A Real-Time Performance Monitoring Tool for Dual Redundant and Resource Augmented Framework of Cruise Control System

Annam Swetha
Amrita Vishwa Vidyapeetham (University)
Coimbatore, India

Radhamani Pillay V
Amrita Vishwa Vidyapeetham (University)
Coimbatore, India

Santanu Dasgupta
Mohandas College of Engineering & Technology
India

Senthil Kumar Chandran
AERB-Safety Research Institute
Kalpakkam, India

ABSTRACT
The computing resources used in safety-critical systems have stringent timing requirements due to mission critical nature of their tasks. A fault in these systems could lead to mission failure and catastrophic consequences. To avoid this various redundancy schemes are built in to mission critical applications to ensure the overall success of the system. The usual industrial practice is to employ fault tolerance using hardware redundancy where costs are highly exorbitant depending on the mission. In this paper, a prototype tool has been designed and developed for testing and evaluation of a framework for adaptive fault tolerance on an existing dual hardware redundancy with resource augmentation. This proposed model gives enhanced resource management and improved system performance under normal runtime and provides minimal safe functionality under permanent fault condition. It has been implemented with a practical case study of Cruise Control System using NXP LPC2148 processors. The results demonstrate the better performance and process speedup (execution time of process) vis-à-vis over a traditional dual redundant processor system and the high performance that can accrue by applying this model to an m-processor redundancy model.

General Terms
Redundancy, Safety-critical systems, Real-time scheduling, Hardware implementation

Keywords
Fault tolerance, Resource management, Cruise control system, Process speedup, Prototype tool

1. INTRODUCTION
Safety-critical systems are those systems whose failure could result in loss of life, significant property damage, or damage to the environment. There are many well known examples in the application areas such as medical devices, aircraft flight control, nuclear systems and modern automotives. Current automotive systems employ up to 100 electronic control units (ECUs) exchanging more than 2500 signals over different communication systems [1]. These ECUs control and monitor many subsystems of a vehicle such as cruise control, chassis control, vehicle stability and engine control. The development of control software and scheduler for such systems has become one of the greatest challenges in the automotive domain due to the increasing complexity of automotive systems [1]. The computing systems used in such applications have to perform several critical tasks and any failure in their execution could lead to complete failure of the system with catastrophic effects. Due to real-time nature of such systems, they also need to satisfy the timing correctness in addition to the correctness requirements of the generated output.

A typical fault tolerant system need to work correctly even under specified fault condition and should provide high reliability. To increase reliability there are two commonly used approaches [7], one is fault avoidance strategy by using highly reliable components, conservative design practices and an extensive testing to eliminate flaws. The second method is fault tolerance which depends on redundancy either hardware or software or both to overcome the effects of failure. In general based on the criticality of the system the level of redundancy is increased which directly increases the cost of the system. The major disadvantages of such system are high power consumption, increased weight and size and incomplete used of resources under fault free condition [13]. An innovative fault tolerant framework called Enhanced Resource Management Scheme (ERMS) has been designed and analyzed in [2]. This scheme uses task level criticality, replicates the critical tasks on all the processors and non-critical tasks are shared between the processors. An application where tasks are parallelizable can effectively use the redundant computing capacity during fault free operation. A Global Real-time Executive Manager (GREM) acts as a master/server node and plays a major role in task allocation and providing fault tolerant real-time scheduling. An algorithm has been developed for the given framework and implemented with simulation results. A comparison has been done vis-à-vis a traditional dual redundant system. Validation of the framework has been carried out with a case study of Cruise Control System (CCS), an important aspect of safety critical structure in automotives.

The gain accrued in terms of high performance and process speedup as seen in the framework laid out in [2] emphasizes the significance of this model in any safety-critical application. In this paper a hardware implementation of dual redundant model and ERMS strategies have been undertaken on ARM LPC2148. A prototype tool for testing and evaluation of the proposed fault tolerant algorithms has been designed and developed for a cruise control system. This tool

can form the basis for testing the scheduling performance of safety-critical systems in terms of improvement in resource utilization and process speedup.

The rest of the paper is organized as follows. Section 2 references the related work and Section 3 gives the background study for the hardware implementation of fault tolerant real-time systems and the case study of cruise control system. In section 4, a brief understanding of the framework is given and implementation of the case study with the experimental setup has been elaborated. In Section 5 the results of the implementation with performance evaluation is presented. The conclusion and future scope is given in Section6.

2. RELATED WORKS

J. Von Neumann [3], E. F. Moore, C. E. Shannon, [4] and their successors developed theories of using redundancy to build reliable logic structures from less reliable components, whose faults were masked by the presence of multiple redundant components. The theories of masking redundancy were unified by W. H. Pierce as the concept of failure tolerance in 1965 [5]. In 1967, A. Avizienis integrated masking with the practical techniques of error detection, fault diagnosis, and recovery into the concept of fault-tolerant systems [6]. Further a fault tolerant scheduling algorithm for multiprocessors has been analyzed by Ghosh [7]. Krishna and Shin [8] proposed a fault tolerant scheme for quick recovery of tasks from failure. Oh and Son [9] proposed a scheme that enhances the fault tolerance in static realtime scheduling.

Resource augmentation [10], allows for the use of extra resources so that scheduler can provide a specific guarantees with respect to some constraints where the scheduler under study is given extra resources such as more number of processors or faster processors, such that a certain goal is achieved. Kalyanasundaram [10] introduced resource augmentation, which studied the effectiveness of online scheduling of real-time tasks by augmenting the processor with more speed. The effectiveness of fixed priority scheduler using resource augmentation, scheduling all the feasible tasks is studied by Davis et. al [11]. A resource augmentation bounds on the processor speed-up required for a fixed priority scheduler to schedule all the task sets scheduled by an optimal scheduling algorithm was also derived by Davis. Abilash et.al., [12] focused on guaranteeing fault tolerance under error bursts on uni-processor systems by resource augmentation. A novel paradigm for fault tolerance of a permanent fault is explained in detail in [13,14,15], using task level criticality for redundancy. This ensures safe functionality of the system even during the occurrence of fault by operating in different modes where certain optional tasks and non-critical tasks are dropped.

3. BACKGROUND STUDY

3.1 Safety-critical systems

Safety critical systems are hard real time systems whose failure might lead to catastrophic effects, substantial economic loss, or cause extensive environmental damage Knight [16]. With the advent of increased development in on-chip technology multiprocessors came into existence for highly complex and sophisticated systems like safety critical systems. A worth mentioning example of safety-critical system in medical domain is modern heart pacemaker with specialized peripherals.

Task set consists of a set of tasks in an application classified as critical, non-critical and optional tasks, based on the criticality level of the task. In some sense, non-critical and optional tasks can be considered as soft real time tasks. In general, all controlling and actuating tasks are considered as critical tasks and sensing tasks are considered as non-critical tasks. One such critical task in the patient monitoring system is the task which controls the oxygen supply to the patient in ICU. Tasks in safety critical systems, periodic tasks are time driven tasks which occur at regular intervals of time, example - task which monitors the temperature of the patient in a patient monitoring system. The important task attributes are computation time (C_i), deadline (D_i), time period (T_i).

3.2 Faults and Fault tolerance

Dependability is a vital system requirement, in safety critical due to the potentially catastrophic consequences of failures [12]. Dependability of the system is typically achieved by use of fault tolerance mechanism using redundancy, which involves the execution of tasks on both primary and redundant units. Faults can be broadly classified according to their duration as permanent, intermittent and transient faults. Permanent faults exist indefinitely if no corrective action is taken. Intermittent faults appear, disappear and reappear repeatedly. It is very difficult to predict such faults. Transient faults appear and disappear quickly, they are mainly due to random environmental disturbances such as EMI [12]. In real-time safety-critical systems fault tolerance is typically provided by redundancy either hardware or software or both.

3.3 Scheduling paradigm

Static table-driven approach-In most of the safety-critical systems a static table-driven approach is followed where schedulability analysis is performed offline and the resulting schedule is used at runtime to decide which task must begin execution [18].

Static priority driven preemptive approach-This approach also produces static schedulability analysis but unlike the table driven approach no explicit schedule is constructed. During runtime whichever the highest priority task in the ready queue will be executed [18].

From the above mentioned scheduling approaches *Static table driven approach* is selected for implementation in this work, where a table is constructed with task attributes that indicates the start and completion of each task and tasks are dispatched accordingly.

3.4 Performance Metrics

Many performance metrics have been used for validating the effectiveness of scheduling paradigms for real time systems. Amongst these, improving the execution time and schedule length are mainly considered in this work. Effective Utilization (U_e) is the normalized utilization of the process during the execution of an application [19]. Process Speedup (S_p) indicates the overall execution time for the process [20].

4. SYSTEM MODEL

4.1 Assumptions

- Non-critical tasks being low priority over critical tasks are preemptable.

- An appropriate watchdog mechanism is assumed to be present that enables the detection of processor failures with a bounded latency.
- Execution time of each task is assumed to be worst case execution time(WCET) and includes time overheads for context switches and communication costs.

The fault tolerant model with a dual processor system and a master node for the given safety-critical system [2] is given in Figure 1.

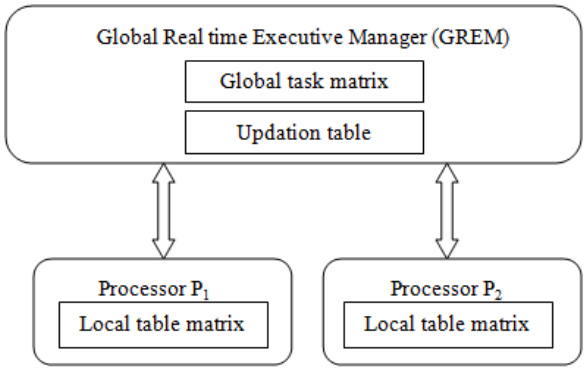

Fig 1: Schematic representation of fault tolerant model

Health check - A self check logic circuit in each processor periodically sends the health status of the processor as an ALIVE signal to the global real time executive manager.
Global Real time executive manager (***GREM)*** - The ***GREM*** acts as a master node, maintains the global task table matrix, updation table matrix during runtime and monitors the health status of the processors. Absence of the **ALIVE** signal indicates the permanent failure of the processor under which a fault recovery algorithm reallocates the tasks of the failed units to the rest of the units by dynamic online reconfiguration.

Global table matrix - This table consists of all the tasks with its task attributes, nature and criticality of the task.

Updation table - This table indicates the remaining utilization of the processors and the level of criticality of each task that is executed in each processor. It is updated for every instance of time based on the information from each processor.

Local table matrix – Each processor is provided with a local table matrix which indicates the tasks that are to be executed by the processor.

4.2 Dual Redundant Scheme (DRS)

The fault tolerant scheme with dual redundancy employs two independent, identical processors for performing the same computations as shown in Figure 2. Error detection is done by a synchronized internal hardware by a specified voting mechanism and if there is failure, the hardware disables that system. If the primary processor fails then the corresponding redundant processor takes over the operation in hot standby. Such an implementation ensures that in case of failure, the system continues to function without any break in the computations, thus meeting the real time constraints essential for such critical missions.

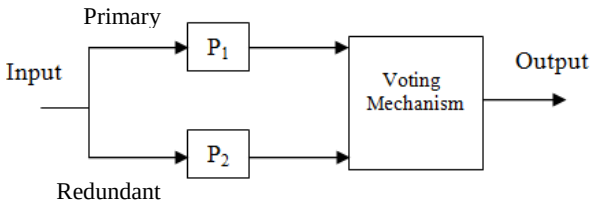

Fig 2: Dual Redundant scheme

4.3 Enhanced Resource Management Scheme (ERMS)

In ERMS scheme, Swetha et.al., [2] describes an innovative paradigm for load sharing using task level criticality shown in Figure 3. Critical tasks ($\mathcal{C}$) are duplicated in both the processors and non-critical tasks ($\mathcal{N}$) are shared among the processors. The additional slack time which is made available by the ERMS can be effectively utilized for scheduling any arrivals of aperiodic events and extra optional tasks. An algorithm **TaAl - ERMS** represents the task allocation in ERMS considering a dual processor system [2].

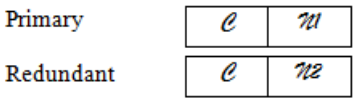

Fig 3: ERMS

Algorithm for Normal mode - Task allocation and scheduling (**TaAl - ERMS**)

Input: τ is a given task set of periodic tasks stored in ***GREM***

Output: ERMS schedule in normal mode

1: **for** i = 1 to n **do**

2: Check the criticality of task

3: Set time (P_1), time(P_2) corresponding to the required workload of the processor

4: **if** (Non-critical) **then**

5: **if** (time(P_1) > time(P_2))

6: Add task to Processor P_1

7: **else**

8: Add task to processor P_2

9: **else**

10: Add task to both P_1 and P_2

11: **for** each clock pulse **do**

12: Trigger the transmission of ALIVE signal in P_1 and P_2

13: Update ***GREM***

Under fault mode seen in Figure 4 where one of the processors fails permanently, ***GREM*** reallocates all the non-critical tasks of the failed processor to the functioning processor and schedules the tasks in this processor dynamically keeping all precedence constraints intact.

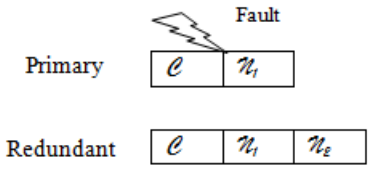

Fig 4: ERMS – Fault Mode

Algorithm for fault mode

Input: An external or internal cause leading to permanent failure of a processor

Output: Fault tolerant ERMS schedule

1: **for** i =1 to hyper period **do**

2: **if** (Alive signal (P1) is absent) **then**

3: ***GREM*** reallocates the non-critical tasks of P1 to P2 without violating the precedence constraints

4: **if** (Alive signal (P2) is absent) **then**

5: ***GREM r***eallocates the non-critical tasks of P2 to P1 without violating the precedence constraints

The periodic tasks are allocated to the processor and table driven scheduling is performed in each processor during runtime, where the task table gives the start and stop time of the tasks along with the task attributes. On the execution of each task, GREM is updated with the status of the executed task.

4.4 Case study: Cruise Control System

A ***Cruise Control System (CCS)*** which is one of the major safety critical unit in automotives is taken as a case study for the implementation of the proposed scheme. The main function of the CCS is to maintain a constant speed which is set by the driver there by reducing the driving load of the driver.

The task set of CCS with its task attributes (time in units) is given in Table 1. The periodic tasks τ_1 to τ_5 are classified as non-critical sensing tasks that can be parallelizable. Because these tasks are parallelizable, this scheme utilizes the otherwise redundant computing capacity for scheduling. Tasks τ_6, τ_7 are critical tasks and are precedence constrained with sensing tasks.

Table 1: Task set

Sl.no	Tasks / nature (P - Periodic)	Processors	C_i	D_i	T_i
1.	Monitoring the Speed τ_{NC1} (P)	P1	3	15	30
2.	Monitoring acceleration τ_{NC2} (P)	P1	2	10	20
3.	Monitoring the CCS clutch τ_{NC3} (P)	P2	2	10	20
4.	Monitoring the brakes τ_{NC4} (P)	P2	3	15	30
5.	Monitoring proximity sensor τ_{NC5}(P)	P1	2	15	20
6.	Computing the control values τ_{C6} (P)	P1,P2	10	55	60
7.	Actuating the throttle valves τ_{C7} (P)	P1,P2	5	30	60
8.	Updating the parameters in τ_{NC8}(P)	P1	10	15	60

C_i – Computation Time, D_i – Deadline, T_i – Time Period

4.5 Experimental Setup

The experimental setup for designing the prototype tool is given in Figure 5. ***GREM*** acts as server node and contains all tasks and task attributes of CCS. The algorithm implemented in ***GREM*** periodically monitors the health condition of working processors P_1, P_2 and ensures all critical task deadlines are met. Processors P_1, P_2 execute the tasks allocated to them and periodically send an ***ALIVE*** signal to ***GREM*** as part of health check process. It updates the ***GREM*** with the current status of task execution.

The microcontroller chosen for the implementation is ARM LPC2148 shown in Figure 6. It is a 32 bit processor with very low power consumption. It has high speed flash memory ranging from 32 kB to 512 kB and has an highly efficient peripherals like 2x10bit ADC, 2xUARTs, Timers, DAC etc supporting In System Programming (ISP).

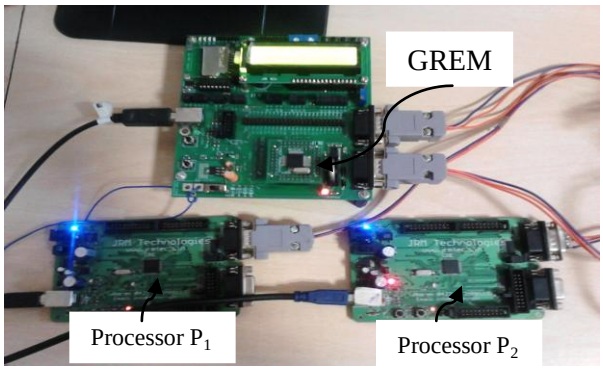

Fig 5: Experimental Setup

Fig 6: ARM LPC2148

A performance evaluation board Figure 7 is used for displaying the scheduling and health monitoring. A series of 8 LEDS are connected to each processor which indicates the execution of 8 tasks. The health status of the processor is indicated by an LED. In order to demonstrate the fault tolerant mechanism, an external fault can be injected into the system by using a push button. Two push buttons have been placed to select one among the two fault tolerant schemes. Based on the selected scheme ***GREM*** allocates the tasks to the processors P_1 P_2. Execution of each task in a processor can be illustrated by the glowing of LED in the corresponding processor and there by the total execution time of the process can be determined using the TIMER0 in each processor.

The operation of the experimental setup is projected in Figure8 as an implementation portarit. Visualisation of the ERMS schedule under normal mode is highlighted in blue in the figure. The duplication of critical tasks and the sharing of

non-critical tasks in both the processors is shown highlighted. Further, the performance evaluation has been obtained by the metrics, Effective Utilization and Process Speedup.

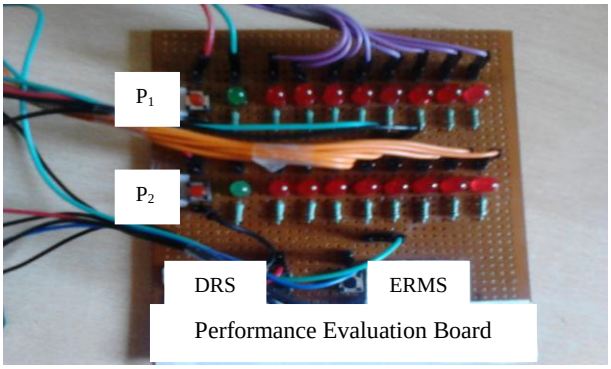

Fig 7: Performance Evaluation Board

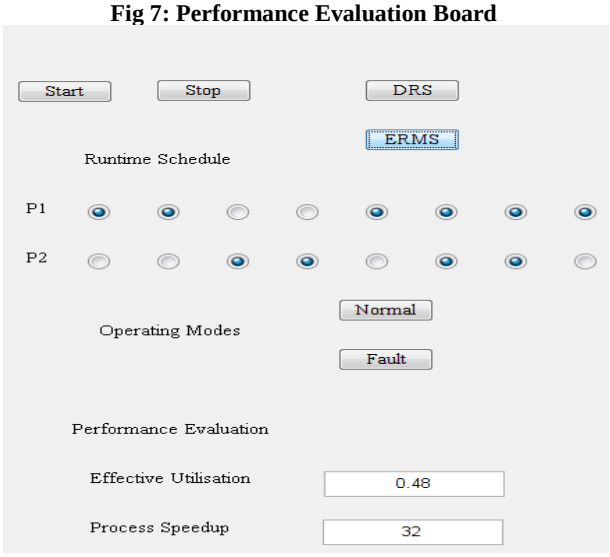

Fig 8: Implementation Portrait

5. RESULTS AND DISCUSSIONS

The execution time of the application by DRS and ERMS under the normal mode is 37 and 32 time units respectively. This proves that ERMS speeds up the process by reducing the total execution time by 13% over the DRS. Under fault mode, the execution time of DRS remains constant as 37 time units, whereas in ERMS it increases by 2 time units. Further, in the next hyper period, execution time of ERMS becomes equal to that of DRS as expected.

Table 2 a) Performance Metrics – DRS

Mode	Normal		Fault	
Metrics	U_e (units)	S_p (units)	U_e (units)	S_p (units)
	0.78	37	0.78	37

b) Performance Metrics – ERMS

Mode	Normal		Fault	
Metrics	U_e (units)	S_p (units)	U_e (units)	S_p (units)
	0.48	32	0.68	34

The normalized utilization (U_e) of the processors under DRS and ERMS have been calculated to be 78% and 48% respectively, thus providing an additional 30% of computing resources for the execution of extra optional tasks as compared to DRS. Under the fault mode, the effective utilization of ERMS increases to 68% due to reallocation of tasks to the working processor.

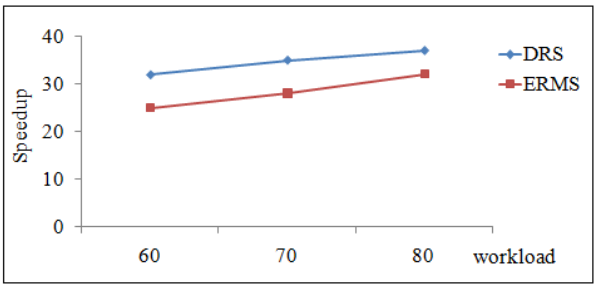

Fig 9: Process Speedup with varying workloads

Variation in process speedup with respect to workload under normal mode for DRS and ERMS models is given in Figure 9, as the workload increases the execution time of the application increases proportionally. Even under varying workload conditions ERMS speeds up the process by a minimum of 5 time units than DRS, thus providing a chance for the execution of optional tasks and aperiodic tasks.

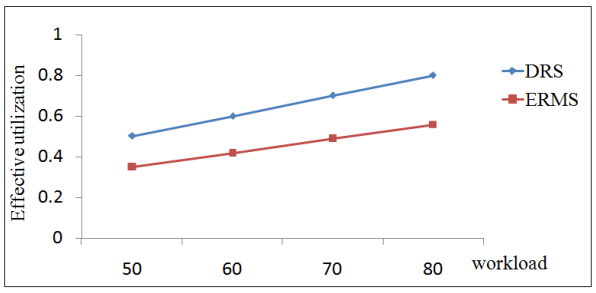

Fig 10: Effective utilization with varying workloads

Effective utilization of the processors with varying workload under normal mode for DRS and ERMS models is represented in Figure 10. In DRS as the workload increases utilization of the processor increases due to replications of tasks. In case of ERMS processor utilization increases with a step size of 0.07 which is based on percentage of critical tasks in an application. Thus this scheme, performs efficiently in applications where percentage of critical tasks are significantly lower than the percentage of non-critical tasks.

6. CONCLUSION

In this paper, a prototype of fault tolerant multiprocessor scheduling of cruise control system with effective resource management has been designed, implemented and tested using LPC2148 processors. Quality timeliness, a schedule with fault tolerance constraints and resource reclaiming are the significant gains of the implementation of this framework. A comparison of performance of ERMS over a traditional dual redundancy scheme (DRS) demonstrates the improvement which is translated in terms of additional computing resources. The ERMS speeds up the process by reducing the total execution time of the process by 13% over the DRS under runtime normal mode. The effective utilization of the processors using ERMS has been enhanced by 30% over DRS. The results emphasizes the tremendous gains that can be obtained in terms of improved performance and process speed up especially if this model is extended to an m-processor redundancy system. The metrics of importance in real time safety critical systems which are minimizing the sum of completion time, minimizing the schedule length and cost have been achieved to an extent.

This work can be further extended by the inclusion of aperiodic arrivals and inclusion of optional tasks for implementing complex algorithms which may be needed to fine tune the computations. A working prototype with a user friendly GUI can make the implementation of this framework more effective for any application. Research on the WCET of the ERMS algorithm which has been developed in this work can be of use for further implementations in different applications.

7. REFERENCES

[1] Anthony Spiteri Staines, "Modeling and Analysis of Real Time Control Systems: A Cruise Control System Case Study", University of Malta,DOI: 10.5772/7397.

[2] Annam Swetha, Radhamani Pillay V, Sasikumar Punnekkat, "Design, Analysis and Implementation of Improved Adaptive Fault Tolerant Model for Cruise Control Mulatiprocessor System", International Journal of computer applications(0975-8887), Volume 86-No 15, January 2014.

[3] J. Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, *Annals of Math Studies*, numbers 34, pages 43-98. Princeton Univ. Press, 1956.

[4] E.F. Moore and C.E. Shannon. Reliable circuits using less reliable relays. *J. Franklin Institute*, 262:191-208 and 281-297, Sept/Oc. 1956.

[5] W.H. Pierce. *Failure-Tolerant Computer Design.* Academic Press, 1965.

[6] Avizienis A., "Design of Fault-Tolerant Computers" Fall Joint Computer Conference 1967 [Aviz67].

[7] D. Mosse, R. Melhem, and S. Ghosh, "Analysis of fault tolerant multiprocessor scheduling algorithm" Proc. IEEE Real Time Systems Symp, pp.129-139, Dec.1981.

[8] C. M. Krishna and K. G. Shin, "On scheduling tasks with a quick recovery from failure" Proc. Fault-tolerant Comput. Symp, pp. 234-239, 1985.

[9] Y. Oh and S. H. Son, "Enhancing fault-tolerance in rate monotonic scheduling," Real-Time Systems, vol. 7, no.3, 1994.

[10] B.Kalyanasundaram and K.Pruhs, "Speed is as powerful as clairvoyance", Journal of ACM, July 2000.

[11] R. Davis, T. RothvoSS, S. Baruah, and A. Burns, "Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling",Real-Time Systems, July 2009.

[12] Abhilash Thekkilakattil, Radu Dobrin, Sasikumar Punnekkat and Huseyin Aysan, "Resource Augmentation for Fault-Tolerant Feasibility of Real-time Tasks under Error Bursts", 20th International Conference on Real-time & network systems, November 2012.

[13] Radhamani Pillay, Sasikumar Punnekkat, Santanu Dasgupta, "An improved redundancy scheme for optimal utilization of onboard Computers", IEEE INDICON 2009, India.

[14] Radhamani Pillay, Senthil Kumar Chandran, and Sasikumar Punnekkat, "Optimizing resources in real-time scheduling for fault tolerant processors", IEEE, International Conference on Parallel, Distributed and Grid Computing (PDGC-2010), Solan India; October2010.

[15] Senthil Kumar Chandran, Radhamani Pillay, Radu Dobrin, and Sasikumar Punnekkat, "Efficient scheduling with adaptive fault tolerance in heterogeneous multiprocessor systems", International Conference on Computer and Electrical Engineering (ICCEE) Chengdu, China; Nov 2010. China.

[16] John C.Knight, "Safety Critical Systems: Challenges and Directions", Department of Computer Science, University of Virginia., Proceedings of the 24rd International Conference on Software Engineering, 2002.

[17] Technical Report: "Adaptive Cruise Controllers – A Literature Review", Stefan Björnander, Mälardalen University, Sweden 2008 C4-01 TR M50.

[18] N. Audsley, A. Burns, "Real time system scheduling", Department of Computer Science,University of York, UK, Predicatably Dependable Computer Systems, Volume 2, Chapter 2, Part II .

[19] Gurulingesh R, Neera Sharma, Krithi Ramamritham and Sachitanand Malewar, "Efficient Real-Time Support for Automotive Applications: A Case Study", Indian Institute of Technology Bombay.

[20] Robert I. Davis, Alan Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems", Real-Time Systems Research Group, Department of Computer Science, University of York, York, UK.

[21] Sherin Abraham, Sivraj.P, Radhamani Pillay, "Hardware Implementation of an Improved Resource Management Scheme for Fault Tolerant Scheduling of a Multiprocessor System", International Journal of Computer Applications (0975-8887), Volume 27-No.2, August 2011.

[22] Akash Anand, Rajendra Y, Shreyas Narayan, Radhamani Pillay, "Modelling, Implementation and Testing of an Effective Fault Tolerant Multiprocessor Real-Time System", IEEE, International Conference on Parallel, Distributed and Grid Computing (PDGC-2012), India.