the gray code does not work very well. The equations for each flip-flop are as shown:

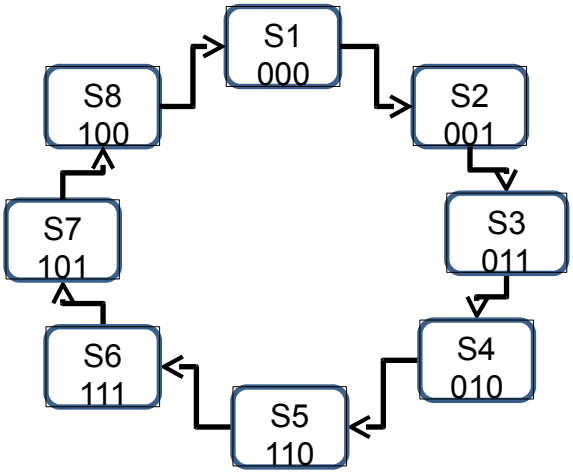


Table 21: A map of an state gray code

A+ = BC' + AC
B+ = A'C + BC'

C+ = A'B' + AB

One Hot Encoding

One Hot Encoding is an encoding sequence that uses as many registers as there are states. With One Hot Encoding, only one of the bits are 1 or "Hot" at any given state. All the other bits are 0. This makes sure only two bits change when moving from one state to the next. With a maximum of two bits changing at any given time less power is consumed. In addition, One Hot Encoding reduces the logic needed to implement each bit. However, it also uses many more logic gates to implement the state design. One Hot is often used where there are many states that need to be implemented in the design. Using binary encoding, the complexity of the design grows with each bit added. With One Hot, the number of flip-flops grows with each state added, but the complexity of each equation does not. Because of this, it is more difficult to accidentally mess up the logic needed to implement the bits. K-Maps are not needed for this type of encoding either. Instead, a truth table is created and used to find the equation for each bit. Using the chart above, the following truth table is created:

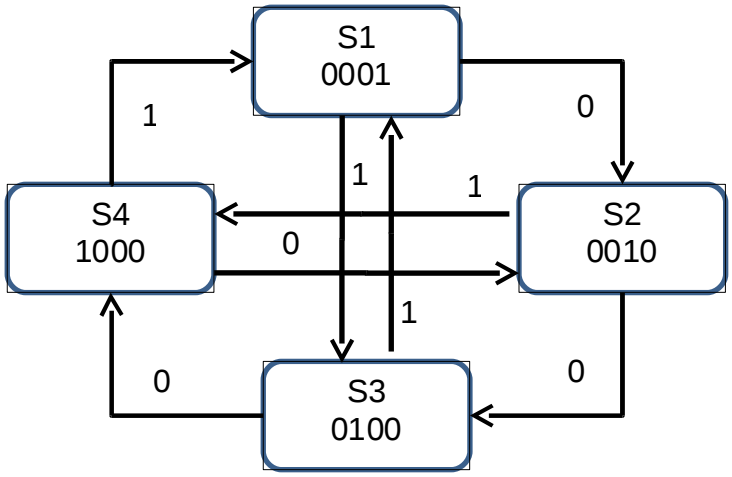


Table 22: One Hot Encoding Map

Input	**Current State**				**Next State**			
X	Q0	Q1	Q2	Q3	N0	N1	N2	N3
0	0	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	0
0	1	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
1	0	0	1	0	1	0	0	0
1	0	1	0	0	0	0	0	1
1	1	0	0	0	0	0	0	1

Table 23: The truth table for a One Hot Encoding

Using the standard method, the following equations can be easily derived by looking at the 1 on each next state. For state N0, the 1's are on rows 3 and 6. By looking at the current states and

the inputs on those rows, the equation for N0 is:
N0 = (X'*Q0'*Q1*Q2'* Q3') + (X*Q0'*Q1'*Q2*Q3')

By using the same method, the equations for the other next states are:
N1 = (X'*Q0'*Q1'*Q2*Q3') + (X*Q0'*Q1'*Q2'*Q3)
N2 = (X'*Q0'*Q1'*Q2'*Q3) + (X'*Q0*Q1'*Q2'*Q3')
N3 = (X*Q0'*Q1*Q2'*Q3') + (X*Q0*Q1'*Q2'*Q3')

Only one of these states are Hot at any given time. This means if Q0 is 1, then Q1, Q2 and Q3 are all 0. Likewise with all bits, if one is a 1, then all others are 0. Because of this, the equations can be reduced to:

N0 = (X' * Q1) + (X * Q2)
N1 = (X' * Q2) + (X * Q3)
N2 = (X' * Q3) + (X' * Q0)
N3 = (X * Q1) + (X * Q0)

This type of encoding uses more bits to encode the states, but reduces the logic needed to glue the bits together. It is a very common coding to use, especially for many states.

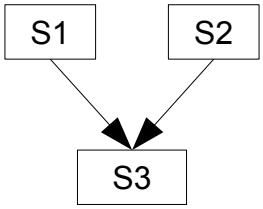
Enumerated

If the engineer only has a limited number of flip-flops and still wants to minimize the logic needed to implement the state machine, then the engineer needs to use an enumerated type of coding. Enumerated takes the states and arranges the binary code so the 1's are somewhat grouped in the K-Map produced for each flip-flop. This reduces the logic needed to glue the bits together.

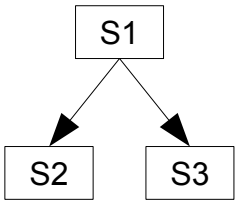
The rules of grouping are as follows:

Assign your states according to three things:

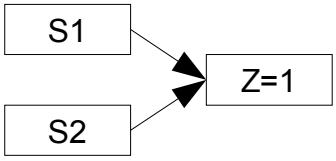
1) States that share a common next state.



2) States that share a common ancestor



3) States that have a common output.



The first task is to create a layout of the different states and where each state goes with each given input. The following is an example of such:

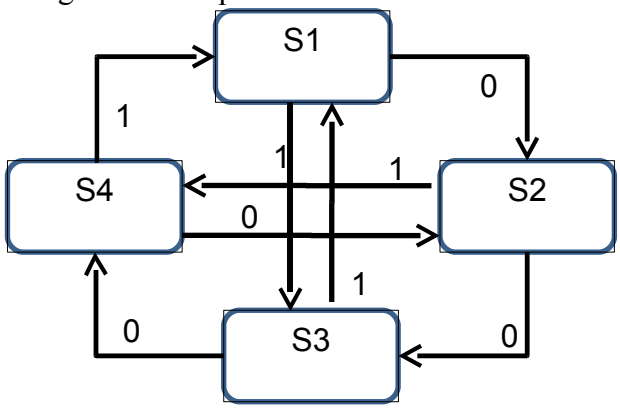


Table 24: State Design

Notice there is no encoding of each state. This has not been created yet.
Next create the state chart that just tells where each state is going for each combination of inputs:

CS	X=0	X=1
S1	S2	S3
S2	S3	S4
S3	S1	S1
S4	S2	S1

Table 25: State Chart of above design

This chart gives a graphical view of the states and their next state on all inputs. By using this chart and a State Map, similar to a K Map, one can easily organize the states so the next states, previous states and outputs line up optimally.

State 1 and 4 have common next states. States 2 and 1 have common next states. States 3 and 4 have common next states and common previous states. Using these observations, a possible map could be:

Q0\Q1	0	1
0	S2	S3
1	S1	S4

The States for this map are as follows.

State	Code
S1	10
S2	00
S3	01
S4	11

Table 26: Resulting codes for the states

The K Maps for the state assignment are as shown:

	00	01	11	10
0	1	1	0	0
1	1	0	0	1

Q0+

	00	01	11	10
0	0	1	0	0
1	1	1	1	0

Q1+

The equations for these assignments are:

Q0+ = (X * Q0') + (X * Q1) + (Q0' * Q1)
Q1+ = (X' * Q0') + (X * Q1')

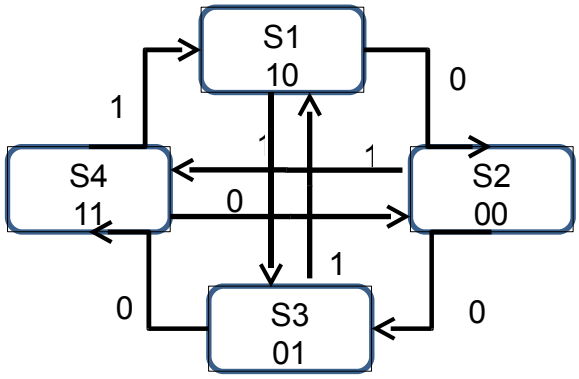


Table 27: The final result of the state design

This greatly reduces the equations from the binary equivalent of:

Q0+ = (Q0' * Q1) + (X * Q0') + (Q0 * Q1' * X')
Q1+ = (X' * Q1') + (X' * Q0) + (X * Q0' * Q1)

Enumerated takes a lot more work than the other types of encoding. However, it reduces the logic needed outside of the flip-flops while maintaining a low amount of flip-flops.

Conclusion

It takes more time and money initially to use these methods to reduce the number of states and logic to implement these states. Because of this, when designing a simple logic circuit, these methods are not needed. If a complex design with many states is required however, it takes more time and money in the end to not these methods. When dealing with state machines that have many inputs and states, the circuit can become very large and complex. The more complex the circuit, the easier it is to make a mistake in the logic. Bugs can crepe in easily and may not be noticed until it is too late. By using the methods above to first reduce the states and second to assign them to an optimal code, more time and money will be saved designing and creating the circuit in the end than to not use these methods in the first place.

References

Encoding State Machines. Retrieved May 11, 2009, from xilinx web site: http://www.xilinx.com/itp/xilinx4/data/docs/sim/vtex9.htm.

Gray code. Retrieved May 12, 2009, from wikipedia web site: http://en.wikipedia.org/wiki/Gray_code.

Grover, J. (2007). *Chapter 15 – State Reduction and State Assignment.* Retrieved May 12, 2009, from Uakron Engineering Web Site: engineering.uakron.edu/grover/web/ee263/handouts/Chapter%2015.pdf

Katz, R. (1993). *Chapter 9: Finite State Machine Optimization*. Retrieved May 11, 2009, from Berkeley Institute Web Site: inst.eecs.berkeley.edu/~cs150/sp00/classnotes/katz-ch9-mod.pdf

One hot encoding for state machines. Retrieved May 11, 2009, from: http://asics.chuckbenz.com/detailed_one_hot.htm.

Whitaker, J. (2005). *The Electronics Handbook 2nd Edition*. In Partitioning (pp. 73-74). CRC Press.