1976-10

Synthesis and applications of boolean memories

Philokyprou, G.

Nuclear Research Center "Democritos"

http://hdl.handle.net/11728/6303

Downloaded from HEPHAESTUS Repository, Neapolis University institutional repository
SYNTHESIS AND APPLICATIONS OF BOOLEAN MEMORIES

G. Philokypri, C. Malatsis, N. Galtanis
Computer Center, Nuclear Research Center "Democritus"
Athens, Greece

This paper presents a systematic method for the synthesis of Boolean Memories, i.e. memories having an arbitrary set of states. The method places no restrictions on the state table of the Boolean memory and now over leads to optimum realisation. A procedure is then developed for the decomposition of Boolean semantics and thus Boolean memories with large number of states can be synthesised. The paper also presents methods for using Boolean memories as memory elements of sequential circuits and special digital systems. These methods lead to the automatic synthesis of circuits having large number of states.

1. INTRODUCTION

The idea of using memory elements with more than two states is old enough. Thus Eckwith [1] and Richards [2] were the first who introduced such elements. The realisation, however, of such elements and the use of those in either special circuits (counters, accumulators, etc.) or as memory elements of sequential machines has received little or no attention. This was mainly due to the fact that a general synthesis method for such elements was not available.

Danielsen [3] in 1968 introduced the term Boolean Memory to denote satisfactory memory elements having any number of states and gave a design procedure for these. According to Danielsen a Boolean Memory is an asynchronous sequential machine having the restricted state table of Fig. 1, and may be realised by a digital net with delay-free feedbacks as in Fig. 2.

Fig.1: State table of a Boolean Memory.

\[
\begin{array}{c|cccccc}
 s_1 & s_2 & \ldots & s_n \\
 \hline
 t_1 & 1 & & & & & \\
 t_2 & 1 & & & & & \\
 \vdots & \vdots & & \ddots & & & \\
 \vdots & \vdots & & & \ddots & & \\
 \end{array}
\]

Danielsen, based on Unger's theorem [4] that "each feedback loop supposed to take the equilibrium state \((0,1)\) must contain an amplifier" has shown:

(a) That each feedback loop of a Boolean Memory net contains at least one NOT operator.

(b) That the Boolean memory net satisfies a system of Boolean equations

\[
\lambda_j = f_j(A_1, A_2, \ldots A_{j-1}, A_{j+1}, \ldots A_n) \quad (j = 1, 2, \ldots n)
\]

where each function \(f_j\) does not depend on the corresponding variable \(A_j\). Equations (1) are called the memory equation system. The solutions of this system is the set of the equilibrium states of the memory, forming the state matrix of the memory. Here \(f_j(i = 1, 2, \ldots n)\) gives the values of \(\lambda_j\) for every equilibrium state of the memory, while column \(i = 1, 2, \ldots n\) gives the encoding of state \(s_i\).

(c) Stability Theorem: Each pair of columns \(i\) and \(j\) in a state matrix must have a stability pair i.e. there must exist at least two rows \(h\) and \(k\) such that

\[
\begin{align*}
\lambda_h & = 0, \lambda_j = 1 \\
\lambda_k & = 1, \lambda_j = 0
\end{align*}
\]

Further to these, Danielsen has shown that given a state matrix satisfying the stability theorem there always exists at least one memory equation system satisfied by and only by that matrix, and has presented a general synthesis method for either a canonical sum-of-products (CSP) or a canonical products-of-sums (CPS) memory equation system of the form

\[
\begin{align*}
\lambda_j & = \sum P_i \mid \lambda_j = 1 \text{ in state } s_i \\
\lambda_j & = \prod S_i \mid \lambda_j = 0 \text{ in state } s_i
\end{align*}
\]

where \(P_i (S_i)\) in the product (sum) of the state variables \(A_j\) which are equal to \(1(0)\) at equilibrium state \(s_i\). The realisations obtained are in general far from optimum and simplifications, based on the fact that
In section II of the present paper we are concerned with the problems of the realisation of Boolean memories. The first problem deals with the realization of a Boolean memory whose state matrix of which does not initially satisfy the stability theorem of Daniels. The second problem concerns the decomposition of a Boolean memory with given state partition into a combination of symmetric boolean memories [3] with one out of a state encodings, called cyclic Boolean memories. The first problem is solved by introducing auxiliary state variables without affecting the original states. Assume, for example, that we are asked to design the most simple memory. That should obviously be a memory with one variable and two states 0 and 1 i.e., with state matrix

\[
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix}
\]

Clearly this matrix does not satisfy the stability theorem by introducing the additional state variable \( \alpha_2 \) we obtain the state matrix

\[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]

This matrix satisfies the stability theorem and possesses the same number of states as the original one. It may be observed here that this matrix represents the simplest possible Boolean memory which is the well known Eichenholzer bistable (or flip-flop) memory element. This element has one variable \( \alpha_1 \) and a hidden (auxiliary) variable \( \alpha_2 \). These facts the memory equation system

\[
\alpha_1 = \beta, \\
\alpha_2 = \alpha_1.
\]

Though any state table which does not satisfy the stability theorem may be transformed to one that satisfies it by simply doubling the number of state variables, a method is presented which, on the contrary, introduces the minimum possible number of additional state variables. The resultant state matrix may then be realized by either the Daniels' method or by our memory decomposition method. Now, given a state matrix not necessarily satisfying the stability theorem a method is presented so that the corresponding Boolean memory can be realized by a combination of cyclic Boolean memories with physical connections among them (second problem stated above). According to these the original state matrix is converted into one decomposable into submatrices of cyclic Boolean memories by the introduction of additional state variables the objective being the addition of the least possible number of additional variables.

The extraction of the set of cyclic Boolean memories into which the original memory is decomposed is quite straightforward. In addition to the opportunities offered by any decomposition synthesis method-modular construction, etc. the decomposition of Boolean Memories into cyclic ones, rather than into other types of Boolean Memories, is favoured by the fact that the cyclic Boolean Memories do not present any critical choice, and are excited in a straightforward manner. It may be observed here that the cyclic Boolean Memories can be considered as a generalization of the Set-Reset flip-flop.

In section III the application of Boolean Memories in the synthesis of sequential machines is examined. Though any of the existing methods of realization of sequential machines that utilize partitions may be readily translated to cope with the use of Boolean Memories as the memory part of a sequential machine, a new method is presented which is believed to offer greater design flexibility. The method introduces the notion of the backward state transition function of \( M \) and of that of the connected and strongly connected covers of \( M \) (CC and SCC of \( M \)) denoted respectively by \( Q(\beta) \) and \( Q \). It is shown that every \( Q(\beta) \) or \( Q \) corresponds to a machine hierarchically to \( M \) and \( Q \) are used for the direct derivation of the Boolean memory which can be used as the memory part of \( M \).

II. SYNTHESIS OF BOOLEAN MEMORIES

The following definitions are set to facilitate our discussion. Let \( \alpha_1, \alpha_2, \ldots, \alpha_n \) be elements in the \( \{0, 1\} \) space. Assuming that \( \alpha \) is an ordering relation, \( \alpha \) is defined in \( \{0, 1\} \) as follows

\[
\alpha_1 \alpha_2 \iff \alpha_1 = \alpha_2 \quad \forall i, j.
\]

Definition 1. A set \( S \subseteq \{0, 1\}^n \) is said to be an equilibrium set iff

\[
\forall \alpha, \beta \in S, \alpha \beta = \beta \alpha.
\]

Definition 2. Let \( S \) be a finite set and \( S(S) \subseteq \{0, 1\}^n \). \( J \) an index set, be a family of subsets of \( S \). \( S \) is called a Completely Separating System (CSS) of \( S \) iff for every two distinct elements \( \alpha, \beta \in S \) there exist \( i, j \in J \) such that

\[
\alpha \neq \beta \quad \text{iff} \quad S_i \cap \beta \neq \emptyset \quad \text{and} \quad \alpha \neq \beta \quad \text{iff} \quad S_j \cap \alpha = \emptyset.
\]

The concept of a CSS was first introduced by Dickeon [5] and is an extension of the concept a Separating System (SS) of \( S \) introduced by Dugundji [5]. According to Dugundji, \( S \in \mathcal{S}(S) \), \( J \) an index set, \( S \in \mathcal{S}(S) \) is called a Separating System (SS) of \( S \) iff for any two distinct elements \( \alpha, \beta \in S \) there exist \( i, j \in J \) such that either \( \alpha \in S_i \) and \( \beta \in S_j \) or \( \alpha \in S_i \) and \( \beta \in S \). Let now \( S(S) \subseteq \{0, 1\}^n \) be a family of subsets of \( S \). \( S \subseteq \{0, 1\}^n \) defined by

\[
S_i = \{ (x_j)_{j=1}^n \mid i, j \in J, i,j \in J, n \text{ defined by (2) then } S \text{ must be a CSS of } S \}.
\]

From Definitions 1 and 2, eq (2) and the stability theorem of Dugundji it is easy to see that an \( n \times n \) state matrix of a Boolean Memory can satisfy the stability theorem is equivalently described by either an equilibrium set or a CSS. Thus if \( E(\{0, 1\}^n) \) is the set of columns of a state matrix, then \( M \) can be an equilibrium set and if \( S(S) \subseteq \{0, 1\}^n \) is defined by eq (2) then \( S \) must be a CSS of \( S \).

The above formulation is helpful in solving the two problems of realization of Boolean Memories stated in the introduction. The problem of constructing a state matrix not satisfying the stability theorem is reduced to one that satisfies it is equivalent to converting a \( S(S) \) of a set \( S \) to a CSS or \( S \), in the following an equivalence and an ordering relation among the elements of the set \( S \) are first introduced and the concepts of CSS and CSS's are redefined in terms of these relations. The concept of the \( S \)-Separating partition is then introduced and is used as the main tool for solving this problem.

Let \( S \) be a finite set and \( S(S) \subseteq \{0, 1\}^n \). \( J \) an index set, be a family of subsets of \( S \). \( S \in \mathcal{S}(S) \), \( J \) an index set, then

\[
x(x) \iff \forall i, j \in J, x(i) = x(j) \iff x(i) = x(j).
\]

An ordering relation \( \prec \) is defined in \( S \) has follows

\[
\forall x, y \in S, x \prec y \iff x(i) < y(i) \quad \forall i \in J.
\]
\[ x \leq y(n) \iff \forall i, j \leq n, x[i] \leq y[i] \]

If for some \( x[i], y[j], \forall j, \) then \( x \leq (y) \forall y[j]. \)

Also, \( x \leq y(s) \iff x \leq y(s) \) and \( x \geq y(s). \)

**Definition 4.** \( \mathcal{S} \) is called a CSS, respectively a CSS, of \( \mathbb{N} \) \( \forall x, y \in \mathbb{N} \)

\[ x \leq y(n) \iff \forall i, j \leq n, x[i] \leq y[i] \]

**Definition 5.** Let \( P \) be a partition on \( \mathbb{N} \). \( \mathcal{P} \) is called an \( S \)-partition of \( \mathbb{N} \) \( \forall x, y \in \mathbb{N} \)

\[ x \leq y(n) \iff \forall i, j \leq n, x[i] \leq y[i] \] and \( x \geq y(s). \)

An \( S \)-partition on \( \mathbb{N} \) is said to be \( S \)-maximal if every \( P \) is not an \( S \)-partition on \( \mathbb{N} \). An \( S \)-partition on \( \mathbb{N} \) is said to be an \( S \)-partition on \( \mathbb{N} \) if \( \exists \) in \( \text{minim} \), \( i \) denotes the number of blocks of \( P \).

If \( P \) denotes the partition on \( \mathbb{N} \) which contains \( x \) and \( y \) in one block and every other element of \( \mathbb{N} \) in separate blocks then it can be shown that the following sequence of sets of partitions on \( \mathbb{N} \) is \( \forall x, y \in \mathbb{N} \)

\[ \mathcal{A}_0 \left( \mathcal{A}_1 = \mathcal{A}_2 = \ldots \right) \]

(b) \( \mathcal{A}_n \) is defined, then

\[ \mathcal{A}_{n+1} \left( \mathcal{A}_n \text{ or } x \leq y(s) \right) \]

Obviously, for some \( n, \mathcal{A}_n = \mathcal{A}_1 \) and \( \mathcal{A}_n \) contains all \( \forall x, y \in \mathbb{N} \)

**Definition 6.** Let \( \mathcal{P} \) be \( \mathcal{P}(\mathbb{N} \times \mathbb{N}) \) be a \( S \)-partition on \( \mathbb{N} \). \( \mathcal{P} \) is said to be \( \mathbb{N} \)-partition on \( \mathbb{N} \).

\[ x \leq y(n) \iff \forall i, j \leq n, x[i] \leq y[i] \]

\( x \geq y(s) \iff x \geq y(s) \) and \( y \geq y(s) \).

\( P \) is said to be \( \mathbb{N} \)-partition on \( \mathbb{N} \) if \( \exists \mathcal{P} \) in \( \mathbb{N} \)-partition, \( x \leq y(s) \) and \( y \geq y(s) \).

It can be shown that for every \( \mathcal{P} \) of \( \mathcal{P} \) there exists \( \forall x, y \in \mathbb{N} \) at least one \( \mathcal{P} \)-partition \( \mathcal{P}(\mathbb{N} \times \mathbb{N}) \) \( \forall x, y \in \mathbb{N} \)

\[ x \leq y(n) \iff \forall i, j \leq n, x[i] \leq y[i] \]

\( x \geq y(s) \iff x \geq y(s) \) and \( y \geq y(s) \).

\( \mathcal{P} \) is called a \( \mathbb{N} \)-partition on \( \mathbb{N} \) if \( \exists \mathcal{P} \) in \( \mathbb{N} \)-partition, \( x \leq y(s) \) and \( x \geq y(s) \).

**Definition 7.** Let \( P \) be a partition on \( \mathbb{N} \). \( \mathcal{C} = \mathcal{C}(\mathbb{N} \times \mathbb{N}) \) \( \forall x, y \in \mathbb{N} \)

\[ x \leq y(n) \iff \forall i, j \leq n, x[i] \leq y[i] \]

\( y \geq y(s) \iff y \geq y(s) \) and \( x \geq y(s) \).

\( \mathcal{C} \) is a minimal \( S \) if \( \forall x, y \in \mathbb{N} \)

\[ \lVert \mathcal{C} \rVert = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \]

From definitions 4, 5, 6 and 7 the following theorems can be derived.

**Theorem 1.** Let \( \mathbb{N} \) be a \( \mathbb{N} \)-partition on \( \mathbb{N} \) if \( \exists x \) in \( \mathbb{N} \)-partition, \( x \geq y(s) \) and \( x \leq y(s) \).

From the above it is deduced that in order to convert a \( \mathbb{N} \)-partition on \( \mathbb{N} \) into a CSS the following steps are required

(a) Form the sequence \( \mathcal{A}_n \) of \( \mathbb{N} \)-partition on \( \mathbb{N} \) if \( \exists x \) in \( \mathbb{N} \)-partition, \( x \geq y(s) \) and \( x \leq y(s) \).

(b) Given \( \mathcal{P} \) an \( \mathbb{N} \)-partition on \( \mathbb{N} \) if \( \exists x \) in \( \mathbb{N} \)-partition, \( x \geq y(s) \) and \( x \leq y(s) \).

In the following we shall be concerned with the problem of decomposition of the Boolean algebra.
Let $H_0(\overline{1}\overline{1})(2)$ be decomposable according to the definition of $\psi_q(1)$. In this case, the set of all maximal partitions of $P_1$ according to which $P_1$ is decomposable into $k$ subsystems of cyclic Boolean memories.

Example: Let the state matrix

$$
\begin{pmatrix}
1 & 2 & 3 & 4 \\
3 & 2 & 1 & 4 \\
2 & 1 & 3 & 4 \\
4 & 3 & 2 & 1
\end{pmatrix}
$$

to be decomposed into subsystems of cyclic Boolean memories.

Step 1. According to the definition of $\psi_q(1)$, it is equal to $\psi_q(1) = \{001010,110100\}$. The resultant state matrix is therefore

$$
\begin{pmatrix}
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4
\end{pmatrix}
$$

decomposed into the subsystems

$$
\begin{pmatrix}
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0
\end{pmatrix}
$$

ends up with the set of all maximal partitions of $P_1$ according to which the set $H_0(\overline{1}\overline{1})(2)$ is decomposable.
As mentioned in section 1.5, a method is developed for using a boolean memory in the realization of synchronous sequential machines.

Let \( \mathbb{S}(x, z, y, f, a) \) be a synchronous sequential machine, where \( S = \{ x, z \} \) are finite non-empty sets of states; \( \{ a \} \), inputs \( x \) and outputs \( y \) respectively and \( f(x, z, y) \) denote the transition and output functions respectively.

Let \( \mathbb{S} \) denote the set of all subsets of \( S \).

**Definition 11** Let \( F \subseteq S \) be the set of all \( F \subseteq S \). For every state \( f(x, z, y) \), the function \( \mathbb{S} \) called the forced state transition function of \( H \).

**Definition 12** Let \( F \subseteq S \) be the set of all \( F \subseteq S \). For every state \( f(x, z, y) \), the function \( \mathbb{S} \) called the prohibited state transition function of \( H \).

Let \( x_{i+1} = \delta(x_i) \) for any input sequence \( \{ x \} \) of length \( n \) as follows:

\[
\delta(x_i) = \delta_{x_i} = \delta(x_{i-1}, \ldots, x_1, x_0, \ldots, x_{i+1})
\]

From definition 1 and 2 the following properties of \( \delta \) and \( \delta \) are easily derived:

1. \( \delta \) is a non-empty subset of \( S \).
2. \( \delta \) is a non-empty subset of \( S \).

In the following we refer to sequential machines with transition functions completely defined.

**Definition 13** Let \( F \) be a non-empty subset of \( S \). The set \( \delta_{\mathbb{S}}(F) \) of all \( \delta_{\mathbb{S}}(F) \) is called a component under \( (F) \) of \( H \) and is denoted by \( \mathbb{T} \).

**Definition 14** A family of subsets \( \{ Q \} \) is called a strongly connected component of \( H \) of index set \( \mathbb{I} \) as \( Q \) is called a strongly connected component of \( H \) is denoted by \( \mathbb{T} \).

**Remarks** (a) \( \delta_{\mathbb{S}}(F) \) and \( \delta_{\mathbb{S}}(F) \) are considered to be trivial SCC's of a machine \( H \) if \( \mathbb{S} \) is a finite, if \( \mathbb{S} \) is also finite and hence it contains at least one SCC, denoted by \( \mathbb{S} \). Let \( \mathbb{X} \) denote the set of all SCC's of the machine \( H \).

**Definition 15** Let \( H \) be an SCC of a machine \( H \). Then \( \mathbb{X} \) is a non-empty SCC of \( H \) if and only if \( \mathbb{X} \) is the only SCC that is contained in a CC of \( H \).

**Definition 16** Let \( \mathbb{X} \) be an SCC of a machine \( H \). Then \( \mathbb{X} \) is called an SCC of \( H \) if and only if \( \mathbb{X} \) is the only SCC of \( \mathbb{X} \) of \( H \). It can be proved that:

- \( \mathbb{X} \) is a strongly connected component of \( H \).
- \( \mathbb{X} \) is a cover of \( \mathbb{X} \).

Using \( \mathbb{X} \) and \( \mathbb{X} \) the operation \( \mathbb{X} \) and \( \mathbb{X} \) may be defined among the SCC's of \( H \) as follows:

- \( \mathbb{X} \) is the sum of \( \mathbb{X} \) and \( \mathbb{X} \) for each \( \mathbb{X} \) and \( \mathbb{X} \) of \( H \).
- \( \mathbb{X} \) is the product of \( \mathbb{X} \) and \( \mathbb{X} \) for each \( \mathbb{X} \) and \( \mathbb{X} \) of \( H \).

Every non-zero SCC of \( H \) is the sum of non-zero minimum SCC's of \( H \).

- \( \mathbb{X} \) is a minimum SCC of \( H \) with \( \mathbb{X} \) then there is no other SCC of \( H \).
- \( \mathbb{X} \) is a minimum SCC of \( H \) with \( \mathbb{X} \) then there is no other SCC of \( H \).
- \( \mathbb{X} \) is a minimum SCC of \( H \) with \( \mathbb{X} \) then there is no other SCC of \( H \).
- \( \mathbb{X} \) is a minimum SCC of \( H \) with \( \mathbb{X} \) then there is no other SCC of \( H \).

The operation for finding all SCC's of \( H \) based on \( \mathbb{X} \) and \( \mathbb{X} \) and \( \mathbb{X} \) and \( \mathbb{X} \) has the following steps.

Step 1. Using \( \mathbb{X} \) find all \( \mathbb{X} \) of \( H \).

Step 2. Find all SCC's of \( H \) that are generated solely from each \( \mathbb{X} \), using the sum operation, \( \mathbb{X} \) and \( \mathbb{X} \) above.

Step 3. Find all SCC's of \( H \) that are generated from different \( \mathbb{X} \)'s using the sum operation.

Let now \( \mathbb{X} \) for any \( \mathbb{X} \) and \( \mathbb{X} \) where \( \mathbb{X} \) and \( \mathbb{X} \) are every \( \mathbb{X} \) and \( \mathbb{X} \) we associate the partitions

- \( \mathbb{X} \) and \( \mathbb{X} \) respectively. The following theorems can easily be
Theorem 4. $Q(p)$ and $P$ are partitions on $S$ with substitution property ($P$).

Theorem 5. For every partition $P = \{B_i\}$ on $S$, if $\forall i, \exists j$ such that $P_{ij} = 0$, then $Q(p)$ is a family.

Theorems 4 and 5 establish the relationship between $P$ and $Q$. Thus, assuming that $Q(p)$ is a family, theorems 4 and 5 may be used to derive the final state space for the sequential machine.

Converting the set of equilibrium states of the machine into a decomposed equilibrium set, we get the state matrix:

$$
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
Y_1 & 1 & 1 & 1 & 0 & 0 & 0 \\
Y_2 & 1 & 0 & 0 & 0 & 1 & 1 \\
Y_3 & 0 & 1 & 0 & 1 & 0 & 0 \\
Y_4 & 0 & 0 & 1 & 1 & 0 & 0 \\
Y_5 & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
$$

If we have a family of $Q$'s $\{Q_{ij}\}$ such that $P_{ij} = 0$ and $Q_{ij} = 0$ for the output of the $i$-th state variable, then we can use a realization of the machine, the following:

$$
\begin{array}{c}
X_t = \sum_{i=1}^{n} x_i Y_i \\
\end{array}
$$

where $x_i$ is the $i$-th state variable.

Theorem 6. The Boolean memory corresponds to a 3-state and a 3-state cyclic Boolean memory. The excitation functions are:

$$
Z_{x_1} = x_1 y_1, Z_{x_2} = x_2 y_2, Z_{x_3} = x_3 y_3
$$

IV CONCLUSIONS.

In this paper, the problem of realization of Boolean memories is reduced to the problem of finding the optimal polynomial (OPF) function for the excitation functions. If we want to decrease the number of state variables, we can use a procedure for selecting the optimum OPF.

The family of $2^m$'s $\{Q_{ij}\}$ with $Q_{ij} = \{1i, 1j, m\}$ and $Q_{ij} = \{16, 5, 31; 136, 136, 233\}$ is found to be an optimum OPF for excitation functions and $P_{ij} = 0$.

The state matrix of the Boolean memory corresponding to $\{Q_{ij}\}$ is:

$$
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
Y_1 & 1 & 1 & 1 & 0 & 0 & 0 \\
Y_2 & 1 & 1 & 0 & 0 & 1 & 1 \\
Y_3 & 1 & 0 & 0 & 1 & 0 & 0 \\
Y_4 & 0 & 1 & 0 & 1 & 0 & 0 \\
Y_5 & 0 & 0 & 1 & 1 & 0 & 0 \\
Y_6 & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
$$

Then the Boolean memory corresponds to a 3-state and a 3-state cyclic Boolean memory. The excitation functions are:

$$
Z_{x_1} = x_1 y_1, Z_{x_2} = x_2 y_2, Z_{x_3} = x_3 y_3
$$

V. REFERENCES.


