Integrating Fault Tolerant Scheme With Feedback Control Scheduling Algorithm At Software Level For Real Time Embedded Systems

Oumair Naseer[1], Akeel Shah[2]

[1,2] School of Engineering. University of Warwick, Coventry, United Kingdom

ABSTRACT

In order to provide Quality of Service (QoS) in open and unpredictable environment, Feedback based Control Scheduling Algorithm (FCSA) is designed to keep the processor utilization at the scheduling utilization bound. FCSA controls CPU utilization by assigning task periods that optimize overall control performance, meeting deadlines even if the task execution time is unpredictable and through performance control feedback loop. Current FCSA doesn't ensure Fault Tolerance (FT) while providing QoS in terms of CPU utilization and resource management. In order to assure that tasks should meet their deadlines even in the presence of faults, a FT scheme has to be integrated at control scheduling co-design level. This paper presents a novel approach on integrating FT scheme with FCSA for real time embedded systems. This procedure is especially important for control scheduling co-design of embedded systems.

KEYWORDS

Fault tolerance, Dependable embedded systems, Feedback based control scheduling, Control scheduling co-design and Check pointing.

1. INTRODUCTION

Real-time embedded systems have become pervasive, in that they can be found in various systems, ranging from safety-critical such as avionics to consumer electronics, mobile phones and waging machines [1]. Over 90% of the embedded microcontrollers are used to control the physical processes and devices [3]. Due to the large number of real time constraints and requirements, embedded computing systems are becoming increasingly complex. Scheduling is the key lever in these computing systems for system performance and resource usage. Classical real time scheduling algorithms used in the embedded system design are Rate Monotonic (RM) and Earliest Deadline First (EDF). From the control perspective all these classical scheduling algorithms are open loop [10]. Also, these algorithms are designed based on the assumption that mapping of the jobs/tasks is known a priori, and that the worst case execution time (WCET) of jobs is known prior to the execution. Due to the open and uncertain environments in which embedded systems are often deployed, the execution time of jobs can vary, such that it is very difficult to accurately predict the timing constraints, such as WCET, of the job before execution. To avoid this uncertainty, feedback control theory is integrated with embedded computing systems [11-14]. Some embedded systems, such as in avionics or cars, have both safety critical and non-safety critical requirements. Faults happening in such systems can occur either in hardware or in software. These faults are further categorised into transient faults and permanent faults [4]. Transient faults occur only for a short period of time whereas permanent faults affect the system forever. To tolerate faults in such systems, fault tolerant (FT) schemes are

DOI : 10.5121/ijics.2013.3401

implemented [5]. Traditionally, FT schemes are based on some notion of redundancy. In real-time systems, hardware redundancy [2] is used to tolerate transient or permanent faults. However, replication method incurs high hardware cost for fault tolerance.

To design for fault tolerance, error detection and an error correction mechanism is required. Some of the most popular FT schemes are: *(i) Active replication,* Fig. 1(a): In this scheme, a job is replicated on different processors and the replicas perform the required services [6*], (ii) Primary backup,* Fig. 1(b): in this scheme, each job has a backup which is executed whenever a fault is detected, and *(iii) Re-execution,* Fig. 1(c): in re-execution, when a fault is detected in an executing task, the task is re-executed from the start. This scheme works only if the fault disappears during re-execution. Each of the above scheme provides fault tolerance, under specific assumptions and failure scenarios, e.g. replication with three 3 replicas will allow for detection and correction, whereas duplication will only allow for detection, and the technique needs to be integrated with a mechanism that allows for error correction.

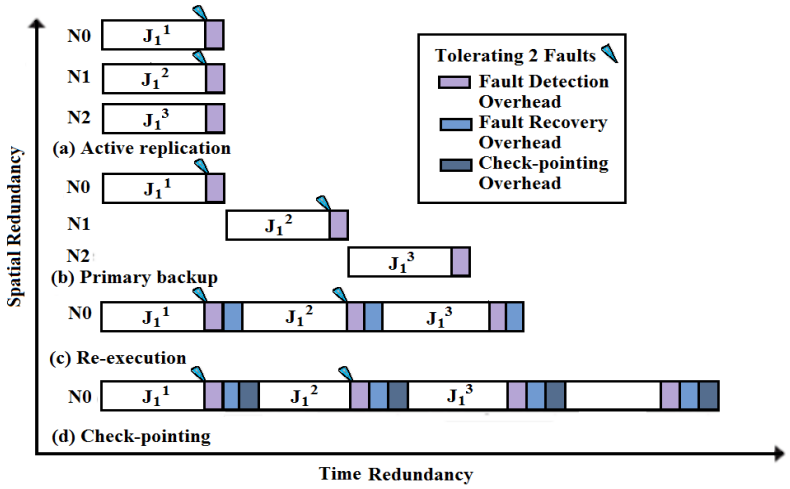

Figure 1: Tradeoffs between different Fault Tolerant schemes.

(iv) Check pointing Fig. 1(d): in check pointing [7], a job is divided into n sub jobs and each sub job contains a checkpoint appended by either a programmer [8] or by the compiler [9], where the state of the system is saved. A fault is detected when the outputs of the two replicas are compared. There is no agreement between the outputs and correction is obtained based on the checkpoints.

Selection of the best suitable FT scheme is based on the system requirements and constraints. However, each FT scheme introduces a delay in the execution time of the task, e.g. *i) Constants delays:* With check pointing (FT scheme), the state of the task is compared for error at each check point. These checkpoints introduce a constant time delay in the execution time of the task. These check points are appended by either a programmer or compiler. Due to the large number of real time constraints, the choice of constant time delays becomes restrictive and less relevant when it comes to the systems having distributed environment where the delays are induced by network. *ii) Bounded time-varying delays:* With re-execution (FT scheme), when a fault is detected the execution of the task starts from the beginning. This FT scheme introduces bounded time-varying delays in the system. Such delays are very common in the systems using networks such as I2C, CAN [24] and flex ray [25] and can be represented as; $0 \leq d(t) \leq d_1$. *iii) Interval time-varying delays:* With the combination of two or more FT schemes such as; Re-execution with check pointing, the time delays introduced in the execution time of the task varies only for a short period of time from one check point to another check point. By saving the state of the task at each

check point, unbounded time variations can be minimized. *iv) Delays with constraints on their first derivative*: For network stability, numerous stability conditions require that the delays functions can vary arbitrarily e.g. the function is strictly increasing. *v) Piece-wise time varying delays:* With networked control systems, the delay functions are not continuous and can be visualised as piece-wise time varying delay. In this paper we have integrated re-execution with check pointing (FT scheme) to provide both error detection and error correction. Table 1 shows the execution of the task at two different processor nodes N0 and N1.

Node	Task	Sub-Tasks	Task Description	Lines of code	Execution Time in Clock Cycles (CC)
N0	J0	J01	Keypad scan	18	29 CC
		J02	Telegram Building	28	28 CC
		J03	Telegram Mapping	30	45 CC
N1	J1	J11	Keypad scan	18	29 CC
		J12	Telegram Building	28	28 CC
		J13	Telegram Mapping	30	45 CC

Table 1: Task J0 is executed on Node No and task J1 is executed on node N1. Each task is divided into three subtasks.

At each sampling interval an error detection processor checks the state of the processor by comparing values of the state variables. If the values of state variable are same then there is no fault in the subtask. However, if the values of state variables are different then there is a fault in the subtask. By saving the state of the task at each sampling interval a lot of re-execution (roll back) time can be saved. This is done by using a check pointing processor and a stable storage memory as shown in Fig. 2.

Figure 2: State of the task is saved in the stable storage by Check pointing processor.

2. RELATED WORK

Feedback performance control for computing systems is presented in [16] which primarily focus on applying control theory to real time scheduling and utilization control. For the real time systems with unknown execution time, a state of the art feedback control scheduling algorithm is presented in [17] which provide the performance guarantee for hard real time tasks. Feedback Dynamic Voltage Scaling (DVS) method to select proper frequency and voltage for Fault tolerant hard real time embedded system is presented in [33-35]. Author also tries to provide QoS by reducing energy consumption and satisfying hard real time constraints in presence of fault. It also provides a technique to integrate DVS and Feedback control theory for hard real time systems. An analysis of distributed control with shared communication and resources utilization for real time system is addressed in [19]. FT scheme check pointing for real time embedded systems is integrated in [7] but this work doesn't talk about control scheduling co-design. A perspective on integrating control and computing for control scheduling co-design is presented [18]. Control design for networked control system; a novel approach for designing control scheduling for the networked systems, is addressed in [20, 28]. An adaptive neural network based feedback control scheduling for soft real time embedded systems is addressed in [13, 14]. In [11] Author provides an approach to recover system from fault mode for parallel systems using check pointing FT scheme and control theory. Trade-offs between reliability/FT and control theoretical methods is addressed in [38]. In [15] author uses a double feedback based control scheduling approach for real time systems to achieve high performance. A control system scheduling for hard real time systems is addressed in [18], but this work doesn't address the Fault detection and Fault recovery mechanism together with feedback control theory. Feedback based control scheduling co-design approach for real time embedded system is presented in [29]. This work shows that closed loop systems are not hard real time systems, although control systems are more robust in nature and uncertain to time variations, but they also suffers from time jitters and data loss. Author also provides different techniques to model time delays in system suffering from data loss over network. In [22, 23] author tries to capture the time variation of Safety Critical (SC) tasks over network for better resource management and bandwidth utilization in correspondence with sampling intervals and time delays to achieve QoS. System response in presence of transient fault and FT schemes for hard real time systems to achieve dependability in X-by-Wire (XBW) systems is addressed in [26 and 27]. A fault tolerant scheduling for hard real time systems is addressed in [38], but this work only focuses on maintaining CPU scheduling with specified scheduling bound by making sure that SC tasks will meet their deadlines. Moreover, this work doesn't capture the task state in fault mode and provides less information about data loss.

Up to date control scheduling algorithms based on Fuzzy logic controller network control is presented in [12]. Author provides a Feedback based Scheduling Control (FCS) framework for adaptive real time systems by developing dynamic model of real time systems. Author demonstrates the robustness of the tuned FCS algorithms when the task execution time varies as much as 100% from the initial estimate. The existing FCSA design technologies only focus on the use of feedback control theory and do not consider the integration of Fault tolerant schemes. To the best of our knowledge, this is the first work that presents the integration of fault tolerant schemes integration with feedback control scheduling algorithms for real time embedded systems.

3. PROBLEM STATEMENT

Due to the increasing design complexity of embedded system, it is very common that several control tasks have to compete for one embedded processor. Therefore, the overall system performance not only depends on the design of control algorithms but also rely on the efficient scheduling of shared computing resources. Unfortunately, the design of embedded control systems is often based on the principle of separation of concerns [15] of control and computing.

This separation is based on the assumption that feedback controllers can be modelled and implemented as periodic tasks that have a fixed time period, a hard deadline, and a known WCET. These assumptions have also been widely adopted by control community for developing sampled control theory which allows the control community to focus on its own problem. For instance, faults associated to embedded control systems can occur at any time, due to which the control task execution time increases than estimated and the control tasks started missing their deadlines. Also many control loop deadlines are not always hard. Instead most practical control systems can tolerate occasional deadline misses due to fault. As a result, the resulting Quality of Control (QoC) of real-time control systems that are designed based on this separation of concerns of control and computing would be worse than possible, and in extreme cases unacceptable with instability. In order to cope with this problem fault tolerant scheme has to be integrated with feedback control scheduling algorithm. In this paper, a novel approach to integrate the FT schemes with FCSA is presented and stability of the system is analysed in the presence of faults. Finally, the verification of this novel approach is investigated on Crane Control System.

4. FEEDBACK CONTROL SCHEDULING ALGORITHM (FCSA)

Feedback scheduler controls the processor utilization by assigning task periods that optimize the overall control performance. This approach is well suited for a "quasi-continuous" variation of the sampling periods of real-time tasks under control of a preemptive Real-Time Operating System (RTOS). Feedback scheduling is a dynamic approach allowing a better using the computing resources, particularly when the workload changes. Fig. 3 gives an overview of a feedback scheduler architecture where control inputs are the periods of the control tasks and output is the measures CPU utilization. CPU activity is controlled according to the resource availability by adjusting scheduling parameters (i.e. periods of the control tasks). An outer loop (the scheduling controller) adapts in real-time scheduling parameters from measurements taken on the computer's activity, e.g. the computing load. FCSA works periodically at a rate larger than the sampling periods of the plant control tasks. System structure evolves along a discrete time scale upon occurrence of events, e.g. for new tasks admission or exception handling.

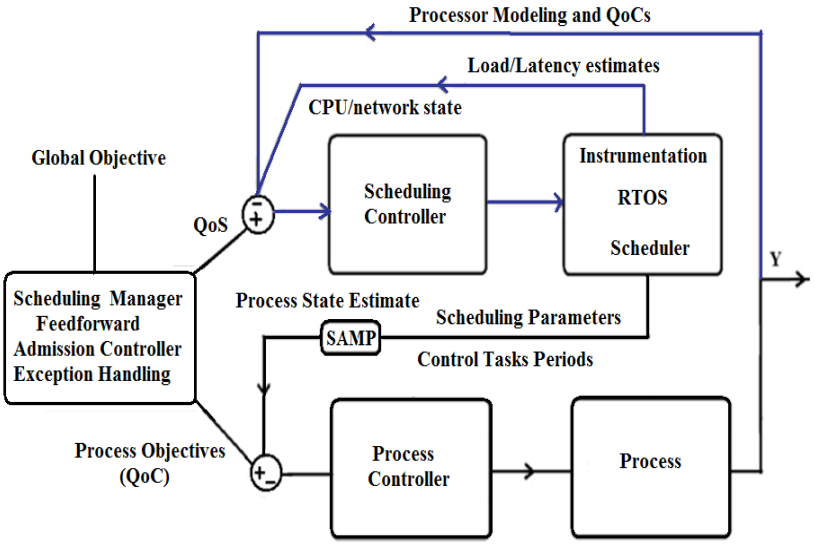

Figure 3: Feedback control scheduling architecture.

The off-line iterative optimization is used to compute an adequate setting of periods, gains and latencies resulting in a requested control performance according to the available computing

resource and implementation constraints. The optimal control is achieved by computing the new tasks periods by the rescaling:

$$h_i^{k+1} = h_i^k \frac{U}{U_{sp}} \tag{1}$$

Where U_{sp} is the utilization set-point, h_i^k is the period of task at time k, U is the estimated CPU load and h_i^{k+1} is the new period of task i at time $k+1$. Processor load induced by a task is defined by $U = \frac{c}{h}$ Where c and h are the execution time and period of the task respectively. Estimated processor load induced by a task for each period h_s of the scheduling controller is defined as:

$$\hat{U}_{khc_s} = \lambda \hat{U}_{(k-1)h_s} + (1-\lambda)\frac{\bar{c}_{kh_s}}{h_{(k-1)h_s}} \tag{2}$$

Where h is the sampling frequency currently assigned to the plant control task (i.e. at each sampling instant kh_s) and $\bar{c}$ is the mean of its measured job execution-time. λ is a forgetting factor used to smooth the measure. $\hat{U}_{(k-1)h_s}$ is the processor load induced by task at time $k-1$ and period h_s . Due to the execution time variation of the task in the presence of faults, estimated CPU load is defined as a function of task periods as:

$$\hat{U}_{(kh_s)} = \frac{(1-\lambda)}{z-\lambda}\sum_{i=1}^{n} \bar{c}_i(kh_s) f_i(kh_s) \tag{3}$$

Where f_i is the frequency of the task i. A single control task system is given in Fig. 4 where the estimated execution-times are used on-line to adapt the gain of the controller for the original CPU system (3) (this allows to compensate the variations of the job execution time).

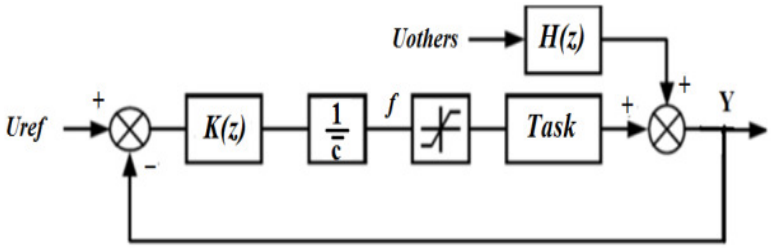

Figure 4: Control scheme for CPU resources.

With this control scheme, design of controller K can be made any control methodology at hand. As $\bar{c}$ depends on the run-time environment (e.g. processor speed, task jitter) a "normalized" linear model of the task i (i.e depend on the execution time), G_i is used for the scheduling controller synthesis where $\bar{c}$ is omitted and will be compensated by on-line gain-scheduling ($1/\bar{c}$) as shown below.

$$G_i(z) = \frac{\hat{U}(z)}{f_i(z)} = \frac{1-\lambda}{z-\lambda}, \quad i = 1,2,\ldots,n \tag{4}$$

Fuzzy Logic Controller based feedback scheduler is showed in Fig. 5 where *U(k)* is the total CPU load measured for each period of scheduling task, *M(k)* is the task deadline miss ratio. *Ud* is the desired load. *Md* is the controller variable, to control the task deadline miss ratio. Adding Feed-forward admission controller allows future tasks cost anticipation and for enhanced transient behaviour.

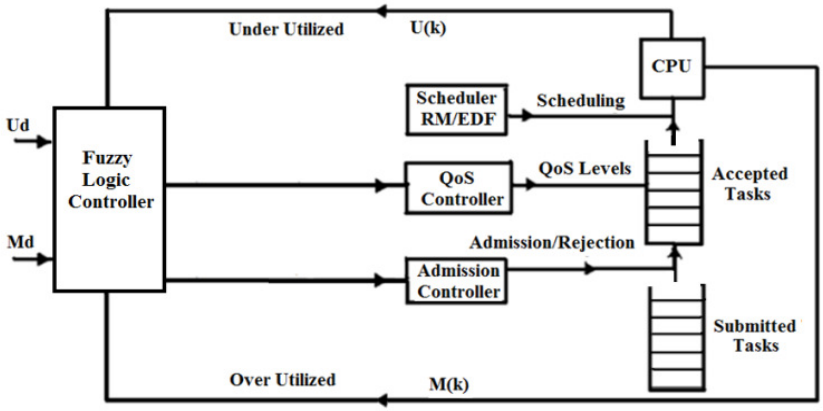

Figure 5: Feedback Control Scheduling Architecture.

5. SYSTEM ARCHITECTURE AND IMPLEMENTATION

FT based FCS is implemented by using the crane control system. System architecture of crane control system constitutes a distributed shared hardware platform with a network topology where every hardware node can communicate with every other node. Fig. 6 shows the high level model of the system architecture and resources elaborating the partitioning concepts.

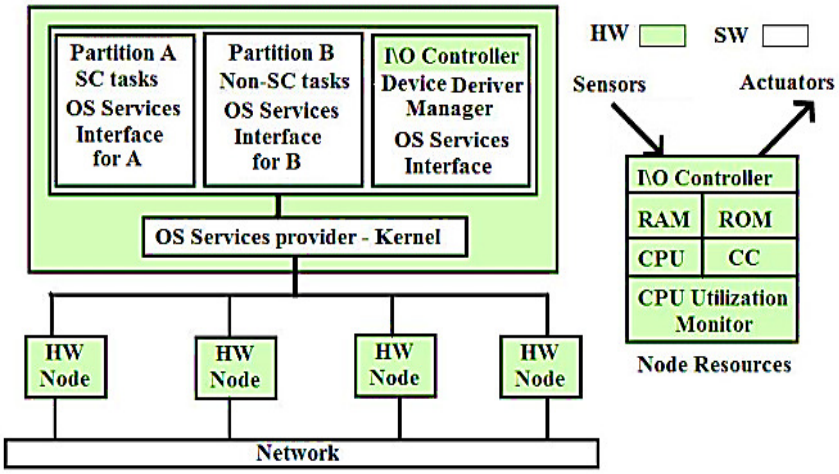

Figure 6: System architecture model follows the integrated approach where each core is divided into two parts. Part A executes SC tasks and Part B executes non-SC tasks. SC and non-SC tasks can execute at any hardware node.

It also describes the application execution environment where nodes are connected through a network bus. Each node has two partitions. Partition A is dedicated for the safety critical jobs and partition B contains non-Safety critical jobs, with shared processor running mixed criticality applications. Node resource consists of a CPU, I/O controller; sensors and actuators, RAM, ROM and a CPU utilization monitor. Every node in the system integrated architecture utilizes the same configuration. Crane control system consists of two major parts; an Operator control unit (OCU) and a Machine Control Unit (MCU) [36-38]. Both OCU and MCU contain two microcontrollers Renesas M16C62p [30], which is the most popular microcontroller used in motor industry for industrial automation and it contains a built-in I2C for multi-processor communication. OCU has multi-level push button keypad installed at the outer surface. Each button is a three steps press push button, to control the speed of the machine attached with MCU. Both OCU and MCU has Radio Frequency Identification (RFID) chip. OCU and MCU communicate through RFID module. [32] RFID chip contains OCU identification number and the address of that particular OCU. This information is transmitted through RFID module in the form of a telegram. Each telegram is 32 byte information and contains a start sequence, telegram identification bytes, timing information, data bytes (information of pressed keys), Cyclic Redundancy Check CRC and the stop sequence. Telegram is sent periodically to MCU. MCU receives that telegram decodes the data bytes and performs action accordingly. One OCU can communicate to several MCU if all MCUs have the same address and the frequency band. 433MHz, 960MHz, and 360MHz are the frequency bands supported by the RFID modules. Both OCU and MCU contain a Liquid Cristal Display (LCD) attached, which shows the current status of the OCU and MCU respectively Fig. 7. There is an emergency stop switch attached to OCU, whenever this switched is pressed MCU should stop instantly.

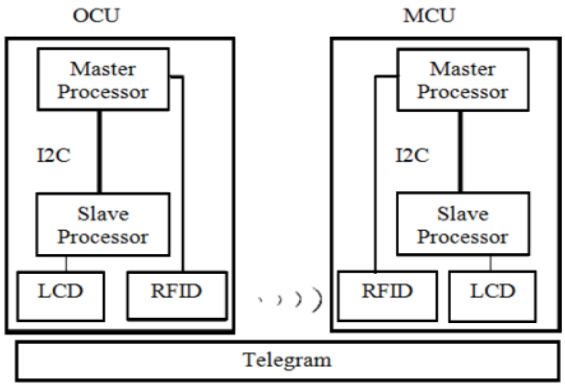

Figure 7: OCU and MCU system architecture.

There are some Safety Critical (SC) and non-SC tasks associated to this system. Degree of the task replication depends upon the safety level of the SC task and is defined by Safety Integrity Level (SIL) Table 2.

SIL	Criticality	System failure	Probability of dangerous failure per hour
4	Safety critical	Catastrophic failure	$10^{-8} - 10^{-7}$
3	Safety relevant	Server failure	$10^{-7} - 10^{-6}$
2	Critical	Major failure	$10^{-6} - 10^{-5}$
1	Non-critical	Minor failure	$10^{-5} - 10^{-4}$
0	No dependability requirements		

Table 2: Safety integrity levels (SIL)

Tasks responsible to display information on LCD are all non-SC tasks. Tasks that scan the push buttons are SC tasks, especially the task associated to monitor emergency stop button. Telegram mapping and transmission tasks are SC tasks. Both master and Slave processors build their own telegram and then validates it before transmission using I2C communication network. Similarly, task that decodes information on MCU is also a SC task. FCSA model is constructed in Matlab/Simulink [31]. Based on this Simulink model a C code is generated with is then integrated with the system code implemented on industrial standards MISRA C in High performance Embedded Workshop (HEW). Transient faults are injected in the system by using test scripts at software level. Steady state response of the system is investigated through Matlab and actual CPU execution time is monitored by using a software time.

6. EXPERIMENT

The purpose of first experiment is to test the robustness (variation of tasks execution in the presence of faults) of the system with and without FT integration. One microcontroller is configured as the master controller and the second microcontroller served as the slave controller. For this experiment, two SC telegram mapping tasks are considered. Both controllers map their own telegrams independently and compare at different sampling intervals using I2C bus network to validate the correctness of the telegram. For this experiment two sampling intervals for each tasks is allocated and a software timer is used to calculate maximum time elapsed (actual execution time) between the two sampling intervals. CPU utilization is monitored both with and without FT integration. Estimated values are verified using Matlab. Aggregate error for the each CPU utilization is calculated by using the equation below when the system is in steady state [21];

$$E = (\sum_{k=D1}^{D2} e_i^2(k))/(D2 - D1) \tag{5}$$

The purpose of second experiment is to investigate the maximum schedulable limit and upper bound of check points. For this experiment, ten SC tasks are considered. Apart from that 30 non-SC tasks on the master processor and 20 non-SC tasks on the slave processor are allocated. Three checkpoints for each SC task are allocated. In this experiment the maximum utilization bound of processor is tested. The purpose of third experiment is to investigate the trade-offs of using fault tolerant based feedback control scheduler against open loop EDF scheduler. For this purpose, four periodic SC tasks are considered. Task T1 scans the keypad associated to OCU. Task T2 and T3 are associated to telegram building. Task T4 is associated to inter-processor communication (I2C read/write). The QoC of the system is investigated by introducing faults in the system at the software level (using test scripts). These faults will eventually increase the execution time of the tasks. Also, if a task is in search for a resource which is occupied by another task then that particular task may miss its deadline.

7. RESULTS

Table 3 shows the values of CPU utilization with and without FT scheme integration of experiment 1. With FT scheme integration, CPU is slightly over utilized, which suggests that SC tasks take more time to execute than estimated. Ratio between the estimated execution time and the actual execution time g is calculated with the help of software timer. For Master microcontroller, CPU utilization turns out to be g=(0.16–0.20), which means that the actual execution time of SC task deviates from 16% to 20% of its estimated completion execution time. For Slave microcontroller, g=(0.10–0.15), which means that the actual execution time of SC tasks on slave processor deviates from 10% to 15% of its estimated execution time. Both

microcontrollers have different values of g because total number of tasks executed on slave microcontroller is more than master microcontroller.

SC Tasks	Non SC Tasks	CPU Type	CPU Utilization with FT scheme integration	CPU Utilization without FT scheme integration	SC task Execution improvement
2	5	Master	0.9213	0.8955	(16–20)%
2	8	Slave	0.9256	0.8713	(10–15)%

Table 3: CPU Utilization with and without FT scheme integration with FCSA for 2 SC tasks including 8 transient faults.

Steady state response of master and slave CPUs are shown in Fig. 8. Both CPUs are robust against uncertain task variation. Initially, both CPUs are under-utilized due to system model estimation inaccuracies but model become more accurate later. Variation of task execution time is evident at sampling interval 300^{th} where g=0.16 (16% execution variation of SC task) and at sampling interval 700^{th} where g = 0.20 (20%) for master CPU and at 300^{th} g=0.10 (10% execution deviation) at 600^{th} g=0.15 (15% execution deviation).

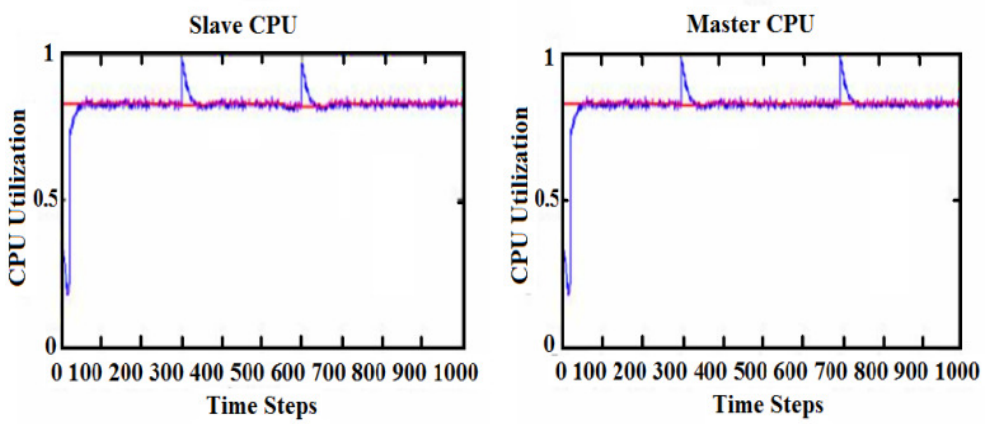

Figure 8: CPU Utilization for Experiment 1.

For the second experiment, execution variation g=(2.0–2.40) for master microcontroller, which means actual execution time for SC task with FT scheme integration is 2.0 – 2.40 times more than estimated completion execution time. Also there are 10 SC tasks are scheduled on the master microcontroller. On slave microcontroller g=(1.8–2.2) which means that actual execution time for SC tasks with FT scheme is 1.8–2.2 times more than estimated completion execution time. Also there are 10 SC tasks are scheduled on the slave microcontroller.

SC Tasks	Non SC Tasks	CPU Type	CPU Utilization with FT scheme integration	CPU Utilization without FT scheme integration	SC task Execution improvement
10	30	Master	0.9107	0.8723	(200–240)%
10	20	Slave	0.8987	0.8421	(180–220)%

Table 4: CPU Utilization with and without FT scheme integration with FCSA for 10 SC tasks including 15 transient faults.

Fig. 9 shows a variation in CPU utilization. At sampling interval 200^{th} g = 2.00, shows 200% execution time deviation and at 700^{th} g = 2.40, shows 240% execution time deviation for master CPU and for slave CPU sampling interval 300^{th} g = 1.80, shows 180% execution time deviation and at 700^{th} g = 2.20, shows 220% execution time deviation.

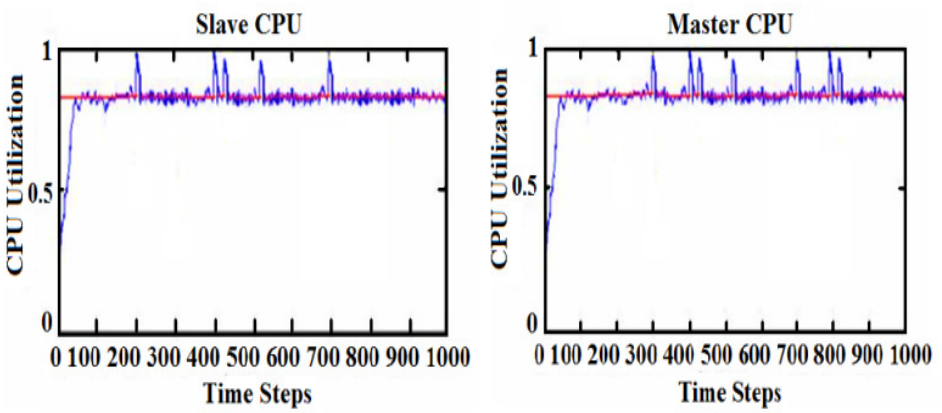

Figure 9: CPU Utilization for Experiment 2.

It is also observed that beyond g=6.98 the scheduler becomes instable and after g=7.0 CPU waveform starts to oscillate (showing instability of scheduler) as shown in Fig. 10.

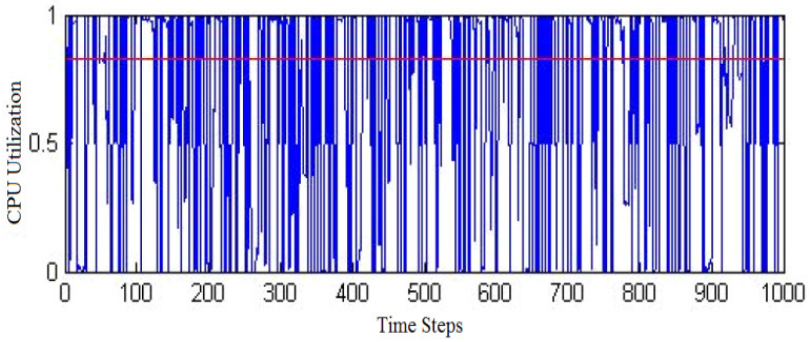

Figure 10: At g= 6.98 CPU wave-form starts to oscillate.

Fig. 11 shows the result of third experiment (scheduling of four tasks T1–T4) using open loop Earliest Deadline First (EDF) scheduling. At time step 3.25 µsec, when task T4 is introduced, the execution of the task is suspended by the scheduler till time step 4.0 µsec. At time step 4.08 µsec task T4 gets a chance to execute. Also, between time steps 3.50–4.0 a fluctuation in the CPU utilization wave form is observed which clearly shows the instability of the scheduler. When

actual CPU utilization exceeds the desired set point, this means a task is taking too much time to execute (because the fault introduced in the system has increases the completion time of the task) as in the case of task T4 at time step 3.25 µsec. When actual CPU utilization is below the desired set point, this means task has completed its execution before its estimated completion time as happened at time step 3.625 µsec. Task T2 has completed its execution before estimated time.

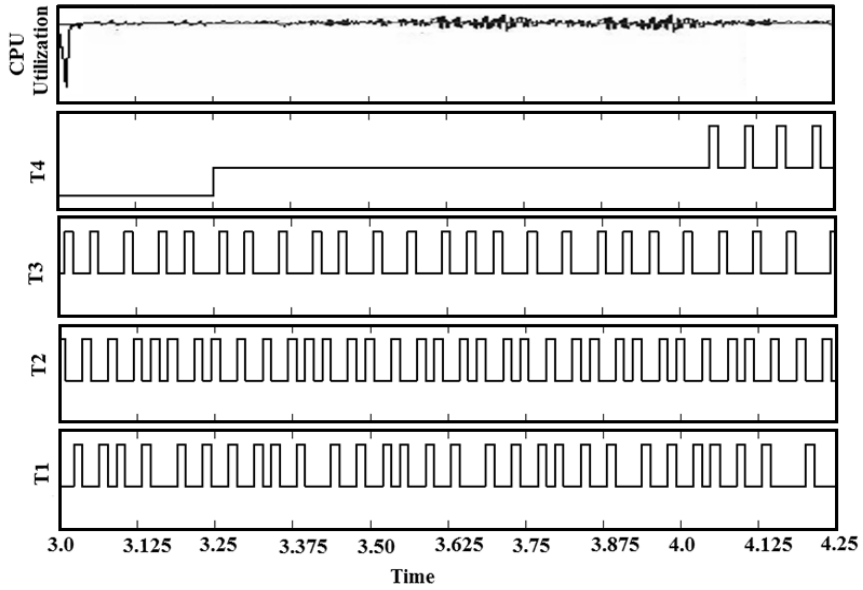

Figure 11: Scheduling of four tasks (T1-T4) using open loop EDF scheduler. A high level means task gets a chance to execute. A low level means task has completed its execution and is terminated. A middle level means that execution of the task is suspended.

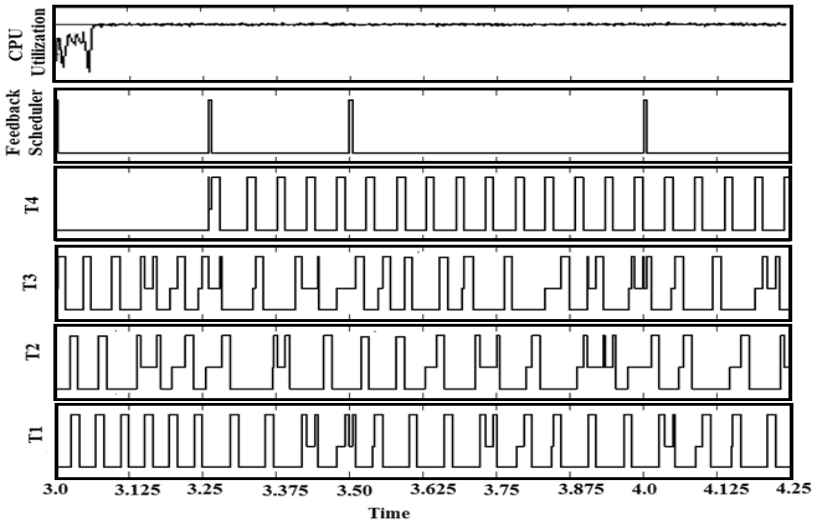

Fig. 12 shows the execution of the same four tasks (T1-T4) using the closed loop FT based Feedback based Control Scheduler.

Figure 12: Scheduling of four tasks (T1-T4) using closed loop FCS. At time step 3.25 when Task T4 is introduced feedback based control scheduler activates and schedules task T4 immediately.

At time step 3.25 µsec when task T4 is introduced, the feedback based control scheduler gets activated and instead of suspending the task execution, the task period is updated such that task

T4 gets a chance to execute as soon as it is introduced. Also between time steps 3.50-4.0 µsec CPU utilization waveform is much more stable.

8. CONCLUSION

This paper provides a novel technique of integrating FT scheme with feedback control scheduling algorithms. System architecture presented in this paper is more robust in terms of the execution variation (CPU utilization) of tasks (schedulable) to a great extent. It is also evident from the experiments that in order to achieve a system with higher reliability and fault tolerant, tradeoffs have to make between the CPU utilization and the number of SC tasks to be scheduled on a particular processor. It is also observed that from g=0.05–7.0, integrated fault tolerant FCSA remains robust (schedulable) after that the number of sampling intervals exceeds the upper bound. The completion time of SC tasks exceeds their WCET and SC tasks started missing their deadlines. Greater number of sampling intervals leads to higher reliability and FT but on the other hand the task execution time increases. Increasing sampling intervals beyond required bound can also leads to network instability. To achieve high QoS (CPU utilization and resource allocation) a balance has to be made by the designer between the numbers of SC tasks to be scheduled on a particular processor, the check-points(sampling intervals) assigned for each SC task, CPU utilization and bandwidth utilization of communication network.

9. FUTURE CONSIDERATIONS

A. Timing delay models:

In this paper delay time is modelled as the bounded time varying delay, however if sampling intervals are known such that there exists two scalar values *d1* and *d2* and the variation exists between these two scalar values then this kind of delays can be modeled as Interval time varying delay.

$$0 < d_1 < d(t) < d_2.$$

Also if the sample interval time function varies in a piecewise manner than Piecewise time varying delay model will be very helpful. For example an increasing sequence of signal $(S_i)_j$ can be seen as a delayed signal with;

$$S(t) = t - t_1.$$

B. Heterogeneous System:

This paper only focuses on the system having the identical processor and same CPU utilization model is adapted for both processors. However, if system has different hardware nodes in terms of processor speed, power and dedicated ASIC application, then time delay model has to capture these constraints as well while keeping the system stability intact.

10. REFERENCES

[1] B. Bouyssounouse, J. Sifakis, Embedded Systems Design: The ARTIST Roadmap for Research and Development, Springer, 2005.

[2] P. Agrawal. Fault tolerance in multiprocessor systems without dedicated redundancy, IEEE transactions on computers, 37:358-362, March 1988,

[3] P. A. Bernstein. Sequoia: A fault-tolerant tightly coupled multiprocessor for transaction processing, Computer, 21:37-45, February 1988.

[4] J-C., Laprie, & B. Randell, Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on DependableSecure Computing (TDSC), 1(1), pages 11{33, 2004.

[5] R. M. Keichafer, C.J. Walter, A.M. Finn & P.M. Thambidurai, The MAFT Architecture for Distributed Fault Tolerance, IEEE Transactions on Computers, 37(4), pages 398{405, 1988.

[6] S. Poledna, P. Barrett, A. Burns, & A. Wellings, Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems, IEEE Transactions on Computers, 49(2), pages 100{111, 2000.

[7] S. Poledna, P. Barrett, A. Burns, & A. Wellings, Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems, IEEE Transactions on Computers, 49(2), pages 100{111, 2000.

[8] Avi Ziv, jehoshua Bruck, Analysis of checkpointing schemes for multiprocessor systems, 13th Symposium on Reliable Distributed Systems, 1994.

[9] K. M. Chandy and C. V. Ramamoorthy, Rollback and recovery strategies for computer programs, IEEE Transactions on computers, 21:546-556, June 1972,

[10] J. Long, W. K. Fuchs, and J. A. Abraham. Fowrawd recovery using checkpointing in parallel systems. In the 19th International Conference on Parallel Processing, pages 272-275, August 1990.

[11] C. Lu, J.A. Stankovic, G. Tao, S.H. Son, "Feedback control real-time scheduling: framework, modeling, and algorithms", Real-time Systems, Vol.23, No.1/2, pp. 85-126, 2002.

[12] Sha, L., T. Abdelzaher, K.-E. Årzén, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, A. Cervin, J. Lehoczky, A. Mok, "Real-time scheduling theory: A historical perspective", Real-time Systems, Vol.28, 2004.

[13] A. Goel, Walpole, and M. Shor. "Real-rate scheduling," in proceedings of the 10th IEEE Real-Time and Embedded technology and Applications Symposium (RTAS), pp. 434-441, 2004.

[14] S. Lin and G. Manimaran. "Double-Loop Feedback-Based scheduling Approach for Distributed Real-Time Systems," in proceedings of the High Performance Computing (HiPC), pp. 268-278, 2003.

[15] J.A. Stankovic, T. He, T.F. Abdelzaher, M. Marley, G. Tao, S.H. Son, and C. Lu. "Feedback Control Real-TimeScheduling in Distributed Real-Time Systems," in proceedings of the IEEE Real-Time Systems, 2001.

[16] K.E. Årzén, B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson, and L. Sha, Integrated control and scheduling. Technical Report ISRN LUTFD2/TFRT7586SE. Lund Institute of Technology, Sweden, 1999.

[17] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," J. ACM, vol 20,no. 1, pp. 46-61, 1973.

[18] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms,"Real-Time Systems J., vol. 23, no. 1/2, pp. 85-126, 2002.

[19] Feng Xia and Youxian Sun, Control-scheduling codesign: A prespective on integrating control and computing. Dynamics of Continuous, Discrete and Impulsive Systems - Series B, vol. 13, no. S1. 2008.

[10] Jianguo Yao and Xue Liu, Mingxuan Yuan, Zonghua Gu, Online Adaptive Utilization Control for Real-Time Embedded Multiprocessor Systems, ACM, 2008.

[21] Payam Naghshtabrizi and Jo˜ao P. Hespanha. Analysis of Distributed Control Systems with Shared Communication and Computation Resources, American Control Conference, 2009.

[22] J. Liu, Real-Time Systems: Prentice Hall PTR 2000.

[23] C. Lu, X. Wang, and K. X., "Feedback utilization control in distributed real-time systems with end-to-end tasks," Parallel and Distributed Systems, IEEE Transactions on, vol. 16, no. 6, pp. 550-561, 2005.

[24] CAN Specification, Controller Area Network Specification and Implementation, Robert Bosch GmbH, http://www.semiconductors.bosch.de/pdf/can2spec.pdf, 1991.

[25] The FlexRay Group, FlexRay Communications System Protocol Specification, Version 2.1, http://www.°exray.com/, 2005.

[26] V. Claesson, S. Poledna & J. Soderberg, The XBW Model for Dependable Real-Time Systems, International Conference on Parallel and Distributed Systems (ICPADS), pages 130{138, 1998.

[27] X-by-Wire Project, Brite-EuRam 111 Program, X-By-Wire – Safety Related Fault Tolerant Systems in Vehicles, Final Report, 1998.

[28] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "Survey of recent results in networked control systems," Proc. of IEEE, vol. 95, no. 1, pp. 138–62, Jan. 2007.

[29] P. Naghshtabrizi, "Delay impulsive systems: A framework for modeling networked control systems," Ph.D. dissertation, University of California at Santa Barbara, Sep. 2007.

[30] Renesas M16C, family for microcontrollers Platform: http://www.renesas.com/media/products/.../m16c/M16C_Family_Catalog.pdf

[31] Stephen J. Chapman (2004). MATLAB Programming for Engineers, Third edition. 2004.

[32] Khan, A.A.; Yakzan, A.I.E.; Ali, M.; , "Radio Frequency Identification (RFID) Based Toll Collection System," Third International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), pp.103-107, 26-28 July 2011.

[33] Ali Sharif Ahmadian, Mahdieh Hosseingholi, and Alireza Ejlali, A Control-Theoretic Energy Management for Fault-Tolerant Hard Real-Time Systems, Real-Time Systems Symposium (RTSS), 2011.

[34] S. Ghosh, R. Melhem, and D. Mosse, "Fault-Tolerant Scheduling on a Hard Real-Time Multiprocessor System," in Proc. 8th Int. Symp. Parallel Processing, pp. 775-782, 1994.

[35] Y. Zhang and K. Chakrabarty, "Dynamic Adaptation for Fault Tolerance and Power Management in Embedded Real-Time Systems," ACM Trans. Embedded Computing Systems, vol. 3, no. 2, pp. 336-360, 2004.

[36] Oumair Naseer, Arshad jhumka, Atif Ali Khan, Dependability driven feedback control scheduling for real time embedded systems, The 2012 International Conference on Embedded Systems and Applications (ESA'12). 2012.

[37] Oumair Naseer, Atif Ali Khan, Online adaptive fault tolerant based feedback control scheduling algorithm for multiprocessor embedded systems, Internationl journal of embedded systems and application (IJESA), 2012.

[38] Oumair Naseer, Akeel Shah, Feedback control scheduling for crane control system. IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS) 2013.