ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2008, Vol. 37, No. 1

HANDLING MULTIPLE FAILURES IN PROCESS NETWORKS

Jonas Čeponis, Egidijus Kazanavičius

Department of Computer Engineering, Kaunas University of Technology
Studentu st. 50-213, LT-51368 Kaunas, Lithuania

Lina Čeponienė

Kaunas University of Technology, Information Systems Department
Studentu st. 50-315a, LT-51368 Kaunas, Lithuania

Abstract. Process network is a computation model used in digital signal processing. This model is effective for solving various tasks, which can be divided into multiple concurrent processes. These processes communicate through unbounded FIFO buffers and can be executed simultaneously. In real time digital signal processing applications execution time is often infinite. However, various hardware/software failures can occur. These failures can be single or multiple. In our work, dynamic run-time reconfiguration is introduced into process network which ensures single or multiple error handling, avoiding deadlocks, continuous and on-time result delivery. In this paper, we extend Fault Tolerant Process Network with algorithm for avoiding multiple simultaneous failures, which is also suitable for avoiding a single failure.

Keywords: Process Networks, Multiple Failures, Failure Handling, Dynamic Reconfiguration, Concurrent Processes, Labelled Transition Systems.

1. Introduction

Process networks are often used for solving various digital signal processing (DSP), multimedia or control tasks. Usually tasks of this type are called data flow applications. If the task can be divided into a few smaller tasks and they can be executed simultaneously, process network is a suitable implementation solution.

Depending on implementation, the application may be control oriented and/or event driven. Such systems respond to changes of internal states or environment. In response to these changes, the system must adequately react. Changes of internal states can be caused by various hardware/software failures [1]. A well-designed system should always have a powerful mechanism for handling failures [21].

Our previous work on Fault Tolerant Process Network (FTPN) or Error-Proof Process Network (EPPN) is presented in [4, 5]. In FTPN, network reconfiguration is used to ensure handling a single failure in process network. FTPN solution does not ensure that all failures will be handled and processed and network will continue its execution. There was no solution for handling multiple failures in FTPN.

Error-proof data flow applications should use general computational model for design, modelling, analysis and implementation in various platforms. This ensures rapid application design and utilisation of existing software/hardware components. Such computational models are Synchronous Data Flow (SDF) [13], Parameterized Synchronous Data Flow (PSDF) [12], Discrete-Event (DE) [12], Kahn Process Network (KPN) [3, 11], etc.

Kahn Process Network is a subset of more general Process Network (PN) model. PN consists of concurrent processes communicating over first-in first out unidirectional queues. PN is useful for modelling and exploiting functional parallelism in streaming data applications. The PN model maps easily onto multiprocessor and/or multi-threaded targets. KPN ensures completely dynamic execution of nodes. In KPN, there is no need for describing scheduling during system design, because scheduling does not affect the functional behaviour of the nodes. These features of KPN are very useful in dynamic network reconfiguration [23].

Synchronous Data Flow network is useful for modelling simple dataflow systems without complicated flow of control. In SDF, nodes (processes) read and write a fixed number of tokens each time they are executed. The number of tokens and execution time are defined during system design. Process Network differs significantly from SDF, as PN uses completely dynamic execution of nodes. In PN, nodes are executed asynchronously, but the result of network

execution is deterministic. Determinism is ensured by blocked reading from channel and non-blocked writing into channel.

Parameterized Synchronous Data Flow is useful for modelling dataflow systems with reconfiguration. PSDF represents a design point between complete static scheduling in SDF and completely dynamic execution in Process Networks.

Discrete-Event model differs from other discussed models as it supports time oriented models of systems such as queuing systems, communication networks and digital hardware.

In this paper, the improved Fault Tolerant Process Network (FTPN) model is presented, which is based on Kahn process network model. Because of dynamic reconfiguration used in FTPN, execution results may become non deterministic, but this helps avoiding critical termination of network execution.

DSP real time systems often require non-standard and costly hardware and software solutions. Modern workstation can represent an alternative to develop real time intensive various purposes signal processing applications. Furthermore, the programming model of Process Network corresponds completely to this kind of applications and fits perfectly on multiprocessor systems.

Various systems developed using Process Network can be modelled using workstation and even personal computer. This helps in finding and processing critical network operation points. This also helps to simulate possible network failures, which can be single or multiple in time [20].

Application performance can be improved dynamically changing network parameters. On the other hand, hardware systems (embedded systems, programmable logic, etc.) dedicated to a particular problem class can be used to minimize processing time. Process Network implementation in the hardware system would help in verifying efficiency of the solution in the particular situation.

The remainder of this paper is structured as follows. The next section discusses related works. This is followed by the formal definition of improved Fault Tolerant Process Network model. Next, rules for handling single and multiple failures are discussed. In section 5, we conclude and present suggestions for the future.

2. Related work

Process Networks (dataflow networks) are a suitable model of computation for developing streaming multimedia and digital signal processing applications [14]. The most popular process network models for streaming applications are Synchronous Data Flows and Kahn Process Networks.

Synchronous Data Flow network is applicable to simple dataflow systems without complicated flow of control. In SDF, a node produces and consumes a fixed number of data tokens on each of its outgoing and incoming channels during each activation. For activation of the node it must have at least as many tokens on its input channels as it needs to consume. The number of tokens and execution times must be defined during system design.

KPN is a computation model in which many concurrent processes communicating over first-in first-out unidirectional queues can be executed simultaneously [18]. KPN is mostly applicable for parallel processing of streaming data. Many of the existing KPN modelling tools (Ptolemy [12], Compaan/Laura [22, 24]) are commonly used for simulation rather than implementation. In analyzed dataflow process networks modelling methods, two important issues were considered: detection of critical execution points and dynamic network reconfiguration [9].

Since the PN model is Turing complete, memory requirements cannot be predicted statically. In general, any bounded-memory scheduling algorithm for this model requires run-time deadlock detection. The few PN implementations that perform deadlock detection detect only global deadlocks. Not all local deadlocks, however, will cause a PN system to reach global deadlock. Olson and Evans [17] presented local deadlock detection algorithm for PN models based on the Mitchell and Merritt algorithm. Their algorithm is suitable for both parallel and distributed PN implementations.

In order to capture the interaction between input events and execution units as well as reconfiguration in dynamic stream processing, Reactive process Networks (RPN) are introduced [7]. The foundation for RPN was laid by efforts to integrate dataflow model and its reactive behaviour [8]. Reactive behaviour in these models is commonly specified using hierarchical state machines.

Another means for specifying dynamic network reconfiguration during run-time is parameterizable SDF model [2, 6]. In PSDF, node execution is characterized by iterations that fire subprocesses in a particular order. Node execution can be reconfigured between iterations at run-time [19].

Yet another approach for dynamic process network reconfiguration is presented in the work of Neuendorffer and Lee [16]. It is concentrated on reconfiguration as a particular kind of event handling. The states of the network, in which reconfigurations are allowed, are named quiescent states. The FIFO channel communication is used for sending and receiving events or parameters and for dividing input ports into streaming input ports and parameter input ports. This work focuses primarily on reconfiguration of SDF networks. This work along with the other discussed works on dynamic reconfiguration focuses on using reconfiguration for increasing efficiency.

FTPN presented in our work differs from the above discussed approaches because it presents another point of view to the purpose of process network

dynamic reconfiguration. This point of view is based on the idea that we must specify critical moments in network execution in order to avoid critical termination. These critical moments appear when communication between network nodes is broken or node fails. In initial stage, algorithms for single failure handling were developed and successfully tested [4, 5]. However, multiple failures can occur in real world and we trying to improve fault tolerant process network with the ability to handle these failures. Fault Tolerant Process Network proposed in this work can be reconfigured dynamically in order to capture and process single and multiple simultaneous failures of network execution.

3. A Model of Fault Tolerant Process Network

In our research the Fault Tolerant Process Network is developed. This process network is dynamically reconfigurable and the main purpose of dynamic network reconfiguration is reaction to failures in process network and handling of these failures. In [4] only handling of single failure is described. In this paper we present the enhanced Fault Tolerant Process network which is able to handle multiple failures.

First, some common definitions and notations for process network are described. For FIFO channel specification there is universal finite set of channels CH and for every channel $ch \in CH$ there is a corresponding finite channel alphabet Σ. Each channel $ch \in CH$ is described by its length L and pointer $ch(p)$ which refers to the last data record in the channel. The actions for data transfer through the channel are $ch \rightarrow a$, $ch \leftarrow a | ch \in CH,\ a \in \Sigma$. The action $ch \leftarrow a$ denotes input of data into channel. The action $ch \rightarrow a$ denotes output of data from channel. These actions form the set of actions for data transfer $Ac=\{ch \rightarrow,\ ch \leftarrow a | ch \in CH,\ a \in \Sigma\}$.

For network nodes specification, a universal finite set of nodes N is used and for every node of this set $n \in N$ there is a corresponding set of atomic actions Act. All actions of all network nodes are defined by the set A and the actions of every node $Act \subseteq A$. Every node has a set of input and output channels $(ch_{in}, ch_{out}) \in CH$. The duration of execution of each atomic action is specified. This time is used to ensure that final result will be reached in proper time. The operation of each node must be defined as a sequence of atomic actions. Such specification is necessary for implementation of dynamic reconfiguration and change of network parameters when operations must be allocated from one node to another. The set of operations for each node Act_n is specified as a sequence of atomic actions $(Act_1, Act_2, \ldots, Act_{k-1}, Act_k)$, where k is the number of atomic actions in the node. The nodes in FTPN can perform operations $Cntrl_n$ which are used for network reconfiguration and changing network parameters.

All node operations can be divided into two groups – communication and computation operations. During communication operations the node performs data input and result output. After successful data input the node performs computation operations. When computations are over, the node writes output data to output channels ch_{out}. Control operations are executed in case of dynamic network reconfiguration during handling of failures. Execution of control operations can be planned (if reallocation of network resources is required) or unplanned (in case of failure of network element).

The node can be in one of the states $Ns = \{ns_{\perp}, ns_r, ns_{br}, ns_w, ns_{bw}, ns_{ex}, ns_c\}$. Initial node state $ns_{\perp}$ denotes the starting point of node execution. In this state, initial working parameters and values of variables are set for the node. Afterwards, the node moves to reading state ns_r and tries to read data from input channels $ch_{in} \in CH$. If at least one $ch(p)=null \mid ch \in ch_{in}$, the node transits to state ns_{br} and waits until data is available in the channel. When the node has successfully read data from input channels, it moves to state ns_{ex} and executes actions $Act \subseteq A$. After finishing execution, the node transits to writing state ns_w and writes results to its output channels $ch_{out} \in CH$. If at least one $ch(p) \neq null \mid ch \in ch_{out},\ p=L$, the node moves to blocked writing state ns_{bw} and waits for available free space in the output channel. When the node has finished writing data to channels, it moves to state ns_r.

The change of network execution parameters may be required in two cases: a node is in blocked reading or blocked writing states (ns_{br} or ns_{bw}) and timeout occurs; or external request is received. In any of these two cases the node moves to control state ns_c and changes required parameters. Afterwards, the node transits to reading or writing state (ns_r or ns_w) and continues execution.

In FTPN all nodes are divided into two groups: internal nodes (having both input and output channels) and interface nodes (having only input or only output channels).

The FSP specification of the node which has both input and output channels is presented below:

```
range T = 1..2
NodeInOut = Initial,
Initial = (parameters_reading -> Reading),
Reading = (reading_done -> Execution |
channel_empty -> BlockedReading),
Execution = (execution_done -> Writing),
Writing = (writing_done -> Reading | channel_full ->
BlockedWriting),
BlockedReading = (channel_notempty -> Reading |
reading_timeout -> Control[1]),
BlockedWriting = (channel_notfull -> Writing |
writing_timeout -> Control[2]),
Control[t:T] = (when (t==1) reading_continue ->
Reading | when (t==2) writing_continue -> Writing).
```

The FSP specification can be mapped to LTS [10] graph using the tool LTSA 2.2 [15]. The LTS graph for the specified internal node is presented in Fig. 1.

The channel can be in one of the states defined in set $Sc=\{sc_{\perp},sc_{w},sc_{cf},sc_{ce},sc_{pop},sc_{push},sc_{inc},sc_{dec}\}$. After setting initial parameters in state $sc_{\perp}$, channel transits to waiting state sc_{w}. In this state channel waits for requests from the nodes. When request from writing node arrives, channel moves to state sc_{cf}, in which it checks the availability of free space in channel memory. If there is a free space in memory, channel moves to the state sc_{push}. During this state the incoming token is written to the channel. When request from reading node arrives, channel moves to state sc_{ce}, in which it checks the availability of data in channel. If there are at least one data token, channel moves to the state sc_{pop} and sends first data token to reading node. When the token from the writing node arrives and the channel is full, channel transits to the state sc_{inc} in which the length of the channel is increased. After successful increasing of the channel length it moves to state sc_{push}, otherwise it moves to state sc_{w}. The channel can transit to state sc_{dec} when a request from reading node arrives and other channels require to be increased. In state sc_{dec}, the length of the channel is decreased thus freeing memory for other channels.

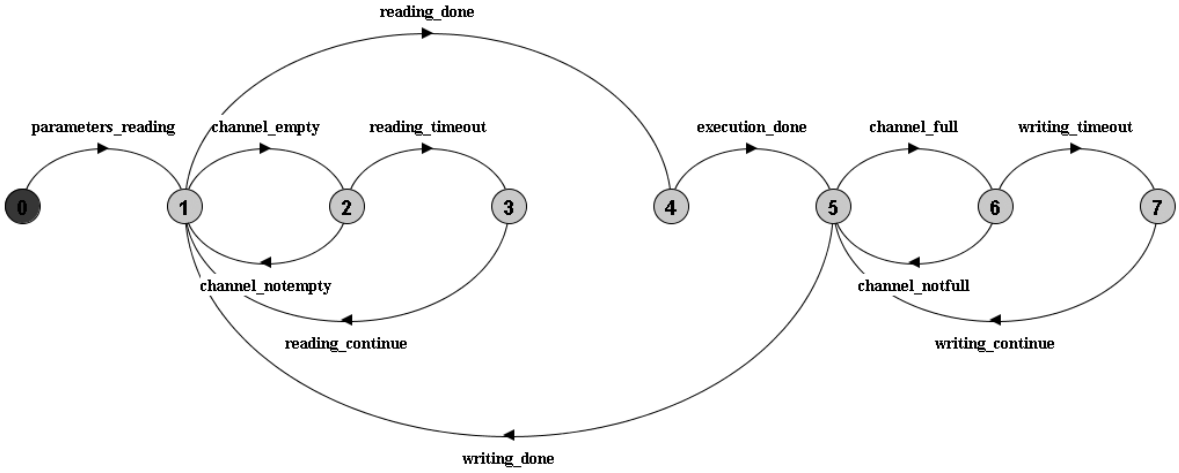

Figure 1. The States of Internal Nodes in FTPN

The FSP specification of channels in FTPN is presented below:

Channel = Initial,

Initial = (set_parameters -> Waiting),

Waiting = (writing -> CheckFullness | read -> CheckEmptiness),

CheckFullness = (full -> Waiting | notfull -> Writing | increment -> IncrementSize),

Writing = (write_ok -> Waiting),

IncrementSize = (increment_ok -> Writing | increment_nok -> Waiting),

CheckEmptiness = (empty -> Waiting | notempty -> Reading | decrement -> DecrementSize),

DecrementSize = (decrement_ok -> Reading),

Reading = (reading_ok -> Waiting).

This FSP specification is mapped to LTS graph which is presented in Figure 2.

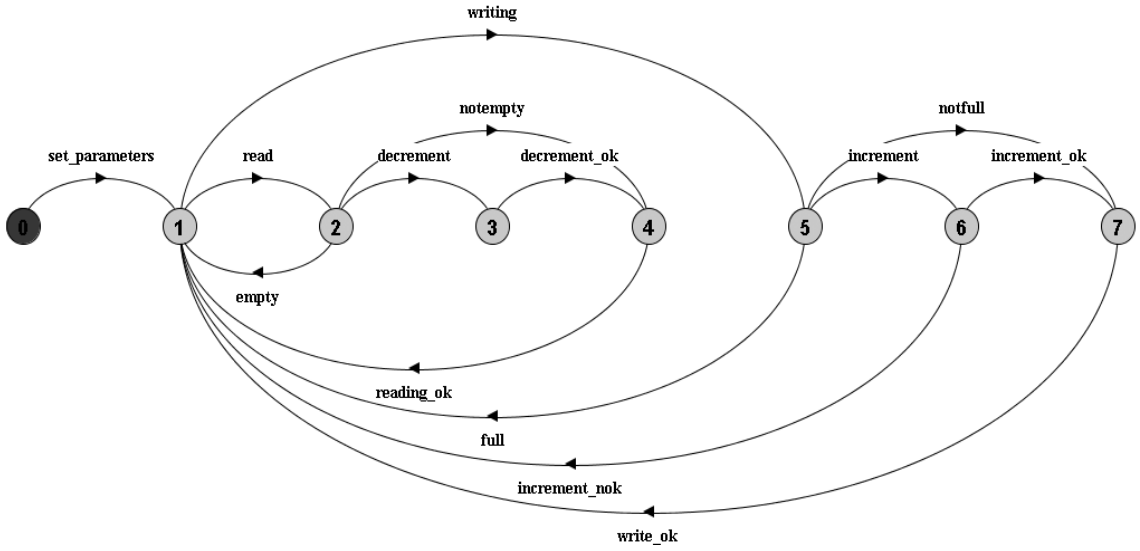

Figure 2. The States of Channels in FTPN

4. Algorithms for Handling Multiple Faults in Fault Tolerant Process Network

The algorithms for handling faults of elements in process network must be specified to ensure that network continues execution after failure. As the main elements of the process network are nodes and channels, the algorithms are specified for the cases of network node failure and for network channel failure. Network nodes are grouped into internal nodes (having input and output channels), interface nodes having only input channels, and interface nodes having only output channels.

In case of network node failure we need to redistribute its actions to other node. We also need to redistribute the input and output channels of the faulty node. In order to minimize data loss, the reconfiguration must follow these rules:

1. All nodes connected with faulty node N_i perform their actions until:
   a) ch_{in} of the node N_i become full;
   b) ch_{out} of the node n become empty.
2. The nodes connected with faulty node N_i transit to blocked reading or blocked writing states.
3. After timeout the nodes connected with faulty node N_i transit to control state. In this state:
   a. actions of the faulty node N_i are transferred to node N_{i+1} which first transits to control state and is connected to output channel of N_i;
   b. the channel between N_i and N_{i+1} is destroyed;
   c. output channels of N_i are connected to N_{i+1};
   d. input channels of N_i are connected to N_{i+1};
   e. the data lost during failure is compensated.
4. The nodes which are in control state move to writing or reading states and reconfigured network continues execution.

These rules are applicable to network nodes which have input and output channels $(ch_{in}, ch_{out}) \subseteq CH$. There are two exceptions when these rules cannot be applied: interface nodes having only output channels and interface nodes having only input channels.

In distributed process network, channel failure is also critical for network execution and can cause a global deadlock. In case of network channel failure, the change of network parameters must follow these rules:

1. The nodes connected by faulty channel transit to blocked reading and writing states ns_{br} and ns_{bw}.
2. After timeout the nodes connected by faulty channel transit to control state ns_c.
3. Creation of a new channel is initiated by the node which first transits to control state.
4. The new communication channel is connected to the reading node as its input channel.
5. The new communication channel is connected to the writing node as its output channel.
6. The data, lost during failure, is compensated.
7. The nodes which are in control state move to writing or reading states and reconfigured network continues execution.

Single failure handling implementation was described in [4, 5]. In this paper we try to improve FTPN to handle multiple failures at the same time. In case of multiple network node failures we need to redistribute their actions to other nodes. We also need to redistribute the input and output channels of the faulty nodes. Data loss can occur each time of network element failure. To solve this problem, we introduce the default value to compensate lost data. This conflicts process network determinism feature but enables further execution and helps to synchronize data.

In Figure 3 a fragment of process network is presented. For multiple network failure simulation, we assume that nodes $ni_{1,m}$, $nout_1$ and channel between them fails (dashed lines in picture). In this case, network node $ni_{1,m-1}$ goes to ns_{bw} state when channel between $ni_{i,m-1}$ and $ni_{1,m}$ has become full. Similarly nodes $ni_{2,m}$ and $nout_2$ go to ns_{br} state, when channels connecting nodes $ni_{1,m}$ and $ni_{2,m}$ and nodes $ni_{1,m}$ and $nout_2$ have become empty.

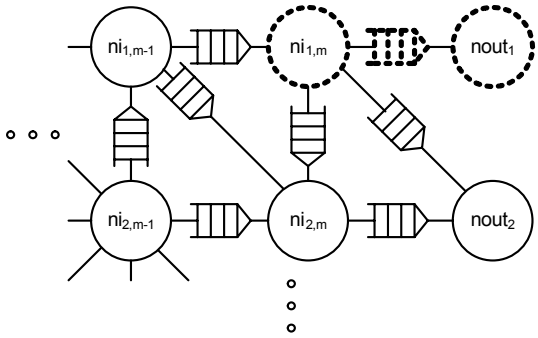

Figure 3. Multiple failures in process network

After timeout one of the blocked nodes goes to control state ns_c and begins a recovery from network deadlock situation. Recovering starts with finding faulty network elements, then transferring actions of faulty nodes to working node, redistributing channels, consuming data from unneeded channels and at last deleting unneeded channels. In our case node $nout_2$ goes to control state and starts recovery from network deadlock. Node timeout originates during reading operation from channel between nodes $ni_{1,m}$ and $nout_2$. Therefore node $nout_2$ starts to search for failures in this direction. First of all, node $nout_2$ finds out that node $ni_{1,m}$ is not working. Then node $nout_2$ follows network path in all directions from faulty node $ni_{1,m}$ and tries to find all failed network elements. In our case node $nout_2$ finds that failed elements are nodes $ni_{1,m}$, $nout_1$ and the channel between them. After finding all failed elements, node $nout_2$ starts redistributing actions and channels. Network model after redistribution is presented in Figure 4.

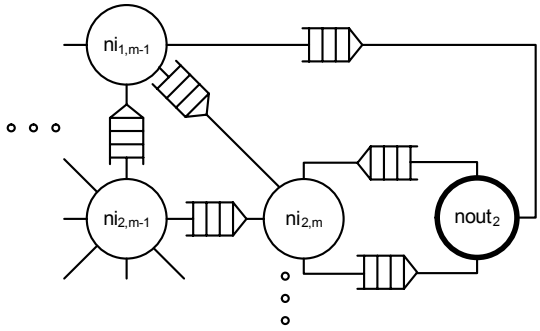

Figure 4. Process network after failures handling

In order to minimize data loss, the change of network parameters in case of multiple failures must follow these rules:

1. The nodes perform their actions until input channels become empty or output channels become full.
2. Nodes connected with failed elements transit to blocked reading or blocked writing states.
3. After timeout only one of the blocked nodes transits to control state. In this state:
   a. node follows network path in the direction of the faulty element and finds all faulty elements;
   b. actions of the faulty nodes are transferred to the node which first transited to control state;
   c. operational output channels of failed nodes are included into the set of output channels of the node which first transited to control state;
   d. operational input channels of failed nodes are connected to node which first transited to control state;
   e. if the sequence of failed elements ends with a channel, then the new channel is created and connected according to the rule 3c or 3d.
4. The nodes which are in control state move to writing or reading states and reconfigured network continues execution.

These rules are applicable to handle multiple network elements failure.

In order to analyze the behaviour of process network in case of hardware failure we are using a multithreaded implementation of FTPN. The FTPN was implemented in C# programming language using multiple threads running at the same time and performing different tasks of process network. We used separate thread for each element of the process network (node and channel) and the main program for coordination. The failures of network elements were imitated by destroying a thread of the node or channel.

5. Conclusion and future work

Various process networks are used for solving DSP tasks. In real time digital signal processing applications, execution time is infinite. However, various hardware or software failures can occur. In our work, dynamic run-time reconfiguration is introduced into process network which ensures handling of multiple failures.

Fault Tolerant Process Network presented in this paper is modelled as a labelled transition system. Formal specification of FTPN and the states of network elements during execution is presented. This specification was used as a base for describing the rules for network run-time reconfiguration in case of single or multiple network element failure. These rules minimize data loss and enable further execution of FTPN.

Future works include extended implementation and testing of FTPN in a distributed system and handling various multiple failures.

References

[1] **A. Avizienis, J. C. Laprie, B. Randell, C. Landwehr.** Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, *Vol*.1, *No*.1, 2004, 11-33.

[2] **B. Bhattacharya, S. Bhattacharyya.** Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing,* 49(10): 2001, 2408-2421.

[3] **J. Čeponis, E. Kazanavičius, A. Mikuckas.** Design and analysis of DSP systems using Kahn process networks. *Ultragarsas*, 2002, *Vol*.45, ISSN 1392-2114, 43-46.

[4] **J. Čeponis, E. Kazanavičius, A. Mikuckas.** Fault Tolerant Process Networks. *Information Technology and Control*, ISSN 1392-124X, *Kaunas, Technologija*, 2006, *Vol*.35, *No*.2, 124 - 130.

[5] **J. Čeponis, E. Kazanavičius.** Error-Proof Process Network Model. *Proc.* 3*rd International Conference on Cybernetics and Information Technologies, Systems and Applications: CITSA* 2006, ISBN 980-6560-19-1, 2006, 153-158.

[6] **M. Dyer, M. Platzner, L. Thiele.** Efficient Execution of Process Networks on a Reconfigurable Hardware Virtual Machine. *Proc.* 12*th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM'04), 2004.

[7] **M. Geilen, T. Basten.** Reactive Process Networks. *EMSOFT*'04*,* 2004.

[8] **A. Girault, B. Lee, E. Lee.** Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems,* 18(6):742-760, 1999.

[9] **D. Hofstee, B.H.H. Juurlink.** Determining the criticality of processes in Kahn process networks for design space exploration. *Proceedings ProRISC* 2002, *Veldhoven, The Netherlands*, 2002, 292-297.

[10] **M. Huth.** Labelled transition systems as a Stone space. *Logical Methods in Computer Science*, 2005, 1–28.

[11] **G. Kahn.** The semantics of a simple language for parallel programming. *Information Processing* 74: *Proc. IFIP Congress* 74, 1974, 471-475.

[12] **E. Lee.** Overview of the Ptolemy project. *Technical Memorandum UCB/ERL No*.M01/11, *University of California, EECS Dept.*, 2001.

[13] **E. Lee, D. Messerschmitt.** Synchronous data flow. *IEEE Proceedings,* 75(9): 1987, 1235-1245.

[14] **E. Lee, T. M. Parks.** Dataflow process networks. *Proceedings of the IEEE,* 83(5): 1995, 773-798.

[15] **J. Magee, J. Kramer.** Concurrency: State Models and Java Programs. 2*nd Editon*. ISBN: 0-470-09355-2. 2006, 432.

[16] **S. Neuendorffer, E.A. Lee.** Hierarchical reconfiguration of dataflow models. *In Proc. Second ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE* 2004*), IEEE Computer Society Press*, 2004.

[17] **A.G. Olson, B.L. Evans.** Deadlock detection for distributed process networks. *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Proc., Vol*.5, 2005, 73-76.

[18] **A.D. Pimentel, C. Erbas, S. Polstra.** A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. *IEEE Transactions on Computers*, *Vol*.55, *No*.2, 2006, 99-112.

[19] **M. Sen, S. Bhattacharyya, T.L. Wolf**. Modeling Image Processing Systems with Homogeneous Parameterized Dataflow Graphs. *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing,* (ICASSP '05), ISBN: 0-7803-8874-7 *Vol*.5, 2005, 133-136.

[20] **T. Skeie**. Handling Multiple Faults in Wormhole Mesh Networks. *Proc.* 4*th international Euro-Par Conference on Parallel Processing*. *LNCS*, *Vol*.1470. Springer-Verlag, 1998, 1076-1088.

[21] **D.J. Smith.** Reliability, Maintainability and Risk (6 edition). *Newnes*, 2001, 370.

[22] **T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, E. Deprettere.** System Design using Kahn Process Networks: The Compaan/Laura Approach. *Proc*. 7*th Int. Conf. Design, Automation and Test in Europe* (DATE 2004), 2004, 340-345.

[23] **A. Turjan, B. Kienhuis, E. Deprettere.** A hierarchical classification scheme to derive interprocess communication in process networks. *Proc.* 15*th IEEE International Conference on Application-Specific Systems, Architectures and Processors,* ISBN: 0-7695-2226-2, 2004, 282-292.

[24] **C. Zissulescu, T. Stefanov, B. Kienhuis, E. Deprettere.** LAURA: Leiden Architecture Research and Exploration Tool. *Proc.* 13*th Int. Conference on Field Programmable Logic and Applications* (FPL'03), 2003.

Received December 2007.

DOI: 10.5755/j01.itc.37.1.11899