Synthesizing Fault Tolerant Safety Critical Systems

Seemanta Saha and Muhammad Sheikh Sadi
Khulna University of Engineering & Technology, Khulna-9203, Bangladesh
Email: seemantasaha@gmail.com, sheikhsadi@gmail.com

Abstract- **To keep pace with today's nanotechnology, safety critical embedded systems are becoming less tolerant to errors. Research into techniques to cope with errors in these systems has mostly focused on transformational approach, replication of hardware devices, parallel program design, component based design and/or information redundancy. It would be better to tackle the issue early in the design process that a safety critical system never fails to satisfy its strict dependability requirements. A novel method is outlined in this paper that proposes an efficient approach to synthesize safety critical systems. The proposed method outperforms dominant existing work by introducing the technique of run time detection and completion of proper execution of the system in the presence of faults.**

Index Terms- **Detector, Fault Tolerance, Program, Safety Critical System, Fail-Safe.**

I. INTRODUCTION

Design of safety critical embedded systems is becoming more and more complex as the performance scale of these systems is rapidly increasing. Numbers of transistors in a single chip of these embedded systems are increasing day by day to provide a more powerful controlling system with low cost. Clock frequencies are reaching multiple GHz range because of constant downscaling of CMOS technologies. As a result, these systems are more prone to various types of errors.

But, safety critical systems need to maintain strict dependability requirements. They need to avoid taking steps that violates the system's safety specifications as well as, perform all necessary steps for the completion of desired function of that system. Nuclear power plant or nuclear power monitoring systems is such a safety critical system where a single bit flips in the value of system parameters can cause severe destruction of the environment or death to human life. A heart pace maker which provides adequate heart rate for a patient needs to maintain stringent availability and dependability. If a heart pace maker fails to maintain adequate heart rate, the patient will lose his/her life. When a spacecraft is in flight, it is not allowed to afford a breakdown. Dependability in all these types of safety critical embedded systems is a major concern for the human being and environment as errors in these systems will be catastrophic. As a result, these systems need to perform tolerating faults or any other undesired external perturbation in the presence of any type of soft errors.

Various approaches have been proposed over the years to provide fault tolerance for safety critical systems. Some approaches are based on synthesizing and some are based on transformations. All these approaches have followed the idea of program modification or program rewriting. Arshad et al. [13] proposed fast detector which has been used in distributed embedded system to provide fail safe fault tolerance. They added the concept of the detector with a fault intolerant program to transform it to a fault tolerant program. For all possible errors, they checked that any particular program is reachable to bad transitions are not. If bad transitions are reachable from a program state because of fault transitions then the earliest inconsistent transition from that program state has been removed. As a result, whenever a fault transition occurs, the program halts to the state providing fail safe fault tolerance of the system.

But, the proposed method of this paper included another novel point of view by keeping the earliest inconsistent transition in the program to continue the complete execution of a system. The program will transit from one state to another state to perform its desired functions and in every state it will check that the safety specification of the system is maintained or not. To check the maintenance of safety specification in a system, reachability to program's bad transitions will be checked. This checking will be done by forward traversing of program states during program execution. As a result, the proposed method enforces completion of program execution tolerating the fault in the system due to a soft error. Though the method has been proposed for safety critical systems, in the future it can be also applied to any system to check its proper flow of execution.

The paper is structured as follows: Section II describes related works. Section III presents preliminaries of the proposed method. Section IV presents the proposed methodology of fault tolerance in safety critical systems. Section IV presents an experimental analysis. Section V summarizes the contributions of the paper and discusses future possibilities.

II. RELATED WORK

A good number of works have been performed related to fault tolerance approaches for safety critical systems. Among these approaches, design of effective detectors, error propagation analysis, parallel program design and component based design have great significance. Most of these approaches are software based and proving to be efficient and effective.

Component based design of the multi tolerant system has been proposed by Anish Arora and Sandeep S. Kulkarni [1], [2] where they developed a basis for improved design of dependable system. They illustrated their method by designing fully distributed multi tolerant program for a token ring [1]. They also introduced the idea of detector and corrector [3] to provide fault tolerance for a multi tolerant system. The method of detector and corrector [3] had been used to automate the addition of fault tolerance in the system. The complexity of their method had been also analyzed [4]. Sandeep S. Kulkarni et al. also provided another approach to automate the addition of efficient fault tolerance [5]. They developed the concept of the detector as a system component which converts the system tolerant to a fault. They also proposed approach of fault tolerance [6] by synthesizing concurrent programs.

Zhiming Liu and Mathai Joseph proposed transformational approach of fault tolerance [7], [8]. They designed how a program, constructed considering a fault-free system can be transformed into a fault- tolerant program for a system which is susceptible to failures. They illustrated their approach of fault tolerance by considering the problem of designing a protocol for reliable communication channel. Doron Peled and Mathai Joseph also developed a compositional framework for fault-tolerance by specification transformation [9]. They adopted a recovery algorithm which has been used to convert a basic program with a fault tolerant version. Felix C. Gartner [10] adopted another transformational approach to the specification and verification of fault- tolerant systems.

But, all these approaches were considered common system fault tolerance. But, in case of safety critical systems, safety specification of the system is the major concern and consistency of system performance and proper execution of the system need to be satisfied. Arshad Jhumka et al. [11], [12], [13], [14] adopted the theory of detector developed by Anish Arora and Sandeep S. Kulkarni. They proposed a perfect and fast detector as a system component which is used to detect faults in a system. They converted a fault intolerant program to a fault tolerant program adding detector to the fault intolerant program. They implemented their method to distributed embedded system to provide fail safe fault tolerance, which provided an effective outcome in the field of safety critical system fault tolerance. Arshad Jhumka along with Matthew Leeke [15] also analyzed complexities and issues on the design of effective fail safe fault tolerance. They also proposed the concept of critical variables [16] and analyzed the importance of critical variables in dependable software.

III. PRELIMINARIES

A. State

A state of a program P is a function that holds a particular value of variables in P. The state space S_P of P is the set of all possible states of P.

B. State Relation

A program can be represented using some states, where every state will hold particular value of system variables and a transition function will be used to transit from one state to another state. This representation of a program using state and transition function is called state relation.

C. Safety Specification

Safety specification of a system can be defined as a term of ‘zero occurrence of bad transitions in a program’. If any computation of a program violates safety specification, program transition belongs to that computation must be avoided. A fault intolerant program can violate safety specification as any faulty program transition can reach to a bad transition.

D. Bad Transition

A transition which violates safety specification of a system will be considered as bad transition. Every transition of a program occurs because of a computation of program. If any transition of a program violates safety specification then the computation holding that bad transition also violates safety specification.

Suppose, a program P consists of states a, b, c, d, e, f. Safety specification for that program is S. C is a computation of program P which consists of state transition (b, c), (c, d), (d, e). If (d, e) transition violates safety specification then it is called a bad transition and as computation C holds (d, e), C violates safety specification.

E. Reachability to Bad Transition

If a program is currently on a state x and in future it can be on state y and there can be a transition from state y to state z, where y to z is a bad transition then there is reachability to bad transition (y, z) from state x.

F. *Inconsistent Transition*

If a faulty transition occurs and reachability to a bad transition is found, transitions that occur between reaching bad transition and the occurrence of fault transition, are called inconsistent transitions. Among all the inconsistent transitions, transition that occurs immediately after the occurrence of fault transition is called earliest inconsistent transition.

IV. THE PROPOSED METHODOLOGY TO SYNTHESIZE FAULT TOLERANT SAFETY CRITICAL SYSTEMS

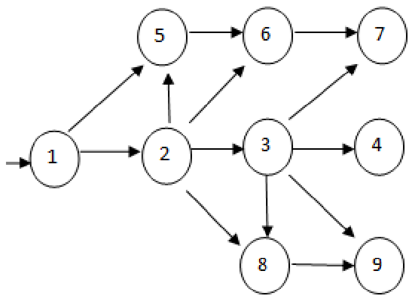
The proposed methodology will be described by following steps which will show how the system will be synthesized and how fault tolerance in the system will be achieved including advancement of existing methodologies.

A. Converting a Program to State Relation

A program consists of a finite set of variables. The values of these program variables depend on various factors such as user input, computations of the program and different conditions of the program. Values of different variables are also responsible for various program conditions. A program executes its desired function step by step depending on different computations and values of program variables. So, a program can be easily designed as a form of some state relations. Every state is assigned with values of some program variables. From the theory of finite automaton, state transition can be denoted by a transition function (state, input). Program transition occurs from one state to another state depending on the input. Values of system variables will be used as input for state transition of program.

```
if (input of x maintains [5<=x && x<=10])
{
program is on state 2;
if (input of y maintains [y<=10 && y<=15])
{
program is on state 3;
Z= x+ y and program is on state 4,
}
else
program is on state 8 and then on state 9;
}
else
program is on state 5 and then on 6 and then on 7;
```

a



b

Fig. 1 (a) A Simple Program, (b) State Relation of the Simple Program

When the values of the program variables are changed and different conditions are arisen depending on the value of variables, the program transits from one state to another state. Any program P can be converted to its state relation by using several states such as P1, P2, P3,…,Pn denotes states. This idea is shown in Fig. 1 (a) a simple program (b) state relation of the simple program, where some program states have been designed based on the value of system variables x, y and output variable z.

B. *Automated Design of Fault Tolerance*

The proposed method provides an automated design of fault tolerance for safety critical systems. The approach can be stated automated as a checking reachability to bad transitions, fault detection, tolerating fault, returning to a safe state, continuation of program execution is done dynamically as the system it advances from one state to another state to perform its desired task.

C. *Fast Detection*

The view of fast detection of a safety critical system has been adopted in the proposed method of detecting error preserving minimal detection latency. Forward checking of state transitions has been used to detect fault transitions in the system. When a program is executed and every time when the program reaches to a new state, reachability to bad transition checks from that state. If there is no error occurrence then there will be no reachability to any bad transition as the program will transit from one state to another state according to its desired execution way. But, if reachability to any bad transition is found then it is considered that a fault transition occurred. The state from which reach ability is found is the immediate program state because of fault transition. As, the fault is immediately detected at zero step after error occurrence, the detection latency is considered as the minimum.

In addition with fast detection, proposed methodology stores the previous state of current state. So, when a fault occurrence is detected at current program state, the program returns back to its previous safe state and continues program execution in the alternate safe way.

D. *Run Time Detection*

The proposed method of run time detection can be described in some basic steps:

- A system is converted to its corresponding state relation considering all possible errors and the values of system variables.
- System execution proceeds through the program transitions from one state to another state. A state takes an input and transits to another state. Input for state transition is considered as the change of values of system variables. When the program reaches to a new state, previous state of the new program state is stored.
- When the program reaches to a new state, reachability to the all bad transitions of the system are checked. The forward checking technique is used to find reachability to bad transitions. If reachability to any of the bad transitions is found, the system detects that an error has occurred.
- When an error is detected in the system, system returns to its previously saved safe state. Then, the program again transits to the alternate state and again reachability to bad transitions is checked to ensure that no safety specification violation will occur in the system.
- The system tolerates fault in the same manner as described in the earlier two steps and eventually proper program execution is achieved.

E. *A Simple Program to Illustrate Fast and Run Time Detection*

Fig. 2 shows a simple system which illustrates the idea of fast and run time detection. This simple system does addition of two variables x, y and output the result to another variable z. Now, this simple system is considered as a safety critical system and assumed that value of z will be between (15, 25), is the safety specification for this simple system. So, to maintain safety specification of this system value of x should be between (5, 10) and value of y should be between (10, 15).

In fig. 2, the system has been converted to state relations following the conditions as in Fig. 1(a). When the system is error free that is there is no occurrence of fault transitions, the system will transit in the way of {(1, 2), (2, 3), (3, 4)} state transitions. Depending on the value range of system variables and different program conditions, 9 possible states have been designed. Possible fault transitions are {(1, 5), (2, 5), (2, 6), (2, 8)}.

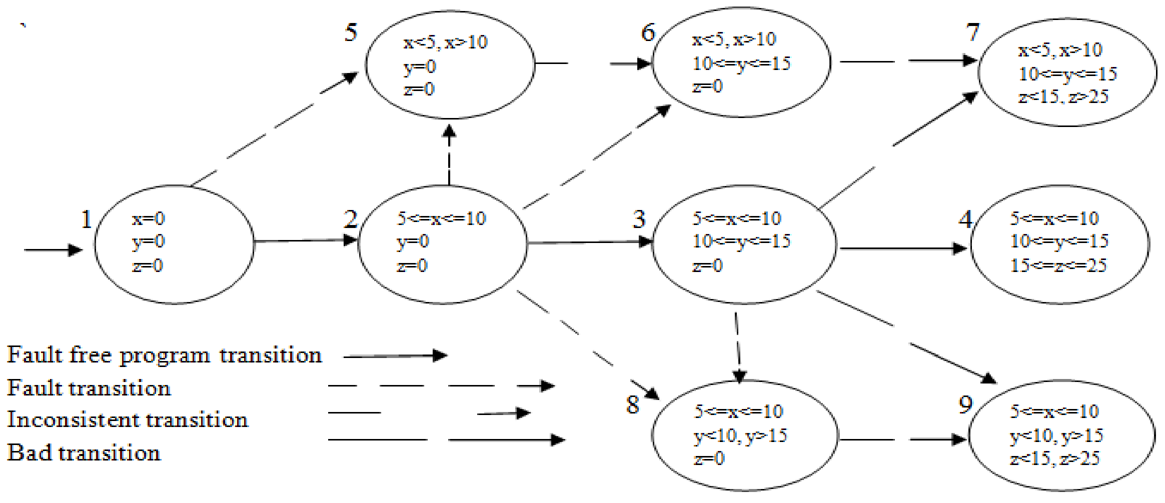


Fig. 2 A Simple System to Illustrate Fast and Run Time Detection

Depending on the safety specification of the system possible bad transitions are {(6, 7), (3, 7), (3, 9), (8, 9)}. There are 4 possible fault transitions in the system. It has been previously said that the proposed method need to consider only about the fault occurrences during the time of program execution. Now, initially the program is on state 1. As the value of program variable x changes, a program transition occurs and program reaches to state 5. But, if there is no error, program will transit to state 2. As program transition has occurred, reachability to the specified bad transitions will be checked. From state 5, reachability to bad transition (6, 7) is found. As, reachability is found, system detects that, a fault transition has occurred during program transition to state 5. This fault occurrence is immediately detected at state 5. This is called fast detection as the error has been detected in zero steps with minimal detection latency. As, the system detects fault during the time of execution, a run time detection is also maintained.

System always stores the immediate earlier state of the current state. So, when fault has been detected in state 5 it immediately returns back to its immediate earlier safe state 1. Without any error, program now transits to state 2 and again checks reachability to any bad transition. No reachability to any bad transition is found. As a result, program decides that no error has occurred and program transition occurs in the proper way. As, the system can detect fault occurrences automatically and continue its operation in a safe manner, automated design of fault tolerance is also satisfied. In this way, any system can be represented to a specific sate relation so that it can always maintain proper program execution and never violates safety specification. So, it can be stated that proposed method provides automated design of fault tolerance with fast and run time detection and correction with cent percent detection coverage of errors. The system is designed in such an efficient way that it will never proceed on such a way that it may eventually violates safety specification for safety critical systems.

F. An Algorithm to Check Reachability to Bad Transitions in Run Time

An algorithm is also proposed which works on the basis of forward checking and provides an efficient and automated approach of fault tolerance with minimal detection latency and detection at program execution. This algorithm has been designed in such a way that it does not need to remove all earliest inconsistent transitions for a system. Faults during program execution are tolerated to continue program execution.

```
Begin reachability_to_bad_transition
Assign 1 to no_transitions;
Assign old_current_state to transition [no_transitions];
Increment no_transitions;
Until state_transitions are complete repeat
Assign transition_function (current_state, input) to current state;
Assign current_state to transition [no_transitions];
Increment no_transitions;
```

```
For each no_transitions do
For each number_of_bad_state do
If (transition [no_transitions] = = bad_state) Return 1;
End of inner loop;
End of outer loop;
End reachability_to_bad_transition;
```

G. *Proof of Fault Detection by Checking Reachability*

When a system is converted to state relations all possible states are considered assuming different case of value range of system variables. Suppose a system can be represented using possible 11 states. Assuming the set of state is X = {A, B, C, D, E, F, G, H, I, J, K} state relation in absence of fault is like in Fig. 3.

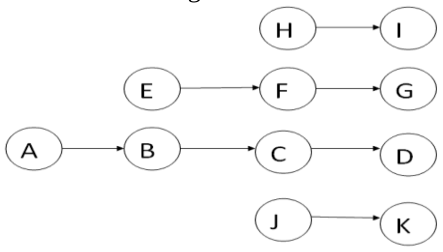


Fig. 3 State Relation in Absence of Fault

But, because of fault or undesired external perturbation some extra transition may occur like (A, E), (B, E), (E, H), (F, H), (F, I), (B, J), (C, G), (C, J), (C, K) as shown in Fig. 4.

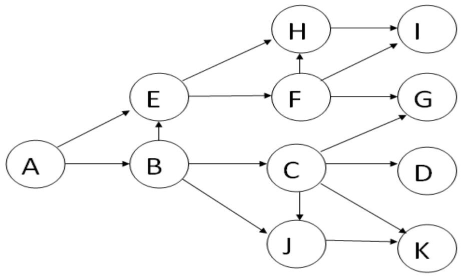


Fig. 4 State relation in Presence of All Possible faults

According to the safety specification of the system transitions (H, I), (F, G), (J, K) are bad transitions. When there is no fault in the system, program transitions are (A, B), (B, C), (C, D). Using the laws of proposition it can be proved that fast and run time fault detection can be done by checking reachability to bad transitions.

Let, R is the relation of reachability from one state to another. Now, applying the 'Transitive' law of propositional calculus on set X, we can write that if 'F is reachable from E' and 'G is reachable from F' then 'G is reachable from E'. That is, bad transition (F, G) is reachable from E. Mathematically the law of transitive relation can be written as:

$$(E\ ^{R}\ F) \wedge (F\ ^{R}\ G) \dashrightarrow (E^{R}G)$$

When there is no fault transition in the system, bad transition is not reachable from any of the system state. In absence of fault, the transition (E, F) is impossible. But, if fault occurs, (E, F) transition occurs. Now, it will be proved that fault transition occurrence can be detected by checking reachability to bad transition. In absence of fault, transition occurs from state A to state B, but when fault occurs, transition may occur from A to E.

Assume, (A, E) transition has occurred during program execution.

Let,

Hypothesis, H = G is reachable from E e.g. $(E\ ^{R}\ G)$

Conclusion, C = (A, E) is a fault transition.

The proof should be

HC e.g. if H holds, C holds.

By method of contra positive it will be enough to prove not C not H e.g. if C does not hold, H does not hold

'Not C' means '(A, E) is not a fault transition'. So, (A, E) is a valid program transition and program reaches to state E. As it has been said earlier that, when there is no fault transition in the system, bad transitions are unreachable. So, G should not be reachable from E as (A, E) is a valid transition. So, it does not hold H e.g. 'Not H' is true.

So, it can be written that

Not C not H e.g. HC

So, it is proved that fault transition of system can detected by checking reachability to bad transitions.

H. *Proper Flow of System Execution*

The proposed method also enforces the applicable systems to maintain proper way of execution. Execution of any system flows step by step from one state to another. In every state there are changes of system variables, input parameters and output results. The proposed method checks the flow of system execution by checking proper state transition and reachability to final or desired state. The method also directs the system execution flow to avoid wrong execution path and detects whether the system is according to its desired way of execution or not.

V. EXPERIMENTAL ANALYSIS

A. *Experimental Setup*

The proposed method has been implemented by using a software simulator. The simulator is classified into three significant steps to implement the proposed method. These tasks are: 'Developing state relations', 'fault Injection', 'simulation of program execution in presence of errors'. The simulation procedure mainly involves three phases. These three phases are 'detection', 'correction and return to safe state', 'proper program execution'. In the detection phase, error is detected specifying fault transition and inconsistent transitions with a pictorial view. In the second phase, fault is corrected by returning to a safe state of the program. Here, safe state is the previous state from where the fault transition occured. Simulation completes in the third phase when the proper execution of the program is accomplished and the system completes its desired task.

The simulator was developed using "GDI+" method and C#. To develop the state relation for a simple program, idea of deterministic finite automaton has been used. The finite state automaton can be easily designed for any system or any program using the mouse event handler in the simulator. The simulator was developed in a generalized way to develop state relation for simple programs with pre- specified safety specifications. To implement the proposed method, error has been injected pseudo randomly.

State relation for a program can be easily drawn and visualize in the simulator. First of all, states are put on the simulator window. Then path from one state to another state is drawn including input symbol. Initial state, final state and bad transitions for the system are also defined using different tools of the simulator. Then input sequences for error free system is provided to simulate the system in the absence of fault.

In the simulator, error is injected pseudo randomly. Number of errors, sequence of input, and the number of input variable are chosen pseudo randomly. Sequence of input is used for program transition which depends on the value of the system variable in every state. Number of input variable denotes the variable in which error has been injected. Error can also be injected manually to see how the proposed method works for a simple program.

Initially, the program will be on start state. Because of error in the input sequence, the program may take a fault transition and transit into a bad state instead of the desired state. But, reachability to bad transitions will be found and error will be detected and the program will return to previous safe state. All error in the system will be detected and corrected and execution of the program will be completed.

B. Results

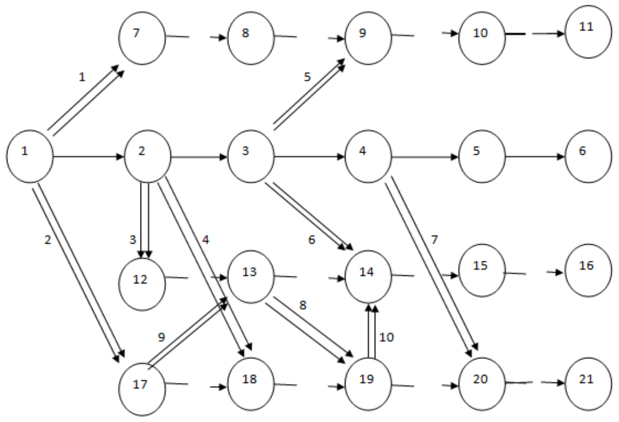


Fig. 5 State Relation of a Simple Program

Fault transitions are {(1, 7), (1, 17), (2, 12), (2, 18), (3, 9), (3, 14), (4, 20), (13, 19), (17, 13), (19, 14)} and have been denoted using 'double arrow'. In this example, bad transitions are {(10, 11), (20, 21)} and have been denoted using 'broken arrow with small gap'. Inconsistent transitions are {(7, 8), (8, 9), (9, 10), (12, 13), (13, 14), (14, 15), (15, 16), (17, 18), (18, 19), (19, 20)} and have been denoted using 'broken arrow with large gap'.

Depending on the number of fault occurrences in a system, dominant existing work of Arshad Jhumka et al. [13] and proposed method shows some differences as shown in Fig. 6 and Fig. 7. When the number of faults in a system is few, the proposed method need more number of steps to complete the system execution maintaining safety specifications with compared to Arshad Jhumka et al. [13] detecting fault occurrences. But, when the number of fault occurrences in a system increases, proposed method needs less number of steps than the number of steps needed in Arshad Jhumka et al. [13].

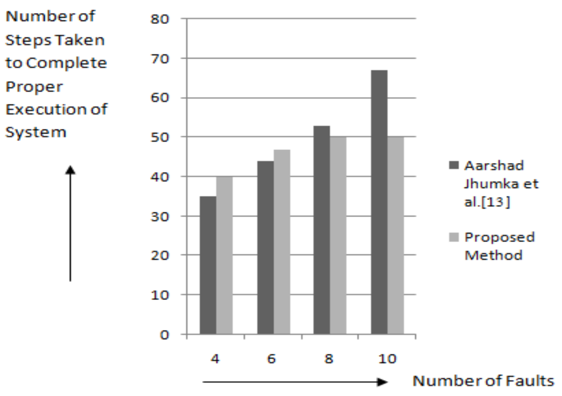


Fig. 6 The comparison between the Proposed Method and Arshad Jhumka et al. [13] with respect to Number of Steps Taken.

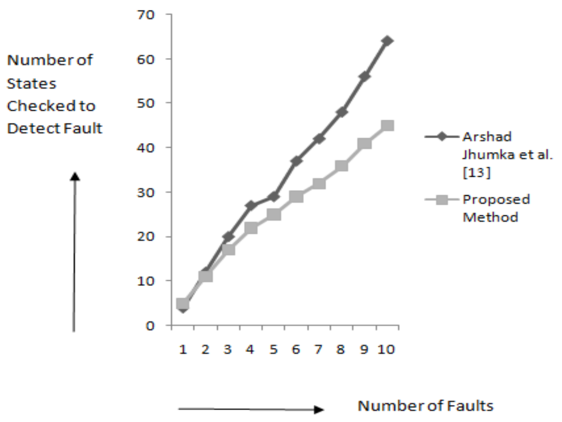


Fig. 7 The comparison between the Proposed Method and Arshad Jhumka et al. [13] with respect to Number of Steps Taken.

When number of fault is 4 and 6 proposed method needs more steps than Arshad Jhumka et al. [13]. But, when the number of faults in the system is 8 and 10 proposed method needs less number of steps than Arshad Jhumka et al. [13] as shown in Fig. 6.The proposed method also needs to check less number of states than Arshad Jhumka et al. [13] with respect to the increase in number of faults in a system. For few number of faults both method need to check approximately same number of states to detect fault. But, when number of fault occurrences increases proposed method needs fewer number of states to check than Arshad Jhumka et al. [13] as shown in Fig. 7.

So, from Fig. 6 and Fig. 7, it can be concluded that when a safety critical system becomes complex with lots of possible fault occurrences proposed method provides a better way of fault detection and program execution. Moreover, the proposed method provides detection at run time considering only the fault occurrences during program execution.

C. Advancements to The Tolerance level of Safety Critical Systems

Table 1 shows some significant differences between the proposed method and the dominant existing work of Arshad Jhumka et al. [13] in which they proposed the idea of fail-safe fault tolerance. The concept of fail-safe fault tolerance was that when a fault occurs in the system it will halt at a state from which it does not proceed to any other states.

TABLE I
REVIEW OF TECHNIQUES OF THE PROPOSED METHOD AND ARSHAD JHUMKA ET AL. [13]

Approaches	Arshad Jhumka et al. [13]	Proposed Method
Adopted methodology	A detector is added as a system component with the fault intolerant program to convert it to a fault tolerant program	Fault is detected using a fast, run time detector and proper execution of system is maintained
Fault handling	All possible faults for a system need to be handled.	Faults during the time of execution need to be handled
Fault Detection method	Reachability to bad transition is checked by backtracking.	Reachability to bad transition is checked by Forward Checking
Fault Removal	Earliest inconsistent transition is removed	Returning to saved safe state
Total number of steps taken to complete system execution	If possible faults are less, less steps are taken than proposed method	If possible faults are more, less steps are taken than existing work
Total number of states need to check	If possible faults are less, less states are checked than proposed method	If possible faults are more, less states are checked than existing work
Contribution	Fail safe fault tolerance	Fault tolerance and proper execution time is needed
Fault Detection	Yes	Yes
Fault Correction	No	Yes
Drawbacks	Program execution is not completed	For less number of faults, much execution

VI. CONCLUSIONS

In this paper, an efficient approach to synthesize a dependable safety critical system has been proposed. The proposed method can be concluded as an efficient approach because of its fast and run-time detection properties and execution of the system in a safe manner avoiding the violation of safety specification.

Though the proposed method need to check reachability into all possible ways and need to store the previous state of the current program state, proper and complete execution of the system is maintained to run time detection. So, the method of this paper outperforms the existing dominant work by proposing corrective measures and continuity of program execution.

This method of fault tolerance using the idea of state relations will open a new efficient door in the field of fault tolerant system. In this paper safety specification has been pre specified. But, if the safety specification for a system can be automatically specified from the functionality, system parameters and other conditions of system environment, then the approach of fault tolerance for safety critical systems will be outstanding and enormous.

REFERENCES

[1] Anish Arora and Sandeep S. Kulkarni, "Component based design of multi tolerant systems," *IEEE Transactions on Software Engineering*. 24(1), January 1998, pp. 63-78.
[2] Sandeep S. Kulkarni, "Component Based Design of Fault-Tolerance," PhD thesis, Department of Computer and Information Science, The Ohio State University, 1999.
[3] Anish Arora and Sandeep S. Kulkarni, "Detectors and correctors: A theory of fault tolerance components," *In Proceedings of the 11th IEEE International Conference on Distributed Computing Systems (ICDCS98).* May 1998.
[4] S. Kulkarni and A. Ebnenasir, "Complexity of Adding Fail-Safe Fault Tolerance," *In Proceedings International Conference on Distributed Computing Systems*. 2002.
[5] Sandeep S. Kulkarni and Anish Arora, "Automating the addition of fault-tolerance," *In Mathai Joseph, editor, Formal Techniques in Real-Time and Fault-Tolerant Systems, 6th International Symposium (FTRTFT 1200) Proceedings*. number 1326 in Lecture Notes in Computer Science, Pune, India, September 2000, pp. 82-93.
[6] Anish Arora, Paul C. Attie, and E. Allen Emerson, "Synthesis of fault-tolerant concurrent programs," *In Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC'98).* 1998, pp. 103-113.
[7] Zhiming Liu, "Fault-tolerant programming by transformations," PhD thesis, University of Warwick, Department of Computer Science, 1991.
[8] Zhiming Liu and Mathai Joseph, "Transformation of programs for fault-tolerance," *Formal Aspects of Computing.* 4(5), 1992, pp. 442-469.
[9] Doron Peled and Mathai Joseph. "A compositional framework for fault-tolerance by specification transformation," *Theoretical Computer Science*, 138,1994, pp. 99-125.

[10] Felix C. Gartner, “Transformational approaches to the specification and verification of fault-tolerant systems: Formal background and classification,” *Journal of Universal Computer Science (J.UCS),* 5(10). October 1999, pp. 668-692.
[11] Arshad Jhumka, Martin Hiller, Vilgot Claesson, and Neeraj Suri, “On systematic design of consistent executable assertions for distributed embedded software,” *In Proceedings of the ACM Joint Conference on Languages, Compilers and Tools for Embedded Systems/Software and Compilers for Embedded Systems (LCTES/SCOPES).* 2002, pp. 74-83.
[12] Arshad Jhumka, “Automated design of efficient fail- safe fault tolerance,” Department of Computer Science, Technische Universitat Darmstadt, 64283, Darmstadt,2003.
[13] Arshad Jhumka, Felix Freiling, Christof Fetzer, and Neeraj Suri. “An Approach to Synthesize Safe Systems”, *Int. Journal. on Security and Network,* 2006, pp.62-74.
[14] Felix C. Gartner and Arshad Jhumka. “Automating the addition of fail-safe fault tolerance: Beyond fusion- closed specifications”, *In Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT).* Grenoble, France, September 2004.
[15] A. Jhumka, M. Leeke, “Issues on the Design of Efficient Fail-safe Fault Tolerance”, In Proceedings IEEE 20th International Symposium on Software Reliability Engineering (ISSRE09), Bengaluru-Mysuru, India, November 18th, 2009, pp. 155–164.
[16] M. Leeke, A. Jhumka, “Towards Understanding the Importance of Variables in Dependable Software”, In Proceedings of the 8th European Dependable Computing Conference (EDCC10), Valencia, Spain, April 28th, 2010, pp. 85-94.

Seemanta Saha received B.Sc. Eng. in Computer Science and Engineering from Khulna University of Engineering and Technology, Bangladesh in 2012. He has published two papers in the field of soft error and safety critical embedded systems. His current research interests are Fault tolerance in Embedded Systems, System Analysis and Design.

Muhammad Sheikh Sadi received B.Sc. Eng. in Electrical and Electronic Engineering from Khulna University of Engineering and Technology, Bangladesh in 2000, M.Sc. Eng. In Computer Science and Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in 2004, and completed PhD (Area: Dependable Embedded Systems) from Curtin University of Technology, Australia in 2010. He is currently Professor at the Department of Computer Science and Engineering, Khulna University of Engineering and Technology, Bangladesh. He teaches and supervises undergraduate and postgraduate theses in topics related to Embedded Systems, Digital System Design, and Soft Errors Tolerance etc. He has published over 30 papers and book chapters in his area of expertise. Muhammad Sheikh Sadi is a member of the IEEE since 2004.