RESEARCH ARTICLE OPEN ACCESS

Design and Structuring of a Multiprocessor System based on Transputers

S. S. Nayak*, R. K. Mishra**, M. N. Murty***, B. Padhy****
*Department of Physics, JITM, Centurion University, Paralakhemundi, Odisha, India,
**Department of Electronic Science, Berhampur University, Berhampur, Odisha, India
***Department of Physics, NIST, Berhampur, Odisha, India, E-mail: mnarayanamurty@rediffmail.com
****Research Student, Department of Electronic Science, Berhampur University, Berhampur, Odisha, India,

ABSTRACT
In this paper the Authors present the design of a high performance transputer based fault-tolerant multiprocessor system for critical applications such as aircraft control, nuclear power station control, satellite applications etc. Fault-tolerant building blocks designed with potential for real time processing. The systematic architecture, regularity and recursiveness enables the system to be more fault-tolerant. Real-time applications have to function correctly even in the presence of faults. Link failure of the network is deeply analysed with reliability analysis.

Keywords: Fault-tolerance, Real-time System, Multitransputer System, Parallel Processing, Reliability Analysis.

I. INTRODUCTION

This design provides transparent protection from permanent module failures based on multiple modular redundancy. Though, transporter is a link based processor, it is easy to design a parallel machine that encapsulate acceptance testing, fault masking, reconfiguring the network. The authors presented a new modular based fault-tolerance technique comprising a single module comprising of four IMS T800 transputer for real fault tolerant mechanism interconnecting the four bidirectional links of each module. One line of each module dedicated to link adaptor for I/O to peripheral instruments. The reference becomes crazy due to permanent faults, transient faults, software faults, operation errors etc. Hardware faults due to memory crazy and short circuiting etc.[1],[2] and errors due to information failure[2]. Faults due to severe environmental conditions includes the transient faults[1]. To challenge the above fault scenario, multilink processor[18] is preferred while designing the network.

The design aims at observing

1. Tolerance against intentionally injected faults
2. Tolerance against faulty modules or identical processor in the network of processors.

Hence, the present transputing systems provides dynamic fault recovery applications in MIMD architecture[3]. The multiple link mechanism is adopted for better group communication[4] in the network.

II. TRANSPUTER OVERVIEW

The IMS T800 transputer is a 32 bit CMOS microcomputer with a 64 bit floating point unit and graphics support. It has 4 kbytes on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the Occam model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond. For convenience of description, the IMS T800 operation is split into the basic blocks shown in Fig. 1.

The processor speed of a device can be pin-selected in stages from 17.5 MHz up to the maximum allowed for the part. A device running at 30 MHz achieves an instruction throughput of 30 MIPS peak and 15 MIPS sustained. The extended temperature version of the device complies with MIL-STD-883C.

The IMS T800 provides high performance arithmetic and floating point operations. The 64 bit floating point unit provides single and double length operation to the ANSI-IEEE 754-1985 standard for floating point arithmetic. It is able to perform floating point operations concurrently with the processor, sustaining a rate of 2.2 Mflops at a processor speed of 20 MHz and 3, 3 Mflps at 30 MHz.

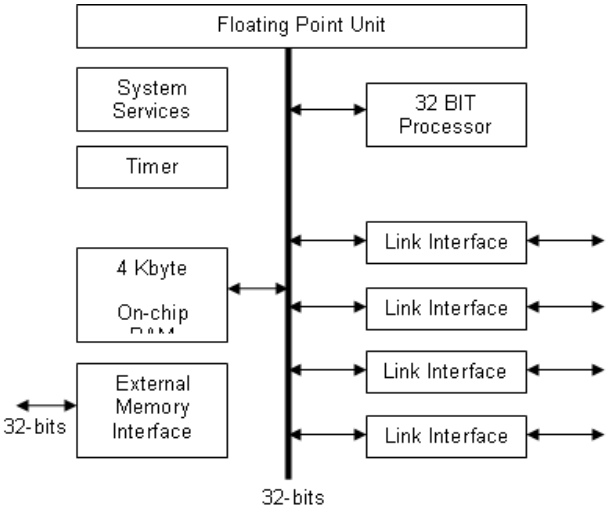


Figure-1 IMS T800

Cyclic Redundancy Checking (CRC) instructions are available for use on arbitrary length serial data streams, to provide error detection where data integrity is critical. Another feature of the IMS T800, useful for pattern recognition, is the facility to count bits set in a word.

The IMS T800 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 100 nanoseconds (40 Mbytes/sec) for a 30 MHz device. A configurable memory controller provides all timing control and DRAM refresh signals for a wide variety of mixed memory systems.

System Services include processor reset and bootstrap control, together with facilities for error analysis. Error signals may be daisy-chained in multi-transputer systems.

The standard INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T800 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 MBits/sec. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec. The transputer is designed to implement the native language Occam but also efficiently supports other languages such as C, Pascal and Fortran of parallel 'C'.

III. SYSTEM DESIGN

The architecture is designed is regular and recursive and the resultant system yields low cost for fault tolerance due to avoidance of roll-back accuracy and small scale of synchronizations.

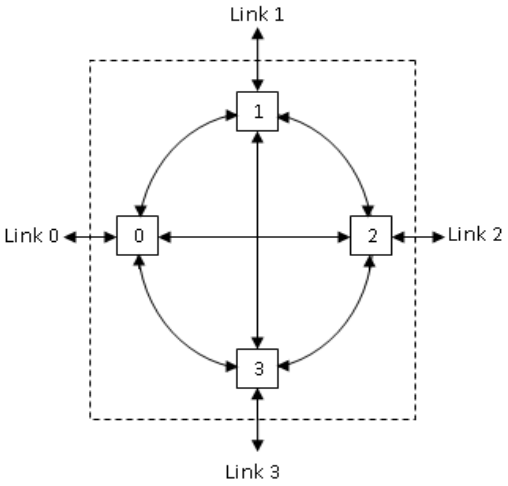


Figure-2 Single Module

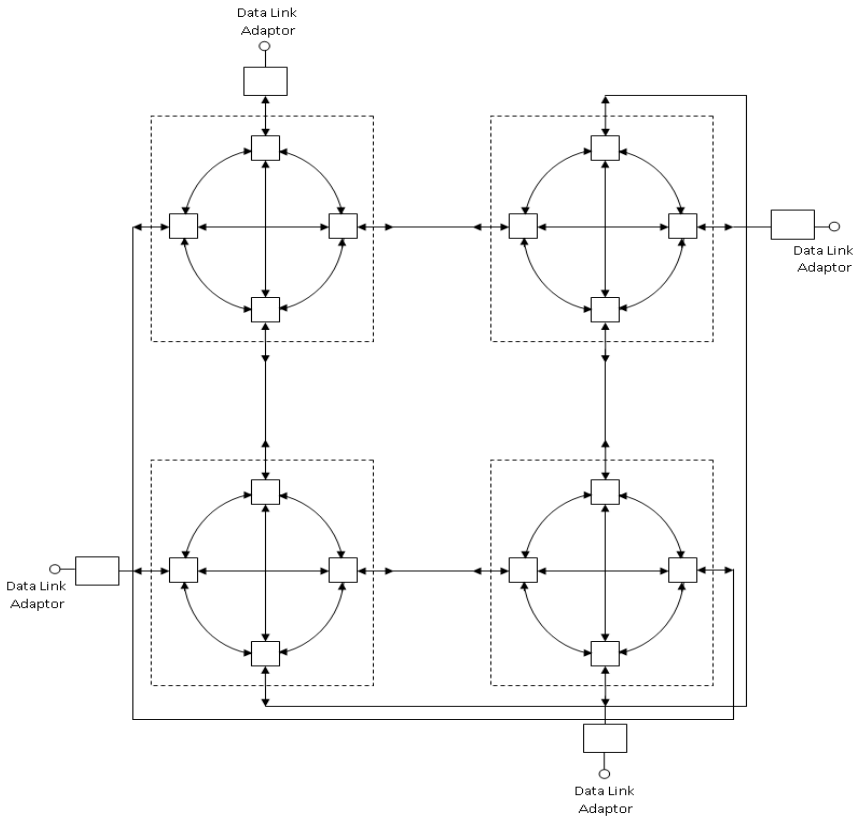


Figure-3: Hardware Architecture [4 Modules]

Each module consisting of four transputer nodes with their own private memory four bidirectional communication links and a copy of operating system. The hardware architecture is shown in figure in the form of grid connection.

a) Operation during multiple link failure

Multiple link failure may result in a network partition. Processor failure is assumed when all the internal links appear to have failed. Consider first the simple one of processors two links filing as in figure. The '0' node scatters the data packet to the available working links, only one node receives it and ACKs are then scattered by the recipient in all the working internal links in response to the packet.

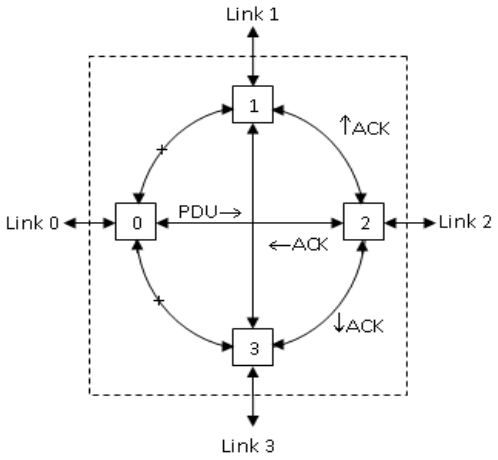


Figure-4 Node '0' scatters PDU. Node '2' receives PDU.

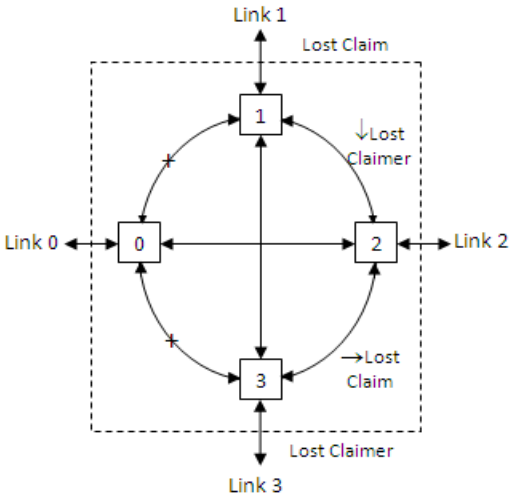


Figure-5 Node '1' and Node '3' observe and claim it. Single Module out of 4 Modules

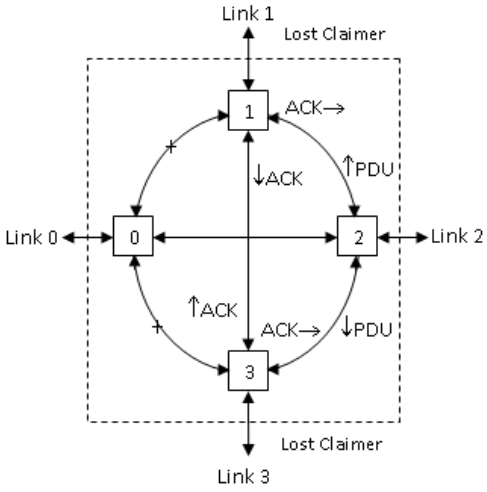


Figure-6 Lost claimers ACK the PDU received.

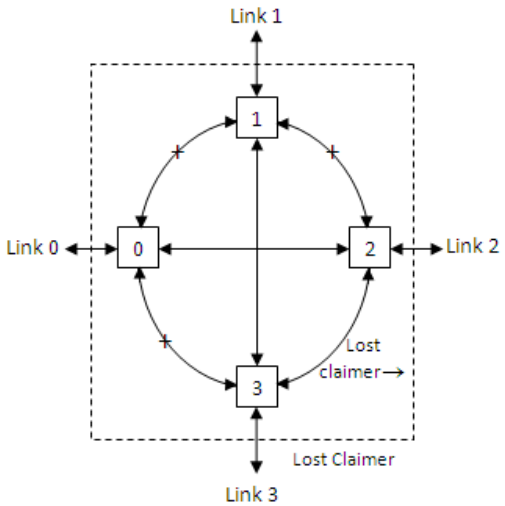


Figure-7 Direct ACKs are returned to the Originator.

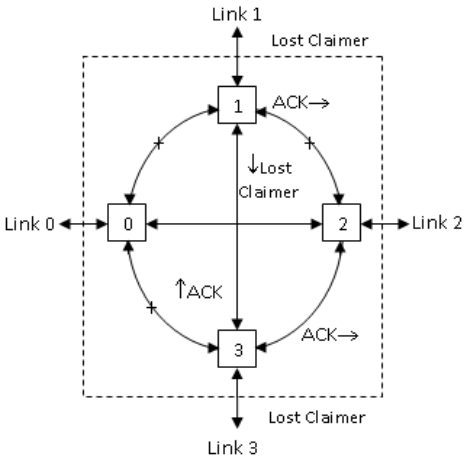
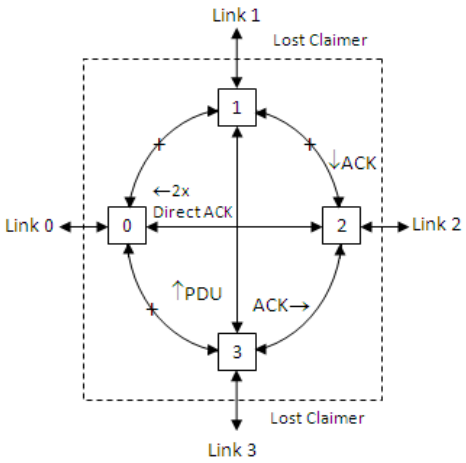
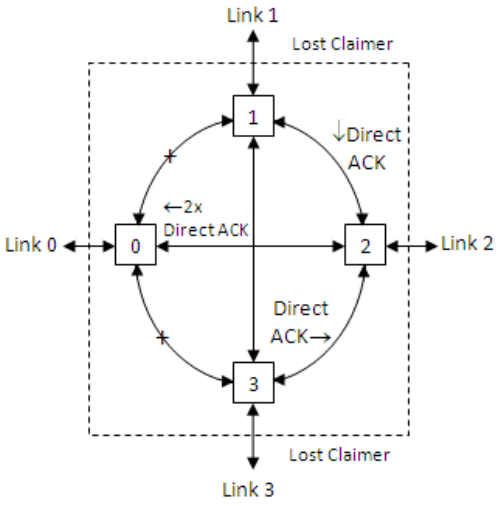


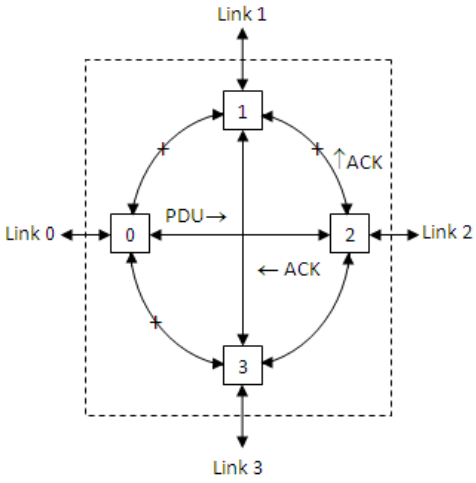
Figure-8 Node '1' scatter PDU and Node '2' receives and ACKs it.



(a)



(b)



(c)

Figure-9(a, b & c): Node 1 receives the PDU and ACKs the reception of the PDU.

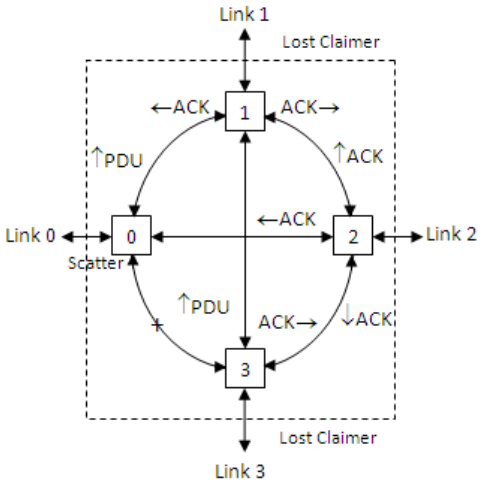
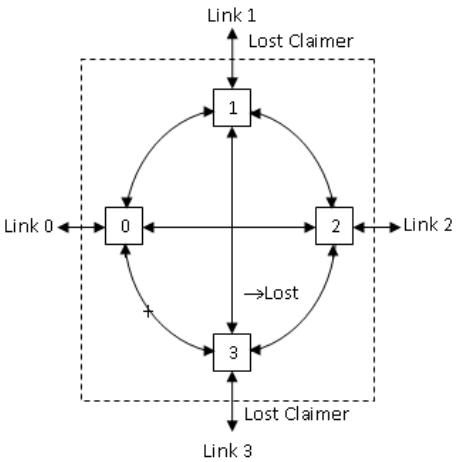
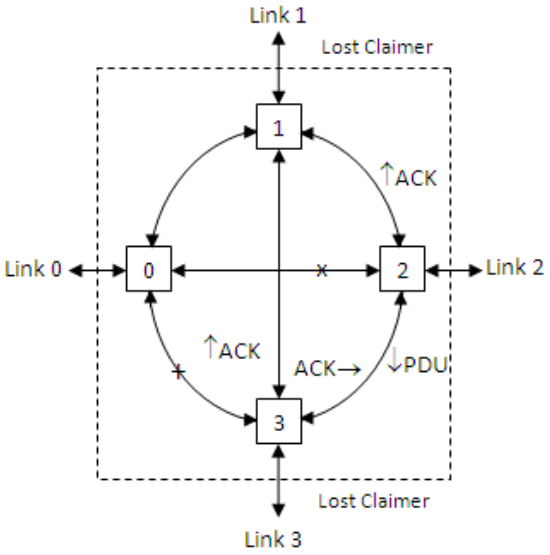


Figure-10: Node '0' scatters PDU and Node '1' and '2' receives and ACKs it.



(a)



(b)

Figure-11(a & b): Node '2' and '2' observe and claim it.

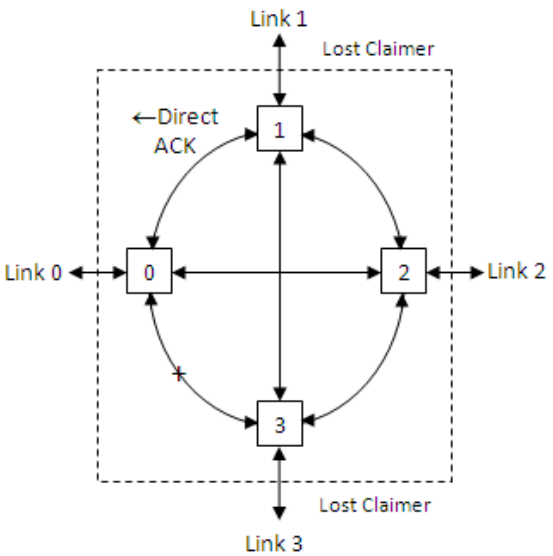


Figure-12: Direct ACK can reach the originator.

The other two links of the recipient are working other two nodes 2 and 1 observe the reception through the ACKs. The recipient sends the data packet to the other two nodes in response to the reception of the lost claim packets. The ACKs are returned to the originator (direct ACK) are also directed to the recipient (Node '0'). When ACKs are scattered form the lost claimer the ACK distributed to the originator is 1st sent to the Node '0' that provided the claimed packet. It is then forwarded by the peer who should have a working link to reach the originator, provided that no further link failure occurred so far.

If the link between the two claimers is working, the observed ACK from either should be received by the other, by now both claimers should have gathered the set of packets i.e. one message packet and two

observed ACKs. The previous recipient should now also have gathered the packets in response to the scatter.

Now, there the case of three links fail, and a single link connection in the network. Same procedure adopted as shown in figure.

The direct ACK is returned to a claimer, but this time the link is not working any longer.

The algorithm is designed so that if any other link is still working than ACK is forwarded to that link otherwise the ACK will have to be returned to the link through which it came.

The signal ACK sender now train the only link left to the processor to reach the originator.

b) Operation in the event of processor failure

The detection of processor failure is very important to stop meaningless waiting or sending attempts. To know the status of processor, each time data packet and control packet is exchanged, the update knowledge about link status of all processor attached to the header of the packet. Each node periodically updates the link status vectors as the exchange of packets takes place with other nodes. A faulty processor will be excluded from the network during the protocol operation until it is repaired.

IV. NETWORK RELIABILITY

The probability that packet exchanges between a pair of nodes can be conducted in the event of link failure in the network is defined as the network reliability R_c [11]. The reliability between two nodes of the network is employed by the very such that a packet sent from one module with two extra ACKs. So, that the receiver to know the packet for proper action, thus the reliability between two modules in the system is increased by the use of redundant ACKs[10].

Let R_1 be the reliability of a transputer link.

R_c be the reliability of the connection between two modules across a link. The probability of a connection failure is

$$F_c = (1-R_1)^2 + 2(1-R_1)^2\ R_1\left[(1-R_1)^2 + R_1(1-R_1^2)\right] \quad (1)$$

Equation (1) can be simplified as

$$F_c = (1-R_1)^3\left(1+2(1-R_1)^2\right)R_1 \quad (2)$$

So, the network reliability R_c is

$$R_c = 1-(1-R_1)^3\left(1+2\left(1-R_1^2\right)R_1\right) \quad (3)$$

There are five possible routes in all possible link failure provided that the system is still connected. The extra ACKs scattered in response to the data packet received, transform the transmission receiver for active actions.

V. SYSTEM RELIABILITY

To keep the system in reliable operation mode the following parameters are used. Let 'C' be the probability that the fault is detected and 'r' be the probability that the fault is repaired after the fault detected in the system. 't' be the processing time and 'λ' be the fault rate during power on. Taking the modular designs, the fault distribution $R_m(t)$ is exponential and identical for all modules i.e. $R_m = e^{-\lambda t}$ since the modules are symmetric independent of each other except when the repair occurs.

Let the time between the consecutive acceptance tests ≤ T. Where T is some constant C and 'r' are constants[5].

To obtain system reliability R_c^T the system failure probability ρ_c^T, has three components No. 1 F_1^1 is due to concurrent faults occur during T. No. 2 F_2^2 is the sequential fault occur over time t where t >> T No. 3 F_3^3 results from fault repairs corresponding to the case when faults occur in two modules. One fault is detected when the other is not and the undetected fault in the repairer.

$$F_1 = (1-R_m)^4 + \left(\frac{4}{3}\right)(1-R_m)^3 R_m \qquad if\ \ t \le T \quad (4)$$

Where F_1 is the probability that all four modules have faultsor three out of four modules have faults during 'T'.

Putting $R_m = e^{-\lambda t}$ in Equation (4).

$$F_1 = \left(1-e^{-\lambda t}\right)^4 + \left(\frac{4}{3}\right)\left(1-e^{-\lambda t}\right)^3 e^{-\lambda t} \qquad t \le T \quad (5)$$

The failure probability F_2 comes from the situation in which the system has suffered two sequential faults. Since all modules are symmetric to each other, so F_2 is the sum of all the sequences[6]. The fault detection probability 'c' and the probability successful repair '4' can change the system failure probability substantially, for four sequences.

$$F_2 = 4\left[(1-c)+c(1-r)\right]^2$$

$$\int_0^t \frac{d}{dt_1}(1-R_m)\int_{t_1}^t \frac{d}{dt_2}(1-R_m)$$

$$\int_{t_2}^t \frac{d}{dt_3}(1-R_m)R_m dt_3 dt_2 dt_1,\ if\ t>T \quad (6)$$

where $(1-c)+c(1-r)$ is the probability that a fault is not detected.

Substituting $R_m = e^{-\lambda t}$ in (6)

$$F_2 = \left[(1-c)+c(1-r)^2\right]\left[1-6e^{-2\lambda t}+8e^{-3\lambda t}-3e^{-4\lambda t}\right]\ if\ t>T$$
$$F_2 = \left[(1-c)+c(1-r)^2\right]\left(1-e^{-\lambda t}\right)^3\left(1+3e^{-\lambda t}\right)\ t>T \quad (7)$$

Let F_3 is the probability that faults occur on only two modules, one of them is detected and other not and the latter appears to have successfully repaired the former using its runtime context[8].

$$F_3 = 2\times\frac{1}{3}\binom{4}{2}(1-c)cr(1-R_m)^2 R_m^2 \qquad t\le T \quad (8)$$

The system failure probability is

$$F_c^1 = \begin{cases} F_1+F_3 & if \quad t\le T \\ F_2 & if \quad t>t \end{cases} \quad (9)$$

Thus the reliability is $R_c^T = 1-F_c^T$.

Equation (7) reflects the contribution to the reliability from the online forward fault repair. It shows that higher values of the probability C or r, lower than system failure probability[7][9].

VI. FAULT INJECTION

Several research papers have been published on fault injection into the live systems [16][17]. Several research groups have developed powerful tools to inject faults by software [12],[13]. The major advantage of simulation based fault injection[14] is the observability of all components which have been module. Our network system depicts a simulation based fault injection approach as in Fig. 13. All systems components have to be modelled in the VHDL hardware description language[15] b using standard synthesis tool and gate level descriptions. Our mechanism uses extended cell library to evaluate fault coverage and fault latency automatically.

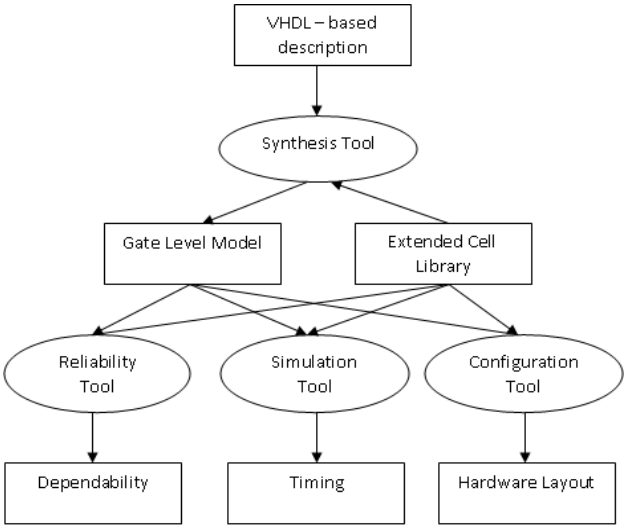


Figure-13 Evaluation of dependability using reliability tool

VII. CONCLUSION

The design exploits parallel processing capability to result in improved reliability. With the mechanism of forward fault repair and redundancy saves the cost of accessing persistent I/O devices. The concurrency control overhead is eliminated due to non-sharing of virtual memory. Definitely this design is highly efficient compared to existing configurations.

REFERENCES

[1] D. Siewiorek, and R. Swarz, "The Theory and Practice of Reliable System Design", 1982 by Digital Press.
[2] R. Koo and S. Toueg, "Check Pointing and Rollback – Recovery for Distributed Systems", IEEE Trans on Software Engineer, Vol. SE-13, No. 1, January 1987.
[3] R. Beton, J. Kingdon and C. Upstil, "Highly Availability Transputing Systems", Proceedings of the World Transputer User Group Conference, April 1991.
[4] F. Christian, B. Dancey and J. Dehn., "Understanding Fault-Tolerant Distributed Systems", Invited Paper, 20th Annual International Symposium on Fault-Tolerant Computing, June 1990.
[5] K. Kim and J. Yoon, "Approaches to Implementation of a Repairable Distributed Recovery Block Scheme", Ann Int. Symp. On Fault-Tolerant Computing, 1988.
[6] O. Selin, "Fault-Tolerant Systems in Commercial Applications", Computer, IEEE, August 1994.
[7] J. Ortiz, "Transputer Fault-Tolerant Processor", Proceedings of the Third Conference of the North American Transputer Users Group, 1990.

[8] R. Oates, J. Kerridge, "Adding Fault Tolerance to a Transputer-based Parallel Database Machine", Proc. Of the World Transputer User Group Conference, 1991.
[9] R. Strom, D. Bacon, S. Yemini, "Volatile Logging in n-Fault-tolerant Distributed Systems", Ann. Int. Symp. On Fault-Tolerant Computing, 1988.
[10] Y. Chen, T. Chen, "Implementing Fault-Tolerance via Modular Redundancy with Comparison", IEEE Trans. on Reliability, Vol. 39, No. 2, June, 1990.
[11] D. Cheriton and W. Zweanepoel, "One-to-many Interprocess Communication in the V-System", Report STAN-CS-84-1011, Department of Computer Science, Standford University, August 1984.
[12] J. Barton, E. Czeck, Z. Segall and D. Siewiorek, "Fault Injection Experiments using FI-AT", IEEE TOC, Vol. 39, No. 4, 1990, pp. 575-582.
[13] J. Carreira, H. Medeira and J. G. Silva, "Software Fault Injection and Monitoring in Processor Function Units", In: Preprints DCCA-5, Dependable Computing for Critical Applications, Urbana Champaign, 1995, pp. 135-149.
[14] E. Jenn, J. Arlat, M. Rimen, J. Ohisson and J. Karisson, "Fault Injection into VHDL Models: The MEFISTO Tool", In: Proc FTCS-24, 1994, pp. 66-75.
[15] P. Ashenden, "The VHDL – Cookbook", Technical Report, Univ. of Adelaide, South Australia, 1990.
[16] R. K. Iyer, "Experimental Evaluation", In: Proc. FTCS-25, 1995, pp. 115-132.
[17] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson and U. Gunneflo, "Using Heavy-ion Radiation to Validate Fault – Handling Mechanisms", IEEE Micro Vol. 14, No. 1, 1994, pp. 8-23.
[18] Inmos, "The Transputer Data Book", Second Edition, 1989.