Synthesis of Reversible Asynchronous Cellular Automata for Pattern Generation with Specific Hamming Distance*

Sukanta Das[1], Anindita Sarkar[1], and Biplab K. Sikdar[2]

[1] Department of Information Technology
sukanta@it.becs.ac.in, anindita.sarkar10@gmail.com
[2] Department of Computer Science & Technology
Bengal Engineering & Science University, Shibpur, West Bengal, India, 711103
biplab@cs.becs.ac.in

Abstract. The reversibility issue of 1-dimensional asynchronous cellular automata (ACA) has been reported in [4]. The ACA rules are classified to synthesis reversible ACA. Characterization has been done to explore the update patterns of cells forming the reversible ACA. This work reports synthesis of reversible ACA for pattern generation, where specific Hamming distance between two consecutive ACA states of a cycle is maintained. A hardware realization of ACA is also reported that can effectively be utilized in VLSI circuit design, testing of asynchronous circuit, and Hamming code generation.

Keywords: Asynchronous cellular automata (ACA), reversibility, ACA cycle, reversible ACA rules.

1 Introduction

The reversibility of (synchronous) cellular automata has been studied extensively for years [1,2,5]. However, reversibility of asynchronous cellular automata (ACA) is almost an unaddressed issue. A very few works on reversibility of 2-dimensional ACA have been reported in the literature [3]. This scenario motivates us to explore the issue for 1-dimensional two-state 3-neighborhood ACA in [4].

In ACA, the cells are updated independently. So, in a single step any number of cells can be updated. Based on the updates of ACA cells, we define *update pattern* for reversibility of ACA. A number of CA 'rules' [6] has been identified as the *irreversible rules* that can not design reversible ACA for any set of update patterns. On the other hand, the *reversible rules* can synthesize reversible ACA for some set of update patterns. A scheme is developed to find an update pattern of ACA with specific cycle. In this work, we report synthesis of reversible ACA having some cycles for pattern generation, where two consecutive states maintain some specific Hamming distance. A hardware realization of ACA is also reported that can effectively be utilized in VLSI circuit design and testing of asynchronous circuit.

* This work is supported by AICTE Career Award fund (F.No. 1-51/RID/CA/29/2009-10), awarded to Sukanta Das.

2 Cellular Automata

A cellular automaton (CA) is the discrete spatially-extended dynamical systems. It evolves in discrete space and time. In its simplest form, as it is proposed by Wolfram [6], CA consist of a lattice of cells, each of which stores a discrete variable at time t that refers to the present state of the CA cell. The next state of a cell in 1-dimensional two-state 3-neighborhood (self, left and right neighbors) CA is $S_i^{t+1} = f(S_{i-1}^t, S_i^t, S_{i+1}^t)$, where f is the next state function; S_{i-1}^t, S_i^t and S_{i+1}^t are the present states of the left neighbor, self and right neighbor of the i^{th} CA cell at t. The function $f : \{0,1\}^3 \mapsto \{0,1\}$ can be expressed as a look-up table as shown in Table 1. The decimal equivalent of the 8 outputs (next state) is called 'rule' [6]. There are 2^8 (256) CA rules in two-state 3-neighborhood dependency. Two such rules, 60 and 51 are shown in Table 1. From the view point of *Switching Theory*, a combination of the present states (first row of Table 1) can be viewed as the *Min Term* of a 3-variable $(S_{i-1}^t, S_i^t, S_{i+1}^t)$ switching function. So, each column of the first row of Table 1 is referred to as Rule Min Term (RMT).

The collection of states of all the cells $(S_1^t, S_2^t, \cdots, S_n^t)$ at time t is the state of n-cell CA at t. If the left most and right most cells are the neighbors of each other (that is, $S_0^t = S_n^t$ and $S_{n+1}^t = S_1^t$), then the CA are *periodic boundary* CA. On the other hand, in *null boundary* CA, $S_0^t = S_{n+1}^t = 0$ (null).

Table 1. Look-up table for rule 60 and 51

Present state :	111	110	101	100	011	010	001	000	*Rule*
(*RMT*)	(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)	
(i) Next State :	0	0	1	1	1	1	0	0	60
(ii) Next State :	0	0	1	1	0	0	1	1	51

If all the cells of a CA update their states simultaneously, the CA is *synchronous*. In *asynchronous CA* (ACA), the cells are updated independently and any number of cells can be updated in a time step.

$$\begin{tikzcd}
15 \arrow[r, "2"] & 13 \arrow[r, "1,4"] & 5 \arrow[r, "2,4"] & 15 \arrow[r, "3"] & 11 \arrow[r, "2,4"] & 9 \arrow[r, "3,4"] & 9 \arrow[r, "2"] & 11 \arrow[r] & \cdots
\end{tikzcd}$$

Fig. 1. Partial state transition of rule 60 ACA. Cells updated are noted on the arrow

The next state of ACA depends not only on the cell rule, but also on the cells that update their states at that time instant. We denote the set of cells updated at t as u_t. Therefore, $U = \langle u_1, u_2, \cdots, u_t, \cdots\rangle$ is the *update pattern* for the ACA. If the ACA rule, U and an initial state are given, the state transitions of the ACA can be identified. A partial state transition diagram of 4-cell rule 60 ACA with null boundary condition is shown in Fig. 1. The set $U = \langle\{2\}, \{1,4\}, \{2,4\}, \{3\}, \{2,4\}, \{3,4\}, \{2\}, \cdots\rangle$ and initial state is 15.

In ACA, a single state transition diagram may not cover all the CA states. A set of update patterns can illustrate the transition of all the states.

3 The Reversibility of ACA

The state transition diagram classifies the CA states as *cyclic* and *acyclic* states. If a CA state lies on a cycle in state transition diagram, the state is cyclic; otherwise it is acyclic. The CA is *reversible* if all the CA states are cyclic. The reversibility, in synchronous domain, guarantees that each CA state has unique predecessor and successor.

Definition 1. *For a set of update patterns, an ACA is* **reversible** *if each CA state can uniquely be reached from the state itself. Otherwise, it is* **irreversible**.

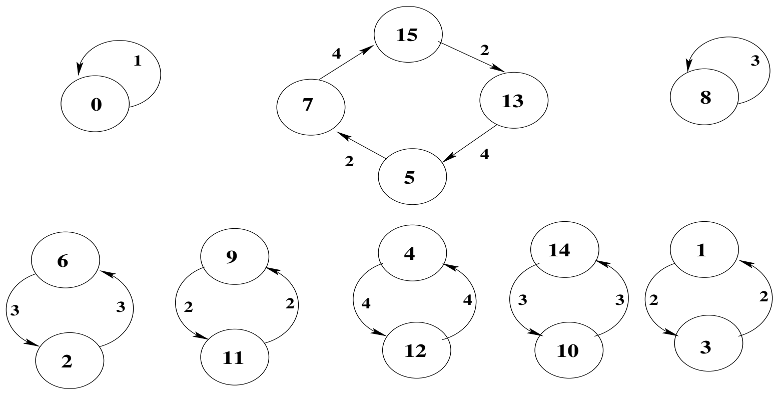


Fig. 2. 4-cell rule 60 reversible ACA in null boundary condition

Fig. 2 depicts the state transition diagram of 4-cell null boundary reversible ACA. The 8 update patterns (one for each cycle) are $\langle\{1\}\rangle$, $\langle\{2\}, \{4\}, \{2\}, \{4\}\rangle$, $\langle\{3\}\rangle$, $\langle\{3\}, \{3\}\rangle$, $\langle\{2\}, \{2\}\rangle$, $\langle\{4\}, \{4\}\rangle$, $\langle\{3\}, \{3\}\rangle$, and $\langle\{2\}, \{2\}\rangle$. The update patterns cover all the $2^4 = 16$ states of the ACA.

Definition 2. *A CA rule* R *is an* **irreversible rule** *if there is at least a state of ACA, designed with R, can never be cyclic for any update pattern. Otherwise,* R *is a* **reversible rule**.

For example, rule 77 (01001101) in null-boundary condition is an irreversible rule. The all-0 state of rule 77 ACA is acyclic for all possible update patterns. On the other hand, rule 60 is a reversible rule in null and periodic boundary condition (Fig. 2). The following theorem [4] characterizes the irreversible rules in periodic boundary condition.

Table 2. Irreversible ACA rules in periodic boundary

0	2	4	6	8	10	12	14
16	20	24	28	64	66	68	70
72	74	76	78	80	84	88	92
141	143	157	159	173	175	189	191
197	199	205	207	213	215	221	223
229	231	237	239	245	247	253	255

Theorem 1. *An ACA rule R is irreversible if and only if all-0 or all-1 state of ACA, designed with R in periodic boundary condition, is acyclic for all possible update patterns.*

Corollary 1. *[4] An ACA rule R is irreversible if (i) the RMTs 0, 2, 7, and either RMT 3 or 6 of R are 1, or (ii) the RMTs 0, 5, 7, and either RMT 1 or 4 are 0, in periodic boundary condition.*

There are (*i*) 24 rules where RMTs 0, 2, 7, and either RMT 3 or 6 are 1, and (*ii*) 24 rules where RMTs 0, 5, 7, and either RMT 1 or 4 are 0. The list of such 48 irreversible ACA rules are noted in Table 2. The rest 208 rules are reversible. Each of these can design reversible ACA in periodic boundary condition for some update patterns. The following theorem [4] characterizes the irreversible rules in null boundary.

Theorem 2. *An ACA rule R is irreversible if and only if all-0, all-1 or* $10101\cdots1$ *state of ACA, designed with R in null boundary condition, is acyclic for all possible update patterns.*

Corollary 2. *[4] An ACA rule R is irreversible if (i) the RMTs 0 and 2, and either RMT 3 or 6 of R are 1, or (ii) the RMTs 0, 1, 3, 4, 5, 6 and 7 are 0, or (iii) the RMTs 0 & 2 are 0, 5 & 7 are 1, and if RMT 1 is 0 or RMT 3 is 1, then either RMT 4 is 0 or RMT 6 is 1 ACA in null boundary condition.*

Therefore, in null boundary, there are (*i*) 48 irreversible ACA rules while RMTs 0, 2, and either 3 or 6 are 1, (*ii*) 2 irreversible ACA rules while RMTs 0, 1, 3, 4, 5, 6 and 7 are 0, and (*iii*) 9 irreversible ACA rules while RMTs 0 and 2 are 0, RMTs 5 and 7 are 1, and if RMT 1 is 0 or RMT 3 is 1, then either RMT 4 is 0 or RMT 6 is 1. All such (59) irreversible rules are listed in Table 3. The rest 197 rules are reversible in null boundary.

However, the reversibility of ACA depends not only on the rule, but also on update patterns. For example, rule 60 ACA can be irreversible (Fig. 1), as well as reversible (Fig. 2) depending on the update patterns. Since the ACA cells can update independently, it can not be predicted in advance that ACA designed with a reversible rule is reversible.

Theorem 3. *[4] It is impossible, in general, to synthesize reversible an 1-d ACA.*

The following section investigates the update patterns for the cycles of reversible ACA.

Table 3. Irreversible ACA rules in null boundary

0	4	13	15	29	31	45	47
61	63	69	71	77	79	85	87
93	95	101	103	109	111	117	119
125	127	141	143	157	159	160	168
170	173	175	189	191	197	199	205
207	213	215	221	223	224	229	231
232	234	237	239	240	245	247	248
250	253	255					

4 Identifying Update Pattern for a Cycle

The reversible rules require different sets of update patterns for reversible ACA. Even, for a particular reversible rule, different sets of update patterns can be identified that result in reversible ACA. The following theorem [4] characterizes the states forming a cycle of ACA.

Theorem 4. *A sequence of l distinct n-bit CA states $\langle S_1, S_2, \cdots, S_l \rangle$ form a cycle of length $l \geq 2$ if, $\forall i$ $(1 \leq i \leq n)$, the i^{th} bit of the CA states does not flip or flips even number of times.*

To get a cycle of the reversible ACA, an update pattern along with an initial state is required. Since the states of a cycle of length l are to be distinct, the update pattern should be designed in such a way that at least one bit of a state flips to get the next state. Moreover, in any sub-sequence of states, the bits of states are not to be flipped in even number of times (Theorem 4). If they flip, the l states can not be distinct.

Therefore, distinct states depend not only on the update pattern, but also on the initial state. Because, the initial state may not allow an arbitrary cell of an ACA to flip. Rule 51 (Table 1) is the only rule that always allows a cell to flip its state when updated. So, rule 51 ACA do not depend on the initial state to form a cycle. The following rule is designed to generate an update pattern for a cycle of length 2^i ($1 \leq i \leq n$, n is the number of ACA cells) while updating a single cell at a time.
To get a cycle of length 2^i $(1 \leq i \leq n)$ of rule 51 ACA, we form a sequence of i cells, to be updated, arbitrarily. Then we start with an arbitrary state and update $(2^{j-1})^{th}$ state by updating j^{th} cell $(1 \leq j \leq i)$ of the sequence. We repeat the update of j^{th} cell after each 2^j states, where $j < i$. The next step is the update of i^{th} cell after 2^{i-1} states to get a cycle of length 2^i.

Example 1. To design a full length cycle for 4-cell rule 51 ACA ($length = 2^4$), all the cells are to be updated in some sequence. Consider, the sequence of update is $SEQ = \langle 1, 2, 3, 4 \rangle$ and the initial state is 0100. Each j^{th} cell of SEQ is selected for the first time to update $(2^{j-1})^{th}$ state. Hence, to get the second state, the first bit of the initial state ($(2^{j-1})^{th}$ state, where $j = 1$) is updated. Similarly, the second, third and fourth cells are selected for the first time to update the second, fourth and eighth states respectively. The first cell is again selected to update third, fifth, and all odd states (that is, after each 2^j states where $j = 1$). After the first time update, the second and third cells are selected repeatedly to update after every 2^2 and 2^3 states respectively. The last cell is updated for the second time after 2^3 states (2^{i-1} states where $i = 4$) to complete the cycle. Therefore, the sequence of states in the cycle is ⟨0100, 1100, 1000, 0000, 0010, 1010, 1110, 0110, 0111, 1111, 1011, 0011, 0001, 1001, 1101, 0101, 0100⟩. The update pattern is ⟨{1}, {2}, {1}, {3}, {1}, {2}, {1}, {4}, {1}, {2}, {1}, {3}, {1}, {2}, {1}, {4}⟩. Here, the update pattern is independent of the initial state, but depends on SEQ (the update pattern and cycle of rule 51 ACA are same for both the boundary conditions).

The cycles can also be formed by updating multiple cells simultaneously.

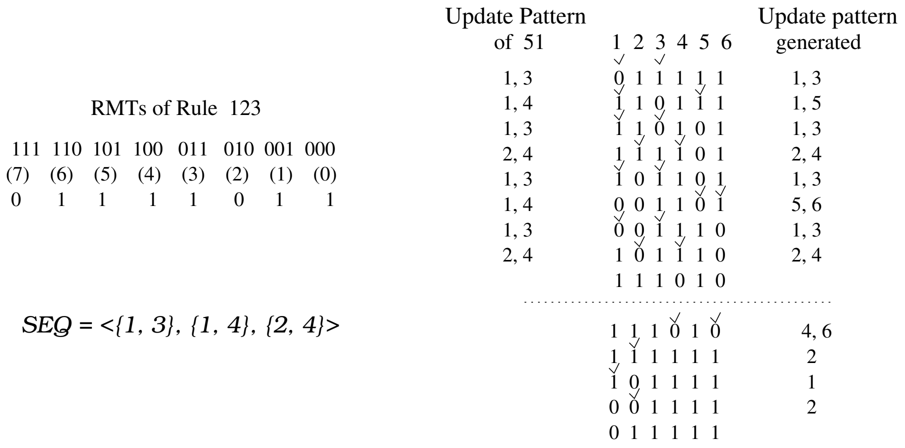
Theorem 5. *Rule 51 ACA with n cells can form a cycle of maximum length 2^{n-m+1} if m cells ($1 \leq m \leq n$) are updated simultaneously.*

In such case, the same rule of single cell update to get a cycle can be followed with an exception that each entry in the sequence of cells (SEQ in Example 1) in a set of m cells.

Example 2. Let us consider, $n = 4$ and $m = 2$. To get an 8 length (2^{n-m+1}) cycle of the ACA, a sequence $SEQ = \langle\{1, 2\}, \{2, 3\}, \{3, 4\}\rangle$ of cells is formed arbitrarily. Consider that the initial state is 0100. The first and second bits are updated to generate the second state (1000). Similarly, the cells of second and third entries of SEQ are selected to update the second and forth states. Like Example 1, the cells of first set ({1, 2}) are repeatedly selected to update the odd states. The cells of second set ({2, 3}) are selected again to update the sixth state. Therefore, a sequence ⟨0100, 1000, 1110, 0010, 0001, 1101, 1011, 0111, 0100⟩ of states is obtained and the update pattern is ⟨{1, 2}, {2, 3}, {1, 2}, {3, 4}, {1, 2}, {2, 3}, {1, 2}, {3, 4}⟩.

The update rule, designed for rule 51 reversible ACA, guides us to develop Algorithm 1 which finds the update pattern for a cycle of the reversible ACA. It first forms a sequence of i unique sets (arbitrarily). The sets are also designed arbitrarily with m ACA cells per set. If no bit flips during the update of a set of m cells, another set of m cells is searched so that at least one bit flips. While 2^i states are covered but no cycle is formed, the algorithm attempts to form a cycle with fewer states.

Algorithm 1. FindACACycle
Input: R (rule), n (# cells), 2^i (cycle length, $1 \leq i \leq n$), S (initial state), m (# cells updated in each step)
Output: Update pattern with cycle length
Step 1: Form a sequence, SEQ of i unique sets of m ACA cells, arbitrarily.
Step 2: Load the ACA, designed with R, with S.
Step 3: For $k = 1$ to 2^i repeat Step 4 to Step 9.



RMTs of Rule 123

111	110	101	100	011	010	001	000
(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)
0	1	1	1	1	0	1	1

SEQ = <{1, 3}, {1, 4}, {2, 4}>

Update Pattern of 51	1 2 3 4 5 6	Update pattern generated
1, 3	0 1 1 1 1 1	1, 3
1, 4	1 1 0 1 1 1	1, 5
1, 3	1 1 0 1 0 1	1, 3
2, 4	1 1 1 1 0 1	2, 4
1, 3	1 0 1 1 0 1	1, 3
1, 4	0 0 1 1 0 1	5, 6
1, 3	0 0 1 1 1 0	1, 3
2, 4	1 0 1 1 1 0	2, 4
	1 1 1 0 1 0	
	1 1 1 0 1 0	4, 6
	1 1 1 1 1 1	2
	1 0 1 1 1 1	1
	0 0 1 1 1 1	2
	0 1 1 1 1 1	

Fig. 3. Generation of cycle for rule 123 ACA ($m = 2$)

Step 4: If $k = 2^{j-1}$ $(1 \leq j \leq i)$, select the j^{th} set of the SEQ.
If $k = 2^i$, select the i^{th} set of SEQ.
If $k = 2^{j-1} + p * 2^j$ (p is a positive integer and $1 \leq j < i$) select j^{th} set.
Step 5: Update ACA cells of the selected set.
Step 6: If no cell flips during update, find a set of m cells so that
a) at least one cell flips, and b) the generated state is unique.
Otherwise, goto Step 9.
Step 7: If no such set is found in Step 6, goto Step 14.
Step 8: Update the ACA cells according to the set, designed in Step 6.
Step 9: Print the ACA cells that are updated to generate the next state of k.
Step 10: If no cycle is formed, identify the bits of $2^i + 1$ state which differ from the initial state, S. Otherwise, go to Step 15.
Step 11: Update the ACA cells to flip the identified bits.
Step 12: If few cells flip, print those. Update the nearest cells of rest bits (one-by-one or more than one at a time) so that the S is reached within few steps.
Step 13: If cycle is formed, goto Step 15.
Step 14: Print 'Cycle is not possible', and exit.
Step 15: Print the length of cycle and exit.

Example 3. Let us consider, $R = 123$, $n = 6$, cycle length $= 8$ (2^3), $S = 011111$ and $m = 2$. Formation of cycle following Algorithm 1 is shown in Fig. 3. Firstly, a sequence of 3 sets $SEQ = \langle\{1, 3\}, \{1, 4\}, \{2, 4\}\rangle$ is formed arbitrarily (Step 1). The ACA is designed with rule 123 in null boundary condition. To get the next state of 011111, the first and third cells are updated (Steps 4 and 5). In Fig. 3, update pattern of rule 51 ACA is noted on the left side of the states, and the update pattern generated for rule 123 is shown on the right side. To update the second state (similarly sixth state), according to the update pattern of rule 51 ACA, the set $\{1, 4\}$ is selected. Since no cell flips following RMTs of rule 123, another set $\{1, 5\}$ is searched (Step 6). After generation of 8 states, cycle is not formed. So, another 4 states are generated to form a cycle (Steps 10 – 12). Therefore, length of cycle is 12.

We have experimented with different reversible rules. It is found that for a number of reversible rules, update pattern can be extracted utilizing Algorithm 1 to get a full length cycle (by updating a single cell at a time). Few of such rules are: 3, 19, 35, 83, 115, 131, 147, 163, 179, 211, 243.

5 Pattern Generation by ACA

The states generated by ACA can be considered as binary patterns. So, the ACA can be treated as a pattern generator. If the ACA follow update pattern produced by Algorithm 1, it can be predicted that a specific Hamming distance between two consecutive patterns in the sequence of states is always maintained. Such patterns can be utilized in various fields, like generating Hamming code, testing VLSI circuits, etc. However, these applications demand hardware realization of ACA, so that pattern generation can be fast and cost effective.

For hardware realization of ACA, a cell is to be implemented in hardware. To do this, we need a combinational logic for implementing the next state logic (rule), and a flip-flop to store present state of the cell. Such a cell is to be

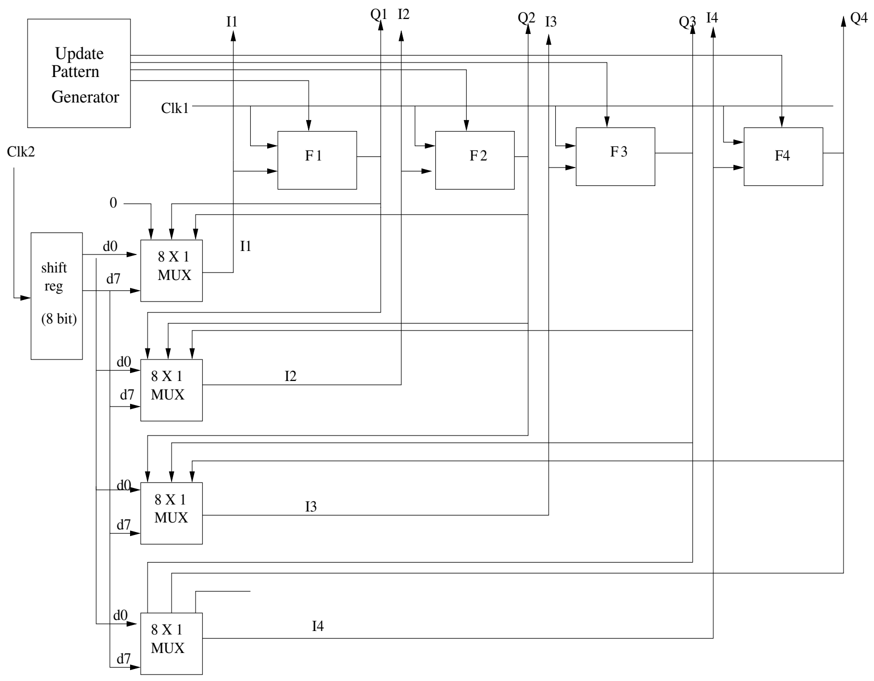


Fig. 4. The 4-cell ACA hardware

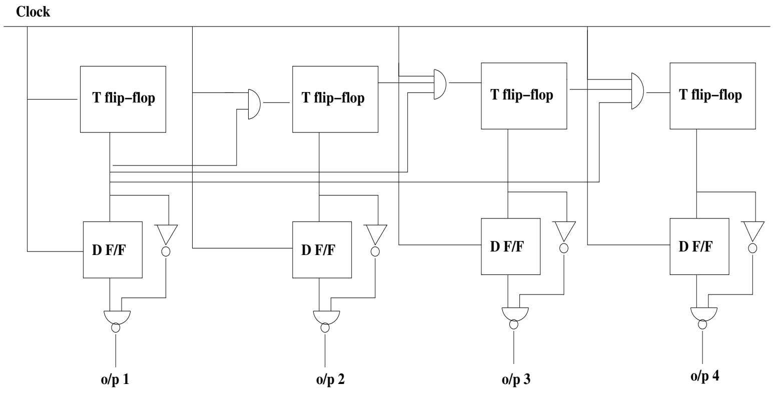


Fig. 5. Update pattern generator

placed into n positions to realize n-cell ACA. However, to optimize the space requirement, we can use an 8-bit shift register to store the rule, and a multiplexer (MUX) at each cell position to output state of the cell. Such implementation also enables us to design ACA with any rule.

The detail hardware of a 4-cell ACA is shown in Fig. 4. The rule is kept in the shift register, D flip flops (F1, F2, F3 and F4) are used to store the states of cells, an 8×1 MUX selects an RMT to get the next state, and *Update Pattern Generator* generates update pattern for the ACA. The logic diagram of update pattern generator is noted in Fig. 5. The generator is designed using T and D flip-flops.

We have simulated the logic diagram (Fig. 4 and Fig. 5) of ACA in FPGA (Field-Programmable Gate Array). Device utilization of such implementation of 4-cell and 16-cell ACA is reported in Table 4.

Table 4. Device utilization of 4-cell and 16-cell ACA

# cells	Name of unit	Unit Utilized	Available Unit	% of Utilization
4	Number of Slices	14	1920	0
	Number of Slice Flip Flops	20	3840	0
	Number of 4 input LUTs	25	3840	0
	Number of IOs	8	-	-
	Number of bonded IOBs	8	173	4
	Number of GCLKs	2	8	25
16	Number of Slices	56	1920	2
	Number of Slice Flip Flops	56	3840	1
	Number of 4 input LUTs	108	3840	2
	Number of IOs	20	-	-
	Number of bonded IOBs	20	173	11
	Number of GCLKs	2	8	25

6 Conclusion

We have addressed the reversibility of ACA in this paper. The CA rules are classified as reversible and irreversible. A reversible ACA may be synthesized with a reversible rule if certain *update pattern* is followed. The ACA can be utilized as an efficient pattern generator for various applications like VLSI circuit testing, Hamming code generation, etc. The paper has finally proposed a hardware realization of ACA, which can be utilized as a cost effective pattern generator. However, a new research on the reversibility of ACA may be initiated considering the random update of cells.

References

1. Amoroso, S., Patt, Y.N.: Decision procedures for surjectivity and injectivity of parallel maps for tesselation structures. Journal of Computer and System Sciences 6, 448–464 (1972)
2. Das, S., Sikdar, B.K.: Classification of *CA* Rules Targeting Synthesis of Reversible Cellular Automata. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 68–77. Springer, Heidelberg (2006)
3. Lee, J., Peper, F., Adachi, S., Morita, K., Mashiko, S.: Reversible Computation in Asynchronous Cellular Automata. In: Calude, C.S., Dinneen, M.J., Peper, F. (eds.) UMC 2002. LNCS, vol. 2509, pp. 220–229. Springer, Heidelberg (2002)
4. Sarkar, A., Mukherjee, A., Das, S.: Reversibility in asynchronous cellular automata. Accepted in Complex Systems (2012)
5. Toffoli, T.: Computation and construction universality of reversible cellular automata. J. Comput. System Sci. 15, 213–231 (1977)
6. Wolfram, S.: Theory and applications of cellular automata. World Scientific, Singapore (1986) ISBN 9971-50-124-4 pbk