Transactions of the Korean Nuclear Society Spring Meeting
Chuncheon, Korea, May 25-26 2006

Interface Testing for RTOS System Tasks based on the Run-Time Monitoring

Ahyoung Sung,a and Byoungju Choi,a
a Dept. of Computer Science & Engg, Ewha University, Seoul, Korea
aysung@ewhain.net, bjchoi@ewha.ac.kr

1. Introduction

Safety critical embedded system requires high dependability of not only hardware but also software. It is intricate to modify embedded software once embedded. Therefore, it is necessary to have rigorous regulations to assure the quality of safety critical embedded software.

IEEE V&V (Verification and Validation) process [1] is recommended for software dependability, but a more quantitative evaluation method like software testing is necessary. In case of safety critical embedded software, it is essential to have a test that reflects unique features of the target hardware and its operating system.

The safety grade PLC (Programmable Logic Controller) is a safety critical embedded system where hardware and software are tightly coupled. The PLC has *HdS* (Hardware dependent Software) [2] and it is tightly coupled with RTOS (Real Time Operating System). Especially, system tasks that are tightly coupled with target hardware and RTOS kernel have large influence on the dependability of the entire PLC.

Therefore, interface testing for system tasks that reflects the features of target hardware and RTOS kernel becomes the core of the PLC integration test. Here, we define interfaces as overlapped parts between two different layers on the system architecture.

In this paper, we identify interfaces for system tasks and apply the identified interfaces to the safety grade PLC. Finally, we show the test results through the empirical study.

2. Interfaces of the Safety Grade PLC

The safety grade PLC is based on the TI TMS320C32 DSP (Digital Signal Processor) board [3]. It downloads, executes, and controls the application programs using the RS-232C communication [4].

Figure 1 represents the architecture of the safety grade PLC. As shown in Figure 1, hardware is physical devices on the TMS320C32 DSP board and software is *HdS* of HAL (Hardware Abstraction Layer) [5], RTOS kernel [6], and system tasks.

The *HdS* of the HAL is to handle the context switch, to manage the ISR (Interrupt Service Routine) vectors, and to manage the critical regions. The HAL is implemented to access and to control target hardware directly.

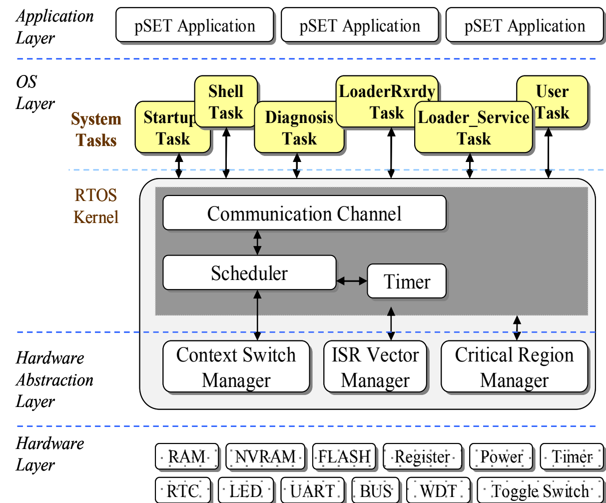

Figure 1. Architecture of the safety grade PLC

The target software focused in this paper is system tasks. The system tasks are meaningful and executable software that occupies CPU. In case of the safety grade PLC, there are five system tasks including the Startup Task, the Shell Task, the Diagnosis Task, the LoaderRxrdy Task, and the Loader_Service Task [4].

Since system tasks associated with the kernel are loaded and executed on the target board, there are two types of interface as follows:

Hardware interface based on *HdS* and the kernel interface based on RTOS API (Application Program Interface).

■ Hardware Interface

The system tasks read and write values in RAM or Register to control the target hardware. Therefore, the interface between the system tasks and the target hardware is defined as the location where software reads and writes values to control.

Register, RAM, FLASH and NVRAM have direct interface, while physical devices like LED and Timer have indirect interface through RAM or Register by using in-line assembly or global variables.

■ Kernel Interface

System tasks have kernel interface using the RTOS API. RTOS Kernel has services including the task management, the inter-task communication based on semaphore, queue, mailbox, Interrupt Handling, Time Management, and Memory Management [6].

Taking 'Task Management' and 'Inter Task Communication' as examples, APIs such as

OSTaskCreate() of ‘Task Management’, OSMboxPsot(), OSSemPost(), and OSQPost() access data structures such as TCB (Task Control Block), ECB (Event Control Block), and QCB (Queue Control Block) in the kernel and their corresponding RAM area.

3. Interface Testing for RTOS System Tasks

We perform interface testing for the system tasks based on run-time monitoring. There are synchronous and asynchronous monitoring techniques for the run-time environment [7]. Synchronous monitoring stops the current program and examines the particular point in the program. Asynchronous monitoring examines the entire system by separate daemons during the execution of the real-time tasks. However, since the daemon is executed periodically as a separate task, asynchronous monitoring may provide imprecise information.

In this paper, we perform interface testing based on the synchronous monitoring. We set break points on the particular location in the source code and observe the monitored results. The break points represent the mapped location of hardware interface and kernel interface in the source code. We determine ‘pass’ if the monitored results satisfy the expected output, and ‘fail’ otherwise.

The core of interface testing is to select the break points on the source code and to determine the symbols to be monitored. As shown in Table 1, the test cases of interface testing have input and expected output. We define the break points as input and the monitored symbols as expected output.

Table 1. Format of Test Cases

	Input (Break Points)	Expected Output (symbols to be monitored)
HW Interface	Inline Assembly or global variable	RAM address corresponds with in-line assembly or the global variable
Kernel Interface	API of Task Management	TCB
	API of Inter Task Communication	TCB, ECB, and QCB (only for Queue management)
	API of Interrupt Handling	TCB, ISR_ID, and Registers
	API of Time Management	TASK_ID_DELAY_TICK
	API of Memory Management	TCB and its RAM address

4. Interface Test Results and Conclusive Remarks

As an empirical study, we performed the interface testing for system tasks including the Startup Task, the Shell Task, the Diagnosis Task, the LoaderRxrdy Task, and the Loader_Service Task in Figure 1.

The interface testing is performed on the Code Composer [8] that supports the TMS320C32 DSP board and RTOS kernel. To monitor the real results, we use the ‘Watch window’ menu and the ‘View memory map’ menu of the Code Composer.

Table 2 demonstrates the interface test results for the system tasks in Figure 1. As shown in Table 2, we selected the total number of 212 test cases. To test the hardware interface and kernel interface, we selected 162 test cases and 50 test cases, respectively. Through this empirical study, we didn’t detect faults in kernel interface while we detected 38 faults in hardware interface.

Table 2. Results of Interface Testing

System Tasks in Figure 1	Test Case # (Detected Faults #)		Total
	HW Interface	Kernel Interface	
Startup Task	52 (16)	2 9 (0)	81 (16)
Shell Task	23 (8)	2 (0)	25 (8)
Diagnosis Task	69 (8)	1 (0)	70 (8)
LoaderRxrdy Task	1 (1)	12 (0)	13 (1)
Loader_Service Task	17 (5)	6 (0)	23 (5)
Total	162 (38)	50 (0)	212 (38)

Since system tasks associated with the kernel are loaded and executed on the target board, they have hardware interface and kernel interface. In this paper, we proposed interface testing based on the synchronous monitoring and applied the interface testing to the system tasks in the safety grade PLC.

In the future, we are going to perform empirical studies that show the excellence of the proposed interface testing technique.

REFERENCES

[1] IEEE Std. 1012-1998, “IEEE Standard for Software Verification and Validation Plan”, IEEE, 1998.

[2] S. Yoo and A.A. Jerraya, “Introduction to Hardware Abstraction Layers for SoC,” in Proc. of Design, Automation and Test in Europe Conf. and Exhibition (DATE), IEEE, 2003, pp.336~337; 2003.

[3] TMS320C32 Digital Signal Processor available in http://www.ti.com/, Texas Instrument, 1998.

[4] KNICS-PLC-SDS331-01, Software Design Specification for the PLC Processor Module, KAERI, 2004.

[5] A. Jerraya, W.Wolf, “Hardware/Software Interface Codesign for Embedded Systems”, IEEE Computer, pp63~69, Feb., 2005.

[6] J.J Labrosse, MicroC/OS-II, The Real-Time Kernel, CMP Books, 1999.

[7] S.E.Chodrow, F.Jahnian, and M.Donner, “Run-Time Monitoring of Real-Time Systems”, in the Proc. of Run-Time Systems Symposium (RTSS), IEEE, pp.103-112, 1991.

[8] SPRU296, Code Composer User’s Guide, Texas Instrument, 1999.