Pak. J. Biotechnol. Vol. 13 (special issue on Innovations in information Embedded and Communication Systems) Pp. 25-28 (2016)

SOFTWARE BASED TESTING OF MICROCONTROLLER USING REAL TIME OPERATING SYSTEM

V.Prabhu* and K.T.Ilayaraja

*M.E Embedded System, Department of Electronics and Communication Engineering
Sathyabama University, Chennai prabhu27390@gmail.com
Department of Electronics and Communication Engineering, Sathyabama University, Chennai
ilayarajaa.tsi@gmail.com

ABSTRACT

Microcontrollers have made their way into embedded applications such as automobile electronics, industrial automation and peripherals for computer systems. For the microcontroller to perform efficiently it is necessary for a test engineer to check the proper operation of a product. Manufacturing test is one of the most difficult tasks in the semiconductor industry. This paper presents a novel approach for testing a microcontroller with software based tests using RTOS (Real Time Operating System). The methodology exploits existing manufacturing test programs designed for software based self test and enhances them by using a new approach with RTOS. Experimental results are reported in this paper showing the reliability and effectiveness of the method in detecting faults in a microcontroller.

Key words: Built in self test, debugging, diagnosis, testing, software based self test

I. INTRODUCTION

Testing is used to measure the quality of a manufactured chip before shipping it to the customer. The microcontroller can be tested based on its functionality and structural design. Functional testing is used to check the operation of the microcontroller as per design by generating test patterns which excites the instruction sets and observes the response. Structural testing is used to check whether the chip is manufactured as per design by testing the circuit elements and their interconnections. Structural testing is used to locate the physical faults in a chip. Accurate location of faults is useful for improving the manufacturing process. In manufacturing most of the faults which occur are due to fabrication errors affecting the interconnections between components. The testing methods applied to detect the faults must be cost efficient and should not exceed the manufacturing cost. Testing can be done by software or hardware methods or a combination of both. Automatic Test Equipments (ATE) can be used to apply a test pattern and store the response back in their memory [1]. These external hardware equipments are more expensive and hence are not the ideal method for testing. Software tools such as simulators and monitors do not require external hardware equipments. Incircuit emulators and Incircuit debuggers integrates the use of both hardware and software methods in testing. The emerging method of self testing allows the microcontroller to test itself by using pseudorandom pattern generators, linear feedback shift registers or by making use of the processor's instruction set architecture [2].

Software based self test (SBST) is a low cost testing methodology which utilizes the resources of the device under test and reduces the need for ATE. The software based test using Real Time Operating System (RTOS) can be seen as an advancement of SBST [3]. The SBST process can be performed at speed without using external hardware equipment and they do not require any hardware structure to be included in the circuit. As a case study, results obtained on a widely used ARM (Advanced Risc Machine) based microcontroller demonstrate the effectiveness of the proposed strategy in terms of detecting fault. In this paper Section II provides a background in software based self test and RTOS. Section III details the proposed approach. Section IV presents some experimental results and Section V provides the conclusion of this paper.

II. BACKGROUND REVIEW

A. Software Based Self Test

The key advantage of SBST is that it uses on-chip programmable resources to run programs that test the processor itself. The microcontroller generates test patterns using its own instruction set, eliminating the need for additional test specific hardware. In the first step of an SBST application the test code is downloaded into a microcontroller. Then the processing unit executes test programs at its actual speed. Finally the test responses are stored in the data memory to indicate the faults in the test unit [4].

The other advantages of SBST which make them suitable for microcontroller testing are:

1. It is obtrusive and does not need any extra hardware. It also consumes power efficiently in normal operation mode.
2. It allows the test to be performed at the processor's actual speed, detecting faults that are not detectable at lower frequencies.
3. It allows programs from manufacturing test to be reused in the field throughout the lifetime of the product.

B. Real Time Operating System

Traditionally developers of small embedded applications had to write all the code that runs on the microcontroller, which is in the form of interrupt service routine triggered by an interrupt. Now with the latest advancements in microcontrollers we have low cost, high

performance devices with large amount of internal memory. This makes it possible to introduce more advanced software development techniques in programming. Introducing a Real Time Operating System (RTOS) into test programs has its advantages. With an RTOS, all functional blocks in the program are developed as tasks and are scheduled by the kernel. This provides a detailed design analysis of the final program structure at the beginning of the development [5]. Each of the program tasks can be developed and tested separately before integration into the complete system. Each RTOS task is then easier to document and reuse.

III. PROPOSED METHODOLOGY

The proposed methodology has an effective set of test programs, taking advantage of an initial test set of the SBST programs devised for processor manufacturing testing. The RTOS used is RTX from Keil and the ARM processor based microcontroller used is LPC2148. A new project is created in Keil IDE (Integrated Development Environment) by selecting the microcontroller from the device database. This step is necessary to configure the right complier and debugger. The configuration settings for RTOS like time slice for each task and stack size are modified through the RTX_Config.c file which is added to the project. A total of six tasks are created by assigning individual task ID's. They are executed by cooperative multitasking supported by the scheduler. These tasks are the same as C functions with the exception that they are executed in an endless loop. The scheduler allots time slices to each task which by cooperative multitasking appears to be executing simultaneously.

The main function in the program is used to start the RTOS and also initializes the external interrupt, peripherals such as ADC (Analog to Digital Converter), LCD (Liquid Crystal Display) and serial interfaces UART (Universal Asynchronous Receiver Transmitter) and I2C (Inter-IC). After reset the microcontroller executes the main function and the first task, which is assigned with default priority. The first task is used to create the other five tasks. All tasks are assigned with the same priority. Since all tasks have equal priority they are allotted run time in a round robin fashion. However when a task becomes ready to run with a high priority the scheduler halts the current task and executes the task with high priority. After creating all the tasks the first task deletes itself in order to reduce context switching overhead. Each task checks for an event flag which is use to trigger the task corresponding to an interrupt. The interrupt code is executed as tasks within the RTOS**.**

The microcontroller is connected with a PC (Personal Computer) through UART0 port. The user communication is established through the tera term program available in the PC. This program is an open source terminal emulator which supports serial port connections. The individual flags are enabled by entering a number through the UART0 serial interface. The hex file generated from the Keil IDE is downloaded to the microcontroller using flash magic tool available in the PC. The flash magic tool is configured with the required baud rate before programming the microcontroller. The functions of individual task are explained separately. The I2C based Electrically Erasable Programmable Read Only Memory (EEPROM) is tested in the main function itself by writing consequent numbers to each address of the memory and verifying it by reading back the values. This confirms that each location in the memory is working properly. The result of the memory test is displayed on the LCD as pass or fail.

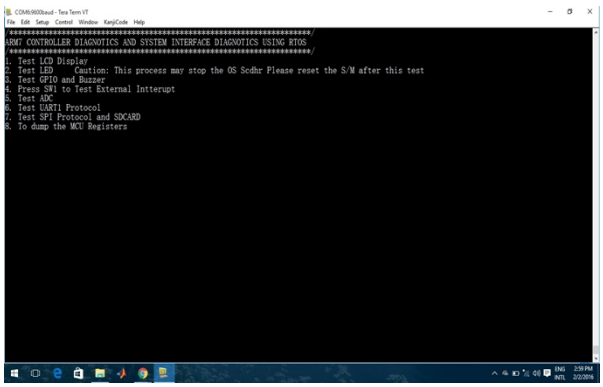

Figure.1 Tera Term application through which the user interacts with the microcontroller

Figure.2 I2C Test result displayed in LCD

A. Task 2

This task performs the testing of special function registers and Test LED's present on the chip. It checks for the display entry flag, register dumping flag and test led flag. The display entry flag is set upon reset of the microcontroller and it displays the user interactive command line interface on the hyper terminal program in PC. The command line interface allows the user to enter a number which triggers a specific task by setting the appropriate event flag. The number 8 is used to set the register dump flag which triggers the task to dump the values of the UART, I2C, ADC, SPI and Interrupt registers on the display. Comparing with the default values to be present in the registers confirms the working of storage locations in memory. The value present in UART0 register is the number currently pressed by the user. These two functions are performed inside the same task by lowering the priority of the task after sending the text to the display and passing the execution to another task. Thereby when the register dump flag is enabled by

pressing 8 the same task gets executed and dumps the values of registers. The number 3 is used to check the test LED's connected through SPI serial interface. The same process of low prioritizing the task and passing the execution is repeated after every function. By combining functions which does not interfere with each other in the same task reduces the number of tasks to be created and thereby reducing the context switching overhead.

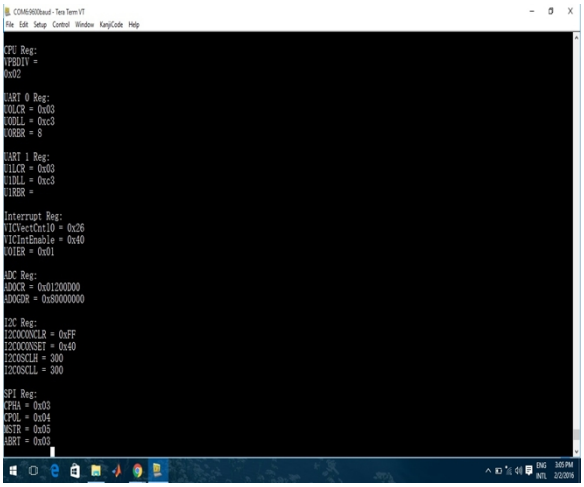

Figure.3 Dumped register values

B. *Task 3*

This task has two functions to test the LCD integrated with the microcontroller and UART1 interface. The function of the LCD is tested by generating the commands and sending the string to be displayed. Although it is not necessary to test the LCD separately as it displays some string once the device is powered on, this function can be used for microcontrollers having more than one LCD. The UART1 serial interface is tested by connecting to a PC through UART1 port and sending some data to be displayed. The data is sent as per the UART protocol.

C. *Task 4*

This task also contains two functions, to test the buzzer integrated with the microcontroller and the external interrupt. These two functions are performed by turning on the buzzer for a particular time interval and by turning it on when the external push button is pressed.

D. *Task 5*

This task is used to test whether a Secure Digital (SD) card is detected or not. The SD card is accessed through SPI interface. By initializing the card through SPI protocol the status of the card is verified and the result is displayed on the LCD.

E. *Task 6*

This task is used to test the ADC peripheral in the microcontroller. These converters act as an interface between sensors connected to the microcontroller and the processing unit by encoding continuous analog signals into digital values. The value of the ADC is changed manually through a potentiometer provided in the microcontroller. The analog value of the ADC is read, converted into digital value and displayed in the LCD. These values are compared with the expected range of values thereby verifying the ADC.

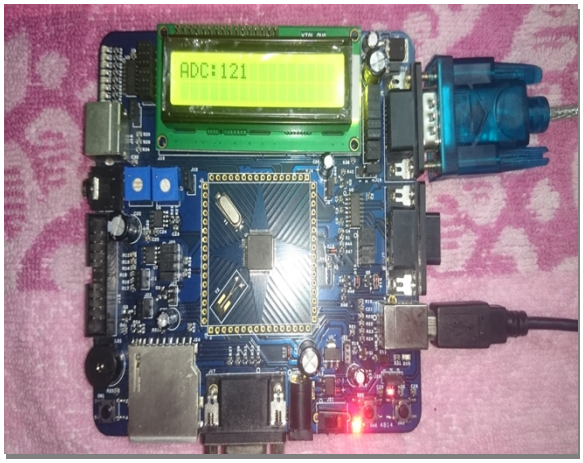

Figure.4 ADC value displayed in LCD

IV. EXPERIMENTAL RESULTS

The proposed method was tested on LPC2148, a 32 bit ARM7TDMI-S microcontroller. The structural faults in the device were detected efficiently and displayed to the user. The tests were performed with minimum interrupt latency and context switching overhead. This method of testing shows that test programs for complex microcontrollers can be executed in real time with each function getting ensured time slices to execute. This method can also be enhanced by modifying the tasks to be executed in a round robin manner or preemptive manner. Thus software based testing using RTOS has shown that it is more effective than the traditional testing methods.

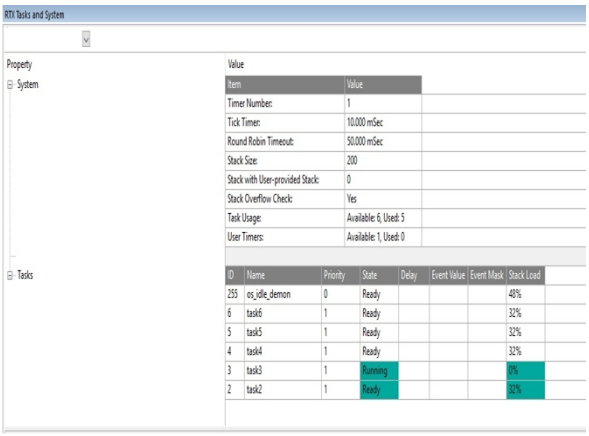

Figure.5Task status and memory utilization

V. CONCLUSION AND FUTURE SCOPE

In this paper, we presented a software based test method which uses RTOS to test the structural faults in a microcontroller. This method significantly reduces the interrupt latency and requires only minimum manual intervention. The usage of RTOS significantly reduces the code complexity and increases reliability. The

method has been experimentally validated on an ARM based microcontroller LPC2148. The obtained results clearly show the advantages over the traditional methods. As future work, we plan to implement this method to execute online as a part of device function.

REFERENCES

[1] Wen, C.H.P., L.C. Wang and K.T. Cheng, Simulation-Based Functional Test Generation for Embedded Processors, IEEE Trans. Computers **55**(11): 1335-1343 (2006).

[2] Shen, J. and J.A. Abraham, Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation, *Proc. Int'l Test Conf. (ITC 98), IEEE CS Press Pp.* 990-999 (1998).

[3] Thatte,S and J. Abraham, Test generation for microprocessors, *IEEE Trans. Comput.* C-**29**(6): 429–441 (1980).

[4] Mihalis Psarakis, Dimitris Gizopoulos, Ernesto Sanchez and Matteo Sonza Reorda, Microprocessor software based self testing, *IEEE CS* Pp. 2-14 (2010).

[5] Tomiyama, H., S.Chikada, S.Honda and H.Takada, An RTOS based approach to design and validation of embedded systems, *IEEE VLSI Test Symp: Pp.* 185-187 (2005).

[6] Manjunath T.N., T. D. Sunil, M. Z Kurian and Imran Rasheed, Design and Development of JTAG for Trace and Debug of Controller and Implementation on FPGA, *(IJARCET) Volume 3 Issue 4, (2014)***.**

[7] Acero, J., D. Navarro, L.A. Barragán, I. Garde, J.-I. Artigas and J.-M. Burdío, "FPGA-based power measuring for induction heating appliances using sigma-delta A/D conversion", IEEE Trans. Ind. Electron **54**(4): 1843–1852 (2007).

[8] Parvathala, P., K. Maneparambil, and W. Lindsay, FRITS: A Microprocessor Functional BIST Method, *Proc. Int'l Test Conf. (ITC 02), IEEE CS Press Pp.* 590-598 (2002).

[9] Gurumurthy, S., S. Vasudevan and J.A. Abraham, ''Automatic Generation of Instruction Sequences Targeting Hard-to-Detect Structural Faults in a Processor, *Proc. Int'l Test Conf. (ITC 06), vol. 2, IEEE CS Press Pp.* 776-784 (2006).

[10] Bayraktaroglu, I., J. Hunt and D. Watkins, Cache Resident Functional Microprocessor Testing: Avoiding High Speed IO Issues,' *Proc. Int'l Test. Conf. (ITC 06), IEEE CS Press Pp.* 769-775 (2006).

[11] Batcher,K. and C.Papachristou, Instruction Randomization Self Test for Processor Cores, *Proc. 17th IEEE VLSI Test Symp. (VTS 99), IEEE CS Press Pp.* 34-40 (1999).

[12] Lai, W.C. and K.-T. Cheng, Instruction-Level DFT for Testing Processor and IP Cores in System-on-a Chip, *Proc. 38th Design Automation Conf. (DAC 01), ACM Press Pp.* 59-64 (2001).

[13] Nakazato, M., Design for Testability of Software-Based Self-Test for Processors, *Proc. 15th Asian Test Symp*. (ATS 06), *IEEE CS Press Pp.* 375-380 (2006).

[14] Christou, K., A Novel SBST Generation Technique for Path-Delay Faults in Microprocessors Exploiting Gate- and RT-Level Descriptions, *Proc. 26th IEEE VLSI Test Symp. (VTS 08), IEEE CS Press Pp.* 389-394 (2008).

[15] Chen, L. and S. Dey, Software-based diagnosis for processors, *Proc. IEEE/ACM Des. Autom. Conf. Pp.* 259–262 (2002).

[16] Bernardi, P. *et al.*, An effective technique for minimizing the cost of processor software-based diagnosis in SoCs, *Proc. IEEE Conf. Des. Autom. Test Eur. Pp.* 412–417 (2006).

[17] Corno, F. *et al.*, Automatic test program generation: A case study. *IEEE Des. Test Comput.* **21**(2): 102–109, Mar.–Apr.(2004).

[18] Bernardi, P. *et al.*, Using infrastructure IPs to support SW-based self test of processor cores, Proc. IEEE Int. Workshop Microprocess. Test *Verification Pp.* 22–27 (2004).

[19] Sánchez, E. *et al.*, On the transformation of manufacturing test sets into on-line test sets for microprocessors, *Proc. IEEE Symp. Defect Fault Tolerance VLSI Syst. Pp.* 494–504 (2005).

[20] Monmasson, E. and M. N. Cirstea, FPGA Design Methodology for Industrial Control Systems: A Review. IEEE Trans on Industrial Electronics **54**(5): 1824-1842 (2007).

[21] Naouar, M.W., E. Monmasson, A. A. Naassani, I. Slama-Belkhodja and N. Patin, FPGA-Based Current Controllers for AC Machine Drives—A Review. IEEE Trans on Industrial Electronics **54**(4): 1907- 1925 (2007).

[22] Berto, S., A. Paccagnella, M. Ceschia, S. Bolognani and M. Zigliotto, Potentials and pitfalls of FPGA application in inverter drives—A case study, *Proc. IEEE ICIT, Maribor, Slovenia Pp.* 500–505 (2003).

[23] Paolo Bernardi, Edgar Ernesto Sánchez Sánchez, Massimiliano Schillaci, Giovanni Squillero, and Matteo Sonza Reorda, An Effective Technique for the Automatic Generation of Diagnosis-Oriented Programs for Processor Cores. *IEEE Trans. Computers* **27***(3):* 1-5 (2008).