

Synthesis of On-Line Testing Control Units: Flow Graph Coding/Monitoring Approach

Serge N. Demidenko¹, Eugene M. Levine², Vincenzo Piuri³

¹Institute of Information Science & Technology, Massey University, New Zealand,
S.Demidenko@massey.ac.nz

²Rainbow Technologies Inc., Belarus, lev@by.rainbow.com

³Department of Electronics and Information, Politecnico di Milano, Italy, piuri@elet.polimi.it

Abstract

The paper deals with design of concurrently (on-line) self-testing control units. The control units of the micro-program type are considered. At the same time, the material can easily be extended and applicable to cover other control unit architectures.

1: Introduction

Traditionally, *Micro-Program Control Unit* (MPCU) is designed around a micro-program memory containing micro-instructions [1-3]. Each micro-instruction consists of a control code field and a next address code field. The sequence of micro-instructions is determined by the sequence in the next address generator (Fig. 1). The generator produces a new address depending on the code in the next address field of the current micro-instruction, on the external control signals (start, stop, set at specific address, etc.), and on the values of logic signals (flags) coming out of the controlled device or system (these flags are required to organize conditional jumps in the program), as shown in Fig. 2.

Two major approaches in on-line monitoring of MPCU are usually employed in order to satisfy high dependability criteria. The first one is related to checking the control unit instruction outputs by employing the techniques developed for the highly dependable data path architectures [4]. The other includes techniques ranging from MPCU state description based on error-detecting codes to control-flow monitoring by on-chip and/or off-chip watchdogs [5-8]. The approaches are significantly different in terms of their fault coverage, hardware overhead, performance degradation, and error detection latency.

In this paper we deal with the MPCU self-monitoring method based on employing special check tokens (keys) incorporated into the body of the micro-program. The key (it is usually some binary word) is put into correspondence to each micro-instruction or to a set of micro-instructions. The keys are then interleaved with micro-instruction during the program execution. This allows checking the control flow by comparing the actual sequence of the check keys with a stored (or calculated) reference sequence corresponding to fault-free operation of the control unit. Different key generation and key/data compression techniques can be used to process of sequences of reference and actual keys. It leads to a wide variety of the possible architectural realizations for self-monitored MPCUs. Some examples of them are discussed in [7,8].

Two particular subjects will be discussed in this paper. The first is related to the fault coverage by the compression of the check keys (tokens) in MPCU where LFSR-based compactor is employed. Then we will discuss a strategy of assigning the test tokens to the micro-instructions of the monitored program.

2: On-line program flow monitoring

A control unit can be described using a flow graph representing its operation in terms of succession of instructions generated at the output. Every vertex (denoted Q_i) corresponds to one instruction (in some cases the vertex could correspond to a group of instructions, however here such a case is not discussed). The check keys are put into correspondence to the operator vertices of the flow graph.

Generally, the flow graph can be represented by set of k linear disjoint sections ($k \in \{1, 2, 3, \dots, N\}$ where N is the total number of the vertices) with i -th section having l_i vertices ($l_i \in \{1, 2, 3, \dots, N\}$ and $i \in \{1, 2, 3, \dots, k\}$). Thus j -th vertex of i -th section of the graph can be denoted as Q_{ij} . Faults of different types can occur during the program execution because of errors in MPCU. They are as follows:

- (a) disappearance of a micro-instruction(s) in the sequence generated by the MPCU;
- (b) appearance of an additional micro-instruction(s) in the sequence;
- (c) erroneous transition on logical conditions (i.e., in a conditional vertex of the flow-graph);
- (d) erroneous transition in a functional vertex;
- (e) substitution of one micro-instruction by another.

Defects of types (a) and (b) are lead to an alteration of the micro-instruction sequence, and thus they can be grouped together. Basically, all the defects from (a) to (e) can also be considered as belonging to a general "false transition" type (where (e) error is a double false transition – the program erroneously proceeds to some incorrect micro-instruction while returning back to the correct path afterwards).

Faults in the instruction fetch sequence correspond to faulty transitions between flow graph vertices, denoted, for instance, as $Q_{ij} \rightarrow Q_{yz}$, or $Q_i \rightarrow Q_j$ if just a linear flow graph is used (Fig. 3). The fault detection probability value P_d is normally used for estimation for the quality of the flow graph checking, and this quality quantitative is of the main interest. Expression $r_d(Q_{ij} \rightarrow Q_{yz})$ is used to denote the probability that a single faulty transition from vertex Q_{ij} to vertex Q_{yz} is detected by means of monitoring. The value of this probability depends on the flow graph itself, on the type and distribution of the check keys, on the chosen checking strategy, and on the employed check key processing methods. In every particular case the probability $r_d(Q_{ij} \rightarrow Q_{yz})$ is either equal to one (when the faulty transition from Q_{ij} to Q_{yz} is detected), or otherwise equal to zero (when the faulty transition remains undetected).

The above notations can be used to get a general description for the fault detection in the MPCU program flow [9].

- (i) Complex event of detection of the faulty transition $Q_{ij} \rightarrow Q_{yz}$ includes the following three sub-events:
 - (a) occurrence of a fault in some vertex Q_{ij} of the program flow-graph,
 - (b) occurrence of the faulty transition from vertex Q_{ij} to vertex Q_{yz} ,
 - (c) detection of the transition by means of on-line (built-in or external) tools.
- (ii) Probability that a fault is occurred in some vertex of a flow graph depends on the reliability characteristics of the MPCU hardware, on the flow graph pattern, on the ranges of the input data as well as on the variation and distribution of the data over these ranges, and on the probabilities of generation of various logic signals (flags) used in conditional jump operations (case statements, "if", "while"). The numerical evaluations for the probabilities can be obtained theoretically (on the basis on analytical reliability characteristics of the MPCU components, theoretical characteristics of the input data, and parameters of the expected operation of the system controlled by MPCU) or empirically from observations on operation of an actual MPCU device. The values can be summarized in the form of a one column matrix - vector F (*Vector of Fault Occurrence*) of dimension

N (N is the number of flow graph vertices). Each element of F corresponds to a vertex of the flow graph and is equal to the probability that a fault occurs there.

- (iii) Probability that the faulty transitions occur between two specific vertices can be presented by the square matrix T (let us call it *Transition Matrix*) of dimension $N \times N$. Each element of the matrix is equal to the probability of a faulty transition between the two vertices. The probability evaluations can be found empirically by operating actual MPCU. Alternatively, they can be obtained analytically or from computer simulation using different fault models and their insertion into different parts of the MPCU.
- (iv) Probability of detecting faulty transitions by means of on-line checking tools can be given by a square matrix D (*Detection Matrix*) of dimension $N \times N$. Values of the elements of the matrix D are determined by the employed method of on-line checking (flow graph execution monitoring). For example, if assuming that only a single transition error occurs, and the checking is performed on every cycle of MPCU operation, then an element of such a matrix is equal to 1 when a faulty transition is detected; otherwise it is equal to zero. A very simple example of such a case is presented in the Fig. 4 where the flow graph is coded using 1-bit keys and the checking is realized at each vertex by means of the parity calculation. If the checking is organized in the interval-by-interval mode (i.e., just one reference value is used for a group of linearly linked vertices), and if an occurrence of multiple errors is possible [8], the detection probabilities may have values in the range between zero and one.
- (v) Matrix expression for the probability of detection of a control flow faulty transition can be presented as follows: $P = F \times T \cdot D$, where “ $\cdot$ ” denotes the element-by-element multiplication (not the conventional matrix-by-matrix multiplication operation). The elements of the resultant matrix P are the values of the probabilities of detecting faulty transitions between the corresponding vertices of the flow graph.

3: Synthesis of self-checking MPCU with predefined fault coverage

The general procedure [8] for synthesis of self-monitoring MPCU with pre-specified level of fault coverage can be presented as a sequence of the following steps:

- (i) By analyzing the flow-graph, the architecture of MPCU as well as the other relevant data (probability of external control signals coming to MPCU, probabilities of conditional jumps in the program initiated by the flag signals, etc.) elements of the vector F and matrix T are found.
- (ii) General attainability of the fault detection is checked by calculating the maximal possible fault coverage level for given V and T . At this stage it is assumed that all the elements of matrix D are ones, that is all possible faulty transitions can be detected by on-line monitoring. If it is found that the required level of the fault coverage can not be reached, then the algorithm, instruction set, data range, and so on must be modified, while the MPCU architecture must be re-configured, as necessary. After that - return to the step (i). Once the required coverage is found to be attainable - go to step (iii).
- (iii) Elements of the matrix D are defined from the required fault coverage and from the values of elements of the vector V and matrix T .
- (iv) Checking method and strategy is selected and the values of the check keys are assigned to the flow graph vertices.
- (v) Corresponding changes are introduced into the architecture of the MPCU hardware and software to realize the checking with the use of the assigned keys and selected strategy.

The presented methodology allows also evaluating the fault coverage/latency/hardware overhead of the self-checking MPCU when the reliability parameters, check key values, method of fault detection are given *a priori*.

A simple example can be used to illustrate the above methodology. Suppose the flow graph of the program is presented as in the Fig. 5 along with the vertex check keys and the reference values. Let the probability of a fault occurring be $1/2$ in the vertex Q_1 ; $1/8$ in the vertex Q_2 ; $1/16$ in each of the vertices Q_3 and Q_4 , and $1/4$ in the vertex Q_5 . The column vector F then takes the form presented in Fig. 6. Let the faulty transition probability graph and the transition probability matrix T be as shown in Fig. 6. According to the flow graph and the check keys allocation the following faulty transitions will be detected by diagnostic means: $Q_1 \rightarrow Q_4$; $Q_1 \rightarrow Q_5$; $Q_2 \rightarrow Q_1$; $Q_2 \rightarrow Q_2$; $Q_2 \rightarrow Q_3$; $Q_2 \rightarrow Q_4$; $Q_3 \rightarrow Q_1$; $Q_3 \rightarrow Q_2$; $Q_3 \rightarrow Q_3$; $Q_3 \rightarrow Q_5$; $Q_4 \rightarrow Q_1$; $Q_4 \rightarrow Q_2$; $Q_4 \rightarrow Q_3$; $Q_4 \rightarrow Q_4$; $Q_5 \rightarrow Q_1$; $Q_5 \rightarrow Q_2$; $Q_5 \rightarrow Q_3$; $Q_5 \rightarrow Q_4$. All other faulty transitions remain undetected. Accordingly, the matrix D will be as presented in Fig. 6.

Finally, the matrix P of the detection probability will be as follows:

$$P = \begin{bmatrix} 1/2 \\ 1/8 \\ 1/16 \\ 1/16 \\ 1/4 \end{bmatrix} \times \begin{bmatrix} 1/2 & 0 & 0 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 \\ 1/4 & 0 & 1/2 & 0 & 1/4 \\ 1/4 & 0 & 1/2 & 1/4 & 0 \\ 1/2 & 0 & 0 & 0 & 1/2 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1/8 & 1/8 \\ 1/32 & 1/32 & 1/32 & 1/32 & 0 \\ 1/64 & 0 & 1/32 & 0 & 1/64 \\ 1/64 & 0 & 1/32 & 1/64 & 0 \\ 1/8 & 0 & 0 & 0 & 0 \end{bmatrix}$$

And the value of the total probability of detecting faulty transition can be obtained by summing all the elements of matrix P :

$P_d = (1/8+1/8)+(1/32+1/32+1/32+1/32)+(1/64+1/32+1/64)+(1/64+1/32+1/64)+1/8=5/8$,
i.e., with the probability of 62.5% faulty transitions in the presented MPCU will be detected.

4: Some properties of matrices V , T , and D

Fault occurrence vector F can be represented as element-by-element (middle) product of two column-vectors F' and F'' . The first specifies the probabilities of reaching the corresponding flow graph vertices in course of program execution. The second – the probabilities of a fault occurrence while the program is in the vertices.

Vector F' depends mainly on the program flow. For example, for a linear flow-graph all the elements of the vector are of equal value, while for the flow-graph having conditional transitions or loops some vertices have higher reaching probability than the other ones. Higher reaching probabilities correspond to the vertices that are inside loops or those of the convergence of different branches.

Values of the components of vector F'' depend mainly on the parameters of reliability of the hardware used to implement the MPCU, as well as on the probability of the external interference leading to the fault occurrence. The results of calculations based on the traditional reliability and failure prediction methods can be used as the first approximation or initial data when other sources to get the information are not available.

Transition matrix T depends, first of all, on the character and the place of the physical fault occurrence within MPCU, as well as on the flow-graph of the micro-program. When no *a priori* data is available, all the transitions can be taken as equally probable. However, this could be used only as a very last resource. The use of such results would ultimately lead to an overall improvement for the fault coverage, however not as fast as it is required in the majority of cases. More realistic data can be obtained by considering simple fault models (stuck-at, bridging, inverting, etc.) inserted into different parts of the MPCU architecture.

As an example, let the linear program fragment consists of eight consecutive vertices, while their addresses are changed in the program in the 'normal' (i.e., incrementing by one) way.

The initial address is 000_2 , the final – 111_2 . No transitions are originated from the final vertex of the fragment, i.e., the micro-instruction generation is suspended after the program reaches the vertex having address 111_2 . Let the fault in MPCU causes a single inversion error in the 3-bit address field (***) of the micro-instruction register (Fig. 1). Finally, let the probabilities of occurrence of the inversion error are equal for all the bits of the address field. In such a case, the state transition diagram and the matrix T will be as shown in Fig. 7.

Some specific features of the matrix corresponding to the linear fragment of the flow-graph are obvious: zeros in the matrix diagonal above the main one; interleaving values of $1/N$ (N is the number of the address bits) and 0 in the main diagonal; interleaving pairs of $1/N$, $1/N$ and 0 , 0 in the diagonal below the main one; all zeros in the diagonal two rows below the main one; all zeros in the bottom row of the matrix; and so on.

A particular form of the matrix T would obviously be of different type for different structure of the flow-graph, different fault models, and once the fault occurrence is considered in the other part of the MPCU architecture. Obtaining the matrix for a particular MPCU can be significantly simplified and facilitated by using computer-aided simulation tools. However, even on the basis of the simple example above, one important conclusion can be made regarding difference between the errors in the MPCU and those in digital logic circuits. In digital circuits testing errors in the output sequences of the unit-under-test are normally of the type of bit value distortion. At the same time, the total length of the sequence normally remains unchanged. In contrast, fault occurrence in MPCU leads to the errors that are not only in the distortion of bits in the output sequence, but also in the *shortening* or *lengthening* the sequence itself. This essentially questions the efficiency of the use of some compression techniques (for instance, LFSR-based, or syndrome compaction) adopted in many earlier designs.

Several dozens of million experiments using computer simulation have been performed. In half of the experiments, before compression the sequence was shortened by removing some bits, while in the others the sequence was extended by adding bits. Each experiment included simulation for all possible combinations for the sequence being compressed for given length. The compression was performed using the following algorithms: LFSR-based signature compression for several lengths of the signature register, parity checking, syndrome compression, counting $0 \rightarrow 1$ transitions, counting both $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions. An example of the experimental results is presented in the Table 1. Fault of the “shortening” (missing bits) type is considered, the length of the initial sequence is 13, and five different types of compression are used: two LFSR-based compressions (with the characteristic polynomials of degrees 4 and 6), counting 1's in the sequence, counting $0 \rightarrow 1$ transitions, and counting both $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions.

The results shows rather significant difference between fault coverage provided by the compression methods for different error configurations. For example, in the MPCU environment one-bit error (i.e., disappearance of one bit from the sequence, or addition of one bit into it) can be undetected, i.e., masked when using LFSR-based compression. This is in contrast to the traditional LFSR-based *Signature Analysis* where all 1-bit and 2-bit errors are detected with 100% reliability. Indeed, shortening a sequence under compression for 1 or 2 bits in case of MPCU could lead to an arbitrary configuration of the error polynomials, and thus to error masking. In contrast, in case of digital testing an alteration of 1 bit (for instance, some l -th bit of the sequence) or 2 bits (some l -th and r -th bits) of the compressed sequence, while the length of the sequence remains intact, will produce an error polynomial of the following types: $e(x) = x^{l-1}$ or $e(x) = x^{l-1} + x^{r-1}$. It is known that division of these polynomials by an irreducible and primitive polynomial of LFSR (so-called characteristic polynomial) will always lead to some residue (i.e., to non-zero signature), and thus to fault detection. Special compression techniques are needed for the implementation in MPCU self-monitoring circuitry.

Detection matrix. Matrix D presents the main tool of the test designer in providing the required level of the fault coverage. Because of that, the properties of the matrix D are of special importance and interest. To reduce the space the properties are given without formal proof, as it is rather straight forward.

Property 1. If some elements of i -th row of matrix D are 0's ($d_{i,j} = d_{i,q} = \dots = d_{i,r} = 0$), while all other elements of the same row are 1's, then

$d_{j-1,i+1} = d_{j-1,q} = \dots = d_{j-1,r} = d_{q-1,i+1} = d_{q-1,j} = \dots = d_{q-1,r} = \dots = d_{r-1,i+1} = d_{r-1,q} = d_{r-1,j} = 0$, while all other elements of the $(j-1)$ th, $(q-1)$ th, ..., $(r-1)$ th row are equal to 1. Besides, if there are logic condition vertices among those having indexes $(j-1)$, $(q-1)$, ..., $(r-1)$ then the number of 0 elements corresponding to them is equal to the number of vertices succeeding the conditional vertices.

Property 2. If $d_{i,j} = 0$ then $d_{j-1,i+1} = 0$, and if $d_{i,j} = 1$, then $d_{j-1,i+1} = 0$ (to remind, value of d is within the inclusive interval between 0 and 1).

Property 3. If there is a transition from Q_i to Q_j in the MPCU program flow graph, then $d_{j-1,i+1} = d_{i,j}$, such elements of the matrix D can be called self-conjunct. Fault detection coverage does not depend on the values of self-conjunct elements of matrix D .

It is important to be able to construct the detection matrix in the manner that would provide required level of fault detection. The perspective approach to solve this problem can be based on the use of sorting algorithms with global or local optimization. Global optimization gives the best (in terms of the value of total detection probability) distribution of the check keys in the flow graph for the chosen method of the fault detection. However, since the algorithm uses combinatorial procedures, it leads to exponential increasing of the required computations to find the optimal solution with the growth in a number of flow graph vertices. For this reason, the global optimization algorithm though interesting and important in theoretical terms, can be impractical in real MPCU design. We will specially return to the global optimization algorithm, and will discuss it in all particulars in one of our future publications.

One of the possible local optimization-based algorithm for the case of first level fault detection (fault detection without using sequence compression, i.e., only the correctness of the transitions between two vertices are monitored, not all the 'history' of the process prior to the vertices [8]) includes the following steps.

1. Calculation of the primary limits (*min* and *max*) for the number of faulty transition to be detected for each vertex of the flow-graph. If possible find exact value for at least few elements of the detection matrix
2. Grouping elements of D while taking into account known elements of the detection matrix (given *a priori*, or found as a result of the step 1). The number of groups gives the number of different values of check keys that will be assigned to the flow graph vertices.
3. Completing the vertex groups while taking into account the number of detectable transitions for each vertex. Checking that the grouping is realizable. If it is the case – proceed to step 4, otherwise – return to step 1, and start all over again while changing the initial
4. Group modifying in correspondence with the properties of the matrix D . Checking that the grouping is realizable. If it is the case – proceed to step 5, otherwise – return to step 1.
5. Entire checking that the detection is realizable with the given limits on the number of the check keys. If the required number of the check keys is within the initially pre-specified limit – proceed to step 6, otherwise – return to the initial step 1.
6. Final vertex grouping and the group simplification.
7. Final completion of the detection matrix.

Example of detection matrix completion. Consider the algorithm of Fig.8. The flow-graph can be partite into linear fragments, e.g., from vertex Q_0 to Q_6 ; from Q_7 to Q_{12} ; and Q_{13} to vertex Q_{16} . Assume the following initial conditions:

- a) the *min* number of detectable faulty transitions for each of the vertices of the group Q_0 to Q_6 , Q_{11} , Q_{12} , Q_{15} , and Q_{16} is 2, for each of the vertices of the group Q_{13} , and Q_{14} is 4, and for each of the vertices Q_7 to Q_{10} is 3;
- b) the *max* number of different check keys that could be used to code the flow graph is 5;
- c) the values of the following elements of the detection matrix D are given $d_{0,2}=1$, $d_{4,9}=1$ as detection of the corresponding faulty transitions are of imperative importance.

To complete the detection matrix D that satisfies the above conditions, the following steps are performed. By taking into account the above condition c) the vertices Q_0 and Q_1 are put into different vertex groups, as well as the vertices Q_3 and Q_8 . The vertices are grouped into sets that would guarantee the required *min* level. The final total number of the groups is 5, as determined by the *max* number of the available check keys. The group providing 2 as *min* number of the detectable transitions should have at least three vertices; the group with *min* = 3 has four vertices, and so on. However, since the vertices Q_1 and Q_4 have two vertices for possible transition, they can be counted as substituting two components in a group. The first grouping can be done as in the table of Fig. 8. The first iteration does not satisfy the requirements as the number of the groups is too high, while the third, fifth and sixth groups are not completed. The second iteration could include transferring Q_{10} from the fourth group into the third, Q_{11} - from the fifth group to the fourth, and Q_4 - from the sixth group into the fifth one. Five groups are completed. There are still six groups in the partition and the group number six is far from providing required coverage, i.e., the initial conditions are not satisfied. If the requirements towards the number of available check keys are strict, there can be slightly eased the requirements towards the detection means overhead, then the new partition can be performed. The group six can be disbanded by transferring Q_{13} into the third group, while vertex Q_{14} into the fourth one. The third iteration grouping is not final - as the coverage for the vertex Q_{13} is lower than required. After the fourth iteration, there are five groups, while the level of guaranteed faulty transition detection is excessive (it means not optimal) for four vertices Q_1 , Q_3 , Q_4 , Q_8 , and Q_{16} . The final matrix D is shown in Fig. 8.

5: Conclusions

This paper presented some of topics in MPCU self-checking - synthesis of the control units providing pre-defined fault coverage. We focused on the properties of the matrixes describing fault occurrence/detection in the self-checking MPCU architecture. Hardware overhead, fault detection latency and fault coverage are closely linked in our synthesis.

Acknowledgement

The authors would like to thank Dr. G. Muravyov, Brest Institute of Contemporary Knowledge, for his contribution into the experimental material of the paper (computer simulation of different types of data compression techniques).

References

- [1] C. D. Kraft, W.N. Toy, "Microprogrammed control reliable design of small computers", Prentice-Hall, 1981
- [2] M. M. Mano, "Computer System Architecture", Prentice-Hall, 1993
- [3] W. Stallings, "Computer Organization and Architecture", Prentice-Hall, 2000
- [4] K. Pradhan, "Fault-tolerant computer system design", Prentice-Hall, 1996
- [5] R. A. Parekhji, G. Venkatesh, S. D. Sherlekar, "Concurrent error detection using monitoring machines", IEEE design & Test of Computers, Vol. 12, No. 3, 1995, pp. 24-31
- [6] M. A. Schutte, J. P. Shen, "Exploiting instruction-level parallelism for integrated control-flow monitoring", IEEE Trans. on Computers, v. 44, pp. 129-140, 1994
- [7] A. Mahmood, E. J. McCluskey, "Concurrent error detection using watchdog processors - a survey", IEEE Trans. on Computers, v. 37, pp. 160-174, 1988
- [8] S. N. Demidenko, E.M. Levin, K.V. Lever, "Concurrent self-checking for microprogrammed control units: an analytical survey", IEE proceedings, v. 138, n.6, pp. 377-388
- [9] S. N. Demidenko, E.M. Levin, K.V. Lever, "New approach to synthesis of self-checking microprogrammed control units with specified fault-detection probabilities", IEE proceedings, v. 138, n.6, pp. 389-396

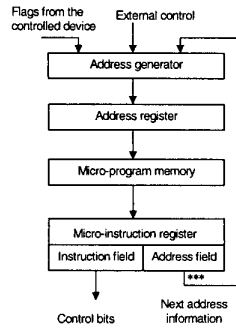


Fig.1. Micro-programmed control unit

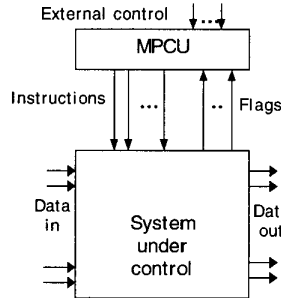


Fig.2. MPCU-controlled system interaction

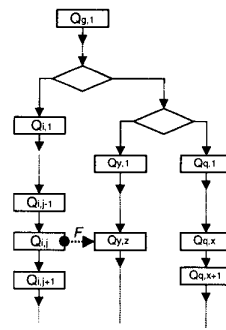
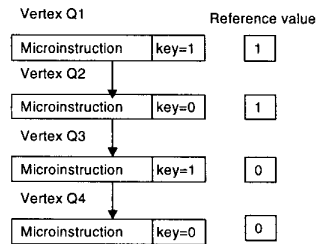
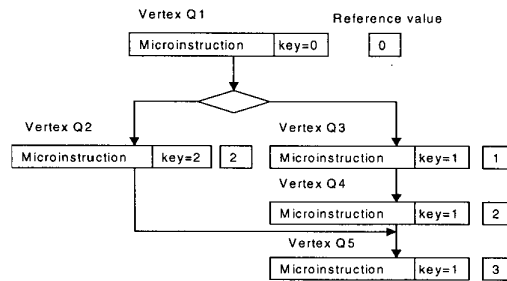


Fig.3. Program flow-graph with a single faulty transition (F)



Checking algorithm $\Sigma \text{key} \pmod{2}$

Fig. 4. Coded flow-graph (parity checking)



Checking algorithm: Reference value = Σkey

Fig. 5. Coded flow graph (summing)

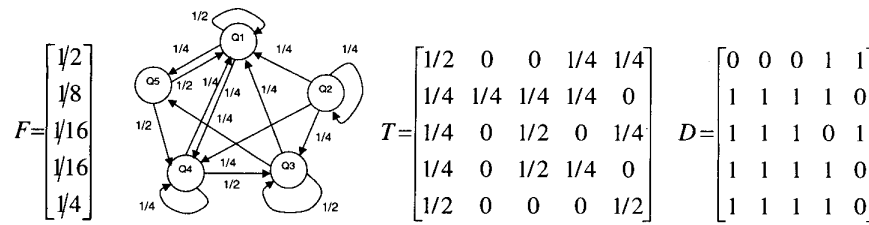


Fig. 6. Fault occurrence vector F , fault transition diagram and matrix T , detection matrix D

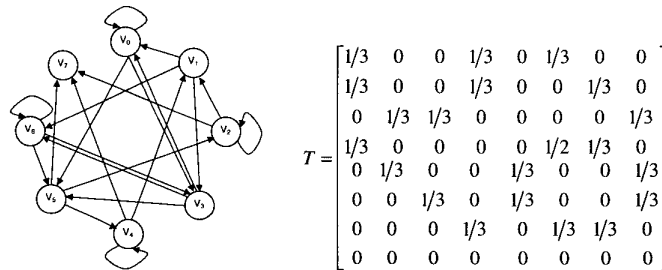


Fig. 7. Transition diagram and corresponding transition matrix T

**Table 1. Fault detection probabilities for 'missed bits' type faults
(length of the initial sequence is 13 bit)**

# of missed bits	# of possible faults (experiments done)	Compression technique				
		LFSR (deg4)	LFSR (deg6)	# of 1's	# of 0 → 1	# 0 ↔ 1
1	98292	0.92	0.95	0.50	0.25	0.27
2	90101	0.93	0.97	0.75	0.50	0.52
3	81910	0.94	0.98	0.88	0.69	0.71
4	73719	0.94	0.98	0.94	0.81	0.83
5	65528	0.94	0.98	0.97	0.89	0.90
6	57337	0.94	0.98	0.98	0.94	0.94
7	49146	0.94	0.98	0.99	0.96	0.97
8	40955	0.94	0.98	1.00	0.98	0.98
9	32764	0.94	0.98	1.00	0.99	0.99
10	24573	0.94	0.98	1.00	0.99	1.00
11	16382	0.94	0.98	1.00	1.00	1.00
12	8191	0.94	0.98	1.00	1.00	1.00

Group No	Group components	Number of detection	
		Actual	Required
First iteration			
1	Q_0, Q_5, Q_{12}	2	2
2	Q_2, Q_6, Q_{15}	2	2
3	Q_3, Q_{16}	1	2
4	Q_1, Q_8, Q_{10}	2	2
5	Q_7, Q_9, Q_{11}	2	3
6	Q_4	2	2
	Q_{13}, Q_{14}	1	4
Second iteration			
1	Q_0, Q_5, Q_{12}	2	2
2	Q_2, Q_6, Q_{15}	2	2
3	Q_3, Q_{10}, Q_{16}	2	2
4	Q_1, Q_8, Q_{11}	2	2
5	Q_4	2	2
	Q_7, Q_9	3	3
6	Q_{13}, Q_{14}	1	4
Third iteration			
1	Q_0, Q_5, Q_{12}	2	2
2	Q_2, Q_6, Q_{15}	2	2
3	Q_3, Q_{16}	3	2
	Q_{10}, Q_{13}	3	3
	Q_{11}	3	4
4	$Q_1,$	3	2
	$Q_8, Q_{11},$	4	3
	Q_{14}	4	4
5	$Q_4,$	2	2
	Q_7, Q_9	3	3
Fourth iteration			
1	Q_0, Q_5, Q_{12}	2	2
2	Q_2, Q_6, Q_{15}	2	2
3	Q_3, Q_{16}	3	2
	Q_{10}, Q_{11}	3	3
4	$Q_1,$	3	2
	$Q_8,$	4	3
	Q_{13}, Q_{14}	4	4
5	$Q_4,$	3	2
	Q_7, Q_9	3	3

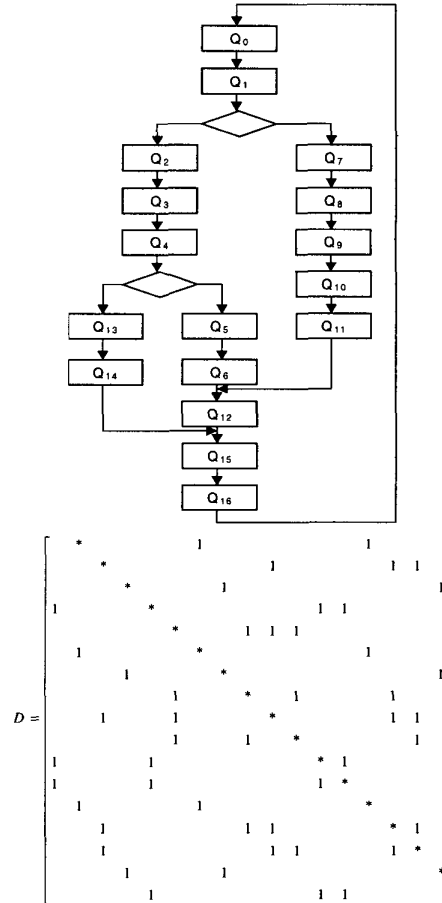


Fig. 8. Initial flow graph, table of iterations, and final detection matrix D .