Hardware Implementation of an Improved Resource Management Scheme for Fault Tolerant Scheduling of a Multiprocessor System

Sherin Abraham
Amrita Vishwa Vidyapeetham
Coimbatore, India

Sivraj .P
Amrita Vishwa Vidyapeetham
Coimbatore, India

Radhamani Pillay
Amrita Vishwa Vidyapeetham
Coimbatore, India

ABSTRACT

Safety-critical systems have to be fault tolerant and also meet stringent temporal constraints. Various redundancy strategies are built into such mission-critical applications to ensure the overall success of the mission. This paper implements a fault tolerant scheduling scheme on a dual processor system, wherein the redundancy is made at the task level. The system continues to function with graceful degradation under failure conditions. The redundancy management employed in the proposed scheme enhances the performance capability of the system. Based on this approach, the scheme is implemented with hardware simulation using LPC-2148 development boards. This simulation when used for implementing any practical safety-critical application can contribute to efficient utilization of computing resources and can prove to be highly cost effective as the number of processors increase.

General Terms

Fault-tolerance, Resource management, scheduling

Keywords

Redundancy, Multiprocessor, Fault-tolerance, Safety-critical

1. INTRODUCTION

Embedded systems often have significant energy constraints, and many are battery-powered. So, embedded systems use low power consuming processors with small memory size, to minimize costs and energy consumption. Some of the features that safety-critical embedded systems possess are fault-tolerance, temporal deadlines, etc. In safety-critical applications, a compromise in the performance or a fault in the system cannot be tolerated. Fault-tolerant design, also known as fail-safe design, is a design that enables a system to continue operation, possibly at a reduced level (also known as graceful degradation)[1], rather than failing completely, when some part of the system fails. A fault-tolerant system has to tolerate fault like transient, intermittent or permanent hardware faults. In this paper, we assume a permanent failure. A fault-tolerant design has to be such that, in case of faults, at least its minimal functionality should be maintained. One approach to hardware fault recovery is the use of redundancy. Both software and hardware redundancy have been used for fault tolerance. One such approach, *N*-version programming, uses static redundancy in the form of independently written programs (versions) that perform the same functions, and their outputs are voted at special checkpoints. In a hardware redundancy, parallel resources employed like dual redundancy, triple modular redundancy, etc. are expected to make the system fault tolerant. Real-time systems like safety-critical systems have to meet stringent deadlines[4],[5]. Missing any deadlines can be catastrophic.

The microcontroller chosen for the implementation of this fault tolerant multiprocessor system is the LPC 2148. The LPC2148 microcontrollers are based on a 32/16 bit ARM7TDMI-S CPU with real-time emulation and embedded trace support, that combines the microcontroller with embedded high speed flash memory ranging from 32 kB to 512 kB. A 128-bit wide memory interface and unique accelerator architecture enable 32-bit code execution at the maximum clock rate. For critical code size applications, the alternative 16-bit Thumb mode reduces code by more than 30 % with minimal performance penalty. Due to their tiny size and low power consumption, LPC2148 are ideal for applications where miniaturization is a key requirement, such as access control and point-of-sale. A blend of serial communications interfaces ranging from a USB 2.0 Full Speed device, multiple UARTs, SPI, SSP to I2Cs, and on-chip SRAM of 8 kB up to 40 kB, make these devices very well suited for communication gateways and protocol converters, soft modems, voice recognition and low end imaging, providing both large buffer size and high processing power. Various 32-bit timers, single or dual 10-bit ADC(s), 10-bit DAC, PWM channels and 45 fast GPIO lines with up to nine edge or level sensitive external interrupt pins make these microcontrollers particularly suitable for industrial control and medical systems.

2. MOTIVATION

For applications that require very high dependability, the costs for providing high redundancy can be very prohibitive.

Any strategy that can minimize the amount of computing resources leading to significant improvement in terms of reduction of size, weight, volume and power can be highly beneficial in aerospace applications. For example, it can mean decreased payloads in space vehicles.

3. BACKGROUND

Hardware redundancy may be provided in one of the following ways:One for One Redundancy, N + X Redundancy, Load Sharing

One for One Redundancy: Every hardware module will haves a redundant hardware module[6]. The hardware module which performs the jobs under normal conditions is called Active and the redundant unit is called standby.

N + X Redundancy: In the scheme, if N hardware modules are required to perform system functions, the system is configured with N + X hardware modules; typically X is much smaller than N. If any of the N modules fail, one of the X modules takes over its functions. Since health monitoring of N units by X units at all times is not practical, a higher level module monitors the health of N units. If one of the N units fails, it selects one of the X units (It may be noted that one for one is a special case of N + X) [9].
Load Sharing: In this scheme, under zero fault conditions, all the hardware modules that are equipped to perform system functions, share the load. A higher level module performs the load distribution. It also maintains the health status of the hardware units [12]. If one of the load sharing modules fails, the higher level module starts distributing the load among the rest of the units. There is graceful degradation in performance with hardware failure. Our scheme has used a similar strategy but modified.

4. FRAMEWORK

In the traditional dual redundant scheme, the hot standby takes over and becomes active, if the primary unit fails. In the proposed improved scheme, under normal conditions, the critical tasks are allocated in a duplicated manner in both processors while the non-critical tasks are shared between the two processors[2]. The slack available allows for additional flexibility for allocation of optional tasks[11]. If there is a failure, the improved scheme provides two modes, Mode 0 and Mode 1.

Mode 0: All non-critical tasks of the failed system are reassigned and executed in the operating processor and no optional tasks are executed.

Mode 1: Some non-critical tasks and optional tasks in the failed processor are dropped. In our paper we have chosen to implement the Mode 0.

The critical tasks are duplicated in both the processors. This is to ensure that even when one of the processors fail, the most vital functions and those that affect the safety of the users and the device are carried out. In order to maintain the functionality of the system even under failure, the processors have to recognize the working condition of each other continuously. Therefore, we require a system with two serial ports. One port connects to the hardware facilitator and the other port connects to the second processor for mutual health check.

5. IMPLEMENTATION

5.1 Hardware Overview

We implement our dual processor system using a prototype board with NXP LPC2148 ARM7TDMI processor and can be clocked up to 60 MHz. This processor has 40kB of RAM and 512kB of internal flash. We chose the board because it is inexpensive and has hardware interfaces for the components we require. ARM is also a fairly simple architecture. Our prototype device is given in Figure 5.1. LPC2148 offers features like USB 2.0 device, 2xUARTs, RTC, 2x10bit ADCs each with multiple channels, 1xDAC,2xI2C, 1xSPI, 1XSSP, 2x32-bit TIMERS, 6XPWM, FAST I/0 support and WDT. It also supports In System Programming (ISP). It runs on µCOS-II, a highly portable real-time operating system and is expected to achieve very high speed

5.2 System Components

Figure 5.2 illustrates the actual implementation and the other requirements. The memory size required for our scheme is very small. As illustrated, the system consists of three LPC 2148 Processors mutually connected through serial cables. Switches are used to simulate the fault condition. An LED extension board is connected to the hardware facilitator to display a task set. The hardware facilities on a development board, as always, cannot be tailored according to our application specific requirements. The scheme requires that the resources are shared.ie, for example, if it is an automatic door operating system, both the processors have to monitor and operate the same door [8]. The drawback of the development boards is that there is only one connector to access each port. Moreover, the task hardware source, be it an LED or LCD display (these are only some examples that can be interfaced to the LPC2148) has to be connected to both the processors P1 and P2. Hence we go for a hardware facilitator that will act as the common resource to which both P1 and P2 can connect, to control the task hardware resource (LEDs). The LPC-2148 board possesses two serial ports (UART communication) and hence is best for the implementation of the simultaneous communication with the two other boards[10]. It also is portable with µCOS-II which is a real-time operating system.

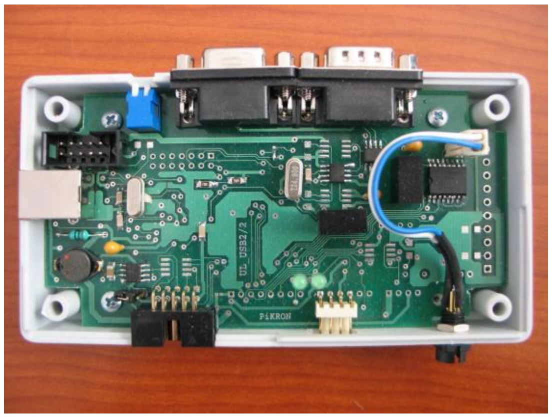
Figure 5.1. Embedded Board Integrating an ARM LPC 2148

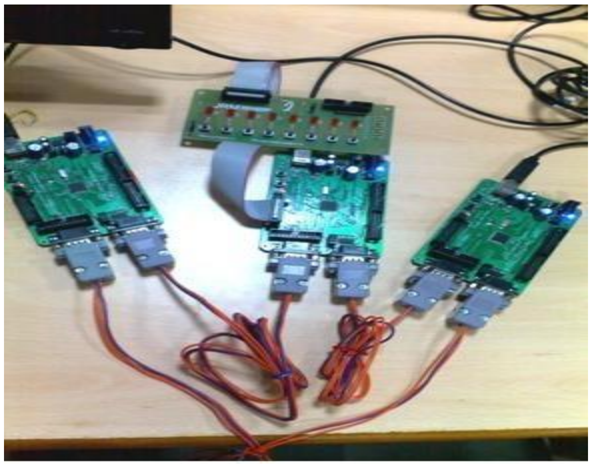
Figure 5. 2 Actual implementation

The block diagram has been shown in Figure 5.3. The processor in the middle is the hardware facilitator , which is connected to the LED extension board. The processor on the left and right were programmed as processor P1 and P2 respectively. The serial port connections serve as a means to do the 'health-check' of the processors.

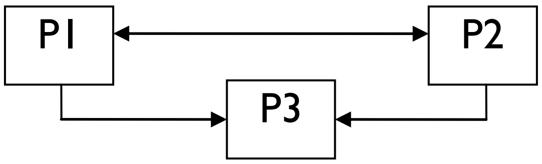

Figure 5.3. Block diagram

5.3 System and Task Model:

The proposed scheme is experimented with simulated task sets. A sample task set generated is given below:

Task.No	C_i(sec)	T_i(sec)	C / NC
1	0.2	0.45	C
2	0.5	0.12	C
3	0.9	2.19	NC
4	1.2	2.37	NC

where C indicates critical tasks and NC indicates non-critical tasks.

The tasks were taken as the blinking of four LEDs of an LED extension board(as the board does not have extra LEDs). The LPC 2148 microcontroller was programmed to utilize the five LEDs of an extension board which was connected to the facilitator so that every time either processor finishes scheduling of a particular task, it can execute the LED tasks. The serial port connections serve as a means to do the 'health-check' of the processors.

- The two processors are running on the µCOS-II operating system.
- The same version of the operating system is working in both the processors.
- The processors have two serial ports (that can be used to check the health status of each other).

µC/OS-II provides many features such as the addition of a fixed-sized memory manager, user definable callouts on task creation, task deletion, task switch and system tick, supports TCB extensions, stack checking, etc. [7]µC/OS-II has developed portable versions for various of ARM CPUs such as ARM7TDMI, ARM9, Strong ARM and etc. It supports ARM kernel based CPUs produced by Atmel, Hynix, Intel, Motorola, Philips, Samsung, Sharp and etc.

5.4 Approach

5.4.1 Assumptions

The two processors are identical and tightly coupled.

1. All tasks considered are independent.
2. All tasks are periodic and critical tasks are non-pre-emptable.

5.4.2 Operating scenarios:

1. Fault free scenario

In fault-free condition, the system offers the operation of both the processors in such a manner that the critical tasks are duplicated and the non-critical tasks are shared in between the two processors P1 and P2. The health status of the processors is mutually checked by each other. The processors schedule all the tasks and when the scheduling is over, an indication in the form of a UART transmission of a character is done to notify the hardware facilitator to execute the task.

2. Fault condition

The functioning status or 'health-status' of the processors is observed by continuous mutual check conducted by the processors on each other. When a fault occurs, the working processor includes some of the tasks from the taskset of the faulty processor into its own and executes the previous and new tasks as a whole taskset. Since the critical tasks were already included in the taskset(critical tasks duplicated), the system continues with safe operation . In this way the functionality of the system is maintained. This is done by continuously monitoring the UART transmission.

6. ALGORITHM IMPLEMENTATION

P indicates one of the processor P1,P2 or P3 where, P3 is the Hardware Facilitator.

Algorithm for processors P1 and P2 **:**

```
Begin
A.  If P = P1            //for processor P1
  •  Mutual Health Check
     If DATA SENT = 'B'
        Schedule tasks of taskset. Transmit a character by
        UART in each task to enable P3 to identify the task
     Else
        Schedule tasks of taskset and some tasks of P2
  •  Execute tasks in P3
B.  If P = P2            //for processor P2
  •  Mutual Health Check
     If DATA SENT = 'A'
        Schedule tasks of taskset. Transmit a character by
        UART in each task to enable P3 to identify the task.
     Else
        Schedule tasks of taskset and some tasks of P2.
  •  Execute tasks in P3.
End
```

Algorithm for the hardware facilitator:

```
Begin
```

- *Check for reception of UART character from both P1 and P2.*
- *Identify the tasks from the UART character received.*
- *Execute the task if the character is received from either P1 or P2.(critical tasks are scheduled in both).*

End

7. RESULTS

For, the total execution time in fault-free condition: **2.8ms,** execution time in faulty condition: **2.1ms** and therefore available time for execution for of more tasks (optional):**0.7ms.** The proposed scheme provides an efficient way to utilize the available resource while providing the required reliability and functionality.

In the implementation given below, execution time of one processor in fault-free condition is 2.1ms which means that there is still scope for accepting more tasks. And hence, in case of occurrence of fault, the working processor, having more time available, can accommodate some of the tasks of the faulty-processor, (upto 0.7ms)and hence maintaining the critical functionality of the system.

8. CONCLUSION AND FUTURE WORK

In this paper we have presented a dual processor system which can tolerate one processor fault. This scheme can be extended to an m-processor system with faults upto the level of m-1 processor failures. The system provides fault tolerance as well as better resource utilization. As future work, performance metrics can be measured to show the system capability. This implementation can also be further extended to schedule aperiodic tasks . This can mean that this strategy can be applied to any generic taskset for a real-time application like an automotive systems.

9. REFERENCES

[1] Avizienis, et al. Dependable Computing and Fault-Tolerant Systems Vol. 1: The Evolution of Fault-Tolerant Computing. Vienna: Springer-Verlag.

[2] Radhamani Pillay, Sasikumar Punnekkat, C Senthilkumar “Optimizing Resources in Real-time Scheduling for Fault Tolerant Processors. INDICON2009,India,2009

[3] Bushnell, M. L., Agrawal, V. D.: *Essential of Electronic Testing for Digital, Memory & Mixed-Signal Circuits*. Springer, 2000, 712 p., ISBN 0-7923-7991-8.

[4] Cheng,A.M.K.:*Real-TimeSystems:Scheduling,Analysis, and Verification*. Wiley, 2002, 552 p., ISBN 0-471-18406-3.

[5] Cottet, F., Delacroix, J., Kaiser, C., Mammeri, Z.: *Scheduling in Real-Time Systems*. John Wiley & Sons, 2002, 266 p., ISBN 0470847662.

[6] Kandasamy N., Hayes, J. P., Murray, B. T.: *Time-Constrained Failure Diagnosis in Distributed Embedded Systems: Application to Actuator Diagnosis*. IEEE Transactions on Parallel and Distributed Systems, 16(3), pp. 258 270.

[7] Micrium: *Micrium.com: Embedded Software Components*. Available on-line at <http://www.micrium.com

[8] J. Nieh and M. Lam. The Design of SMART: A Scheduler

for Multimedia Applications. Technical Report CSL-TR-

96-697, Computer Systems Laboratory, Stanford University,June 1996.

[9] Rupe, D., Kenny, J., R.: *Two Competitive FPGA Methodologies for Run-Time Reconfiguration*. Technology Feature. 4 p., 2008.

[10] Rushby, J.: *A Comparison of Bus Architectures for Safety-Critical Embedded Systems*. NASA/CR-2003-212161 Contractor Report, 63 p., 2003.

[11] Rubel, P., Gillen, M., Loyall, J., Gokhale, A., Balasubramanian, J., Paulos, A., Narasimhan, P., Schantz, R.: *Fault-Tolerant Approaches for Distributed Real-Time and Embedded Systems.* In: Proceedings of Military Communication Conference, 2007, 8 p. ISBN 1 4244-1513-06.

[12] Strnadel, J.: *Testability Analysis and Improvements of Register-Transfer Level Digital Circuits*, In: Computing and Informatics, 25(5), 2006, Bratislava, pp. 441-464, ISSN 1335-9150.