

Mirror Procedure for One-Shot-Coding of finite state machines

Dr. Wilfried Eisele

Addr.: Blumenstr. 3, D-79194 Gundelfingen, Germany

Email: w.eisele@t-online.de

Abstract:

In this paper a procedure is shown to modify a finite state machine in such a way that an One-Shot-Coding of their states is possible. The characteristic of One-Shot-Codes for finite state machines result in some properties of the realized sequential networks. The main advantage is less power dissipation, better electromagnetic compatibility and very fast circuits because only one bit changes between all successive states.

1 Introduction

In the past tense (mostly in the 60th) a lot of researchers, for instance Unger[4], had investigated many kinds of Finite State Machine (FSM) state codes. State encoding is a necessary part in the design of digital sequential circuits because all signals must be mapped to two possible electrical states: high voltage state or low voltage state. Hence, every state of a FSM has to be assigned at least to one binary tuple. This binary tuple represent points in a Boolean hypercube.

The characteristic of One-Shot Codes (see also [4]) is that neighbour states in the automata graph has to be assigned to neighbour points in the Boolean hypercube. The reason is that during a state transition only one bit shall change its value. So the only allowed ways through the hypercube are parallel to the dimension axes (edges along the hypercube). For this it is very important to define neighbourhood in state graphs and in Boolean hypercubes. The first neighbourhood type has to be mapped to the second one.

The advantage of One-Shot coded sequential circuits is that signal transitions will be minimized because only one bit of the state bits changes its value. In fact of this, the power dissipation is minimized and less generated internal thermal stress. Additionally the electromagnetic susceptibility is decreased because of more stable internal signals and the switching noise is reduced.

The usage of this state encoding is not limited to synchronous circuits. With some additional requirements a similar (near One-Shot) Code is possible for asynchronous circuits. More about this can be found in Eisele[2] and similar aspects for asynchronous Finite State Machines in Beister[3].

2 Preliminaries

The most important criteria for One-Shot coded FSMs is the constant distance aspect. The state graph has to be modified and to be mapped to the Boolean hypercube. Every node (state) in the state graph will be mapped to one (or more) points (numbered states) and every transition will be mapped at least to one edge in the Boolean hypercube. The distance between the nodes in the state graph are defined as the shortest way through the state graph without considering the transition direction. The distance between points in the Boolean hypercube are the shortest way between the points using only the edges of the hypercube (no diagonal transitions in the hypercube).

The main rule for One-Shot-encoding is, that distances of one between two nodes in the state graph shall be mapped to distances of one between their counterparts in the Boolean hypercube.

The following example illustrates, that often modifications of the state graph are necessary because many graph structures (e.g. cycles of odd length) are not possible to map in the Boolean hypercube.

The example (fig. 1) illustrates the fact, that it is not possible to encode three states so that every code word is a neighbour from every other code word. The trick is, to duplicate every node (state) so that there is a 6 node cycle. This is a possible structure in the Boolean hypercube. This example illustrates that it is often not possible to find a One-Shot-Code without modifying the state graph.

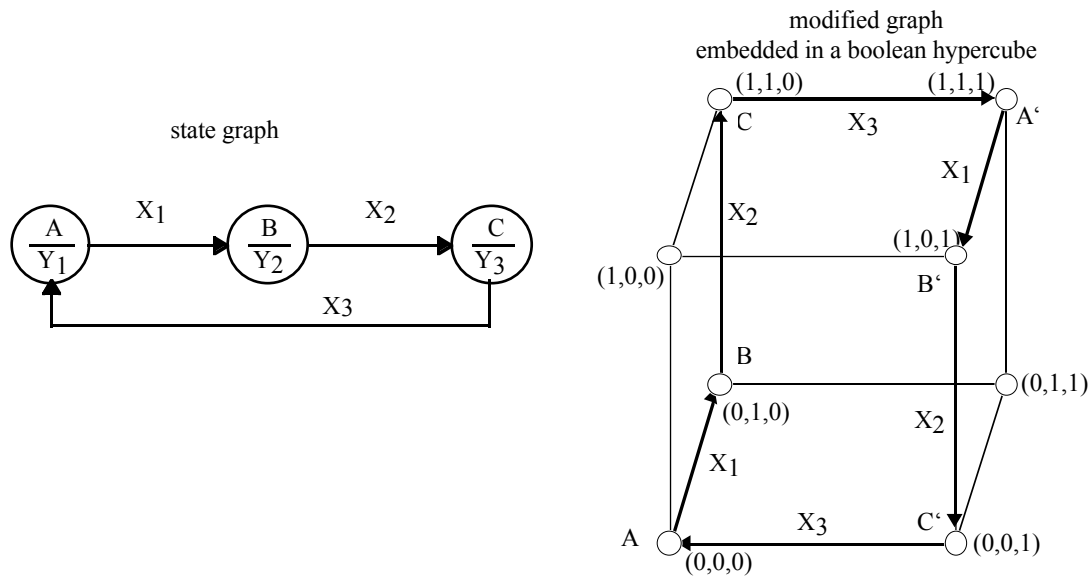


Fig. 1: Encoding of a three state FSM

3 The Basic Mirror Procedure

The basic idea of graph mirroring is not new. Kantabutra and Andreou [1] has shown this idea in 1994. This basic (or simple) mirror procedure is presented here.

First of all, the states (nodes) in the state graph has to be enumerated (encoded) in such a way that as few as possible would produce bad transitions. We will call bad transitions those transitions where more than one bit is changing its value. Furthermore, all bad transitions can be classified in types of bad transitions. Now define the type of a bad transition to be the set of bit positions at which the numberings of two states at the ends of the transition differ. For example, suppose the bit positions are counted from right to left. Then a transition between node $A=(1,0,0,0)$ and node $B=(1,0,1,1)$ would be a transition of type $\{1,2\}$. All bad transition types has to be notified.

All this bad transitions must be eliminated. How can this be done ?

The answer is, for every type of bad transition do the following:

One state variable q_{r+1} has to be added (until now there are r state variables). The code words of all states have to be duplicated. One with $q_{r+1} = 0$ and one with $q_{r+1} = 1$. Now have a look at the bad transition type for instance $A - B = (1,0,0,0) - (1,0,1,1) = \{1,2\}$. After the first step there are two code words for A ($(0,1,0,0,0)$ and $(1,1,0,0,0)$) and for B ($(0,1,0,1,1)$ and $(1,1,0,1,1)$). The code words with $q_{r+1} = 0$ are called the original states (eg. A, B) and the other ones are called the mirrored states (e.g. A' , B').

To eliminate all bad transitions of one specific type, all numbers (states X') with $q_{r+1} = 1$ have to be modified. All bit positions in which the actual bad transition differs will be inverted with respect to their original value. With this change, there is made a neighbourhood between A and B' and also B and A' .

It is further important to set the state graph transitions in a correct way between the states. There are two types of code words which differ in q_{r+1} . All state transitions which are not of the referenced type remain between the states with the same q_{r+1} . Only the transitions of the bad type will be changed and set between both halves of the hypercube along q_{r+1} .

The main idea is illustrated in the following figure.

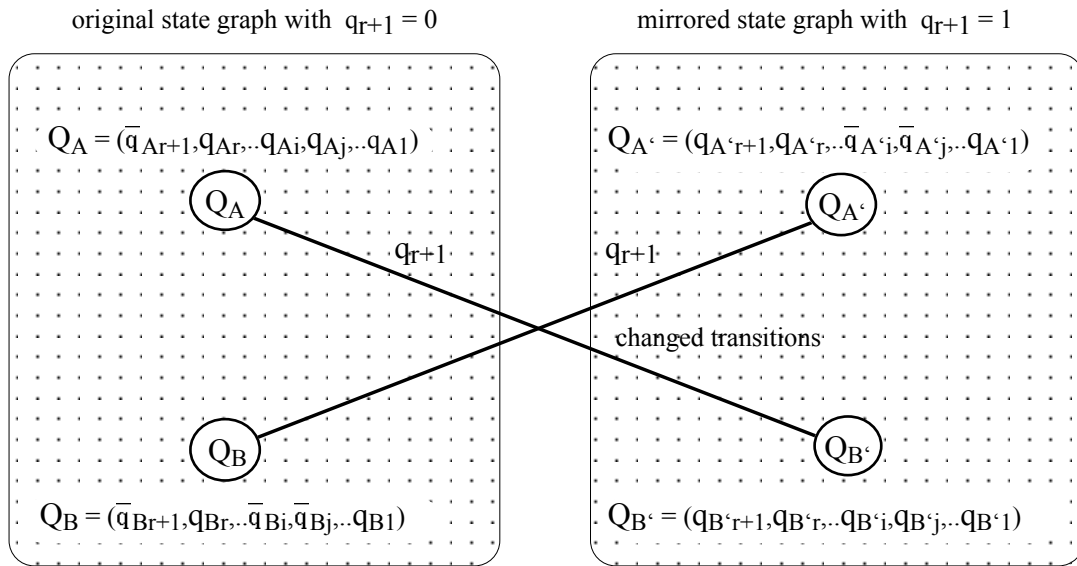


Fig. 2: General changes during one mirror step (bad transition A - B)

General procedure for replacing of bad transitions of one specific type by good ones:

Assume the actual bad transition type are between state A and B.

1. step: Adding new state variable q_{r+1} .
2. step: All existing code words obtain $q_{r+1} = 0$.
3. step: For every code word Q_X a copy $Q_{X'}$ has to be generated. $Q_{X'} = Q_X \oplus \delta Q$. Where $\delta Q = Q_A \oplus Q_B \oplus (1, 0, \dots, 0)$. The EXOR operator shifts all copies in the same direction through the Boolean hypercube.
4. step: All transitions which are not of the actual bad type are left unchanged inside their particular q_{r+1} hypercube half. The selected bad transition type must be replaced by good ones across both q_{r+1} hypercube half (e.g. $Q_A - Q_{B'}$ and $Q_B - Q_{A'}$).

An example (fig. 3 and 4) will illustrate the procedure.

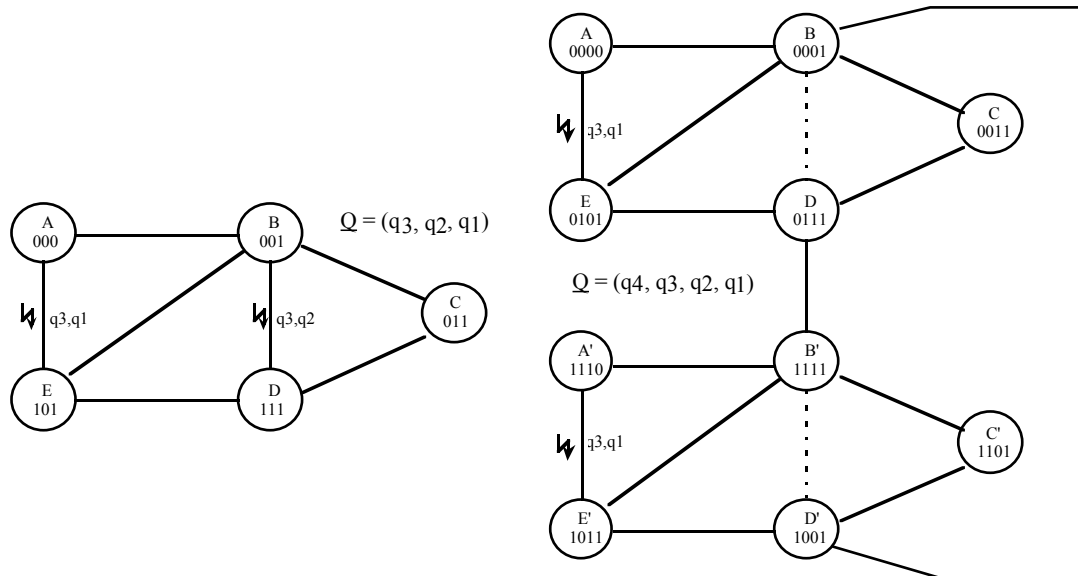


Fig. 3: Given state graph and modified state graph after first mirror procedure

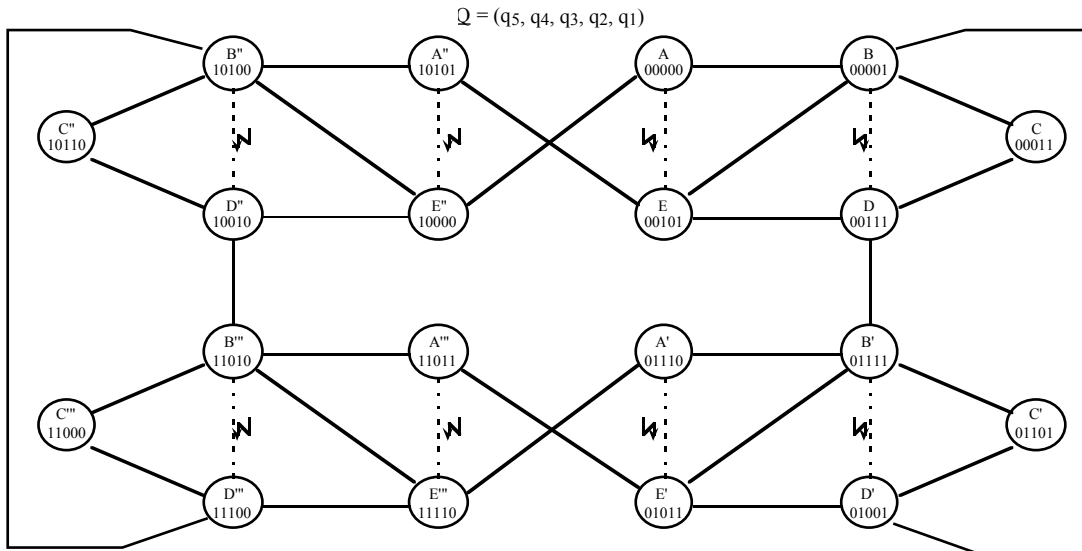


Fig. 4: State graph after second mirror procedure

It can be shown, that in the worst case, the number of mirror procedures are so much as Huffman[7,(also presented in [4]) for his universal solution.

Proof:

The considered finite state machine contain z states.

Hence, at least $r = \lceil \log_2 z \rceil$ variables for state coding are necessary.

The maximum k_m of transition types, is as follows

$$k_m = \sum_{i=1}^r \binom{r}{i} = \sum_{i=0}^r \binom{r}{i} - \binom{r}{0} = 2^r - 1$$

(Sum of all partitions of i state variables from a set of r state variables).

For instance, FSMs of 5..8 states needs maximal 7 state variables and FSMs with 9..16 states needs maximal 15 state variables. In such cases every neighbour of a code word represent an other original state. The Huffman algorithm generates a universal code in such a way that every type of transition is mapped to one axis in the Boolean hypercube

$$Q = (q_7, q_6, q_5, q_4, q_3, q_2, q_1)$$

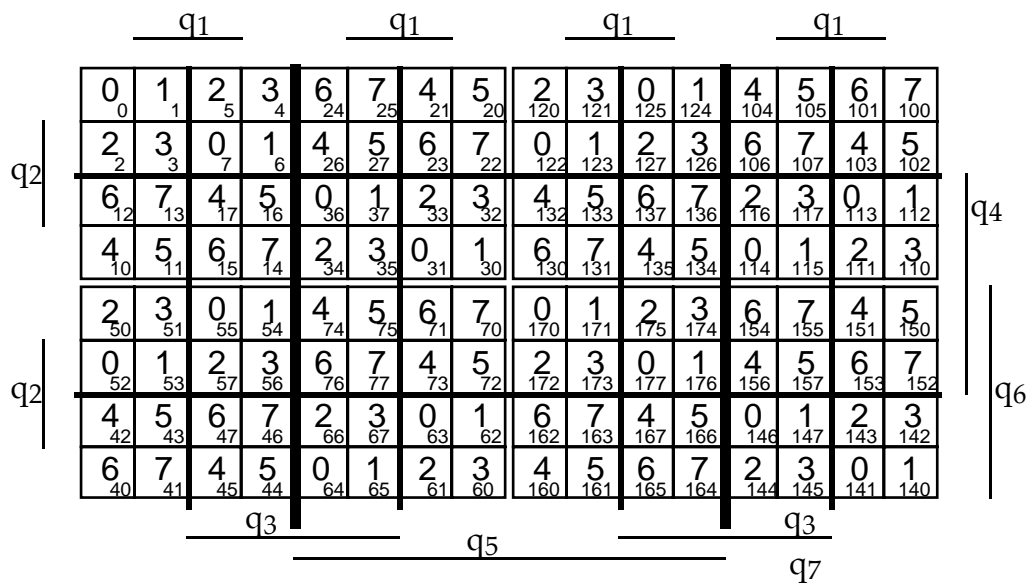


Fig. 5: Karnaugh diagram of the universal solution (worst case) for 8 state FSM

4 The Complete Mirror Procedure

The basic mirror procedure reflects only the possibility, that some bits can be inverted in that way that code words became neighbours. There is an additional aspect. Its also possible to change the sequence of bit positions in the way to get neighbour positions of the referenced points in the Boolean hypercube. Figure 6 illustrates a FSM graph with two bad transitions (A-F and D-C). The following figures 7 and 8 illustrate the difference between exchanging and inverting procedures at the mirror side. The code words are different between them.

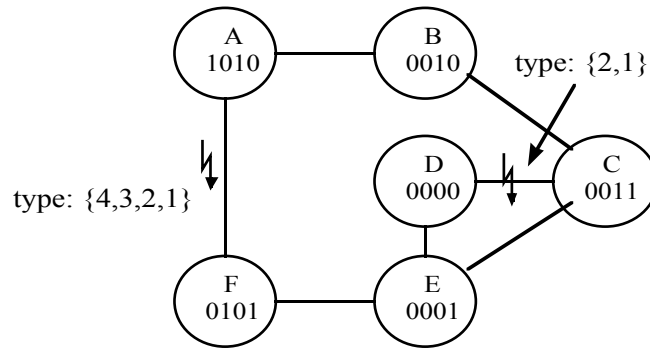


Fig. 6: Original state graph

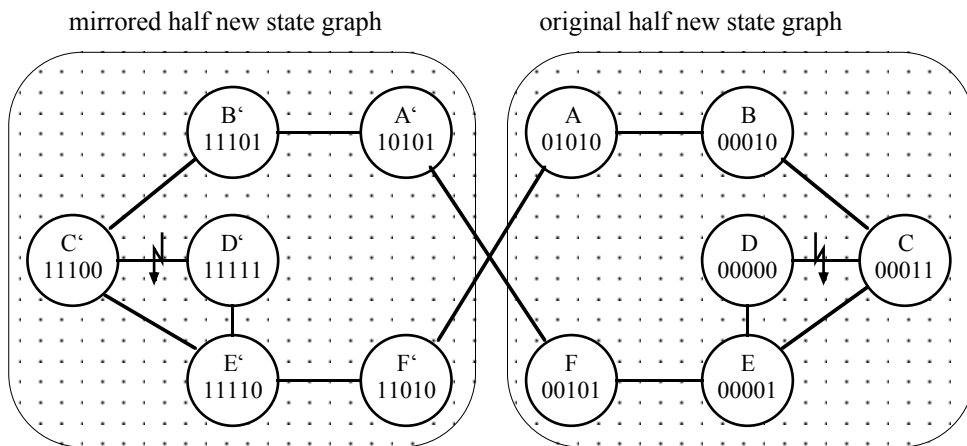


Fig. 7: Eliminating A-F transition by inverting of $q_1 \dots q_4$ at the mirror side

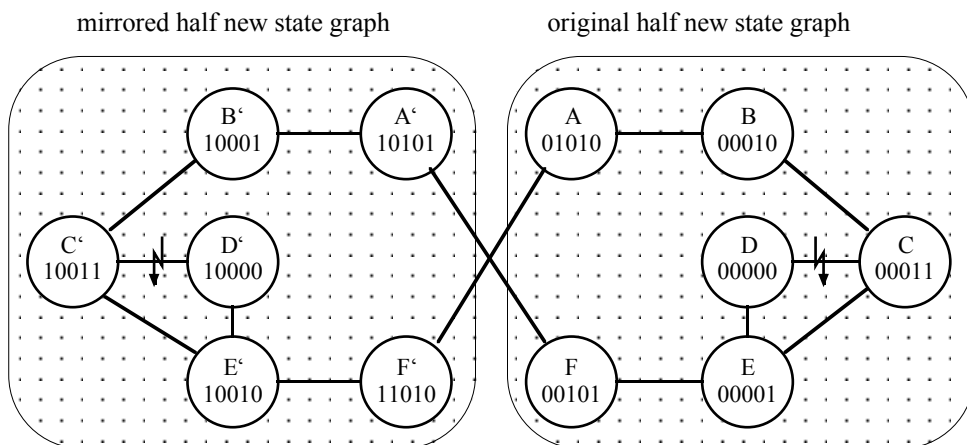


Fig. 8: Eliminating A-F transition by exchanging of q_4-q_3 and q_2-q_1 at the mirror side

Figure 9 illustrates, that this technique can generate better results than the simple procedure. With the basic mirror procedure, there are 2 mirror actions (4 times states) necessary. With the combination of complement of bit positions one can get a result with only a single mirroring operation. For further investigation how to get better results (less mirroring procedures) types of transitions have to be defined.

The main type is defined by the variables which changes its value. For instance in figure 7 the transition D-C is of the type {1, 2} because bit 1 and 2 changes their value.

The subtype gives information which changing bit positions have the same value in the considered code words. Table 1 illustrates the two subtypes of two variable transitions in a more detailed way.

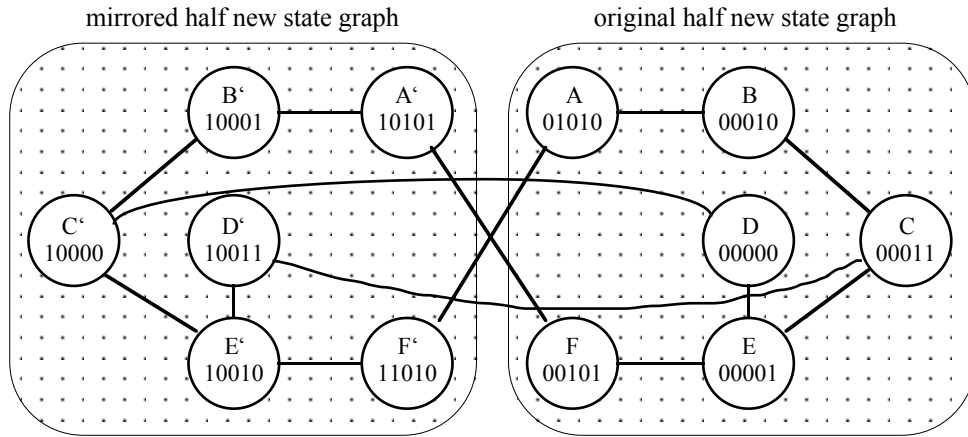


Fig. 9: Exchange of q_4 with q_3 and inverting of q_2 and q_1 at the mirror side

relation	type	$\delta(Q_A, Q_B)$	$\delta(Q_A, Q_{B'})$ $\delta(Q_{B'}, Q_A)$	$Q_A =$ $(q_{A1} \dots q_{A_i} \dots q_{A_j} \dots q_{A1})$	$Q_B =$ $(q_{B1} \dots q_{B_i} \dots q_{B_j} \dots q_{B1})$
$q_{A_i} = q_{A_j}$ $q_{B_i} = q_{B_j}$ $q_{A_i} \neq q_{B_i}$	$q_i = q_j$	2	3	$(\dots 0, \dots 0, \dots)$ $(\dots 1, \dots 1, \dots)$	$(\dots 1, \dots 1, \dots)$ $(\dots 0, \dots 0, \dots)$
$q_{A_i} \neq q_{A_j}$ $q_{B_i} \neq q_{B_j}$ $q_{A_i} \neq q_{B_i}$	$q_i \neq q_j$	2	1	$(\dots 0, \dots 1, \dots)$ $(\dots 1, \dots 0, \dots)$	$(\dots 1, \dots 0, \dots)$ $(\dots 0, \dots 1, \dots)$

Table 1: Fine distinction between subtypes of transitions

In the first case the variables inside the tuples have the same value. So if they are exchanged, nothing changes and the result is the distance of 3 (with additional variable) between them. In the second case the variables inside the tuple are different. If they are exchanged in one tuple, the tuple become the same. One variable has to be added and the result is to have the good case of distance 1.

Its seen that the exchange action works selective. In the first case of transition, there are no difference and in the second case the result is the wished one. Hence, it is necessary to get an additional criteria for distinguishing the transition types.

The transition types are shown as the partion of the changing bit positions. The subsets contains the bit positions with the same values. For instance $\Pi = (5; 2)$ means that variable 5 and 2 changes their value in the transition and they have different values in the code words, while $\Pi = (5, 2)$ means that variable changes their value in the transition and they have the same value in both code words.

transition ($q_p \dots q_1$)	transition type	correction	example new transition
<u>11000</u> – <u>01010</u>	$\Pi = (5; 2)$	$(5 \leftrightarrow 2)$ or $/ (5, 2)$	011000 – 111000
<u>010</u> – <u>101</u>	$\Pi = (3, 1; 2)$	$((2 \leftrightarrow 1)$ and $/3)$ or $((2 \leftrightarrow 3)$ and $/1)$ or $/ (3, 2, 1)$	0010 – 1010
<u>10001</u> – <u>00000</u>	$\Pi = (5, 1)$	$/5$ and $/1$	010001 – 110001
<u>111000</u> – <u>001011</u>	$\Pi = (6, 5; 2, 1)$	$((6 \leftrightarrow 2)$ and $(5 \leftrightarrow 1))$ or $((6 \leftrightarrow 1)$ and $(5 \leftrightarrow 2))$ or $((6 \leftrightarrow 2)$ and $/ (5, 1))$ or $((6 \leftrightarrow 1)$ and $/ (5, 2))$ or $((5 \leftrightarrow 2)$ and $/ (6, 1))$ or $((5 \leftrightarrow 1)$ and $/ (6, 2))$ or $/ (6, 5, 2, 1)$	0111000 – 1111000

Table 2: Correction of bad transition types

(In table 2 the character „ \leftrightarrow “ means the exchange of bit positions and „/“ means the inverting of bit positions).

In the basic mirror procedure only one state transition at one time is looked for. To get better results it is necessary to eliminate at least 2 bad transition types with one mirror procedure. In preparation for this, shared and local variables shall be distinguished. Local variables changes their value only in one of the considered transitions. Shared variables changes their value in more than one considered transitions.

Definition: Matching of two transitions

Two bad transitions match each other, if the can replaced by good ones with one mirror action.

pair of transition	type of transition	transition variable		both bad transitions eliminable together
		shared	local	
0001 – 1010 0000 – 1100	$(q_4, q_2; q_1)$ (q_4, q_3)	q_4	q_2, q_1 q_3	no
1011 – 0100 1100 – 1111	$(q_4, q_2, q_1; q_3)$ (q_2, q_1)	q_2, q_1	q_4, q_3	compl. q_2, q_1 exchange q_4, q_3
0000 – 1100 1010 – 0101	(q_4, q_3) $(q_4, q_2; q_3, q_1)$	q_4, q_3	q_2, q_1	compl. q_4, q_3 exchange q_2, q_1
1011 – 0100 1000 – 1011	$(q_4, q_2, q_1; q_3)$ (q_2, q_1)	q_2, q_1	q_4, q_3	no
0001 – 1011 0001 – 0010	(q_4, q_2) $(q_2; q_1)$	q_2	q_4 q_1	no
01101 – 10111 10001 – 00100	$(q_5, q_2; q_4)$ $(q_5, q_1; q_3)$	q_5	q_4, q_2 q_3, q_1	compl. q_5 exchange q_4, q_2 and q_3, q_1

Table 3: Examples of (bad) transition pairs (some matching, some not)

Local variables change their value in the change transition (end points are change points) and keep their value in the nonchanging transition (end points are nonchanging points).

Requirements for matching of two transitions:

1. For shared transition variables, only the complement operation is allowed (global effect).
2. A pair of local transition variables can be exchanged in the new mirror half of the hypercube if
 - both variables have different values in the tuples beyond the change transition and
 - both variables have the same value in the tuples of the nonchanging transition.

With this requirements it is possible to construct a compatibility matrix for all bad transitions in a FSM for getting a minimal coverage of transition (pairs). For every transition (pair) a mirror procedure is necessary.

5 Conclusion

Coding theory is still an important aspect in the digital synthesis theory. With a good coding style of finite state machines it is possible to get properties like low power (and heat) dissipation, a fast circuit and fewer EMC dissipations. Its seen, that the One Shot Code is well suited for this properties. With this work it has been shown, that with relatively simple FSM mirroring procedures fast state assignment using One Shot Codes can be found. In the most cases the results will be better (with fewer states) than with the universal method of Huffman or with the simple method of Kantabutra et. al..

Further aspects and methods is shown in [2].

Acknowledgments

This work was done during the time I was research assistant at the University of Kaiserslautern by Prof. Dr.-Ing. J. Beister. He helped with many discussions and suggestions for this work. I want also to thank Mr. Prof. Dr.-Ing. D. Bochmann for his helpful remarks as referee of my dissertation.

References

- [1] V. Kantabutra, A. G. Andreou: A State Assignment Approach to Asynchronous CMOS Circuit Design. IEEE Trans. on Computers, Vol 43, No. 4, 1994
- [2] W. Eisele: Zustandscodierung sequentieller Schaltungen mit einschrittigen, wettlauffreien Codes. Dissertation, Univ. Kaiserslautern, 2000
- [3] J. Beister, G.Eckstein, R.Wollowski: From STG to Extended-Burst-Mode Machines. Proc. of the 5th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, Barcelona, 1999
- [4] Unger, S. H.: Asynchronous Sequential Switching Circuits, Wiley-Interscience, New York, 1969
- [5] D. Bochmann.: Automatengraphen. Akademie Verlag, Berlin, 1982
- [6] D. A. Huffman: „A study of the memory requirements of sequential switching circuits“, Tech. rep. 293, MIT Cambridge, 1955