View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by CiteSeerX

Research on Multi-Dimensional Cellular Automation Pseudorandom Generator of LFSR Architecture

Yong Wang[1,2], Dawu Gu[2], Junrong Liu[2], Xiuxia Tian[1] and Jing Li[1]

[1]Shanghai University of Electric Power,
[2]Shanghai Jiao Tong University
China

1. Introduction

Linear feedback shift register (LFSR) is widely used in pseudorandom generator. Chien described an optimized BIST scheme which has a configurable 2-D LFSR structure and presented a synthesis procedure for this test generator. Experimental results show that the hardware overhead is considerably reduced compared with 2-D LFSR generators [1]. Erik H. presents a new test response compaction technique with any Number of Unknowns using a new LFSR Architecture in the test response bits [2].

Cellular automata (CA) are used in modern cryptography. R. Breukelaar research on the way using a genetic algorithm to evolve behavior in multi-dimensional cellular automata [3]. Sheng-Uei Guan proposed a variation of two-dimensional (2-D) cellular automata (CA) variation with asymmetric neighborship for pseudorandom number generation [4]. S. Nandi deals with the theory and application of Cellular Automata (CAI for a class of block ciphers and stream ciphers. Which has been proposed as running key generators in stream ciphers. Both the schemes provide better security against different types of attacks [5].

We research on three-Dimensional CA algorithm and LFSR hardware device pseudorandom g feasibility and efficient generator [6]. In order to find feasibility and efficient of multi-dimensional or multi-rank cellular automata (CA) algorithm with LFSR, we design more ways to test. The experiment result show they also can provide better pseudorandom key stream and pass the FIPS 140-1 standard tests.

2. Multi-dimensional cellular automation definition

2.1 One-dimensional cellular automation definition

A simple one dimensional cellular automation (CA) is an 8-cell array. For examples, when the CA is initialized to 0100 1011, each cell changes its state based on some rule. One possible rule, for example, CA could be defined by the immediate neighborhood of each cell. The cell depends on the current value and the values of its left and its right cells [7]. The CA is assumed to be connected in a circle, so the cell to the left of cell 0 is cell 7,and the cell to the right cell 7 is cell 0.The CA Neighborhood is as follows:101 010 100 001 010 101 011 110.A

specific rule for this neighborhood could be:Neighborhood 000 001 010 011 100 101 110 111. New state: 0101 0110.
Rules for this type of CA are identified by converting the new state bits into a decimal number. The new state for the preceding rule is 0101 0110, which in decimal is 86.Applying this rule to the initial CA results in the succeeding CA value becoming 1001 0111.A CA can be used to generate random bits by selecting a rule, a CA size, an initial seed and the cell to provide to the random bit [1]. For example, we choose the 7th cell to produce the random bit. The first 5 step to produce the random bits are 10100 [7].

2.2 Two-dimensional cellular automation definition

A two-dimensional cellular CA offers a better random number generator at the expense of additional complexity. A two-dimensional cellular CA is an array of one-dimensional cellular, where a cell's value is updated by some function of this current neighborhood which consist of the cells above, below, to the right, and to the left of the target cell [1]. A general rule structure is defined as follow:

$$\begin{gathered} S_{i,j}(t+1) = Xxor\left[C \times S_{i,j}(t)\right]xor\left[N \times S_{i-1,j}(t)\right] \\ xor\left[W \times S_{i,j-1}(t)\right]xor\left[S \times S_{i+1,j}(t)\right]xor\left[E \times S_{i,j+1}(t)\right] \\ \text{Where } X,C,N,W,S,E \in \{0,1\} \end{gathered} \tag{1}$$

If X is 1, this is a nonlinear rule, otherwise, it is a linear rule. C,N,W,S and ,E represent the center, north, west, south, and east cells, respectively. The cells that participate in updating the center cell are determined by values of these five variables. The CA is assumed to be connected in a row circle and column circle as the one-dimensional cellular. A simple two-dimensional 3*3cellular automation is like a double circle array. So the cell to the north of cell [0,0] is cell [2,0],and the cell to the west cell [0,0] is cell [0,2].
If N is 1,then the north cell is used to update the center cell ,The values of all six variables are used to identify each possible rule. If (X,C,N,W,S,E)=(001011), then the rule is defined as Rule 11, because 1011 is decimal 11. the rule looks like this:

$$\begin{gathered} S_{i,j}(t+1) = \left[N \times S_{i-1,j}(t)\right] \\ xor\left[S \times S_{i+1,j}(t)\right]xor\left[E \times S_{i,j+1}(t)\right] \end{gathered} \tag{2}$$

A random stream is generated by assigning a rule to each cell, initializing the CA to a random state, and running the CA using a center cell to produce the bit stream [1], For example, the CA is initialized to {001;000;010},each cell is assigned Rule 11.The value in the center cell is applied to construct the random stream. The cells are randomly initialized, after four steps, the random-bit stream is .01101 [7].

2.3 Multi-dimensional cellular automation architecture

A three-dimensional cellular CA is a cube of circle two-dimensional cellular. A cell's value is updated by some function of its neighborhood, which consists of the cells above, below, to the right, to the left, to the up and to the down of the target cell. A general rule structure can be defined as follows:

$$S_{i,j,k}(t+1) = Xxor\left[C \times S_{i,j,k}(t)\right]$$

$$xor\left[N \times S_{i-1,j,k}(t)\right]xor\left[W \times S_{i,j-1}(t)\right]$$

$$xor\left[S \times S_{i+1,j}(t)\right]xor\left[E \times S_{i,j+1,k}(t)\right]$$

$$xor\left[NW \times S_{i-1,j-1}(t)\right]xor\left[NE \times S_{i-1,j+1}(t)\right]$$

$$xor\left[SW \times S_{i+1,j-1,k}(t)\right]xor\left[SE \times S_{i+1,j+1,k}(t)\right]$$

$$S_{i,j,k}(t+1) = S_{i,j,k}(t+1)xor\left[U \times S_{i,j,k+1}(t)\right]$$

$$S_{i,j,k}(t+1) = S_{i,j,k}(t+1)xor\left[D \times S_{i,j,k-1}(t)\right] \quad (3)$$

If an element is in the corner near the border, its neighbor can't be found. We can imagine the three-dimensional cellular is a cube of circle connection with beginning element and last element. So we can find every element neighbor in the imaged circle. For example element s[0][0][0] up neighbor is s[2][0][0] and it's left neighbor is s[0][2][0].
A simple three-dimensional 3*3*3 cellular automation is shown as Figure 1:

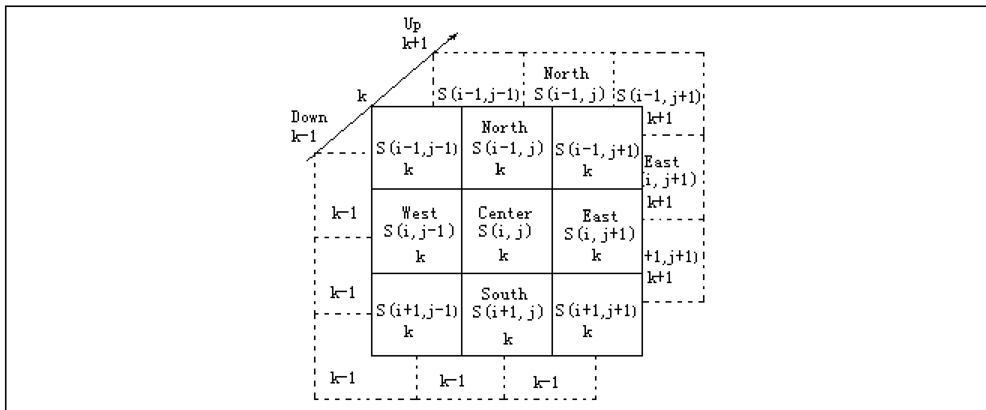

Fig. 1. Three-dimensional cellular automation architecture

Three-dimensional rule array has 0 or 1 variables. We changed the three-dimensional rule array into one-dimensional linear array. For example, element rule[0][0][0] is the first element of linear rule and rule[0][0][1] is the second element of linear rule.
The values of all twenty-seven variables are applied to identify each possible rule.
If the first rule is equal to binary (110 1001 1011 0100 0101 1001 0100), then the rule is also defined as rule 110839188, because the binary value is decimal 110839188. The first bit of the binary 1 is equal to the element of 3 rule array rule[0][0][0]. When the bit of binary rule is zero, it means the corresponding element don't affect the random stream.

If the second rule is equal to binary (111 1111 1111 1111 1111 1111 1111), then the rule is also made as Rule 134217727, because the binary value is decimal 134217727. The first bit of the binary 1 is equal to the element of 3 rule array rule[0][0][0].
A random stream is generated by assigning a rule to each cell, initializing the CA to a random state, and running the CA using a center cell to produce the bit stream.CA is initialized to three-dimensional 3*3*3 array s[i][j][k]={0}, and the initial data is as follows: s[0][0][0]=1;s[0][2][1]=1; s[1][0][1]=1; s[2][1][1]=1;s[2][2][2]=1;
For example, each cell is assigned the first rule. The value s[1][1][1] in the center cell is used to construct the random stream.
Multi-dimensional cellular automation structure is much complicated than 3-dimesinal cellular automation. For example one 4-dimensional and 3 rank cellular automation is assumed to be two cubes connected with each other. We changed the two cubes into linear array by assigning the first cube's then another cube's element to the array in sequence. The Multi-dimensional cellular automation structure is shown in Figure 2:

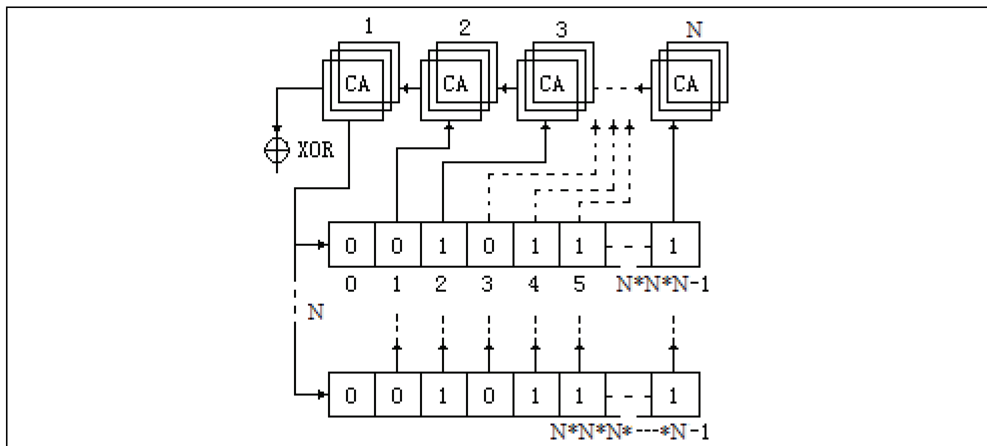

Fig. 2. Multi-dimensional cellular automation architecture

Then by rotating the first cube we can get different sequence. All the cubes is connected with each other, the output terminal will affect with LFSR output bits.

3. Multi-dimensional cellular automation pseudorandom generator of LFSR architecture

3.1 LFSR architecture

The most familiar method is to use a hardware device called a linear feedback shift register (LFSR). Shift register is a very useful, because registers are packed in CPU with very high access speed. This device stores a set of bits. The typical registers are 8-bits and 32-bits registers used in P4 CPU. The bits in register can shift to right by using assembly language instruction. Shift logical right instruction can shift each bit to the right and the leftmost bit is zero. The needed shift register's function is shifting each bits to the right, and the rightmost bit is lost, leftmost bit is replaced with the input bit. Linear feedback shift register (LFSR) choose some bits from the shift register and XOR he input shift in bits. The XOR result is the shift in bits [7].

A general LFSR can be represented by a function of its stored bits and the connections to the shift-in bits. The function is:

$$b_n = c_1 b_1 XOR c_2 b_2 XOR \cdots XOR c_n b_n \quad (4)$$

3.2 Multi-dimensional cellular automation with LFSR architecture

The Multi-dimensional cellular automation LFSR combine the dimensional cellular method and linear feedback shift register (LFSR) to create continues stream bits. The method can be represented by a function of its stored bits and the connections to the shift-in bits from both selected bits and CA random-bit stream. The function is

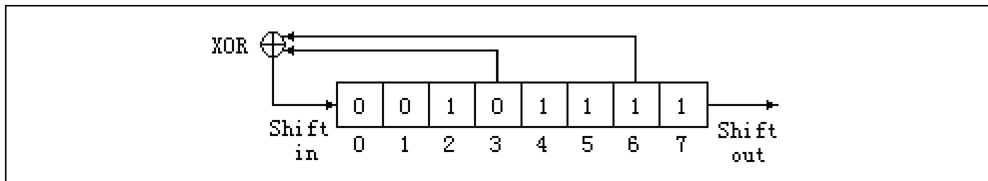

Fig. 3. LFSR hardware device architecture

For example, if 8-bits LFSR is set 0010 1111,in which the shift XOR function among the Shift in bit, cell 3 and cell 6. The LFSR shift procedure includes three steps: the first step is calculating cell 3 and cell 6 by XOR to create shift in bit. The second step is shift bit including shift in bit from left to right [1]. The last step is filling the shift out cell with the cell 7.Then circulate this three steps until the counter is back to 0 [6].

$$b_n = CArb XOR c_1 b_1 XOR c_2 b_2 XOR \cdots XOR c_n b_n \quad (5)$$

CArb stands for cellular automation random bit .Where, if the bit is selected for the XOR operation; otherwise, it is 0. For example the operation of a simple 8-bit LFSR, in which the shift in is the XOR of cells 3 and 6 with s[1][1][1]is shown in Figure 4.

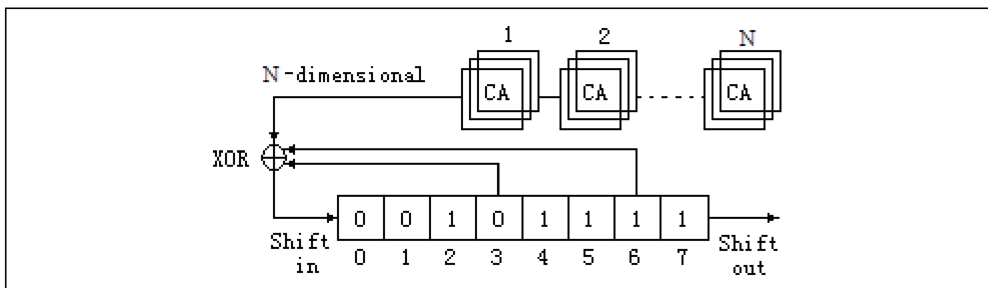

Fig. 4. Multi-dimensional cellular automation pseudorandom generator of LFSR architecture

If 8-bits three-dimensional cellular automation LFSR is set 0010 1111 same as to the CA Rule decimal 110839188, in which the shift XOR function among the Shift in bit, cell 3 and cell 6 with the CA random bit. The procedure is illustrated in Table 2.Then initial data in s[i][j][k] is as CA example.

When loop counter is 20000 times, the 4-dimensional and 3-rank cellular automation LFSR algorithm can create stream bits including 10002 bit zero and 9998 bit one. The result can pass the bit test.

4. Pseudorandom generator program

4.1 Macro definition

```
#define N 3 //rank N=3,4,5,6,7,8,9,10,11,12,13,14
#define RuleInitKey 110839188
#define RuleInitFullKey 134217727
#define E 5
#define Nd N*N*N*N*N
#include<stdio.h>
```

4.2 Variables definition

```
int s[N][N][N]={0},t[N][N][N]={0},i,j,k,count,cx;
int se[Nd]={0},ch,ecx=0,chJudge=N*N*N;//number of N in se array = E dimensional
long low=N*N*N,high=Nd;
int rule[3][3][3]={0},ruleJudge[27]={0};
unsigned long  ruleKey=RuleInitKey,ruleKeyTemp=0;
int ic,jc,kc,icc,jcc,kcc;//near i j k
int LFSR[8]={0,0,1,0,1,1,1,1},shiftIN,shiftOUT,loop;
int nsi=0,ssi=0,wsj=0,esj=0,kup=0,kdn=0,x=1;
// Variables definition for mono test
int number1=0,number0=0;
// Variables definition for poker test
int n[16]={0},splitCounter=0,number=0,pokerCounter=0;
double sumNIsquare=0.0,pokerX=0.0;
// Variables definition for run test
int  runNum[20000]={0},run[20]={0},runLoop,runCounter=1,run1Counter=0,run0Counter=0;
        FILE *fp;
```

4.3 Initialize the data and rule

```
//----------------init data--------------------------
if((fp=fopen("nn_data.txt","a+"))==NULL)
        printf("Can't open file!\n");
   fprintf(fp,"The %d dimensional array with %d depth\n",E,N);
//init the cube data
s[0][0][0]=1;
s[0][2][1]=1;
s[1][0][1]=1;
s[2][1][1]=1;
s[2][2][2]=1;
 //init the rule:  N dimen rule is  i j k sequecnce  using hex
// 110 100 110 110 100 010 110 010 100 =>69B 4594H=>1 1083 9188 D
 count=26; // [0..26]
```

```
ruleKeyTemp=ruleKey;
for(k=2;k>=0;k--)
        for(i=2;i>=0;i--)
                for(j=2;j>=0;j--)
                {
                        rule[i][j][k]=ruleKeyTemp&1;
                        ruleJudge[count]=rule[i][j][k];
                        count--;
                        ruleKeyTemp=ruleKeyTemp>>1;
                       // printf("rule[%d][%d][%d]=%d\t",i,j,k,rule[i][j][k]);
                }
fprintf(fp,"The rule is %ld\n",ruleKey);
for(count=0;count<27;count++)
        fprintf(fp,"%d ",ruleJudge[count]);
        fprintf(fp,"\n");
                //getchar();
```

4.4 N dimensional with N depth cube Using LFSR

```
for(count=0;count<=20000;count++)
{
// loop three count=20000
        //k i j means every element
        ch=N*N*N;
        for(k=0;k<=N-1;k++)
        {
                for(i=0;i<=N-1;i++)
                {
                        for(j=0;j<=N-1;j++)
                        {
//3 cipher Rule is 00101111(          north       south east kdown kup)=decimal Rule 47
//n Rule is 10101111(self center north  west south east kdown kup)=decimal Rule 175
//-----n dimensional-----------
/*
                                        if(i-1<0) nsi=N-1; else nsi=i-1;
                                         if(i+1>N-1) ssi=0;  else ssi=i+1;
//------n dimensional------
                                        if(j-1<0) wsj=N-1; else wsj=j-1;
                                            if(j+1>N-1) esj=0;  else esj=j+1;
                                        if(k-1<0) kdn=N-1;
                                                else kdn=k-1;
                                        if(k+1>N-1) kdn=0;  else kup=k+1
 */
  //----------loop  cube1---------------------
                                        cx=0;
                                        for(kc=-1;kc<=1;kc++)
                                                for(ic=-1;ic<=1;ic++)
```

```
                                    for(jc=-1;jc<=1;jc++)
                                    {
                                        if(i+ic<0)   icc=N-1;
                                        else if(i+ic>N-1) icc=0;
                                            else icc=i+ic;
                                        if(j+jc<0)   jcc=N-1;
                                        else if(j+jc>N-1) jcc=0;
                                            else jcc=j+jc;
                                        if(k+kc<0)   kcc=N-1;
                                        else if(k+kc>N-1) kcc=0;
                                            else kcc=k+kc;
                                        if(ruleJudge[cx++]==1)
                                    t[i][j][k]=t[i][j][k]^s[icc][jcc][kcc];
                                    }
                                        t[i][j][k]=x^t[i][j][k];
//printf("t[%d][%d][%d]=%d",i,j,k,t[i][j][k]);
                                    //getchar();
 /*
t[i][j][k]=x^s[i][j][k]^s[nsi][j][k] ^ s[ssi][j][k]^s[i][wsj][k]^s[i][esj][k];
t[i][j][k]=t[i][j][k]^s[nsi][wsj][k]^s[nsi][esj][k]^s[ssi][wsj][k]^s[ssi][esj][k];
t[i][j][k]=t[i][j][k]^s[i][j][kup]^s[nsi][j][kup] ^ s[ssi][j][kup]^s[i][wsj][kup]^s[i][esj][kup];
t[i][j][k]=t[i][j][k]^s[nsi][wsj][kup]^s[nsi][esj][kup]^s[ssi][wsj][kup]^s[ssi][esj][kup];
t[i][j][k]=t[i][j][k]^s[i][j][kdn]^s[nsi][j][kdn] ^ s[ssi][j][kdn]^s[i][wsj][kdn]^s[i][esj][kdn];
t[i][j][k]=t[i][j][k]^s[nsi][wsj][kdn]^s[nsi][esj][kdn]^s[ssi][wsj][kdn]^s[ssi][esj][kdn];
*/
                        s[i][j][k]=t[i][j][k];
//----------rotate cube-----------
//i<->j  j<->k i<->k
 //s[j][i][k]=s[i][j][k];
                        s[k][j][i]=s[i][j][k];
                        s[j][k][i]=s[i][j][k];
//---------------------- expend n dimensional to higher dimensional-----------------
                        for(ecx=1;ecx<=E-1;ecx++)
                        {
                            if (ch>=low && ch<=high)
                                se[ch++]=s[i][j][k];
// n dimensional cube xor
//printf("ch=%d\tse[%d]=%d\ts[%d][%d][%d]=%d\n",ch,ch,se[ch],i,j,k,s[i][j][k]);
                            if(ch%(chJudge/2)==0)
                            {
                                s[i][j][k]=s[i][j][k]^se[ch];
//printf("ch=%d s[%d][%d][%d]=%d\n",ch,i,j,k,s[i][j][k]);
//getchar();//debug
                            }
                        }
//------n dimensional-------
```

```
//[j][k]=s[nsi][j][k] ^ s[ssi][j][k];
//printf("\t%d ",t[i][j][k]);
//if(j+1>N-1) esj=0;
//else    esj=j+1;
//t[i][j][k]=t[i][j][k] ^ s[i][esj][k];
//printf("%d ",t[i][j][k]);
 //(k-1<0) kdn=N-1;
//else kdn=k-1;
//t[i][j][k]=t[i][j][k] ^ t[i][j][kdn];
//printf("%d ",t[i][j][k]);
//if(k+1>N-1) kdn=0;
//else kup=k+1;
//t[i][j][k]=t[i][j][k] ^ t[i][j][kup];
//printf("\tt[%d,%d,%d]=%d",i,j,k,t[i][j][k]);
                    if(i==N/2&&j==N/2&&k==N/2)
                    {  //LFSR
                            shiftIN=s[N/2][N/2][N/2];
                            shiftIN=shiftIN ^ LFSR[3];
                            shiftIN=shiftIN ^ LFSR[6];
                            shiftOUT=LFSR[7];
                            for(loop=7;loop>=1;loop--)
                                    LFSR[loop]=LFSR[loop-1];
                            LFSR[0]=shiftIN;
//printf("\nshiftIN=%d\t shiftOUT=%d\n",shiftIN,shiftOUT);
//getchar();
```

4.5 Mono bit test program

```
//printf("%d \n",shiftOUT);
                            if(shiftOUT==0) number0++;
                            else number1++;
//poker test
                             splitCounter++;
                            if(splitCounter<=4)
number=number+(shiftOUT << (splitCounter-1));// binary bit shift left
                            else
                            {
//printf("number=%d",number);
//getchar();
                                    n[number]++;
                                    splitCounter=1;
                                    number=0;
                    number=number+(shiftOUT << (splitCounter-1));
                            }
//0110111 run test
                            runNum[runCounter++]=shiftOUT;
                    }
```

```
                    }//end of j
//printf("%d ",j);// endline
              }// end of i
        }//end of k
//printf("\tch=%d",ch);
 // printf("\t%d",count);
        for(k=0;k<=N-1;k++)
              for(i=0;i<=N-1;i++)
                    for(j=0;j<=N-1;j++)
                    {
                          s[i][j][k]=t[i][j][k];
                          t[i][j][k]=0;
                    }
//printf("\n");
//end of the array
}
fprintf(fp,"\nThe bit test including 0 and 1\n");
fprintf(fp,"bit test:number0=%d number1=%d\n",number0,number1);
```

4.6 Poker test program

```
fprintf(fp,"\nThe pokertest pass  1.03<x<57.4 \n");
for(pokerCounter=0;pokerCounter<=15;pokerCounter++)
{
        fprintf(fp,"n[%d]=%d\n",pokerCounter,n[pokerCounter]);
        sumNIsquare=sumNIsquare+n[pokerCounter]*n[pokerCounter] ;
}
fprintf(fp,"sumNIsquare=%f\n",sumNIsquare);
pokerX=(16.0*sumNIsquare)/5000.0-5000.0;
if(pokerX>1.03 && pokerX<57.4)
        fprintf(fp,"pokerX=%f Passes the Poker Test\n",pokerX);
else
        fprintf(fp,"pokerX=%f Failure the Poker Test\n",pokerX);
```

4.7 Run test program

```
fprintf(fp,"\n The runtest including 0 and 1\n");
for(runLoop=0;runLoop<20000;runLoop++)
if(runNum[runLoop+1]!=runNum[runLoop])
        {
              run[runCounter]++;
              runCounter=1;
        }
        else
              runCounter++;
for(runLoop=1;runLoop<20;runLoop++)
        fprintf(fp,"01 run[%d]=%d\n",runLoop,run[runLoop]);
 //------------runTest ------------1
```

```
for(runLoop=1;runLoop<20;runLoop++)
        run[runLoop]=0;
for(runLoop=0;runLoop<20000;runLoop++)

        if(runNum[runLoop]==1)
        {
                run1Counter++;
                run0Counter=0;

        }
        else
        {
                run0Counter++;
                if(run0Counter==1)
                {
                        run[run1Counter]++;
                        run1Counter=0;
                }
        }
for(runLoop=1;runLoop<20;runLoop++)
        fprintf(fp,"1 run[%d]=%d\n",runLoop,run[runLoop]);
 //------------runTest--------------- 0
for(runLoop=1;runLoop<20;runLoop++)
        run[runLoop]=0;
for(runLoop=0;runLoop<20000;runLoop++)

        if(runNum[runLoop]==0)
        {
                run1Counter++;
                run0Counter=0;
        }
        else
        {
                run0Counter++;
                if(run0Counter==1)
                {
                        run[run1Counter]++;
                        run1Counter=0;
                }
        }
for(runLoop=1;runLoop<20;runLoop++)
        fprintf(fp,"0 run[%d]=%d\n",runLoop,run[runLoop]);
fprintf(fp,"\n--------------end--------------\n");
return 0;
}
```

5. Pseudorandom bit tests

5.1 LFSR pseudorandom bit test

There are several ways to prove whether the bit stream has the characteristics expected of a random set of bits. The FIPS 140-1 test suite includes tests, which are in the collection of National Institute of Standards and Technology (NIST).The FIPS 140-1 includes three tests: mono test, poker test and runs test. Three tests suite accept 20,000 bits from a random source.

The first test is mono bit test, which verifies that the number of 1's and 0's are almost equal; The process counts the number of 1's: if it is within the range 9654-10,346, then the bit stream passes the mono bit test[1].

We initial LFSR with 0010 1111 set in which the shift XOR function among the Shift in bit, cell 3 and cell 6.The mono test result is:Cycle = 20000;Ones bits Count = 10060;Passes the One bits Count Test (Mono bit test)

The second test is poker test. Passing the mono bit test does not guarantee that a bit stream is truly random. Poker test is another specified test by FIPS 140-1. For the poker test, the 20,000 bits are divided into 4-bit segments. Each 4-bit segment represents a decimal number between 0 and 15. A truly random sequence of bits should result in a random distribution of the numbers 0-15.Let be the number of occurrences of a number . is the number of 4-bit 0110.These values are substituted into:

$$X = \frac{16}{5000}\sum_{i=0}^{15} n^2 - 5000 \tag{6}$$

The poker test is passed if $1.03<X<57.4$.The LFSR poker test result is:

Cycle = 20000;The number of occurrences of number from 0 to 15 is as follows set:{275, 317, 316, 314, 313, 315, 314, 315, 314, 316, 313, 314, 318, 316, 315, 315}

X=4.8896 ;1.03<X<57.4, Passes the poker test.

A third randomness test is called the runs test. A run is a consecutive sequence of either 1's or 0's.In a truly random-bit stream, there should be a random distribution so maximal-length runs. If the number of each run falls within the following guidelines, then the sequence passes the test. The required interval is illustrated in Table 1.

Length	Required Interval and Run Test Result					
	1	2	3	4	5	6+
min interval	2267	1079	502	223	90	90
gaps count	2517	1261	630	315	157	158
runs count	2519	1259	630	315	157	158
max interval	2733	1421	748	402	223	223

Table 1. Required interval of runs test

Passes the run gap test [7].

5.2 Pseudorandom bits test of multi- dimensional CA and LFSR algorithm

We initial 3-dimensional cellular automation LFSR. CA is initialized to three-dimensional array

s[i][j][k]={0};s[0][0][0]=1;s[0][2][1]=1;s[1][0][1]=1; s[2][1][1]=1; s[2][2][2]=1;
For example, each cell is assigned rule1:110839188 and rule2: 134217727. LFSR is 0010 1111 set in which the shift XOR function among the Shift in bit, cell 3, cell 6 and s[1][1][1] .The poker test is passed if $1.03<X<57.4$.The Multi-dimensional CA LFSR algorithm mono test and poker test result is shown in table 2:

N=3 Rank	Mono Test (Num. of Bit 0)		Poker Test(Value of X)	
	rule1	rule2	rule1	rul2
3	10014	9994	4.37	1.32
4	9957	9969	5.27	7.11
5	9991	9956	13.40	8.33
6	9998	10054	4.50	17.24
7	10085	10096	9.42	9.64
8	10112	9944	14.71	16.14
9	9923	10083	11.97	10.04
10	10042	10010	10.07	8.476
11	10117	10001	19.67	10.25
12	9918	9895	17.72	6.25
13	10014	10077	20.04	15.78
14	10076	10158	10.65	14.72

Table 2. Poker test of three-dimensional and multi-rank CA algorithm

When changed dimensional and rank simultaneously, the algorithm can also pass the poker test as table 3 shows:

poker rank	N=4		N=5		N=6	
	rule1	rule2	rule1	rule2	rule1	rule2
3	12.66	1.75	17.20	1.06	4.58	6.46
4	16.07	18.29	15.20	25.69	5.82	11.37
5	14.59	13.92	18.11	15.97	10.51	10.28
6	10.37	12.85	7.57	13.54	12.45	22.02

Table 3. Poker test of multi-dimensional and multi-rank CA algorithm

All the poker X value is $1.03<X<57.4$, passes the poker test. We can't draw other obvious character from the table. It seems that the result isn't apparently changed with the number of dimensional and rank. It implies that it is not necessary to increase the number of dimensional and rank if we want to get stream bit. Maybe a lower dimensional and rank CA is enough.

6. Discussion

According to the random bits test result, Multi-dimensional cellular automation (CA) linear feedback shift register (LFSR) passed three stream bit test. Three tests suite accept 20,000 bits from a random source. The first test is mono bit test, which verifies that the number of 1's

and 0's are almost equal, the Multi-dimensional CA LFSR algorithm can create 1 bits and 0 bits, different rank from 3 to 14 of 3-dimensional CA LFSR algorithm can create bit stream. The difference between 0 bit and 1 bits is shown in Figure 5:

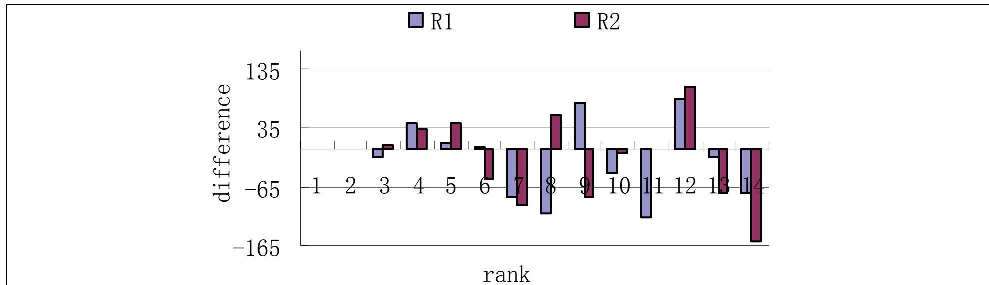

Fig. 5. Bits difference of multi-dimensional and multi-rank CA LFSR algorithm

According the figure, the bits difference can't be decreased by increasing 3-dimensional CA LFSR rank. That's means simple algorithm maybe better than complicated algorithm.
Multi-dimensional CA LFSR algorithm can also pass the poker test. The X value is between 1.03 and 57.4. When 3-dimensional CA LFSR increases its rank from 3 to 14, all the result can pass the poker test. The result is shown in Figure 6:

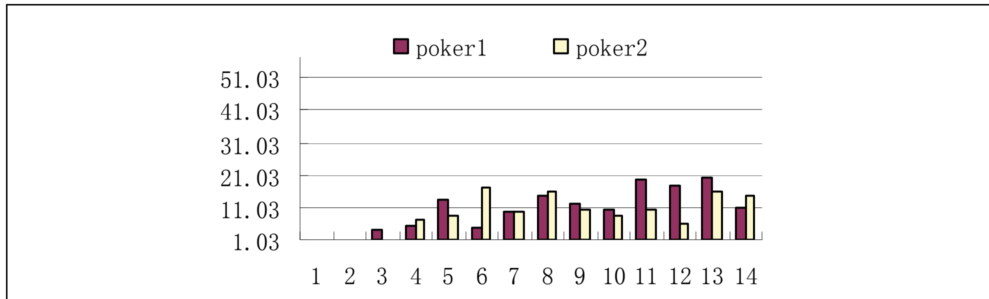

Fig. 6. Poker test of three-dimensional and multi rank CA LFSR algorithm

When increase the dimensional and rank at the same time. The result illustrates the algorithm can also pass the poker test shown in Figure 7:

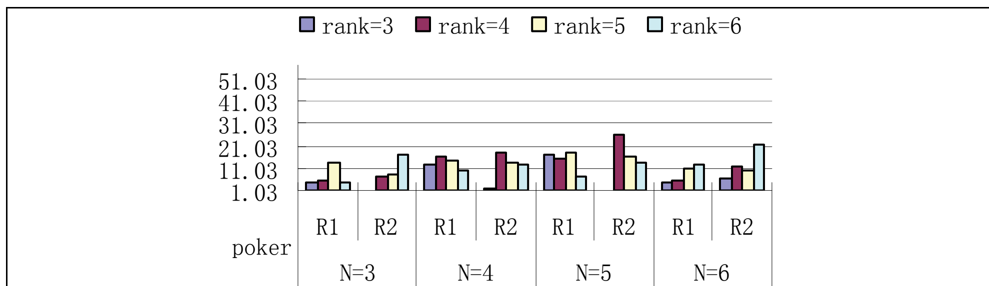

Fig. 7. Poker test of multi-dimensional and multi-rank CA LFSR algorithm

Compared with the Multi-dimensional CA LFSR and 3-dimensional CA LFSR, we can't find the obvious superiority by dimensional or rank increase. The Multi-dimensional CA LFSR algorithm can also pass the runs test, the average 1 bits and 0 bits pass the run and gaps test, the result fills in required interval almost same to the LFSR run gap test. The algorithm result is shown in Figure 8.

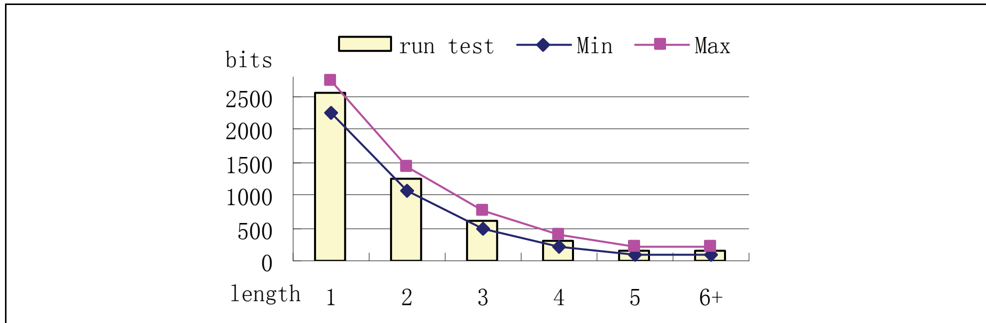

Fig. 8. Run gap test of multi-dimensional CA LFSR algorithm

The Multi-dimensional cellular automation (CA) linear feedback shift register (LFSR) algorithm combined cellular automation method and LFSR method to create random stream bit The design method can pass three FIPS 140-1 standard pseudorandom stream bit test and also can provide better pseudorandom key stream. The results illustrate it is feasible and efficient.

7. Acknowledgment

The paper is supported by the Shanghai Education Commission Innovation Foundation (11YZ192).

8. References

Chien-In & Henry C., Synthesis of configurable linear feedback shifter registers for detecting random-pattern-resistant faults, *ACM ISSS'01*, pp.203-208, Montreal, Quebec, Canada, October 2001. 1-3

Erik H. & Volkerink Subhasish M. Response compaction with any number of unknowns using a new LFSR architecture, *DAC 2005 ACM*, pp.117-122, Anaheim, California, USA. , June 13-17, 2005

Breukelaar R. ; Th. B. & Nutech, S. G. sing a genetic algorithm to evolve behavior in multi dimensional cellular automata, *GECCO' 05*, pp.107-114, Washington, DC, USA，June 25 – 29, 2005

Sheng-Uei G. ; Shu Z. & Marie, T. Q. 2-D CA variation with asymmetric neighborship for pseudorandom number generation", *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* , pp. 378-388, March 2004

Nandi, S. ; Kar, B. K. & Chaudhuri, P.P. Theory and applications of cellular automata in cryptography, *IEEE Transactions on Computers,* pp.1346-1357, December 1994, VOL. 43, NO 12

Yong, W. ; Xinming, G. & YuW. Three-dimensional cellular automation LFSR algorithm, *The Sixth International Workshop for Applied PKC*，pp.188-194, Perth, Australia, 3-4, December 2007
Richard J.S. *Classical and Contemporary Cryptology*, The Tsinghua Press, China, July.2005.

Cellular Automata - Innovative Modelling for Science and Engineering
Edited by Dr. Alejandro Salcido

ISBN 978-953-307-172-5
Hard cover, 426 pages
Publisher InTech
Published online 11, April, 2011
Published in print edition April, 2011

Modelling and simulation are disciplines of major importance for science and engineering. There is no science without models, and simulation has nowadays become a very useful tool, sometimes unavoidable, for development of both science and engineering. The main attractive feature of cellular automata is that, in spite of their conceptual simplicity which allows an easiness of implementation for computer simulation, as a detailed and complete mathematical analysis in principle, they are able to exhibit a wide variety of amazingly complex behaviour. This feature of cellular automata has attracted the researchers' attention from a wide variety of divergent fields of the exact disciplines of science and engineering, but also of the social sciences, and sometimes beyond. The collective complex behaviour of numerous systems, which emerge from the interaction of a multitude of simple individuals, is being conveniently modelled and simulated with cellular automata for very different purposes. In this book, a number of innovative applications of cellular automata models in the fields of Quantum Computing, Materials Science, Cryptography and Coding, and Robotics and Image Processing are presented.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Yong Wang, Dawu Gu, Junrong Liu, Xiuxia Tian and Jing Li (2011). Research on Multi-Dimensional Cellular Automation Pseudorandom Generator of LFSR Architecture, Cellular Automata - Innovative Modelling for Science and Engineering, Dr. Alejandro Salcido (Ed.), ISBN: 978-953-307-172-5, InTech, Available from: http://www.intechopen.com/books/cellular-automata-innovative-modelling-for-science-and-engineering/research-on-multi-dimensional-cellular-automation-pseudorandom-generator-of-lfsr-architecture

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2011 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License, which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.