Cellular Automata Based Test Structures With Logic Folding

Biplab K Sikdar[1] Sukanta Das[1] Samir Roy[2] Niloy Ganguly[3] Debesh K Das[4]
[1]Computer Science & Technology, Bengal Engineering & Science University, West Bengal, India 711103, (biplab,sukd)@cs.becs.ac.in
[2]Computer Science & Technology, Kalyani Government Engineering College, West Bengal, India, samir@kucse.wb.nic.in
[3]Centre for High Performance Computing Technical University, Dresden, Germany, n_ganguly@hotmail.com
[4]Computer Science & Engineering, Jadavpur University, Calcutta, India 700032, debeshd@hotmail.com

Abstract— **This paper presents an efficient test solution for *VLSI* circuits. The test structure is designed with GF(2^p) *CA*. The introduction to an innovative scheme of *logic folding* optimizes the cost of test logic that can not be feasible with the flattened structure of GF(2) CA/LFSR.**

I. Introduction

Existing $BIST$ structures have been typically built around $LFSRs$ [1] and CA [2]. A wide variation of these structures has also been proposed [3], [4]. The major limitation of conventional test logics is that these are typically designed without any consideration to the structure of CUT (circuit under test). Recent work [5] has addressed the problem by proposing a TPG structure based on HCA. A number of $BIST$ schemes targeting their behavioral/RTL descriptions have also been proposed. However, these impose a number of design restrictions.

In this work, we view a $VLSI$ system with single level hierarchy. Rather than considering primary inputs to a circuit at bit level, we look at a set of p bits as a cluster of inputs to a $RTL/functional$ block. The hierarchical architecture of GF(2^p) CA enables modeling of an elegant test structure for such a $VLSI$ circuit that permits introduction of logic folding, customized for the given CUT. The basic architecture of GF(2^p) CA is discussed in *Section III*. A brief on the preliminaries of extension fields, relevant for the design of $GF(2^p)$ CA, is introduced next.

II. Extension Field Preliminaries

In GF(2^p), there exists an element α that generates the non-zero elements $\alpha, \alpha^2, ..., \alpha^{2^p-1}$, of the field; α is the *generator*. The irreducible polynomial of which α is a root is called the *generator polynomial* of GF(2^p).

The generator $\alpha \in$ GF(2^p) can be represented by a $p \times p$ matrix M. Each elements of $M \in \{0,1\} \in$ GF(2). The matrix representation of the element α^j ($j = 2, 3, \cdots, 2^p - 1$) is given by M^j. A column vector of the M^j can be used as the vector notation for α^j. The operations defined in $GF(2^p)$ are addition $\oplus$ and multiplication, under modulo operation of the generator polynomial [5].

Example 1: For an n-cell $GF(2^3)$ CA with $x^3 + x^2 + 1$ as the generator polynomial, the matrix representation of $\alpha \in GF(2^3)$ is $\alpha_{matrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$. The α_{vector} (011) = 3 (3^{rd} column of α_{matrix}). The other elements α^2, α^3, $\cdots$, α^7 of $GF(2^3)$ can be computed from α. For example, $\alpha^5_{matrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

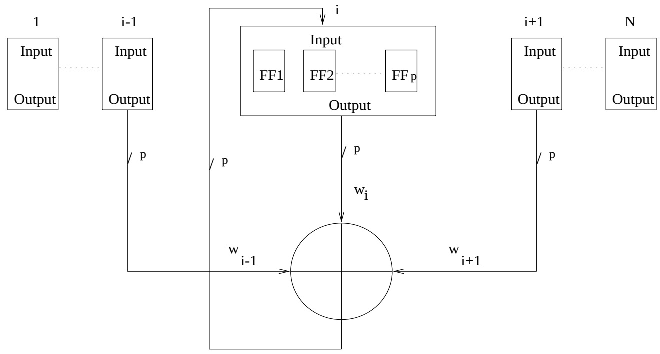


Fig. 1. General structure of a $GF(2^p)$ CA

III. $GF(2^p)$ Cellular Automata

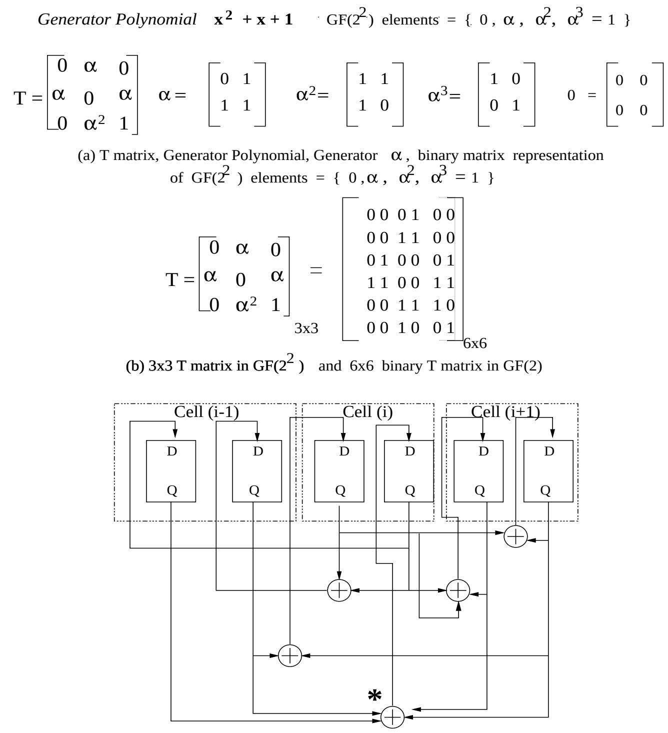
A $GF(2^p)$ CA cell (*Fig.1*) can store values 0,1,2....,($2^p - 1$) - that is, it has p memory elements (FFs). In 3-neighborhood, the next state of the i^{th} cell is $q_i^{t+1} = ((w_{i-1} * q_{i-1}^t), (w_i * q_i^t), (w_{i+1} * q_{i+1}^t))$. The w_{i-1}, w_i and w_{i+1} are belong to $GF(2^p)$ and specify the weights of interconnection among the CA cells.

An n-cell $GF(2^p)$ CA is characterized by an $n \times n$ matrix T [5]. The next state X_{next} of the CA is $X_{next} = T \times X_{current}$, where X_{next} and $X_{current}$ are the n-symbol strings representing the states of the CA. A 3 cell $GF(2^2)$ CA is shown in *Fig.2*. Analogous to the theory noted in [5], as the $det[T] \neq 0$, it is a group CA - that is, all the states lie on some some cycles. A 6×6 matrix (*Fig.2(b)*), can be derived by replacing the elements of $GF(2^2)$ in T with their corresponding 2×2 binary matrix representations (*Fig.2(a)*). It defines the CA hardware.

In a 3-neighborhood $GF(2^p)$ CA, a memory element (FF) has effectively a $3 \times p$ neighborhood. It adds additional computing/modeling power than that of $GF(2)$ CA. This property is exploited to design the $CATPG$.

IV. Design of $CATPG$

Let us consider the example CUT of *Fig.3* with $4p$ primary inputs (PIs). In practice, for all types of CUT with $4p$ PIs, the same $PRPG$ is used as the TPG in conventional designs. That is, the structure of TPG is considered independent of the circuit. It is logical to assume that all the PIs of *Fig.3* are not independent so far as their functionality is concerned. The PIs (0 to p-1) of type a input data to $Block1/Block2$ and assumed to be functionally similar and form *cluster* of PIs. The similar situation exists for type b, c & d. Instead of feeding the PIs of a *cluster* from p cells of a $4p$-cell GF(2)



(a) T matrix, Generator Polynomial, Generator α, binary matrix representation of $GF(2^2)$ elements = { 0, α, α^2, α^3 = 1 }

(b) 3x3 T matrix in $GF(2^2)$ and 6x6 binary T matrix in GF(2)

(c) The CA Structure

Fig. 2. A 3-cell $GF(2^2)$ CA

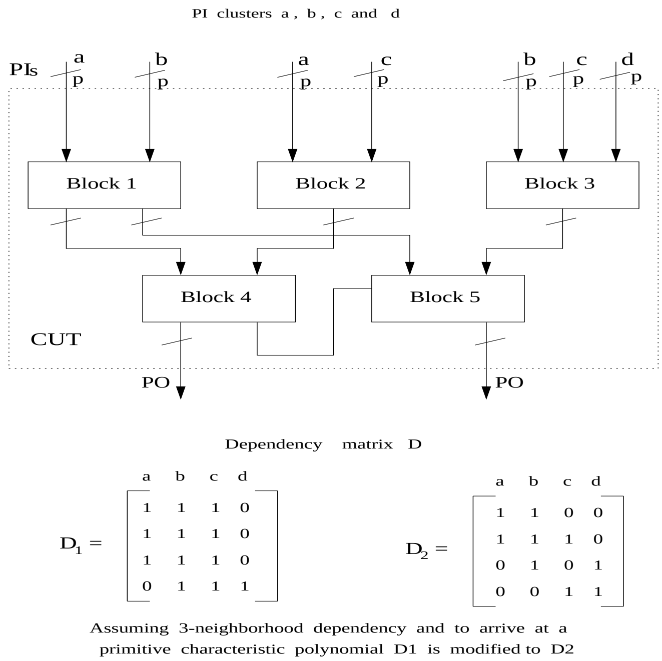


Fig. 3. Construction of dependency matrix D.

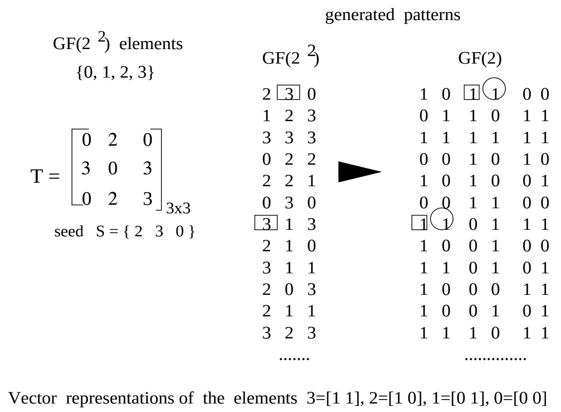


Fig. 4. The symbol string generated by a $GF(2^2)$ CA

$CA/LFSR$, we propose to feed the *cluster* from a cell of 4-cell $GF(2^p)$ CA. That is, rather than considering PIs to a circuit at bit level we look at a set of p bits. To customize the $CATPG$, we further investigate the existence of structural dependencies among the PI-clusters.

Definition 1: If two PI*-clusters* enter into the same RTL block, the structural dependencies is said to exist between them and referred to as *dependent clusters.*

Both the clusters a & b [a & c] carry input to *Block 1* [*Block 2*] and so these PI sets form dependent clusters. Similarly, PI-clusters b, c & d are also dependent clusters. It is observed that [5] the dependent clusters closely interact to detect the faults of a CUT. Therefore, the CA cells feeding the *dependent cluster*s must be interconnected.

Further, high quality of pseudo-random patterns are generated from a $GF(2^p)$ CA. This is due to apparent random phase shifts among the patterns generated from the cells. Moreover, the sub-cells (FFs) inherit the same phase shift that exists between two cells. For *Fig.4*, the phase shift of 0^{th} cell with respect to 1^{st} cell is 6. The relative phase shift between sub-cell 01 & 11 and the sub-cell 02 & 12 is also 6. While testing the CUT of *Fig.3* if phase shift between the patterns fed to $a[0]$ and $b[0]$ is x, then the phase shift between the patterns fed to a[1] and b[1] is also x. That is, the TPG does not differentiate between two lines of a PI-cluster. This justifies the application of $GF(2^p)$ CA in designing the TPG.

Overview of logic folding: The extraction of dependencies among the PI-clusters enables folding of the proposed $GF(2^p)$ $CATPG$. That is, a k input CUT can be tested by an n-cell $GF(2^p)$ $CATPG$, where $n \times p < k$. In the proposed design, the *independent* PI-clusters are supplied test patterns from the same $CATPG$ cell. So, the value of n depends on the dependencies among the PI-clusters of a CUT.

The PI-clusters C & D /C & E/ A & C/$\cdots$ of *Fig. 5* are independent to each other and can be fed from the same $CATPG$ cell, whereas the dependent clusters (A, D & E)/ (A & B) /(B & C) are to be connected to different cells. This effectively results in a 4-cell folded $CATPG$.

Thus, the design of a $CATPG$ demands specification of p & n, generator polynomial, and T of the $GF(2^p)$ CA.

A. Selection of p & n

Selection of p involves partitioning of PIs to form the PI-clusters and identifying the most frequent cardinality.

To decide on the value of n, we find (i) $multi - input\ PI$ cluster - carries inputs to more than one RTL blocks; and (ii) $single - input\ PI$ cluster - car-

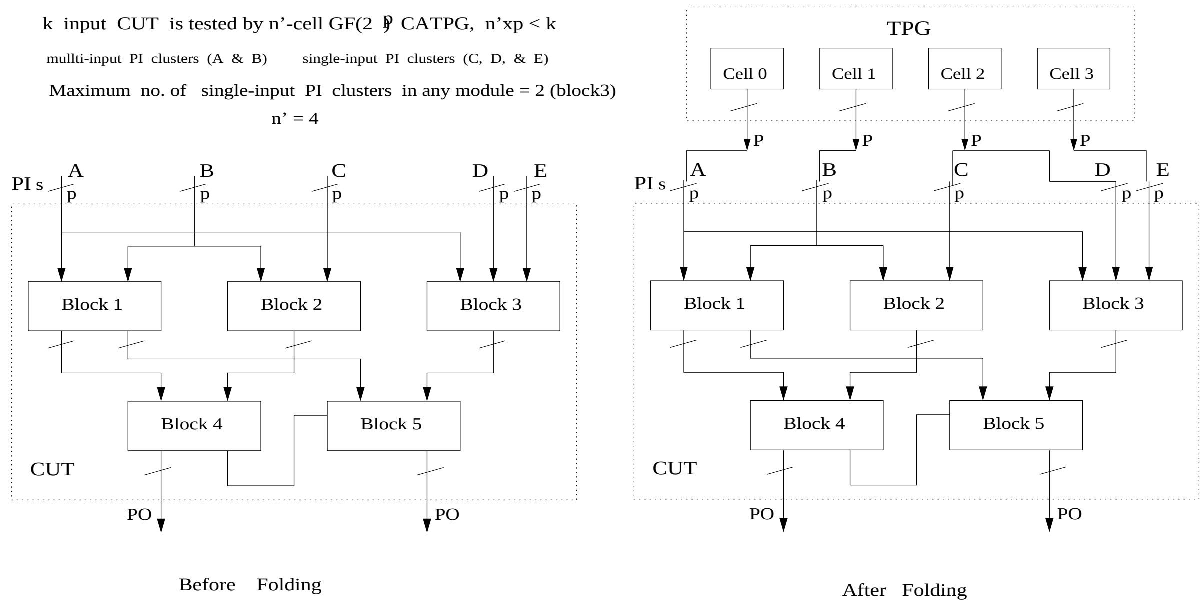


Fig. 5. Folding of a $GF(2^p)$ $CATPG$.

ries inputs to only one RTL block. The length of the $CATPG$ is fixed as $n = N_f + N_c$, where N_c = number of *multi-input* PI-clusters; and N_f = maximum number of *single-input* PI-clusters input to any module. For the CUT of *Fig.5*, $N_c = 2$. The number of *single-input* PI-clusters in *Block2* & *Block3* are 1 & 2. The value of N_f is 2. Therefore, $n = N_c + N_f = 4$.

The above discussion is formalized in *Algorithm 1.* Assume, (i) I is the set of PI-clusters, and M is the set of RTL blocks in a CUT which are fed by the PI-clusters; (ii) $I_c \in I$ is the set of $multi-input$ PI-clusters with $|I_c| = N_c$; and (iii) for $m \in M$, I_m denotes the set of PI-clusters input to m.

Algorithm 1: Find_n_of_CATPG
Input: PI-clusters input to different blocks
Output: n - the number of cells in the $CATPG$
Step 1. Find M, I_c, and N_c for the CUT.
Step 2. For every $m \in M$, find I_m and compute set of $single-input$ PI-clusters I_{mf}, where $I_{mf} = I_m \cap I_c$.
Step 3. Compute the number of $single-input$ PI-clusters, $N_{fm} = |I_{mf}|$, input to module m, $\forall m$.
Step 4. Find $m_{max} \in M$, such that $N_{fm_{max}} >= N_{fm}$ for any $m \in M$.
Step 5. Fix the length of TPG as $n = N_{fm_{max}} + N_c$.
Step 6. Return.

B. Fixing the $CATPG$ Structure

There are two aspects in the design - (i) to identify dependencies of one cell on its neighbors, and (ii) to specify the weight values of this dependencies.

Dependency Identification: In designing the $D_{n\times n}$ matrix for an n-cell $GF(2^p)$ $CATPG$, we extract the relative structural dependencies (*Definition 1*) among the PI-clusters. The D captures the dependent clusters and specifies the dependencies among the TPG cells - that is,

$$D[i,j] = \begin{cases} 1 \ \ (true), & \text{if } i^{th} \ \& \ j^{th} \ PI\text{-clusters} \\ & \text{are the dependent clusters} \\ 0 \ \ (false), & \text{otherwise.} \end{cases}$$

Fig.3 illustrates the dependency identification of a CUT. The clusters a & b, a & c and b, c & d are pair wise dependent clusters. While constructing D for the 4-cell $CATPG$, the dependencies are extracted and then specified in D_1. It can be observed that the D_1 may not get restricted to 3-neighborhood. *A graph algorithm is proposed to construct the 3-neighborhood dependency matrix.*

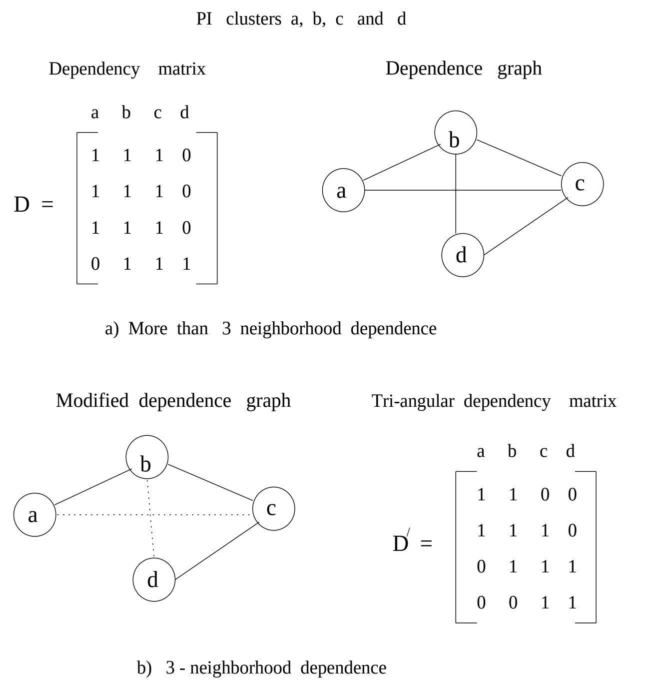


Fig. 6. Construction of dependency matrix D in 3-neighborhood.

From the dependency matrix D, an undirected graph

(*Fig.6(a)*) referred to as *dependence graph* of the CUT is constructed. The rows of D corresponds to the nodes (vertices). An edge between two nodes v_i and v_j exists iff $D[i,j] = 1$. Self dependency of a node is not considered.

Theorem 1: : *The degree of a node in the dependency graph can have maximum value of 2 in 3-neighborhood.*

The conversion, D to D' (3-neighborhood), boils down to the extraction of a subgraph by removing minimal number of edges of the dependence graph, where each node of the subgraph has degree ≤ 2. The scheme finds a number of disjoint subgraphs each of which is a path. The union of these subgraphs covers all the nodes of dependence graph. It starts from a node v_s with the least degree and selects a node v_a adjacent to v_s, where v_a has the least degree among the adjacent nodes of v_s and v_a is not included in the path. Once a path is found for a traversal, all the nodes along with the edges incident on them are removed from the original dependence graph $G(V,E)$. The procedure repeats with the resulting reduced graph $G_r(V_r,E_r)$ until the G_r is null.

The process outputs *Fig.6(b)* from *Fig.6(a)*. To ensure the group property [5] of the CA, employed for the $CATPG$ design, we locally modify the 0s and 1s of the D' and results in D_2 of *Fig.3*.

In designing the D of folded TPG, the *single-input* PI-clusters (c_i & c_j), that are to be fed from the same $CATPG$ cell, are considered as a single unit and called *cluster-pair*(c_i,c_j). A *cluster-pair*(c_i,c_j) is declared dependent on c_k if c_k inputs data to the block fed by the c_i or c_j. In the example design of *Fig.5*, there is only one cluster-pair (C,D).

Weight values of dependencies: *All the non-zero values of a column i of the D matrix are replaced by identical primitive weights* $w_is \in GF(2^p)$ *to arrive at the desired* T. This approach makes the $GF(2^p)$ CA as a group CA with larger length cycle and also minimizes the hardware overhead of the $CATPG$ [5].

V. Experimental Results

Table I provides the fault coverage figures of the $CATPG$. The selected value of p for a CUT is noted in *Column 3*. The fault coverage obtained for the $CATPG$ of length $n = N_f + N_c$ are reported. For comparison, the fault coverage with the full length (= number of PIs) $PSLFSR$ [4] and $GLFSR$ [3] are reported in the columns 6 and 7 respectively. Further comparisons are shown in *Table II*. Here the length of $PSLFSR$ and $GLFSR$ based pattern generators is assumed to be $n = N_f + N_c$.

The overheads have been computed in *Table III*. *Column 3* presents the length of the $CATPG$ in terms of the number of FFs required to implement the $CATPG$. $CATPG$ overheads for the CUTs are noted in *Column 4* in comparison (in %) to that of full length $Phase-Shift$ $LFSR$ based TPGs. We compute gate area only.

VI. Conclusion

This work establishes the application of $GF(2^p)$ cellular automata in $VLSI$ circuit testing. From the analysis of the experimental results, it is established that the $CATPG$ can be a better alternative while designing TPGs for $VLSI$ circuits.

TABLE I

TEST RESULTS WITH $CATPG$

Circuit name	No. of PI	p	CATPG cov(%)	#TV	PSLFSR cov(%)	GLFSR cov(%)
c6288	32	4	99.56	60	99.48	99.56
c1908	33	2	99.47	4000	99.47	99.07
c3540	50	4	96.06	3500	95.83	95.77
c7552	207	8	95.07	12000	94.90	95.37
s35932	35	4	86.17	14000	85.69	85.38
s3271	26	4	99.51	10000	99.57	97.86
s3384	43	4	92.16	8000	91.75	91.86
s4863	49	4	95.24	8000	93.56	93.78
s6669	83	4	100	4500	100	100

TABLE II

TEST RESULTS FOR PATTERN GENERATORS OF LENGTH $n = N_f + N_c$

Circuit name	CATPG Fault cov(%)	PSLFSR Fault cov(%)	GLFSR Fault cov(%)
c6288	99.56	99.30	99.33
c1908	99.47	98.72	98.72
c3540	96.06	95.56	95.30
c7552	95.07	94.29	94.64
s35932	86.17	85.67	85.38
s3271	99.51	99.27	97.86
s3384	92.16	91.66	91.66
s4863	95.24	93.45	93.83
s6669	100	99.84	99.85

References

[1] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-in test for VLSI : Pseudo-random Techniques.* John Wiley & Sons.

[2] P. D. Hortensius *et al.*, "Cellular automata based pseudo-random number generators for built-in self-test," *IEEE Trans. on CAD*, vol. 8, pp. 842–859, August 1989.

[3] D. K. Pradhan and M. Chatterjee, "GLFSR–A New Test Pattern Generator for Built-in-Self-Test," *IEEE Transactions on Computer -Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 319–328, 1999.

[4] J. Rajski, N. Tamarapalli, and J. Tyszer, "Automated synthesis of large phase shifters for built-in self-test," *International Test Conference*, pp. 1047–1056, 1998.

[5] B. K. Sikdar, N. Ganguly, and P. P. Chaudhuri, "Design of hierarchical cellular automata for on-chip test pattern generator," *IEEE Trans. on CAD*, pp. 1530–1539, December 2002.

TABLE III

OVERHEAD OF CATPG

Circuit name	No. of PIs	CATPG size (#FF)	Overhead in % CATPG/PSLFSR
c1908	33	16	53
c3540	50	36	71
c6288	32	24	62
c7552	207	40	35
s3271	26	20	83
s3384	43	16	58
s4863	49	28	76
s6669	83	20	49
s35932	35	20	42