IMTC 2005 – Instrumentation and Measurement
Technology Conference
Ottawa, Canada, 17-19 May 2005

Fault-Tolerance in Micro Programmed Control: Architectures & Schematic Synthesis

Serge Demidenko[1], Eugene Levine[2], Vincenzo Piuri[3], Gourab Sen Gupta[4,5]
[1]School of Engineering, Monash University, Malaysia Campus, 2 Jalan Kolej, 46150 Petaling Jaya, Selangor, Malaysia
Phone: +60-3-56360600, Fax: +60-3-56329314, Email: Serge.Demidenko@eng.monash.edu.my
[2] Rainbow Technologies Inc., Minsk, 220026, Minsk, Belarus
Phone: +375-17-249-8276, Fax: +375-17-248-8812, Email: lev@rainbow.by
[3]Department of Information Technologies, University of Milan, Via Bramante 65, 26013 Crema (CR), Italy
Phone: +39-02-503-30066, Fax: +39-02-503-30010, Email: piuri@dti.unimi.it
[4]Institute of Information Sciences & Technology, Massey University, Private bag 11222, Palmerston North, New Zealand
Phone: +64-6-350 5799, Fax: +64-6-350-2259
[5]School of EEE, Singapore Polytechnic, 500 Dover Road, Singapore
Email: G.SenGupta@massey.ac.nz

Abstract *– The paper deals with architectural and schematic design of concurrently (on-line) self-checking Micro Program Control Units. The checking is organised by using a special check keys added to each microinstruction. The sequence of keys appearing on the output of MPCU during the program execution is then monitored. It allows checking the control flow by comparing the actual sequence of the check keys against a reference sequence corresponding to fault-free operation of MPCU.*

Three main architectures discussed in the paper are: a) based on the use of Compression-in-Space of the check keys; b) based on implementation of Compression-in-Space & Compression-in-Time; and c) Compression & Generation check keys -based. The design procedure for the checking circuitry based on the use of Compression & Generation is discussed in the paper for two types of the check key generators: a standard one (for example, M-sequence generator) and an unique generator (specially synthesised sequential automata producing some required digital sequence).

Keywords *– microprogram control unit, program flow monitoring, check key generation and compression, architectural and schematic implementations.*

I. INTRODUCTION

Micro-Program Control Unit (MPCU) is usually designed around a micro-program memory with microinstructions generated on its outputs [1-2]. A micro-instruction includes a control code field and a next address field. The sequence of microinstructions is determined by the next address generator (Fig. 1).

The generator produces a new address depending on the code in the next address field of the current microinstruction, as well as on the external control signals (e.g., *start*, *go to a specific address*, etc.), and on the values of logic signals (flags) coming from the system under control. The flags are employed to organize conditional jumps in the microprogram.

In order to satisfy high dependability criteria, an on-line monitoring of MPCU can be employed. In this paper we deal with the MPCU self-monitoring (self-checking) method based on the use of special check keys incorporated into the body of the microinstructions. The key (some binary word) is put into correspondence to each microinstruction (or to a set of them). The sequence of keys appearing on the output of MPCU during the program execution is then monitored. This allows checking the control flow by comparing the actual sequence of the check keys against a reference sequence corresponding to fault-free operation of MPCU.

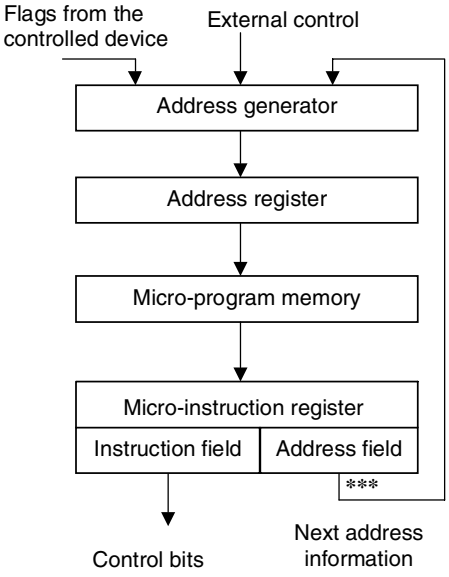

Fig. 1. Microprogram Control Unit

Different check key generation and key/data compression techniques can be used to process sequences of reference and actual keys. It leads to a wide variety of the possible architectural realizations of self-monitored MPCUs. Three main architectural implementations for the fault-tolerant (self-checking) MPCU providing some pre-specified levels of error detection are presented in this paper, the while design

0-7803-8879-8/05/$20.00 ©2005 IEEE

procedure for one of them (so-called Compression & Generation based architecture) is discussed in detail.

The structure of the paper is as follows. The general description and notations for MPCU on-line monitoring is presented in Section II. Main generalised architectures of the control flow monitoring circuitry used in self-checking MPCU are discussed in Section III, while Section IV talks about possible strategies for MPCU self-checking. Sections V and VI are devoted to general step-by-step procedures for checking circuitry implementations for MPCU using standard and unique check key generators.

II. MPCU PROGRAM FLOW MONITORING

MPCU functioning can be presented by using a flow graph of its operation algorithm. Every vertex of the graph corresponds to one microinstruction. The flow graph can be considered as a set of k linear disjoint sections ($k \in \{1,2,3,...,N\}$, where N is the total number of the vertices, i.e., the total number of the microinstructions in the microprogram). Each i-th section of the flow graph contains some l_i vertices ($l_i \in \{1,2,3,...,N\}$ and $i \in \{1,2,3,...,k\}$), and j-th vertex of i-th section can be denoted as $Q_{i,j}$.

Faults can occur during the program execution. Fig. 2 shows some examples of such faults. They are [1-2]:

(a) microinstruction(s) disappearance in the sequence, or appearance of an additional one(s) in the sequence;
(b) faulty transition on logic conditions;
(c) erroneous transition from a functional (non-conditional) vertex;
(d) substitution of one micro-instruction by another.

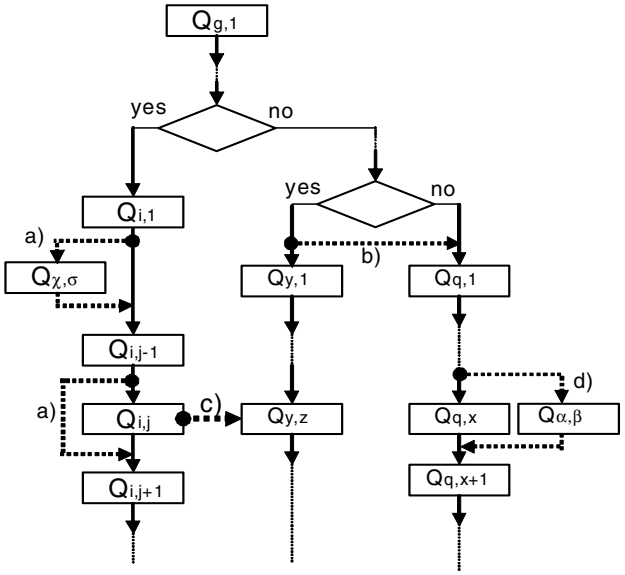

Figure 2. Microprogram execution fault types

All the above faults can be qualified as to be of a general "faulty transition" type (the case (d) can be considered as a double faulty transition: the program erroneously proceeds to some incorrect microinstruction while returning back to the correct path afterwards). These faults in the instruction sequence correspond to faulty transitions between two flow graph vertices, denoted, for instance, as $Q_{ij} \rightarrow Q_{yz}$.

The fault detection probability P_d can be used to estimate the reliability level for the adopted technique of flow graph checking. Achieving and guaranteeing some pre-specified value for P_d is among the main targets of the design of fault-tolerant MPCU.

Notation $P_d(Q_{ij} \rightarrow Q_{yz})$ can be used to denote the probability of detection of a single faulty transition between vertexes Q_{ij} and Q_{yz}. Particular value of the probability depends on such factors as a program flow graph configuration, type and distribution of the check keys, adopted checking strategy and check key processing methods.

The above notations were employed in developing the general description for the fault detection in the MPCU [1].

The detection of the faulty transition $Q_{ij} \rightarrow Q_{yz}$ can be considered as a combination of the following events:

(i) occurrence of a fault in a microinstruction corresponding to some vertex Q_{ij} of the program flow-graph,
(ii) occurrence of the faulty transition between the vertex Q_{ij} and some other vertex Q_{yz} (belonging to the same flow graph, or located outside of it),
(iii) detection of the transition by the on-line program monitoring tools incorporate into MPCU.

The probability that a fault is occurred in a vertex of the flow graph (i.e., in a microinstruction of the program of MPCU) depends on such factors as reliability of hardware, flow graph pattern, probabilities of generation of various external conditional logic signals (flags), etc. The values of such a probability for all vertexes of the flow graph can be summarized by a one-column matrix - vector $\boldsymbol{F}$ (so-called *Vector of Fault Occurrence*) of dimension N. Each element of $\boldsymbol{F}$ corresponds to a vertex of the flow graph and is equal to the probability that a fault occurs while the program in that vertex.

The probability of faulty transitions between the vertexes of the flow graph can be presented by the $N \times N$ square matrix $\boldsymbol{T}$ (*Transition Matrix*). Each element of the matrix is equal to the probability of a faulty transition between two corresponding vertices.

Finally, the probability of faulty transition detection by on-line checking tools can be presented by the square ($N \times N$) matrix $\boldsymbol{D}$ (*Detection Matrix*).

Based on the above notations, the probability of detection of a faulty transition can be expressed as:

$$\boldsymbol{P = F \; x \; T * D}.$$

Here symbol "$*$" denotes the element-by-element multiplication, while "x" - the conventional matrix-by-matrix multiplication operations. The elements of the matrix $\boldsymbol{P}$ present the probability of detecting faulty transitions between the corresponding vertices of the flow graph.

III. FAULT-TOLERANT MPCU ARCHITECTURES

As it was stated earlier, the control flow monitoring (on-line checking) is realized by means of comparing an actual sequence of the check keys generated by MPCU during its program execution against a reference sequence corresponding to fault-free operation of the control unit. Due to restrictions on the size of overhead memory in MPCU for storing the key values (actual and reference) various data compression (compaction) techniques can be employed [2].

First of all, the data compression can be applied to calculate the value (short code) of a single check key associated with every microinstruction.

Secondly, the compression can be used to reduce the amount of the data related to the sequence of the check keys. Here a long sequence can be compressed into a single short code containing information related to the entire prehistory of movement from the initial flow graph vertex to the current one (or flow graph fragment) during program execution.

Based on the use of the compression techniques three generic architectures of program flow monitoring and checking circuitry for MPCU can be proposed.

1. *"Compression-in-Space"-Based Architecture.* The architecture of the checking (program flow monitoring) circuitry of MPCU (Fig. 3) includes the *Parallel Correction Circuit* (PCC) that can be realized by means of combinational logic; *Buffer Register* (BRG) and a *Comparator* (COMP).

The checking circuit is connected to the *Microprogram Instruction Register* (MPIR) of MPCU where two additional fields are added: *Parallel Correction Key* (PCK) field and *Reference Key* (RK) field. The first group of the inputs of the Parallel Correction Circuit is connected to the check (flag) points F_1 of the circuit under control. At each time cycle PCC produces a so-called first level check key $W_k^1(Q_i)$ for the program vertex Q_i under execution.

The Parallel Correction Circuit normally realizes two functions. The first is related to the need to reduce the size (bit count) of the check key and flag data (i.e., *PCK* and *F1*). This is normally achieved by the use one of the known algorithms often called "space compression"[2], i.e., where a high bit count word with all its bits supplied in parallel is reduced to a shorter one by means of some specific bit manipulations. A good example of the "space compression" is a parity bit or Hamming code syndrome calculations.

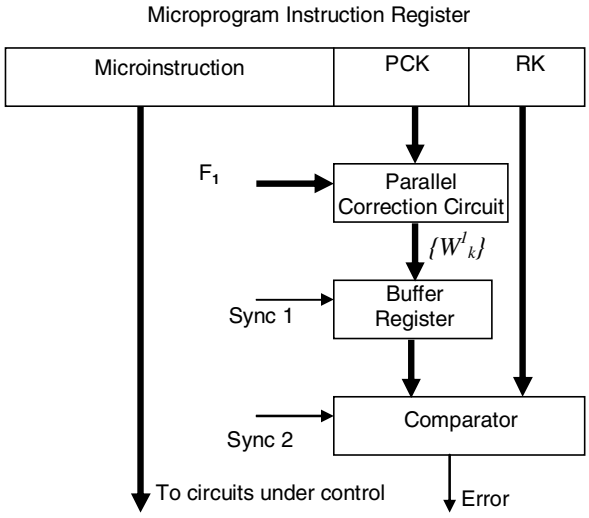

Fig. 3. Self-Checking MPCU using "Compression-in-Space" Technique

The second function is the generation of the required first level check key $W_k^1(Q_i)$ based on the data coming from the PCK field of the MPIR and the flag bits F_1 arriving from the circuit under control (after their compression).

The results of the generation are loaded into BRG using the *First Synchronization Signal* (Sync 1). The calculated (actual) check key is then compared with the reference value coming from the RK field of the next microinstruction. It is synchronized by *the Second Synchronization Signal* (Sync 2). This is to cater for a propagation delay as well as to provide the comparison between the corresponding values (the actual check key that is calculated during the current microinstruction execution and the reference key that is supplied on the next cycle of the microinstruction generation).

The simplified realization of the architecture may not contain the Parallel Correction Circuit. In such a case the content of the PCK filed is a direct value of the Reference Key of the next microprogram instruction.

2. *"Compression-in-Space (CiS) & Compression-in-Time (CiT)"-Based Architecture.* In this case the architecture of the CiS & CiT checking circuitry contains two additional fields in MPIR: *Sequential Correction Key* (SCK) field and *Sequential Correction Control Key* (SCCK) field. In addition a special *Sequential Correction Circuit* (SCC) combined with BRG is introduced (Fig. 4). The SCC has 5 inputs:

1) the input for the first level check keys coming from the Parallel Correction Circuit;
2) the correction key input (the keys come from SCK field of the Microprogram Instruction Register);
3) the control input (connected to SCCK field of MPIR);
4) the external flag input F_2 connected to the check (flag) points of the circuit under control;
5) the input of synchronization (Sync 1).

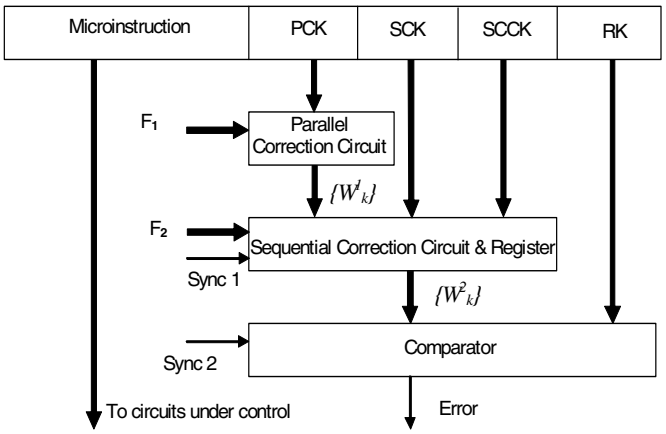

Fig. 4. "Compression-in-Space" cum "Compression-in-Time" Based Architecture

The Sequential Correction Circuit realizes one of the methods of digital data manipulation known as "time compression" (i.e., representation of a set of binary words arriving sequentially in time by just a single word by means of some compression algorithm). Classical examples of the "Compression-in-Time" circuits are *IIR Digital Filters* [3] and *Signature Analyzers* based on a *Linear Feedback Shift Registers* [4]. As a result of performing CiT over a sequence of check keys preceding to some vertex Q_i, a second-level check key $W_k^2(Q_i)$ is derived and put into correspondence to the vertex.

The F_2 input of the Sequential Correction Circuit & BRG block (in the same way the input F_1 of the Parallel Correction Circuit) are connected to different check points (flags) of the circuit under control. The latency of the error detection and the types of the detected errors depend on the selection of the check points in the circuits.

3. "*Compression & Generation"-Based Architecture.* In addition to the Parallel Correction Circuit and the Comparator this basic architecture (Fig. 5) contains a dedicated *Check Key Generator* (CKG).

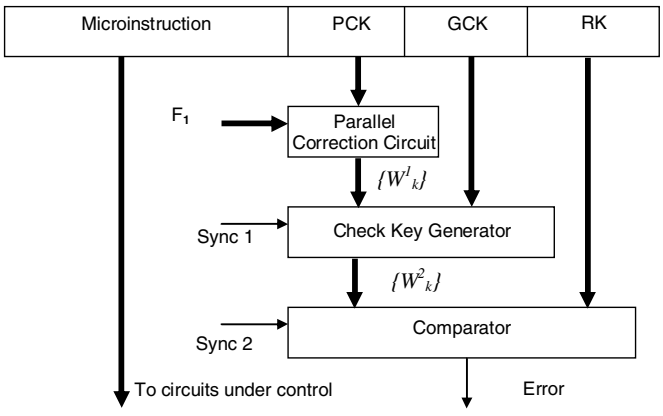

Fig. 5. Compression cum Generation Based Architecture

CKG is controlled at every cycle by the codes coming from the special *Generator Control Key* (GCK) field of the Microprogram Instruction Register. The output sequence of CKG is determined by the control signals coming from GCK field and the sequence of the first level check keys coming from the Parallel Correction Unit (that provides "compression-in-space" type of processing).

IV. STRATEGIES OF MPCU SELF-CHECKING

The presented above architectures can be used to develop original customized solutions providing required latency and fault coverage. For example, for the *cycle-by-cycle* checking strategy [1-2] (that is mainly discussed in this paper), the Compression-in-Space based architecture (shown in Fig. 3) is trivial. However it might not be the best one. The CiS & CiT-based architecture (Fig. 4) also can be applied in this case. At the same time such a solution would require rather substantial calculations for values of the correction and reference check key sequences. Besides, it could lead to quite high bit count for the PCK and SCK fields thus leading to significant hardware overhead.

The paper concentrates on the use of the third architecture (Fig. 5). The main advantage of the architecture is in its versatility: it can be use for both *cycle-by-cycle* and *interval-by-interval* checking modes.

There are two types of possible implementation of this architecture in self-checking MPCU.

The first one is based on the use of some standard digital sequence generators (for example, multi-bit M-sequence generators/signature analysers based on the Linear Feedback Shift Registers, cellular-based generators, etc.) as CKG. In such a case the generator provides a new output code at each cycle of the signal Sync 1 in accordance with its internal organisation. At the same time the state of the generator (and thus the value of its output) can be changed by the correction signal coming from GCK. After the correction, the Check Key Generator continues its operation from the new state (in a way, CKG can be considered as a classic example of Moore-type finite state machine).

The second approach is based on assigning of the pre-specified key sequence to the microprogram under check (the check keys are assigned to the vertices of the flow chart representing the microprogram). Based on the sequence the CKG is then synthesized.

Such a solution could potentially provide excellent results in terms of fault coverage (the initial check key sequence can be chosen as sophisticated as needed to ensure the required fault detection). However, in many cases this may be not possible or feasible (for example, due to a very specific type of a microprogram). Beside that it might lead to an excessive hardware overhead due to a very complicated generator.

The above two approaches have been researched, and the summary of the design procedure for both of them are presented below.

V. MPCU SELF-CHECKING WITH THE USE OF STANDARD DIGITAL SEQUENCE GENERATORS

Let a multi-channel generator (MCG) implemented by means of a parallel M-sequence generator/signature analyser is used as CKG. Such a type of generators/analysers is well-known from the literature and is widely used in test technology [5-6]. A very good example of the circuit of this type is BILBO that was first introduced as early as in the end of 70s [7].

One of the very useful properties of such MCG is that it generates reasonable good quality sequence of multi-bit binary numbers when its inputs are zeroed.

The period of the generated sequence depends on the generator's particular architecture [5, 8].

Let also the design goal would include providing the uniform fault detection probability level (P_0=*const*) across the entire microprogram (i.e., any faulty transition from any vertex Q_i of the microprogram flow graph is to be detected with the guaranteed probability not lover than the given level P_0, i.e.,

$$P_0(Q_i) = P_0 \ \forall i = \overline{1, N} \ ,$$

where N is in the number of the vertexes (microinstructions) in the microprogram). This would ensure the guaranteed fault detection level across the microprogram.

The number of the possible check keys used in the coding of the microprogram can be related to the number of possible states of the MCG. And the fault detection probability can be related to the number of the MCG state repeated within the microprogram (if all the states are unique, the fault detection probability would be of 100%).

The check circuitry design procedure can be presented as a sequence of the following steps.

1. On the basis of the given level of guaranteed fault detection probability P_0 the upper boundary for the number of allowable repetition of the check keys is found.
2. The type of MCG and the number of its bits (hence the period) are found for the given upper acceptable boundary for the number of allowable repetitions of the check keys (found as a result of step 1).
3. The entire flow chart of the microprogram is partitioned into a number of linear disjoint fragments (i.e., the fragments that do not share common vertexes). The partition has to adhere to the following rules:
   - all vertexes (except one) preceding to a vertex of branch merging have to be the end vertexes of their corresponding fragments;
   - all vertexes successive to a logic condition one, are the beginning vertexes of their corresponding fragments;
   - fragments of a loop-type are not allowed.

   Some unique identification number is assigned to each of the fragment (e.g., it can be just simple numbering 1, 2, 3,…).

   A single long linear program thread is constructed by placing the fragments one after another according to their identification numbers in the ascending order. Some arbitrary initial state (seed) is assigned and loaded into MCG. The entire set of the consecutive states of the generator is then obtained assuming that all the MCG inputs values are equal to zero.
4. The states are assigned to the vertexes of the linear program constructed in the step 4. The assigned codes are now the reference check keys for the vertexes.
5. Based of the assigned reference check keys $\{W_{ref}\}$, the following values are calculated for each given vertex Q_i of the linear program:
   - the value at the Generator Control Key (GCK) field of the Microprogram Instruction Register;
   - the value of the first level correction key ($\{W^1_k(Q_i)\}$) coming from the output of the Parallel Correction Circuit (PCC).
6. By taking into account design requirements for the fault detection latency as well as targeted fault types, the check (flag) points F_1 of the circuit under control are selected.

   It is important to mention that properly designed PCC allows to detect not just simple faulty transitions between vertexes of the microprogram flow graph ($Q_i \rightarrow Q_j$), but also faults of several other types such as distortions in the 'body' of the microinstruction codes without affecting the sequence of the microprogram flow [2].
7. On the basis of the required level of the probability of fault detection in the microinstruction codes, the algorithm CiS is chosen for the Parallel Correction Circuit.
8. For the chosen algorithm, and by taking into account the number of check points in the circuit under control (i.e., the bit number of the input F_1) as well the word length of PCC output, the architecture of the Parallel Correction Circuit is developed and the length of PCK field of the microinstruction is chosen.
9. Finally, by taking into account the codes in the fist input (F_1) and those needed to be generated on PCC output at each clock cycle, the value of PCK field are found for each microinstruction.

If detection of faults in the 'body' of microinstruction is not targeted, the Parallel Correction Circuit may not be required. In such a case the PCK codes can be used to directly change the state of the Check Key Generator when transiting from one fragment of the microprogram to another.

The steps 8, 9 and 10 of the above design procedure can be removed, while the step 11 would be changed. The step 11 would request to place into PCK field of the microinstructions the codes providing direct correction of the states of CKG at the vertexes when the microprogram performs a transition from one its linear fragment to another one.

VI. MPCU SELF-CHECKING BASED ON THE USE OF UNIQUE DIGITAL GENERATORS

The main difference between the implementation of the unique generator based approach and that where the standard generators are used (presented in Chapter V), is the need to synthesise the generator (sequential program automata) as well as a control circuitry for it. The synthesized generator has to produce the pre-specified sequence of the check keys. These keys are then compared with the key sequence coming from the RF field of the microinstruction (reference key sequence) during the microprogram execution.

The design procedure includes the following steps.

1. The flow chart is partitioned into a number of non-overlapping fragments (the same as in Step 3 of the standard generator based approach)
2. The probability of faulty transition detection for every vertexes of the microprogram flow graph are found by using the earlier mentioned expression $P = F \times T * D$ as well the techniques known from the literature [2].
3. Based on the above detection probabilities for possible faulty transitions, the distribution and the assignment of the check keys are derived. This can be achieved by using, for example, the techniques presented in [2], or by employing some other algorithms known from the literature [9].
4. Based on the known sequence of the check keys, the CKG is synthesised as a synchronous sequential automata and its control sequence supervising the generation of the required check key at every time cycle is found.

The other steps of the design procedure correspond to the steps 5-11 of the above presented procedure for the design based on the use of the standard generators.

VII. CONCLUSIONS

The presented above material is related to just out of the many interesting topics in the area of MPCU self-checking – synthesis of the control units providing pre-defined fault coverage. In particular, we concentrated on the architectures and general design strategies for the program-flow monitoring circuitry of the fault-tolerant (on-line self-checking) microprogram control units.

There are three important issues that need to be considered when implementing the concurrent self-checking in MPCU, they are: the hardware overhead, fault detection latency and fault coverage. It is important to bear in mind when the signing the checking circuitry that all the three characteristics are closely linked. If higher fault coverage is specified, then generally longer check keys and more comprehensive sequence compression/fault detection algorithms are required. However if the allowed hardware overhead is limited, then the fault latency would generally be sacrificed. If the hardware overhead is limited then the fault coverage or fault latency would suffer, and so on.

The described in the paper three basic architectures provide different levels of fault detection, hardware expenses and latency. The most suitable architecture can be chosen for an every particular case depending on design requirements. In overall the Compression & Generation based architecture could potentially provide the best overall characteristics in terms of hardware overhead and latency for a given fault detection level. And that is why the main attention of the paper has been devoted to that particular architectural solution.

REFERENCES

[1] S. Demidenko, E. Levine and V Piuri: Synthesis of On-Line Testing Control Units: Flow Graph Coding/Monitoring Approach – *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct. 25-27, 2000, Yamanashi, Japan, p. 266-274

[2] S. N. Demidenko, E. M. and K. V. Lever: Concurrent Self - Checking for Microprogrammed Control Units: An Analytical Survey - *IEE Proceedings, Part E,* 1991, Vol. 138, p. 377-388.

[3] R. W. Hamming: Digital Filters, Prentice Hall, NJ, 1989

[4] S. Demidenko and V. Piuri: Signature Compression: A Generalised Description and Its Applications - *Journal of Microelectronic Systems Integration*, 1995, Vol. 3, No 1, p. 19-34.

[5] P. H. Bardell, W. H. McAnney and J. Savir: Built-In Test for VLSI, John Wiley, New York, 1987

[6] M. L. Bushnell, V. D. Agrawal: Essential of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits: Kluwer Academic Publishers, 2000

[7] B. Koenemann, J. Mucha and G. Zwiehoff: Built-In Logic Block Observation Techniques, *Proceedings ITC'79*, Oct. 1979, pp. 37-41.

[8] V. Yarmolik and S. Demidenko: Generation and Application of Pseudorandom Sequences in Random Testing, John Wiley and Sons, 1986

[9] S. N. Demidenko, E. M. Levin and K. V. Lever: Concurrent Self-Checking for Microprogrammed Control Units: An Analytical Survey – *IEE Proceedings, Part – E*, Vol. 138, No 6, 1991, pp. 377-388