Chapter 7

A Few Applications of 2D CA

7.1 Introduction

In the present era of submicron technology, new chips with wider functionality and better throughput are being designed, fabricated and marketed every year. As a result, the design community is under constant pressure to reduce the design turn around time and they look for regular, modular and cascadable building blocks for realizing a VLSI circuit. Further, cost of testing VLSI circuits is becoming prohibitively large both in terms of time and money. CA has been employed in a number of VLSI applications [Barde90, Das89, Horte89a, Horte90a, Horte90b, Chowd92d, Chowd94c]. In this chapter, we have explored a few applications of 2D CA based on the analytical studies proposed by us in earlier chapters.

7.2 Text Compression

In this era of signal processing, alphabets are considered as real life signals and online text compression is the need of the day. In this section we are utilizing rule 170P of 2D CA for text compression.
CHAPTER 7. A FEW APPLICATIONS OF 2D CA

We consider a $2 \times 4$ 2D CA. The experimental results have shown that for $m = 2$ and $n = 4$, the dimension of kernel = 4, that is, in the state transition diagram, each reachable state is having $2^4 = 16$ predecessor states. Fig: 7.1, gives the STD for such a 2D CA and we see that ASCII equivalents of all the uppercase and lowercase letters appear in the second quadrant. Each quadrant is having 4 characters. Only uppercase letters appear in the second quadrant of Fig: 7.1 (a, b, c, d, i, j, k, l). Similarly, lowercase letters appears in the second quadrant of Fig: 7.1 (e, f, g, h, m, n, o, p).

To implement our coding and decoding scheme, we will be having 16 blocks of memory locations each for the state from which we can reach to 0 in one step. There will be one main memory block having 16 bytes and all other 16 blocks will be having 4 bytes. Memory blocks (other than main) contains the values of the second quadrant of their respective state. Main memory block will be having values of the states which can reach to state 0 in one step, starting from first quadrant. The starting address of these blocks other than main can be obtained with the help of some hashing technique. The starting address of main memory block is fixed. As most texts are having lowercase letters in abundance than that of uppercase and other special characters, so we give stress on efficient encoding of only lowercase letters. Generally an uppercase letter comes just after the fullstop(.) or full stop and space, so we encode this also with same efficiency. Also the encoding of space, full stop and comma are done with equal efficiency. Uppercase letters other than those starting just after a full stop or full stop and space are encoded with the help of 10-bits and all other special characters are encoded in 13-bit form. But as the probability of existence of these inefficient codeable symbols in a text is extremely less, we will get a good compression ratio.
CHAPTER 7. A FEW APPLICATIONS OF 2D CA

Fig: 7.1 (a)

Fig: 7.1 (b)

Fig: 7.1 (c)

Fig: 7.1 (d)
CHAPTER 7. A FEW APPLICATIONS OF 2D CA

Fig: 7.1 (e)

Fig: 7.1 (f)

Fig: 7.1 (g)

Fig: 7.1 (h)
CHAPTER 7. A FEW APPLICATIONS OF 2D CA

Fig: 7.1 (m)

Fig: 7.1 (n)

Fig: 7.1 (o)

Fig: 7.1 (p)
7.2.1 Encoding

Our encoder will be having a simple logic to separate codeable and non-codeable symbols. The codeable symbols will be given to CAM which apply rule 170P on the binary value of the symbol to produce a number. We access to the memory block (Table 7.1) belonging to this number and search for input symbol. The two bit index of it in that block is stored in the buffer. Then we search this number in the main memory block. We get the 4-bit index of it in that block and store the 1-MSB and 2-LSB's in the buffer. This is how we encode a symbol (Fig: 7.2).

Our logic circuit will allow only lowercase letters and uppercase letters to reach to the CAM. Space will be converted to ' ', full stop will be converted to '@', or '"', CR will be converted to '" and comma will be converted to '[' before passing it to CAM. As in the memory blocks we have put the decimal value of full stop in place of '@' and '"', the comma in place of '['.
CHAPTER 7. A FEW APPLICATIONS OF 2D CA

, CR in place of “ and space in place of ‘□’, so we will be able to recover the original values in the decoding phase. Whenever an uppercase letter is coming which is not just after a full stop or space we put a 5 bit code 01011( ) in the buffer and then store the code for the symbol in buffer. Whenever a noncodeable character comes we put a 5-bit code 01111(!) before its actual 8-bit representation in the buffer.

Example: 7.1 Now let us try to compress a word with the help of our technique. Let the word be ‘As’. On loading ‘.’ to the encoder, the logic circuit will convert it into ‘@’ or ‘’ (say ‘@’) and supply it to CAM. CAM will apply rule 170P to give 160 (see Fig 7.1 (i)). Now we will search the decimal value of input symbol(‘.’) in the memory block belonging to 160 and find its index which is 00. Then we search 160 in the main memory block and find out its index, which is 1000. Now we omit the 3rd bit from right, the resultant bits are 100, so our complete code will be 10000. As ‘A’ is coming after ‘.’, so it is encoded by the same technique as 11000. Next for ‘s’ we get 01100.

7.2.2 Decoding

To decode the compressed text we start taking the 5-bit chunks from the stream of bits. If it is 01111(!) then we just output the next 8-bits from the stream. If it is 01011( ) then we take the next chunk of 5-bits and decode it as an uppercase letter. If it is a full stop then decode it and output it, now take the next chunk and find whether it is a space, if yes then decode it and check for another space and so on. In doing so as we get a non space 5-bit chunk decode it as an uppercase letter. Thus by above scheme we will be able to distinguish between lowercase and uppercase. If code is for lowercase and its MSB is 0 then add binary 100 to 3-MSB’s else if it starts from 1 then
add 1100, and fetch the value of number at this indexed memory location in the main memory block. Now go to the memory block corresponding to this fetched number and fetch the number indexed by the 2-LSB's of the code. Output the binary equivalent of it. In case of uppercase letters if it starts from 0 then add 0000 to 3-MSB's and if it starts from 1 then add 1000 and do the same procedure as for the lowercase letters.

Table: 7.1. Different memory blocks.

| Main 0 | 0  | 5  | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0     | 0  | 80 | 85 | 90 | 95 | 100| 105| 110| 115| 120| 125| 130| 135| 140| 145| 150| 155| 160| 165| 170| 175|
| 0     | 10 | 112| 117| 122| 127| 132| 137| 142| 147| 152| 157| 162| 167| 172| 177| 182| 187| 192| 197| 202| 207|
| 0     | 15 | 113| 116| 123| 126| 131| 134| 139| 142| 145| 150| 153| 158| 161| 166| 169| 174| 177| 182| 185| 188|
| 0     | 20 | 114| 119| 120| 125| 130| 135| 140| 145| 150| 155| 160| 165| 170| 175| 180| 185| 190| 195| 200| 205|
| 0     | 25 | 115| 118| 123| 126| 131| 134| 139| 142| 145| 150| 153| 158| 161| 166| 169| 174| 177| 182| 185| 188|

7.2.3 Experimental Data

Following text file was taken for experiment:

File:

This document provides supplementary information on configuring Btrieve for use with Visual Basic. Before you import, export, or attach Btrieve tables with Visual Basic, please read this file.

Contents:

General Considerations

Using Compressed Data Files

Using Btrieve in a Multiuser Environment
CHAPTER 7. A FEW APPLICATIONS OF 2D CA

Setting Btrieve Options in WIN.INI

Configuring Novell Network LAN Manager (NLM).

General Considerations

When using Btrieve data with Visual Basic, keep the following considerations in mind:

Btrieve data files must be in version 5.1x format.
You must have the data definition files FILE.DDF and FIELD.DDF, which tell Visual Basic the structure of your tables. These files are created by Xtrieve or another .DDF file-building program.
You must have the Btrieve for Windows dynamic-link library WBTRCALL.DLL, which isn't provided with Visual Basic. This file is available with Novell Btrieve for Windows, Novell NetWare SQL, and some other products for Windows that use Btrieve.
You can't use Btrieve files that have Xtrieve security. To use data files with Visual Basic, disable Xtrieve security.

Using Compressed Data Files

If you're using compressed Btrieve files, you must be sure that the compression buffer Btrieve is using is adequate for your data. The buffer size must be at least as large as the largest record in your data files.

To ensure proper operation, set the compression buffer size option (/u) in the [btrieve] section of your WIN.INI file. The units for this setting are kilobytes, so if your largest record is 2K, you would add /u:2 to the Btrieve options line in WIN.INI.
For more information on setting options, see "Setting Btrieve Options in WIN.INI" later in this file.

End of File.
CHAPTER 7. A FEW APPLICATIONS OF 2D CA

<table>
<thead>
<tr>
<th>No. of 5 bit codes</th>
<th>No. of 10 bit codes</th>
<th>No. of 13 bit codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1564</td>
<td>125</td>
<td>21</td>
</tr>
</tbody>
</table>

Thus total no. of bits in the compressed file = $1564 \times 5 + 125 \times 10 + 21 \times 13 = 9343$ bits.

Whereas total no of bits in the input file = 13304 bits.

Therefore the compression ratio = $29.77\% = 30\%$.

Running the same input file by the adaptive Huffman coding scheme (variable length coding) the compression ratio is little over 30%.

7.3 VLSI Testing

Test technology has failed to match the growth of circuit complexity and size. The only viable option available to the design community is to employ DFT (Design For Testability) techniques. Notable among the DFT techniques is the full scan design. However, with the growth of circuit size, serial scan in and scan out of larger volume of test and response data have become a major bottleneck along with the larger test circuit overhead. Partial scan techniques attempt to reduce this overhead. In both cases, automatic test pattern generators are used – as a result test generation and test application time becomes significant. Built-In-Self-Test (BIST) structures provide an on-chip test generation and test evaluation methodology with an aim to reduce the drawbacks of DFT techniques discussed previously. In this section, we project 2DCA as a BIST structure for VLSI circuits.
7.3.1 Pseudo-exhaustive Testing

For a complex circuit with a large number of inputs, pseudo-exhaustive testing has been found to be suitable where each of the outputs depend only on a subset of the inputs – this results in a test size much less compared to the exhaustive test size $2^n$ for an $n$-input circuit under test (CUT). Given any group 1DCA, [Das93] presents an algorithm for finding out the bit positions where the pseudo-exhaustive patterns are generated. The same result holds for the 2D CA case also. Moreover, due to the incorporation of nine neighbourhood, we have wide variety of 2D CA characteristic matrices and its factor polynomials – this gives us much wider flexibility to employ 2D CA for pseudo-exhaustive test pattern generation.

**Example: 7.2** Let us consider a $2 \times 4$ 2DCA where each cell is configured with the rule 34 or rule 35. For the cells on the row boundaries (except the cell (1,1) and (2,4)) (i.e. cell (2,1)) – its left dependence comes from the output of cell (1,4) – something like a mesh. The corresponding $T$ matrix is:

$$
T = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

The characteristic polynomial is: $f(x) = (1 + x)(1 + x^2 + x^3)(1 + x + x^4)$

The factors of the characteristic polynomial are:

- $f_1(x) = (1 + x)$
• $f_2(x) = (1 + x^2 + x^3)$

• $f_3(x) = (1 + x + x^4)$

A primitive factor of the characteristic polynomial is: $f_2(x) = (1 + x^2 + x^3)$

As per the algorithm presented in [Das93] the pseudo-exhaustive bit positions are cells (1, 1), (1, 2) and (1, 3).

### 7.3.2 Generation of Pseudo-random test patterns

1D CA has been projected as a pseudo-random pattern generator in a large number of publications. As noted in [Chowd94c], the 2DCA structure is expected to generate much better pseudo random patterns so far as randomness is concerned – this is because of the varieties of neighbourhood which are not available in case of three neighbourhood 1DCA. In [Chowd92d] a variety of tests for randomness, viz. Equidistribution Test, Correlation, etc has been considered [Knuth81]. It has been established that 2DCA is a much better PRPG with superior randomness qualities than 1DCA or LFSR. The 2DCA described in [Chowd94c] is RVN-CA (Reduced Vertical Neighbourhood CA) which is a subset of the general 2DCA we have considered in this paper. Hence, it is natural that we achieve improvement in randomness of the patterns by using the general 2D CA structure. Such 2D CA structures can be effectively employed as multiple parallel pseudo-random pattern generator which provide much better randomness of patterns than RVN-CA, 1D CA or LFSR.

However, some circuits exist which are inherently random pattern resistant and thus require large number of pseudo-random patterns for achieving high fault coverage. So, use of weighted-random patterns has been proposed [Strol91]. Next subsection briefly highlights an elegant scheme for generation of weighted random patterns.
7.3.3 Generation of Weighted Random Test patterns

In this section, we propose a scheme for generating weighted-random patterns using 2DCA. Given the neighbourhood dependencies of the cells and the boundary conditions, we can construct the \( T \)-matrix, thereby generating the entire state transition behaviour of the 2DCA. Given a cycle \( C \) belonging to the state transition diagram of the 2DCA, let \( n \) be the total number of states in \( C \) (that is the cycle length is equal to \( n \)) – for any particular bit \( i \), the probability of 1 being output at that bit is \( n_i/n \), where \( n_i \) is the number of states in \( C \) which have the \( i^{th} \) bit equal to 1 – thus choosing any state of \( C \) as a seed and loading it into the CA we can obtain different probabilities at different bits. If the length of the cycle \( C \) (that is \( n \)) is \( 2^k - 1 \), where \( k \) is the number of bits of the CA, then we can obtain the probability of 1 at any bit to be equal to 0.5 by loading the CA with any non-zero seed. On the other hand, if \( C \) is of non-maximal length, then the probability of 1 will be different from 0.5. Thus, corresponding to each cycle \( C \) in the state transition diagram of any group \( m \times n \) 2DCA we can define the following tuple \( r = (\text{cycle length}, \text{probability of 1 in bit position } (1,1), \cdots, \text{probability of 1 in bit position } (m,n), \text{seed }) \). Thus, if we load the CA with the seed and look at any bit position \((i,j)\) we can distinctly tell the probability of getting 1 in that position – this is possible because of the matrix algebraic characterization of 2DCA. We can generate a program for the 2D CA by specifying the rules, the initial seed and the number of cycles for which the 2D CA has to be evolved. On running the 2D CA with such a program, wide varieties of probability of 1 at different bit positions can be generated. The basic architecture is shown in Figure 7.3.
7.4 Cryptography

Two dimensional binary text or image information is a natural state of 2DCA. For an invertible CA, the state transition diagram consists only of cycles, and hence any state lies on a cycle. Let $c$ be a configuration on a cycle of length $L$. If the CA is evolved for $x$ ($0 \leq x < L$) steps starting with initial state $c$, then a new state $d$ is reached. Given the state $d$, it is possible to get back to $c$ by loading the CA with initial state $d$ and evolving for $(L - x)$ steps. Thus, $d$ can be considered to be the cipher text. The evolution for $x$ steps the enciphering algorithm and the evolution for the next $(L - x)$ steps is used in the deciphering algorithm.

Based on this idea, we now describe an enciphering scheme for two dimensional information. A 2D uniform $m \times n$ CA using only rule 170N is invertible if and only if $(m + 1)$ and $(n + 1)$ are coprime. For the case of CA with rule
171, a sufficient condition for invertibility is that \( m = 2^{x_1} - 1 \) and \( n = 2^{x_2} - 1 \), for some \( x_1 \) and \( x_2 \). Given an \( M \times N \) 2D state, we form a partition of \( M \) and \( N \), of the form \( M = m_1 + m_2 + \ldots + m_{r_1} \) and \( N = n_1 + n_2 + \ldots + n_{r_2} \) such that for each pair \((m_i, n_j)\), \((1 \leq i \leq r_1)\), \((1 \leq j \leq r_2)\), we have either

(a) \( \gcd(m_i + 1, n_j + 1) = 1 \) or

(b) \( m_i = 2^{x_1} - 1 \) \& \( n_j = 2^{x_2} - 1 \)

for some \( x_1 \) & \( x_2 \). This will divide the \( M \times N \) into blocks of \( m_i \times n_j \) grids. We will separately apply rule 170N or 171N on these \( m_i \times n_j \) blocks depending on whether condition (a) or (b) is satisfied. The conditions ensure that each individual transformation is invertible.

Such a CA structure is set up at both the sender and the receiver end. To encipher a given information, we load the \( M \times N \) 2D state into the structure described above. For each \( m_i \times n_j \) information block \( (c_{ij}) \), we evolve for \( x_{ij} \) steps with rule 170N or 171N to get \( d_{ij} \) which is also a \( m_i \times n_j \) block. Let \( l_{ij} \) be the length of the cycle on which \( c_{ij} \) lies. Then alongwith \( d_{ij} \) we also send \( l_{ij} - x_{ij} \) to the receiver. The partition of \( M \) and \( N \), \( l_{ij} \), \( x_{ij} \) are kept secret to the adversary.

### 7.4.1 Complexity and Versatility

For any cryptosystem, the invulnerability is equivalent to showing that breaking the cipher text is computationally infeasible. Thus, a cipher is secure under the intractibility assumption of the problem. However, popular cryptosystems like DES are not altogether secure in this sense. In fact, extensive study of DES has shown some potential weaknesses [Patte87] though uptill now it has not been cryptoanalysed.

We first calculate the size of the key space. It is determined by the following factors:
CHAPTER 7. A FEW APPLICATIONS OF 2D CA

1. For each \( m_i \) and \( n_j \) we have the information matrix of size \( m_i \times n_j \). The possible number of one-dimensional map matrices of dimension \( (m_i n_j \times m_i n_j) \) is \( 2^{(m_i n_j)^2} \). The number of bits required to represent the possible number of one dimensional map matrices is \((m_i n_j)^2 = k_1 \) (say).

2. The basic difficulty presented to the adversary is to guess the partition of \( M \) and \( N \). The number of partitions of a positive integer \( K \), \( P_K \) is the coefficient of \( x^K \) in the expansion of \((1 + x + x^2 + x^3 + \cdots)(1 + x^2 + x^4 + x^6 + \cdots)(1 + x^3 + x^6 + x^9 + \cdots) \cdots (1 + x^i + x^{2i} + x^{3i} + \cdots) \cdots = (1 - x)^{-1}(1 - x^2)^{-1}(1 - x^3)^{-1} \cdots (1 - x^i)^{-1} \cdots = 1/((1 - x)(1 - x^2)(1 - x^3) \cdots (1 - x^i) \cdots)

Thus an upper bound on the number of possible divisions of the \( M \times N \) grid is \( P_M P_N \), which is certainly a large number. Suppose \( k_1 \) number of bits are required to fix the partition matrix. Now, for each partition matrix, since the cycle length will be varying, we have to keep provision of \( m_i n_j \) bits (to take care of the maximum length cycle). Thus, the length of the key for each partition matrix is \( (k_1 + l_{ij} - x_{ij} + 1) \) bits – the extra one bit is required to indicate whether rule 170A T or rule 171A T has been applied.

7.5 Conclusion

In this chapter we extended the theory of 2D CA to other application areas. A novel scheme for Text Compression was presented. We projected 2D CA as a BIST (Built-In-Self-Test) structure for VLSI circuits. We also presented Cryptographic application of 2D CA in this chapter.