KKU ENGINEERING JOURNAL 2016;43(S3):483-486 Research Article

KKU Engineering Journal

https://www.tci-thaijo.org/index.php/kkuenj/index

Applying fault-tolerance on multi-microcontroller system with RTOS

Chayanan Kuruwasri and Daranee Hormdee*

Embedded System Research and Development Group, Department of Computer Engineering, Faculty of Engineering, Khon Kaen University, Khon Kaen 40002, Thailand.

Received April 2016
Accepted June 2016

Abstract

This paper proposes a design of fault-tolerance technology, combined with a Real-Time Operation System (RTOS), applied to a multi-microcontroller system. The aim of this work is to demonstrate the above issue via an embedded system with a "fault-tolerance" on a FreeRTOS kernel and driven by a multi-microcontroller system. A simple protocol that used to communicate between microcontrollers is USART with the feature of a high-speed baud rate at 10.5 Mbit/s. Sift-out modular redundancy, one of hybrid redundancy techniques in fault-tolerance algorithm has applied in this work. The experiments covered two types of faults; Slave Fault and Master Fault. The results showed that faults occurred in the proposed multi-microcontroller system can be detected and recovered in order to finish the whole assigned process.

Keywords: Fault-tolerance, Sift-out modular redundancy, Multi-microcontroller system, RTOS

1. Introduction

The essential abilities of current embedded systems would be performance and power efficiency. And the 'heart' of an embedded system is microcontroller.

Modern system aims at connecting and coordinating multiple microcontrollers in order to achieve the desired output. Increasing number of microcontrollers has been used these days for various industrial and domestic applications. With an increase in the number of microcontrollers, it becomes important to maintain proper coordination among them. The issue with such systems is that if one microcontroller fails it is likely to affect the working of other connected microcontroller's.

The aim of this work is to demonstrate the above issue via an embedded system with a "fault-tolerance" ability on a Real-Time Operation System (RTOS) kernel [1] and driven by a multi-microcontroller system.

2. Literature reviews

Two concepts, Fault-tolerance and RTOS are described here. Then previous research related to this work are summarized.

2.1. Fault-tolerance

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components, by detecting failures, and isolate defect modules so that the rest of the system can operate correctly. Fault tolerance can be provided with software, or embedded in hardware, or provided by some combination.

Redundancy [2] is a technique that used for the fault-tolerance. It has four types; there are "Hardware Redundancy, Software Redundancy, Information Redundancy and Time Redundancy". This paper uses a hybrid technique [3] in hardware redundancy technique.

A hybrid technique was made of static technique (passive technique) and dynamic technique (active technique). Static technique using a concept call "fault masking". This technique has designed for achieve fault-tolerance system without any action on the part of any system based on the mechanism of voting and provide spares to replace faulty hardware. Dynamic technique has designed to provide a fault-tolerance system by checking the existence of damage and the performance from some action of the failure device and remove from the system based on fault detection, fault location and fault recovery technique.

Figure 1, Sift-out modular redundancy technique is one of a hybrid technique that use in this paper. "When using a sift-out configuration, the system is organized into L identical channels where L is any integer. The channels are synchronized with one another and perform simultaneous operations. Each channel is active as long as it is fault-free. Whenever one of the channels fail, its contribution to the system output ceases. The system becomes an (L - 1) redundancy scheme. Upon the occurrence of a new failure, the process repeats itself.

Sift-out redundancy has a fault tolerance F = L - 2. (L - 2) channels can fail and the module will still operate correctly. When the module is reduced to two channels and one of them fails, the system is unable to detect which one

*Corresponding author. Tel.: +6681 544 6850
Email address: darhor@kku.ac.th; darhor@gmail.com
doi: 10.14456/kkuenj.2016.162

failed. Sift-out is a 2-out-of-L structure, or more emphatically, an L-down-to-two redundancy." [4].

The advantages of sift-out redundancy as show in Figure 1 include the following:

1. Inherent fault detection.
2. Fault-isolation capability, facilitation diagnosis and self-repair.
3. Adjust order of redundancy.
4. Efficient use of hardware.
5. Straight forward implementation.

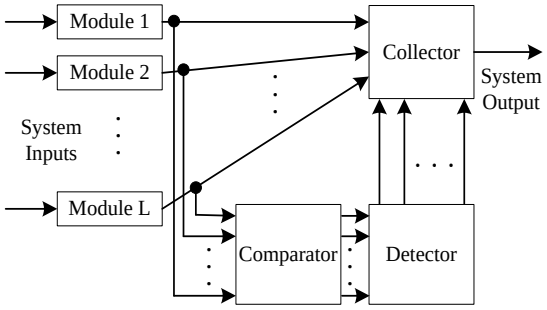

Figure 1 Sift-out Redundancy Model

2.2. RTOS

Real-Time Operation System (RTOS) is an operating system proposed to serve real-time application requests. It can process an income data typically immediately. A key of RTOS is processes that need to be done at a specific time. It is built for multitasking since it can execute two or more tasks concurrently. In fact, CPU only processes one task at a time. The important things in RTOS kernel are Thread and Scheduler. The RTOS has an advance algorithm for scheduling that is Scheduler. It has scheduled to decide at any moment that any threads can be executed and/or switched. The Scheduler is designed to be predictable. Therefore, it is interested by embedded system's designers, because the embedded system's applications require real-time responses to certain events strictly at defined period. It was guaranteed to respond immediately by the prediction of Scheduler.

FreeRTOS [5] was chosen in this work. The reasons behind this decisions include:

1. It is a free real-time operating system kernel, hence its name, aimed at small embedded real-time systems.
2. It is highly portable size, making it possible to run where most (or all) other RTOS will not fit, also with minimal ROM, RAM and processing overhead.
3. It contains a pre-configured example for each port, therefore no need to figure out how to setup a project - just download and compile!
4. Furthermore, It has an ability to allow users to determine the importance of each thread.

Figure 2 illustrates the basic concept of FreeRTOS. The main components are:

a. The **Kernel** is the 'body' and contains all the functions and feature of the RTOS.
b. The **Scheduler** is the 'heart' of the RTOS and decides which thread should run. The scheduler will decide to split the thread according to the task priority.
c. **Threads** ($T_1 \ldots T_i$) are the 'workers' in the RTOS and contain the application code.
d. **Mailboxes** and **Queues** are used to send message between threads.
e. **Events** and **Semaphores** are signaling communications between the different threads.

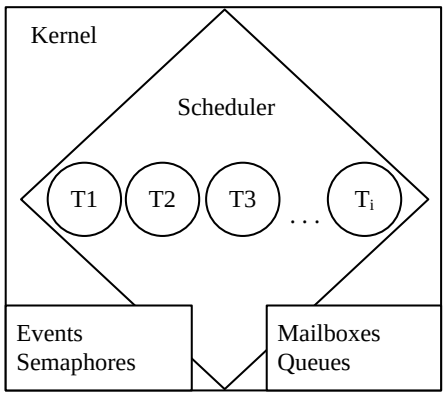

Figure 2 Overall Concept of FreeRTOS

2.3. Related works

Various multi-microcontroller systems have been implemented for the past decade. RTOS is not new to microcontroller system either. Also the attempts to cooperate fault-tolerant techniques into embedded system have been on going.

In 1997, Rennels, et al. [6] have presented a research that implemented fault-tolerance to an embedded computing node by using low-cost commodity microcontrollers combined with software and external logic circuits. Each node of microcontroller can choose a fault-tolerance redundancy technique as duplex, triplex or hybrid by host microcontroller selected. This system aimed to link each node in the distribution network. The final implementation of the fault-tolerance system in microcontroller node uses the same type of microcontroller connected by serial buses. Each node uses microcontroller 2 modules for transient error recovery and fail-safe shutdown upon permanent faults. Using 3 modules to provide uninterrupted operation during a fault or error and 4 modules for applications requiring extended life

In 2010, Chen, et al. [7] introduced double Microcontroller Unit (MCU) parallel control system to obtain high quality, high performance to price of the control system on solar cell lamination machine and improve temperature control precision. They use expert PID control algorithm that make improvement on dynamic and static quality. This system has compare with PLC control system in the control system of lamination machine. It can greatly reduce cost and easier to realize a complex algorithm.

Similarly to a previous work [8] where the tiny but effective FreeRTOS kernel has been applied, this work also explains on how to accommodate a fault-tolerance technique within the system.

3. Overall design

To tolerate hardware physical faults, the channels might be identical, based on the assumption that hardware components fail independently. In this case, there are 2 types of components, "Master" and "Slaves".

The system used low power STM32F4Discovery boards with high-speed baud rate at 10.5 Mbit/s USART interfacing between microcontrollers. The number of total

microcontrollers is L, comprising of one Master and the rest (L-1) are Slaves. The Master first detects all Slaves at the initiation state. Later each process will be assigned broadcast to all Slaves by the Master which will wait for the result from each Slave before assigning more processes until all tasks have been accomplished. Figure 3 exhibits the components in a microcontroller; Master and Slaves along with the details inside each Slaves. All processes have been managed by "Scheduler" with multi-task system of FreeRTOS kernel and the communication task in all microcontroller was designed to transfer data parameters.

Sift-out modular redundancy, one of hybrid redundancy techniques [3] in fault-tolerance algorithm has applied in this work.

4. Experimental results

Using Whetstone Benchmark [9], all experiments have been executed and evaluated on two cases of fault-tolerance; Fault on Slave and Fault on Master.

Per the former case (depicted in Figure 4), The Slave fault test is a random fault for Slave microcontrollers such as process/result fault then recovered and blackout event. This Slave fault test need at least one Slaves must healthy for fault-tolerance algorithm. Faults on Slaves (i.e. incorrect output produced, a fail in task process or unexpected system shutdown) have been generated from a computer. After System initialization period (t_1), Master then assign task out (t_2). At this point all Slaves can process their tasks and return result to Master. However in t_3, Master detects a fault at Slave#1 hence the algorithm fault recovery [10] has been sent off in t_4. Slave#1 can then recover itself and get ready to proceed the remaining task in t_5.

On the other hand, per the latter case, a random fault for Master microcontroller has been tested. All Slaves can detect when Master was fault then all Slaves go to highest priority process is voting algorithm for vote a new Master, after that the Slave who transform to Master resume a previous task then go until the end of work. Fault on Master has been shown in Figure 5. During t_3-t_4, a fault on Master occurred. Each Slave can then detect this via observing no response has been issued back from Master (t_5). Eventually, all Slaves went to fault-tolerance mode. With voting algorithm, a new Master can then be voted (t_6) and the whole system has then been reinitialized in t_7. From this point the new Master sent a signal attempting to recover the old Master (t_8). Once the old Master could recover itself (t_9), it can proceed as a Slave onwards.

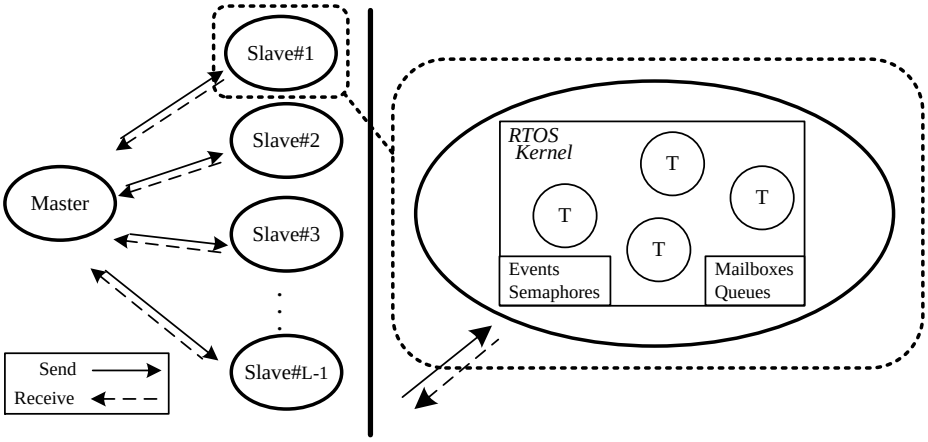

Figure 3 Overall System Design

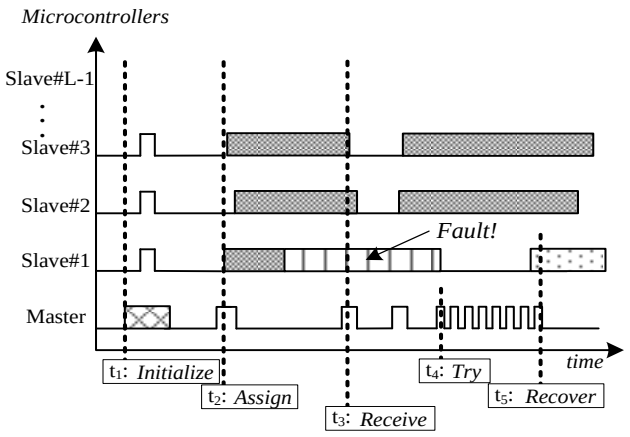

Figure 4 Slave Fault

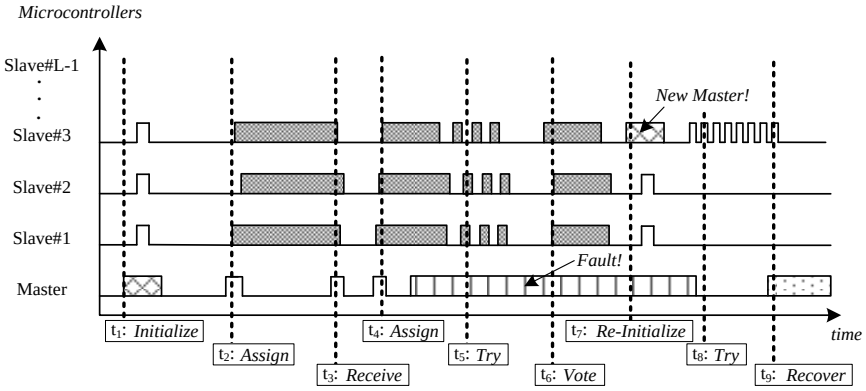

Figure 5 Master Fault

5. Discussion

Multi-microcontroller system in this paper used the FreeRTOS kernel and it has better performance than real-time interrupt (RTI) system because the FreeRTOS kernel has guarantee the chance of the system could be processed in limited time and support multi-tasking ability. USART is a simple communication interface that build in traditional microcontroller and developer can adapt to use in another microcontroller. USART was also managed by FreeRTOS, therefore it could help a developer to develop a program on Multi-microcontroller system easier.

The fault-tolerance is the key here. This enables all processes to work at all the time by themselves. The hybrid technique in hardware redundancy approach provides the ability whether fault-tolerance to be included in this system. As shown in Figure 6, for high efficiency, the shaded box is not present in the system, if fault-tolerance is not included. Otherwise for high availability, fault-tolerance must be included, hence the whole system as shown all in Figure 6.

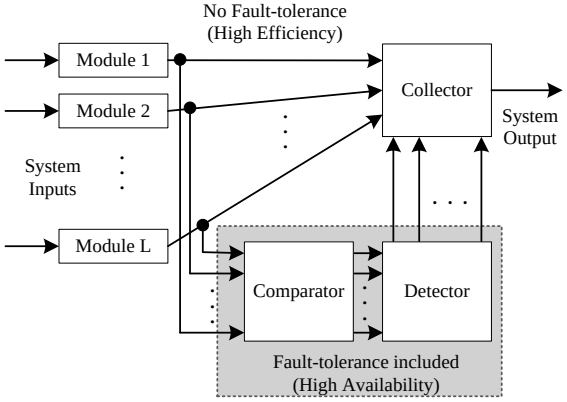

Figure 6 System Modes

Each microcontroller can change itself to Master or Slave if necessary. During a Master fault, one of Slaves need to be promoted as a new Master. A function within each Slave, using n-modular redundancy technique is applied for the voting

6. Conclusions

This paper has demonstrated on how to enhance processing performance of an embedded system by organizing in multi-microcontroller system style with a hardware redundancy technique of fault-tolerance combined with a Real-Time Operation System (RTOS). All the faults, both Slave and Master faults, can be dealt nicely, yielding the whole system to continue working. The experimental results has also illustrated the ability of the entire system to continue process steadily until all tasks have been processed by FreeRTOS kernel.

7. References

[1] Ramezani R, Sedaghat Y. An overview of fault tolerance techniques for real-time operating systems. 3rd International Conference on Computer and Knowledge Engineering (ICCKE 2013); 2013 Oct 31-Nov 1; Mashhad, Iran. IEEE. p. 1-6.

[2] Shooman ML. Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design. USA: John Wiley & Sons; 2003.

[3] Kopetz H. Real-Time Systems: Design Principles for Distributed Embedded Applications. USA: Springer Science & Business Media; 2011.

[4] Sousa PTD, Mathur FP. Sift-Out Modular Redundancy. IEEE Transactions on Computers 1978;C-27(7):624-7.

[5] FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions [Internet]. 2010 [cited 2016 Mar 20]. Available from: http://www.freertos.org/.

[6] Rennels DA, Caldwell DW, Hwang R, Mesarina M. A fault-tolerant embedded microcontroller testbed. Pacific Rim International Symposium on Fault-Tolerant Systems; 1997 Dec 15-16; Taipei, Taiwan. USA: IEEE; 1997. p. 7-14.

[7] Chen L, Shi L, Liu R, Ma S, Liu S, Zheng J, et al. Design of Control System of Solar Cell Lamination Machine Based on Double MCU. International Conference on Measuring Technology and Mechatronics Automation (ICMTMA); 2010 Mar 13-14; Changsha, China. USA: IEEE; 2010. p. 1046-1048.

[8] Kuruwasri C, Hormdee D. Design of Multi-Microcontroller System Based on STM32F4Discovery Boards with FreeRTOS. International Conference on Embedded Systems and Intelligent Technology (ICESIT); 2014 Sept 18-20; Gwangju, Korea.

[9] Whetstone Benchmark History and Results [Internet]. 2014 [cited 2016 Mar 20]. Available from: http://www.roylongbottom.org.uk/whetstone.htm.

[10] Rennels DA, Hwang R. Recovery in fault-tolerant distributed microcontrollers. In: International Conference on Dependable Systems and Networks (DSN 2001); 2001 Jul 1-4; Goteborg, Sweden. USA: IEEE; 2001. p. 475-80