Military Technical College
Kobry El-Kobbah,
Cairo, Egypt

8th International Conference
on Electrical Engineering
ICEENG 2012

Design for Testability Technique for Microcontroller

By

Sherif I. Morsy* Mohamed H. El-Mahlawy* Gouda I. Mohamed*

Abstract:

Testing of embedded system including microcontroller is difficult task with external Automatic Test Equipment (ATE). Therefore, empowering the microcontroller to test itself as software-based self-testing (SBST) looks the suitable solution like the microprocessor testing. Practically, the SBST is not suitable for microcontroller testing. It utilizes large space area in the program memory inside the microcontroller that has limited space area in the available memory. Also, it cannot test all microcontroller internal modules and when it test internal modules it cannot make sure that the General Purpose Input Output (GPIO) of the microcontroller work probably without using external ATE. So the Design for Testability (DFT) methodology that uses Instruction Set Architecture (ISA) of the microcontroller family to generate test subroutines and for the Test Pattern Generator (TPG) and part of the Built-In Self-Test (BIST) control unit and uses the external ATE for the other part of the BIST control unit and for the test response compaction (TRC) and evaluation.

This paper introduces a hybrid testing methodology that combines both SBST and hardware-based self-test (HBST) for microcontroller testing as an efficient DFT methodology. It introduces for either in the field or as part of a production test of a microcontroller as an example of the system of chip (SoC). This DFT methodology is based on divide and conquer algorithm and requires knowledge of the ISA of the microcontroller to test not only the embedded processor found in microcontroller but also test other peripherals found in it using brute force technique. The comparison between the SBST and the presented hybrid methodology is based on memory utilization, number of clock cycles that was taken to complete each test and the number of modules that can be tested using each of them. Experimental results indicate that the presented methodology is superior in memory utilization, test time and can test all microcontroller modules for both 18F4X2 and 16F87X families.

Keywords: Design-for-Tetability (DFT), Built-In Self-Test (BIST)

* Egyptian Armed Forces

1. Introduction:

Almost every complex SoC contains at least one embedded processor. Such processor is surrounded by memory of various sizes used for code and data storage and other peripherals. The complexity of SoC designs consisting of deeply embedded cores with poor accessibility makes their testing process a difficult task. Additionally, the increasing gap between the operating frequencies of external ATE and the operating frequencies of SoC lead to the escape of failures that may be detected only when testing is performed in the actual speed of the IC. The transfer of the SoC test task from an external ATE to an internal built-in self-test (BIST) mechanism provides significant advantages not only for processor but also for other peripherals found on the SoC [1]. Self-test methodologies can be executed either using HBST techniques or SBST techniques. In HBST methodology, special parts of the circuit are used for Test Pattern Generation (TPG), Test Response Compactor (TRC), and BIST controller. In this case, the extra circuit area may be significant, but the most important is the possibility of significant performance loss due to the introduction of extra logic in the critical paths of the circuit. Recent applications of hardware-based commercial logic BIST techniques in large industrial designs and microprocessors [2 - 4] revealed that extensive design changes have to be performed which has a negative impact in the circuit area, performance and power consumption of the SoC.
SBST methodologies for embedded processor cores have the advantage that they utilize the processor functionality and instruction set for both TPG and TRC and thus do not add hardware or performance overheads in the optimized design [5]. SBST achieves high fault coverage without system modifications. The processor executes the SBST program residing in the memory of the SoC (e.g., in a flash memory) at its actual speed (at-speed testing) and very small area, performance or power consumption overheads are induced for the embedded system. So it's is better than HBST for embedded systems testing. The SBST approaches can be classified in two different categories. The first category includes approaches that have a high level of abstraction and are functional in nature [6 - 8]. The second category includes the SBST approaches, which are structural in nature and require structural fault driven test development [4, 9 - 12].
The increasing heterogeneity and programmability associated with the microcontroller architecture together with the rapidly increasing operating frequencies and technology changes are demanding fundamental changes in VLSI testing. The test application using external ATE only poses challenges. Unlike hardware-based self-testing, software-based testing applies tests in the normal operational mode of the circuit. The key idea of SBST is to exploit on-chip programmable resources to run normal programs that test the microcontroller itself. Moreover, software instructions has the ability of guiding the test patterns through the microcontroller using different testing techniques, they have detected fault coverage percentage using fault simulator which is very expensive.

In this paper, the hybrid methodology between SBST and HBST for testing the microcontroller is presented. It is based on a divide and conquers approach, microcontroller components and their corresponding component operations are identified. The knowledge of the ISA of each family constructs a test subroutine that excite all operation for each individual module in this family including the memory, exhaustive test patterns (based on brute force algorithm) are generated targeting structural faults of individual microcontroller modules so there no need for fault simulation.
This paper is organized in five sections. This section gives an introduction to the previous work in this field. Section 2 describes the presented methodology development phases. Section 3 summarizes the microcontroller test modules. Section 4 discusses the experimental results on two different families of microcontrollers then it concludes the paper.

2. Presented DFT methodology

Novices in electronics usually think that the microcontroller is the same as the micro-processor. That's not true. They differ from each other in many ways. The first and most important difference in favor of the microcontroller is its functionality. The microprocessor may be used other components. Memory comes first to be added. The microprocessor is considered a powerful computing machine; it is not adjusted to communicating to peripheral environment. In order to enable the microprocessor to communicate with peripheral environment, special circuits must be used. On the other hand, the microcontroller is designed to include all in one as shown in Figure(1). No other specialized external components are needed for its application because all necessary circuits which otherwise belong to peripherals are already built in it. It saves time and space needed to design a device.

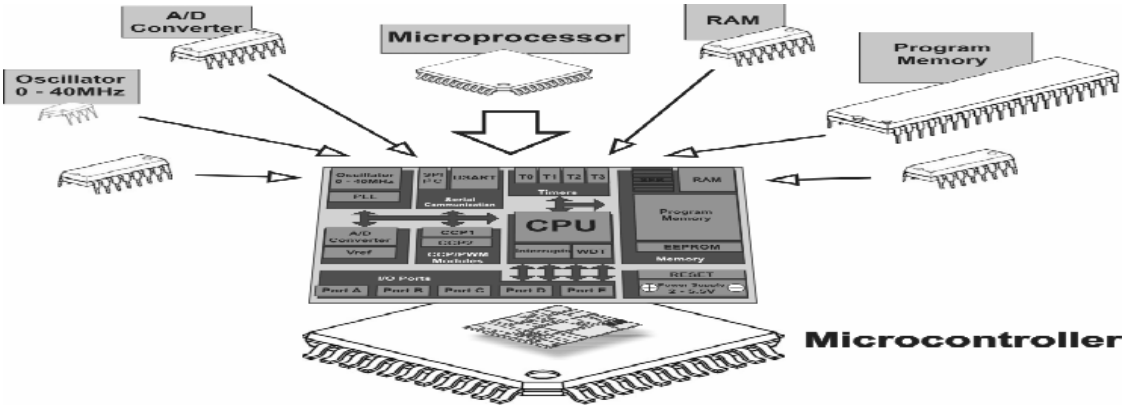


Figure (1): *Microcontroller Core Features.*

The self-test program routines are based on the ISA of the microcontroller core. These routines are considered as development phases, stored in the flash memory of a microcontroller for in-the-field testing or production test. Subsequently, these test

routines are executed at speed to generate the necessary test patterns for testing the complete set of operations performed by the components of the processor and test results are propagated to GPIO pins of the microcontroller and analyzed using the external portable ATE [13 - 14]. Figure (2) depicts the main phases of the presented software-based part of the DFT methodology. These phases includes: Identification and information extraction about microcontroller features, Instruction selection strategy depending on observability and controllability and the last phase is operand selection and used TPG technique.

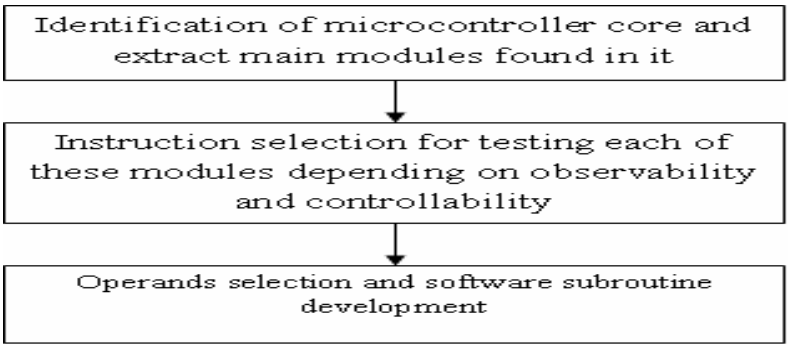


Figure (2): *DFT methodology for microcontroller test.*

The information extraction and component identification phase, studies PIC microcontroller core features and, according to divide and conquer algorithm, it was divided to a number of main modules and collect all available information on every module to be tested effectively. After completing this step, the microcontroller was found that it contain (Processor – Memory – Timers – Serial Port – PWM modules – GPIO – A/D converter (not tested in this paper)). Memory in microcontroller can be divided into (RAM – E^2PROM – Flash Memory) and the processor can be divided into (ALU and multiplier). Using instruction selections phase that based on ISA of the microcontroller, every module M would have a set of operations O_M that module M performs. It was denoted $I_{M,O}$ the set of microcontroller instructions that, during execution, enable the same control signals and cause, module M to perform operation O. It is evident that for each module M there is at least one microcontroller instruction that, during its execution, causes module M to perform operation O, i.e. $I_{M,O} \neq \emptyset$.

The instructions which belong to the same set $I_{M,O}$:

- Have different observability properties since, when operation O is performed, the outputs of module M drive internal microcontroller registers with different observability characteristics.
- Have different controllability properties since, when operation O is performed, the inputs of module M are driven by internal microcontroller registers with different controllability characteristics After identification of the set $I_{M,O}$ for every module operation, select an instruction I of the set $I_{M,O}$ according to the following criteria:

Criterion 1: Discard instructions belonging to $I_{M,O}$ that, when operation O is performed, the outputs of module M do not propagate to an internal microcontroller register. This means that the faulty component output cannot be propagated.
Criterion 2: Between instructions I_A and I_B belonging to $I_{M,O}$, I_A is ranked higher priority than I_B if it requires a smaller instruction sequence to propagate the outputs of module M through the related internal microcontroller register to primary output ports. That means that instruction I_A is more easily observable than I_B and should be preferred over I_B.
Criterion 3: If Criterion 2 ranks two different instructions I_A and I_B belonging to $I_{M,O}$, have the same priority, select the one that requires smaller instruction sequence.
Operand selections phase chooses the appropriate test patterns to use it with software test routines in order to have high fault coverage as possible. Our methodology is based on brute force technique for test pattern generation to achieve very high structural fault coverage for each component in the microcontroller. If part of the test response of a component is not driven to a well accessible internal register (that is the case of flag outputs driving status register or special function registers) an extra instruction sequence is required to propagate first to accessible registers and then to primary outputs GPIO. After completing this step, test subroutines have been developed for each of the microcontroller modules that based on the above 3 criteria using both assembly and C programming languages.

3. Microcontroller Test Modules

The effectiveness of the presented methodology is evaluated on two different types of microcontroller families (PIC16F87X – PIC18F4X2). Table (1) introduces the key features of both families of microcontrollers as a result of information extraction and component identification phase. Test subroutine for each module in microcontroller will be introduced in the following sections.

Table (1): *Microcontroller key features*

Key Features	**PIC 16F87X**	**PIC 18F4X2**
FLASH Program Memory (14-bit words)	8K word	32K word
Data Memory	368*8	1536*8
E^{2}PROM Data Memory	256 bytes	256 bytes
I/O Ports	5 I/O Ports	5 I/O Ports
Timers	3	4
Capture/Compare/PWM Modules	2	2
Serial Communications	USART	USART
multiplier	-	8 * 8
Instruction Set	35 instruction	75 instruction

3.1 Memory Test

There are two main memory blocks in microcontroller. Main memory organization is divided into flash memory and data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle. The data memory can further be broken down into General Purpose RAM and the Special Function Registers (SFRs). The SFRs used to control the peripheral modules found in the microcontroller and RAM are used to hold data that microcontroller need during its normal operation. This RAM can be also divided into smaller banks. Here a different test was produced for each type of memory found in microcontroller using different technique.

3.1.1. Flash Memory Test

Here flash memory is tested by reading it byte by byte and apply the software multiple input shift register (MISR) [13] with primitive polynomial ($X^8+X^6+X^5+X^4+1$) shown in Figure (3). The output signature is compared with signature saved in last location of E^2PROM. The saved signature was generated by C++ program written in Visual Studio Dot Net 2010 that emulate MISR by reading Hex file generated from the mikroC compiler for the test program. If both signatures are equal then program is downloaded successfully and memory is tested as well.

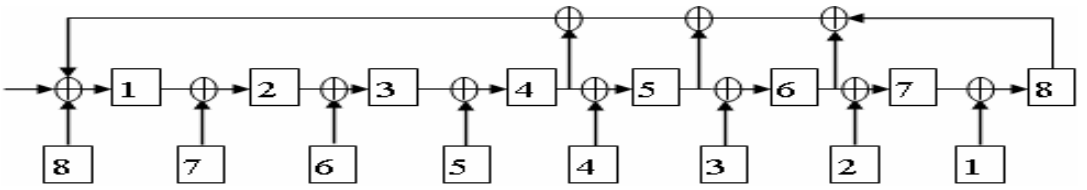


Figure (3): *MISR for Flash memory test.*

3.1.1 Testing RAM

Many Functional fault models (FFMs) for memories have been introduced [18]. Some of the published work focuses on *static faults*. Recent published work reveals the existence of another class of faults in the new memory technologies. It was shown that another kind of faulty behavior could take place in the absence of static faults. This faulty behavior, called *dynamic fault*, requires more than one operation to be performed sequentially in time in order to be sensitized. For example, a write 1 (W1) operation followed immediately by a read 1 (R1) operation will cause the cell to flip to 0, however, if only a single write 1 or a single read 1, or a read 1 which is not immediately applied after write 1 operation is performed, and then the cell will not flip.

RAM faults can be divided into single-cell and multi-cell faults. Single-cell faults consist of fault primitives (FPs) involving a single cell, while multi-cell faults consist of FPs involving more than one cell. Data RAM is the functional memory during the runtime of the microcontroller. When trying to test RAM as one block, it may take

long time so it was divided into smaller modules and test each module individually. RAM can be divided into two smaller modules For PIC16F87X and five modules for PIC18FXX2. RAM has been tested using March test algorithms [15 - 17] and output signature is propagated to GPIO pins and read them by the portable ATE [13 - 14] or by testing it internally and send signal to any GPIO pin that indicate that RAM has passed test.

The March test can be defined as a sequence of March elements, where a March element is a sequence of memory operations performed sequentially on all memory cells. In a March element, the way from one cell to the next is specified by the address order, which can be increasing or decreasing. For some March elements, the address order can be chosen arbitrarily as increasing or decreasing. In a March element, it is possible to perform a write 0 operation (W0), write 1 (W1), read 0 (R0) and read 1 (R1) operation. The 0 and 1 after read operations represent the expected values of the read on the output. An example of a March element is ↑ (R0; W1), where all memory cells are accessed in an increasing address order while performing R0 then W1 on each cell, before continuing to the next cell. By arranging, a number of March elements one after the other, a march test is constructed. Because of their simplicity and linearity with the memory size, all of them are in O(*n*).

Testing RAM using March test SS: Here a structured C subroutine based on March test SS technique was constructed (A Test for All Static Simple RAM Faults) for testing the RAM found in the microcontroller. For multi-cell FPs, March test SS restrict analysis to two-cell FPs (i.e., two-coupling FPs), because they are considered to be an important class for memory faults [15]. March test SS sequence is {↕(W0); ↑(R0;R0;W0;R0;W1); ↑(R1;R1;W1;R1;W0); ↓(R0;R0;W0;R0;W1); ↓ (R1;R1;W1;R1;W0); ↕(R0)}. It has a test length with a complexity of 22n.

Testing RAM Using March test AB: Here a structured C routine based on March test AB was constructed. It is a state-of-the-art March test for realistic static linked faults and dynamic faults in SRAMs technique for testing the RAM found in the microcontroller. March test AB sequence is {↕(W0); ↑(R0;W1;R1;W1;R1); ↑(R1;W0;R0;W0;R0); ↓(R0;W1;R1;W1;R1); ↓(R1;W0;R0;W0;R0); ↕(R0)}. It has a test length with a complexity of 22n.

Testing RAM Using March test BDN: Here a structured C routine based on March test BDN was constructed (A new March test for Dynamic Faults) March test BDN that is extended modified version of March test AB [17]. The newer version is able to improve the fault coverage of March test AB. In particular, it also detects new classes of dynamic faults, while keeping the same complexity of March test AB. March test BDN sequence is {↑(W_a); ↓(R_a;W_b;R_b;W_b;R_b); ↓(R_b;W_a;R_a;W_a;R_a); ↑(R_a;W_b;R_b;W_b;R_b); ↑(R_b;W_a;R_a;W_a;R_a); ↕(R_a)}. It has a test length with a complexity of complexity 22n. March test BDN still has the same coverage of March test AB, March test BDN is able to detect the whole set of realistic static linked and un-linked faults as well as the dynamic faults presented in [18].

3.1.2 Testing E2PROM

The E^2PROM is readable and writable during normal operation. This memory is not directly mapped in the register file space. Instead it is indirectly addressed through the Special Function Registers. There are four SFRs used to read and write this memory. These registers are: EECON1, EECON2 (not a physically implemented register), EEDATA and EEADR. Here E^2PROM is tested using modified algorithmic test sequence (MATS) algorithm. The MATS detects any combination of stuck-at faults (SAF) in RAMs, independent of the decoder design [19]. The resulting test sequence is {↑(W0); ↑(R0,W1); ↑**(**R1)}. E^2PROM is tested internally and signal is sent to GPIO pin to indicate if it pass test or not. User can use E^2PROM to store values during running application and read it after end or may use it to store pass code for the application, so it may contain important data for the user so when testing it, so a read for this data is done from its locations and saving it into a variable before testing it because the methodology does not need to destroy data in E^2PROM. Test program is written in C.

3.2 Testing USART

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules (other is the SSP module). The USART is also known as a Serial Communications Interface or SCI. The USART can be configured as a full duplex asynchronous system that can communicate with peripheral devices such as personal computers, or it can be configured as a half duplex synchronous system that can communicate with peripheral devices such as A/D or D/A integrated circuits, Serial E^2PROMs etc.
In this test, first the baud rate of the USART was set to 38400 bps then its tested simply by sending data from (0 to 255) through transmitter of USART (TX) and loop it back again through MAX232 or through short circuit to receive it through receiver of USART (RX) and then check if the sent data equals to received or not and the signature is measured using the external ATE [13 – 14]. Test code for this module is written in C language. List (1) shows the flow chart of the USART testing algorithm. When calculating error according to equations found in microcontroller data sheet, it was found that when using running frequency 4 MHz and baud rate 38400 bps errors wills be 0.18%.

$$\text{Desired Baud Rate} = F_{OSC} / (64\,(X + 1)) \quad (1)$$

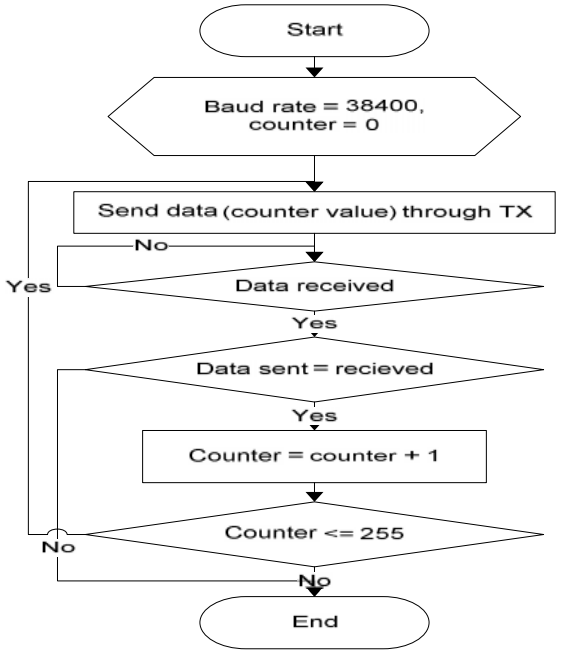
$$X = ((F_{OSC} / \text{Desired Baud Rate}) / 64) - 1 \quad (2)$$

$$\text{Error} = (\text{Calculated Baud Rate} - \text{Desired Baud Rate})/\text{Desired Baud Rate} \quad (3)$$

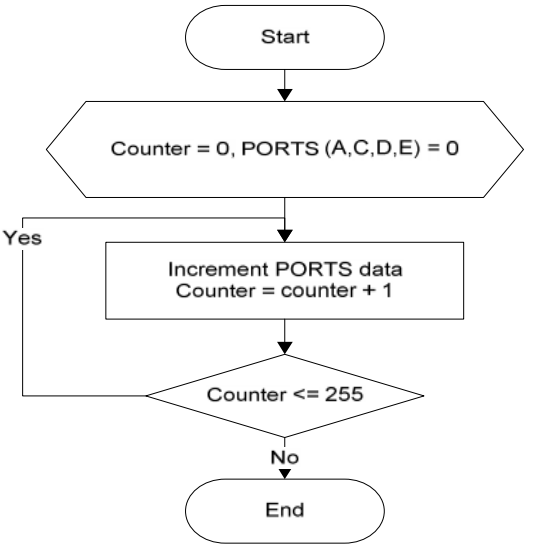
3.3 Testing GPIO

GPIO pins can be considered the simplest of peripherals. They allow the microcontroller to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with an alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

In this test, all GPIO found in the microcontroller family are set to be output ports by setting TRIS register of all PORTS to (0x00). A software loop is constructed that send data from 0 (min) to 255 (max) to these PORTS. Some of microcontroller ports have been used in previous tests but ports test was used to ensure that ports have been tested using all possible combination. The signatures are measured using the external ATE [13 – 14]. PORTB is used as an input for mode selection. Test code for this module is written in assembly language. List (2) shows the flow chart for GPIO test.



List (1): *Test USART Flow chart.*

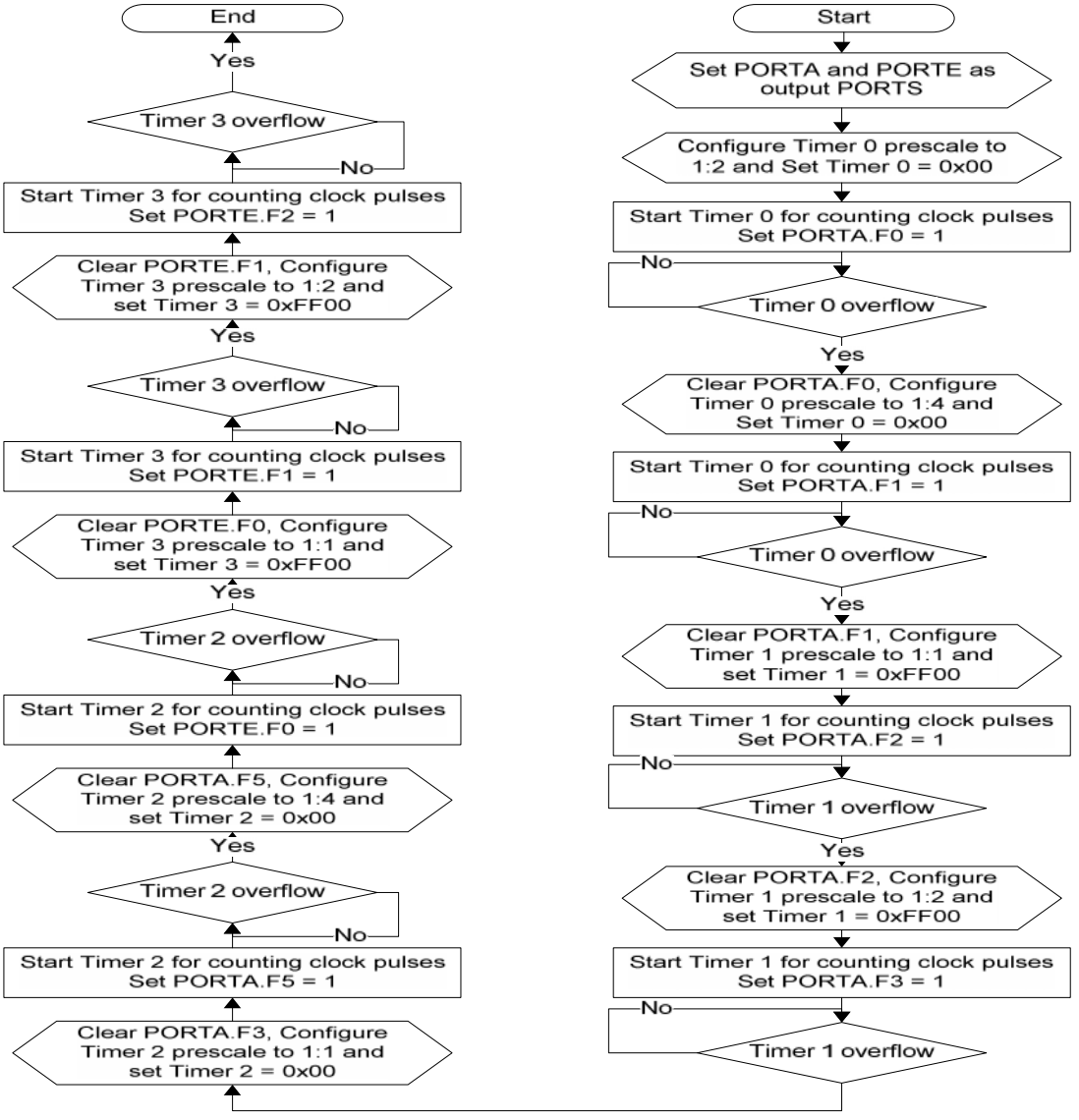


List (2) *GPIO test flow chart.*

3.4 Testing Timers

It is found from extracted information from step 1 that both microcontroller families has more than one timer that can be used as timers/counters depending on external or internal source (microcontroller clock) respectively. These timers have different sizes (8 or 16 bits) and different prescaler, here each timer is tested in two different prescale in order to check the functionality of the timer and it's prescale. For each timer, SFR registers are configured to make timer depend on internal input source (clock) It is set the initial value and prescale (1:1, 1:2 and 1:4). The timer is started for counting and set PORTD Pins to high until timer overflow occur then deactivate

PORTD Pins to low. Here a measure for the on-time that PORTD still high using Portable ATE [20] is done. List (3) shows the flow chart for the timer test.



List (3): *Timers test flow chart.*

3.5 Testing CPU

The CPU can be thought of as the "brains" of the device. It is responsible for fetching the correct instruction for execution, decoding that instruction, and then executing that instruction. The CPU controls the program memory address bus, the data memory address bus, and accesses to the stack. The CPU is responsible for using the information in the program memory (instructions) to control the operation of the device. To operate on data memory, the ALU performs arithmetical and logical operations, controls status bits (which are found in the STATUS register). The result of some instructions forces status bits to a value depending on the state of the result.

Here the test subroutine is constructed that aims at structural faults from the beginning by preparing structural tests for the components in the microcontroller's CPU. Moreover, the instructions are not randomly chosen, but carefully crafted in order to deliver all structural tests to the desired components. As previous approaches, the control unit is not tested because it is tested during CPU test. The extra instructions are used to test status bits after arithmetic and logic operations. It is found from extracted information that microcontroller families have three basic operations in the instruction set architecture. They are byte-oriented file register operations, literal and control operations, and bit-oriented file register operations. Also, it is found that PIC16F87X has 35 instructions in the ISA and PIC18FXX2 has 75 instructions in the ISA. So two subroutines contain most of the two microcontroller instructions are developed to test all basic operations in both processors and an 8*8 multiplier in the case of PIC18FXX2. The signatures are measured using the external ATE [13 – 14] through GPIO.

3.6 Testing Capture/Compare/PWM modules

Both of two families contain two Capture/Compare/PWM (CCP) modules. Each CCP has a 16-bit register which can operate as a 16-bit capture register, as a 16-bit compare register or as a 10-bit PWM master/slave Duty Cycle register. The CCP modules are identical in operation, with the exception of the operation of the special event trigger. Different CCP modes depend on timers in the microcontroller. When working in both (capture and compare) modes it uses Timer 1 and when working in PWM mode it uses Timer 2. Here both CCP modules are tested in PWM mode in order to test functionality and they are not tested in all modes because timers are tested before. Here the CCP is configured to 5 KHz frequency and set the PWM duty to 127 and let it work for 2 ms then stop it. Repeat this procedure with second CCP. For both CCPs, the signature is taken from port C pin 1 and port C pin 2 using Portable ATE [13-14, 20], and compare it with the fault free signature.

4. Discussion and Experiment results

List (4) and List (5) contains the integrated flow chart of the presented test methodology that was applied on both microcontroller families using mikroC compiler. Three testing strategies for two different families of microcontroller were generated. The first testing strategy is the presented hybrid DFT methodology which combines both SBST and HBST. The other testing strategies are based on SBST, one of them uses MISR for the TRC and the other uses LFSR for the TRC. The following section presents the comparison between these sting strategies. This comparison is based on memory utilization, number of clock cycles that was taken to complete each test and the number of modules that can be tested using each of them.

Experimental results in Table (2) and Table (3) and Figures (4), (5), (6) and (7) show that the presented methodology is superior in memory utilization, test time and can test all microcontroller modules for both 18F4X2 and 16F87X families. The

presented charts compare between number of clock cycles taken to finish test for the three testing strategies and the presented pie charts shows memory utilization for them all.

Finally, its concluded that HBST is not preferable to test microcontroller and microprocessor because of number of limitations like area and performance overhead and SBST is practically not suitable also for microcontroller test because it cannot test all microcontroller internal modules and when it test internal modules in microcontrollers it cannot make sure that GPIO of the microcontroller work probably without using external ATE. So this presented DFT methodology that uses ISA of the microcontroller family generates test subroutines and for the TPG and part of the BIST control unit. The external ATE is used for the other part of the BIST control unit and for test response compaction and evaluation. So we can say that a hybrid test methodology between SBST and HBST was created.

Table (2): Test program statistics for PIC 18F4X2

PIC 18F4X2	**Crystal Oscillator**	**Unit**	**Presented Hybrid DFT**	**SBST with Compaction Tech. using MISR**	**SBST with Compaction Tech. using LFSR**
Used RAM	4 MHz	Byte	32 (2%)	35(2%)	56 (3%)
Free RAM		Byte	1479 (98%)	1476 (98%)	1455 (97%)
Used ROM		Byte	2560 (7%)	2622 (8%)	4206 (12%)
Free ROM		Byte	30207 (93%)	30145 (92%)	28561 (88%)
Clk cycles		count	25122732	255374762	701789720
Simulation Time			00' 25.1"	04' 15"	11' 41"
Tested Modules			All Passed	Timers, GPIO and CCP Failed	Timers, GPIO and CCP Failed

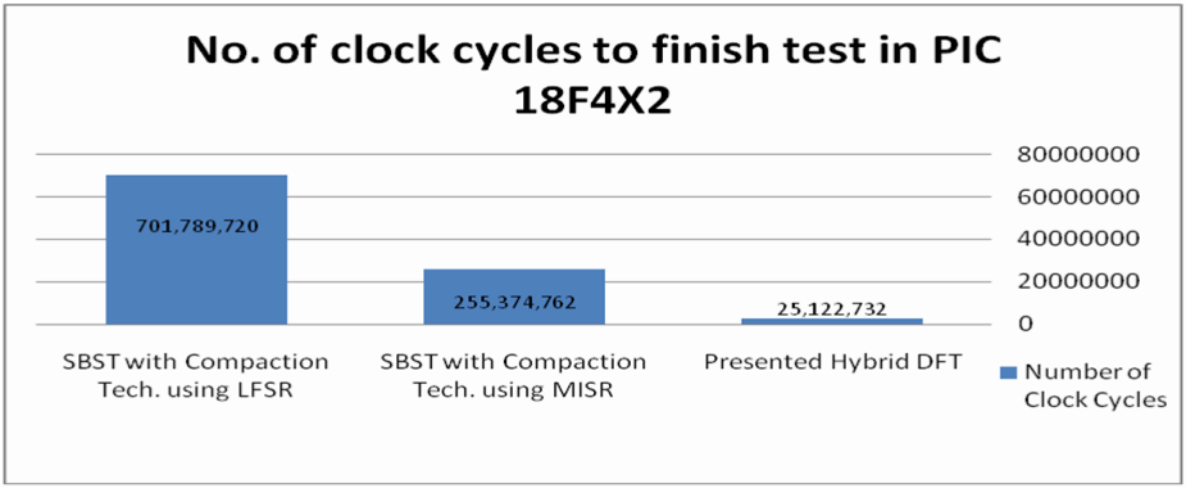


Figure (4): *Number of clock cycles for each test PIC 18F4X2*

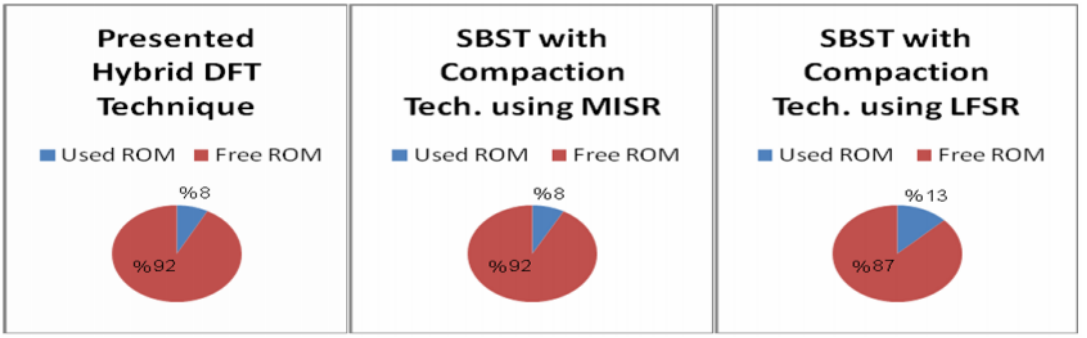


Figure (5): *Memory Utilization for the three techniques for PIC 18F4X2*

Table (3): *Test program statistics for PIC 16F87X*

PIC 16F87X	**Crystal Osc.**	**Unit**	**Presented Hybrid DFT**	**SBST with Compaction Tech. using MISR**	**SBST with Compaction Tech. using LFSR**
Used RAM	4 MHz	Byte	27 (7%)	30(8%)	51 (13%)
Free RAM		Byte	341 (93%)	338 (92%)	317 (87%)
Used ROM		Byte	1699 (20%)	1903 (23%)	2461 (30%)
Free ROM		Byte	6492 (80%)	6288 (77%)	5730 (70%)
Clock cycles		count	23820098	300849152	493812776
Simulation Time			00' 23.8"	05' 00"	08' 13"
Tested Modules			All Passed	Timers, GPIO and CCP Failed	Timers, GPIO and CCP Failed

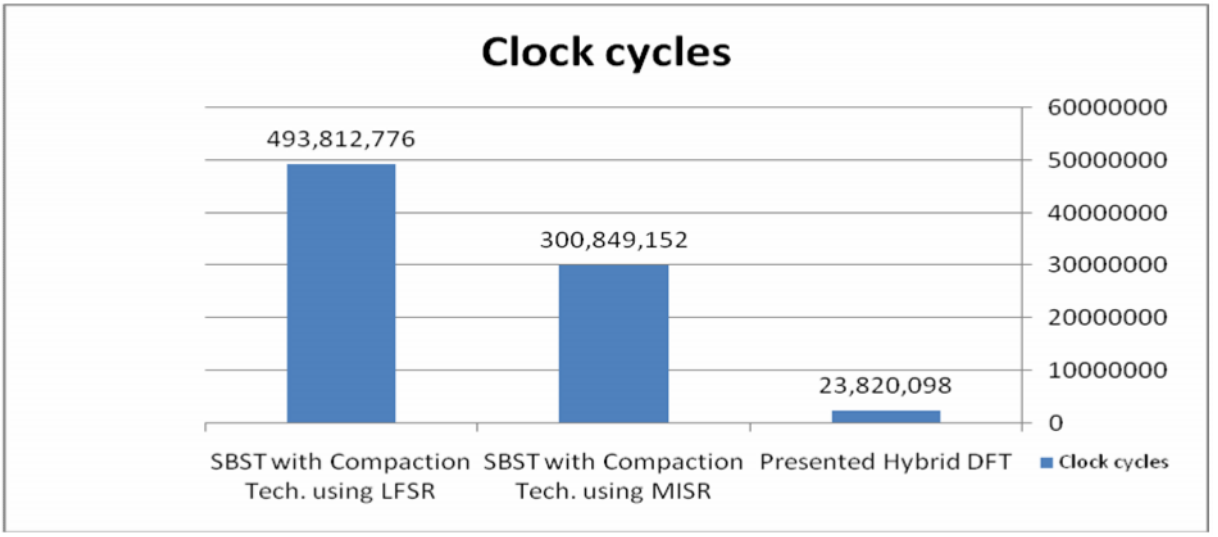


Figure (6): *Number of clock cycles for each test PIC 16F87X*

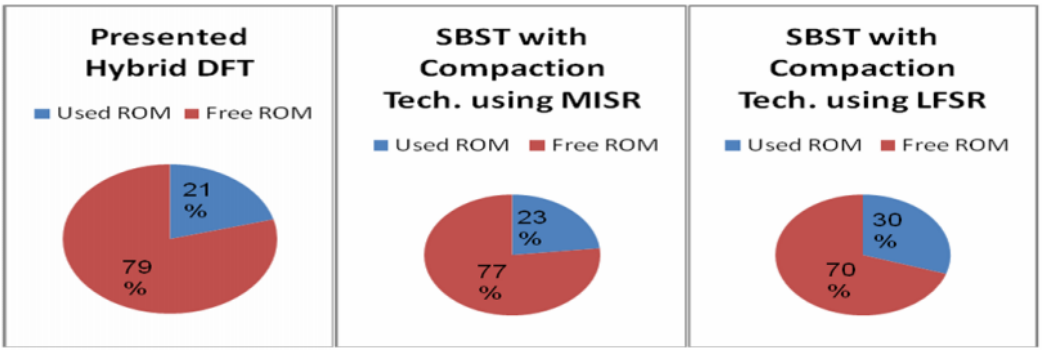


Figure (7): *Memory Utilization for the three techniques for PIC 16F87X*

5. Conclusion

In this paper, the hybrid self-test methodology for microcontrollers is presented. It achieves high structural fault coverage without performance degradation. The methodology targets microcontroller components and applies exhaustive TPG for every component operation. Based on divide and conquer algorithm, the microcontroller is divided into number of main modules and construct subroutine test for each module that excite all module operations. The exhaustive TPG technique is used in order to have high fault coverage without fault simulator. The presented DFT methodology that uses ISA of the microcontroller family generates test subroutines and for the TPG and part of the BIST control unit. The external ATE is used for other part of BIST control unit and for test response compaction and evaluation. The presented methodology is superior in memory utilization; test time and can test all microcontroller modules. The comparison based on clock cycles and the memory utilization concludes that SBST is practically not suitable for microcontroller test because it cannot test all microcontroller internal modules and when it test internal

modules in microcontrollers it cannot make sure that GPIO of the microcontroller work probably without using external ATE.

1. References

[1] Association, S. I. (1999). "The International Technology Roadmap for Semiconductor."

[2] T. G. Foote, D. E. H., W. V. Huott, T. J. Koprowski, B. J. Robbins and M. P. Kusko, (1997). "Testing the 400 MHz IBM Generation-4 CMOS Chip". International Test Conference.

[3] G. Hetherington, T. F., N. Tamarapalli, M. Kassab, A. Hassan and J. Rajski (1999). "Logic BIST for large industrial designs: Real issues and case studies". International Test Conference, Atlantic City, NJ,.

[4] L. Chen, S. D. (2001). "Software-Based Self-Testing Methodology for Processor Cores". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 20: 369 – 380.

[5] N. Kranitis, A. P., D. Gizopoulos and Y. Zorian (2002). "Effective Software Self-Test Methodology for Processor Cores". IEEE Transaction on Design, Automation and Test in Europe Conference and Exhibition.

[6] J. Shen, J. A. A. (1998). "Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation". Proceedings of International Test Conference (ITC).

[7] K. Batcher, C. P. (1999). "Instruction Randomization Self Test for Processor Cores". Proceedings of IEEE VLSI Test Symposium (VTS).

[8] P. Parvathala, K. M., and W. Lindsay (2002). "FRITS—A microprocessor functional BIST method" Proceedings IEEE International Test Conference.

[9] N. Kranitis, D. G., A. Paschalis, and Y. Zorian (2002). ,"Instruction based self-testing of processor cores". Proceedings IEEE VLSI Test Symp.

[10] N. Kranitis, G. X., A. Paschalis, D. Gizopoulos, and Y. Zorian (2003). "Application and analysis of RT-level software-based self-testing for embedded processor cores". Proceedings IEEE International Test Conference.

[11] Zhou, J. (2009). "Software-Based Self-Test under Memory, Time and Power Constraints". Institute of Technology computer science, University of Stuttgart. PhD.

[12] A. Paschalis, D. G. (2005). "Effective software-based self-test strategies for on-line periodic testing of embedded processors". IEEE Transactions on CAD. 24: 88 – 99.

[13] M. H. EL-Mahlawy, A. Seddik , "Design And Implementation Of New Automatic Testing System For Digital Circuits Based On The Signature Analysis.", proceeding of the 12th ASAT Conference, 29-31 May, (2007).

[14] M. H. El-Mahlawy, A. Abd El-Wahab, A.S. Ragab, "FPGA

Implementation of The Portable Automatic Testing System for Digital Circuits.", Proceedings of the 6th ICEENG Conference, 27-29, May, (2008).

[15] S. Hamdioui, A. J. v. d. G. a. M. R. (2002). "March SS: A Test for All Static Simple RAM Faults" IEEE International Workshop on Memory Technology, Design and Testing: 95 - 100.

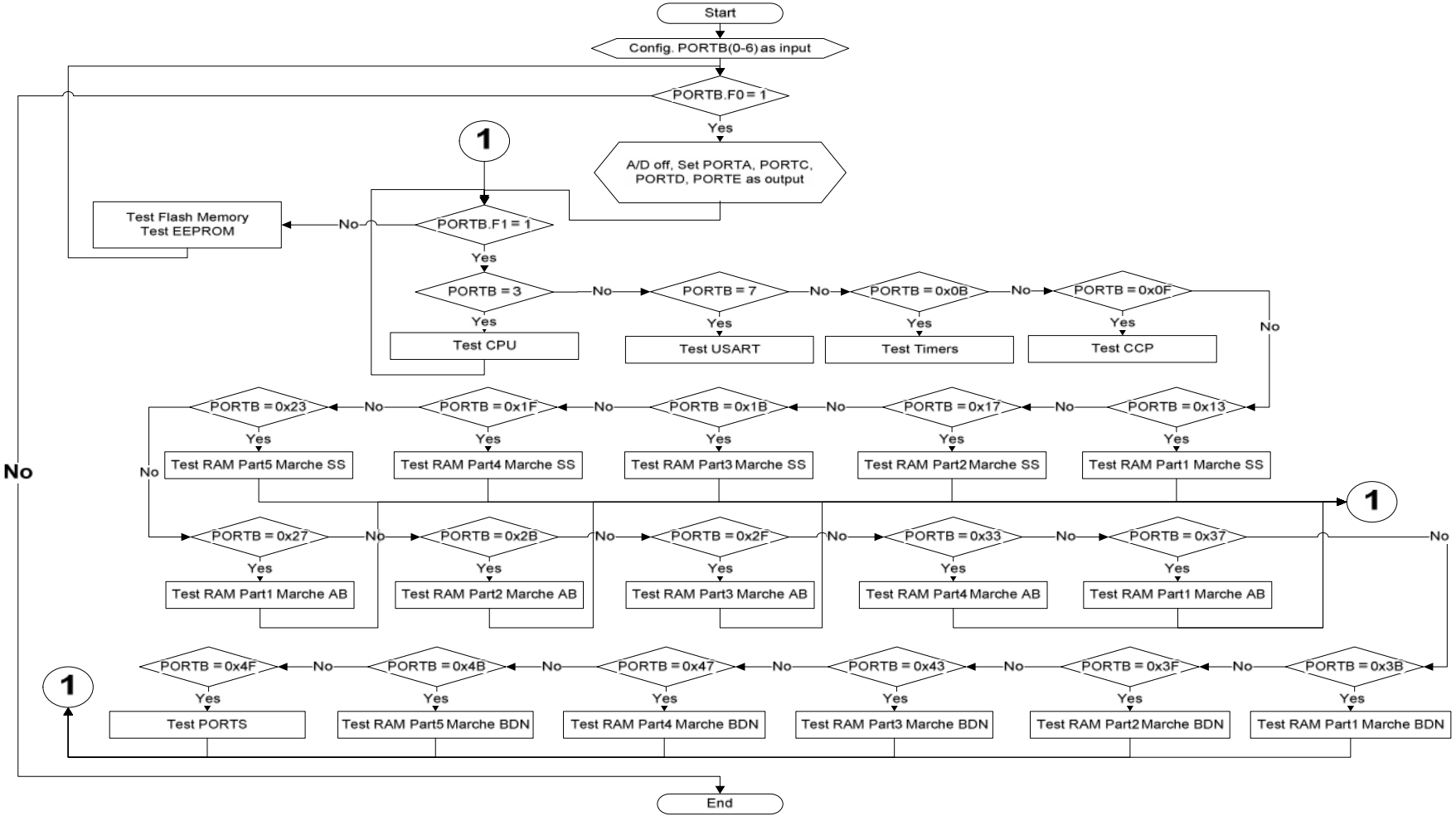
[16] A. Benso, A. B., S. Di Carlo, G. Di Natale and P. Prinetto (2005). "March AB, March AB1: New March Tests for Unlinked Dynamic Memory Faults". IEEE International Test Conference.

[17] Alberto Bosio, G. D. N. (2008). "March Test BDN: A new March Test for Dynamic Faults". IEEE International Test Conference.

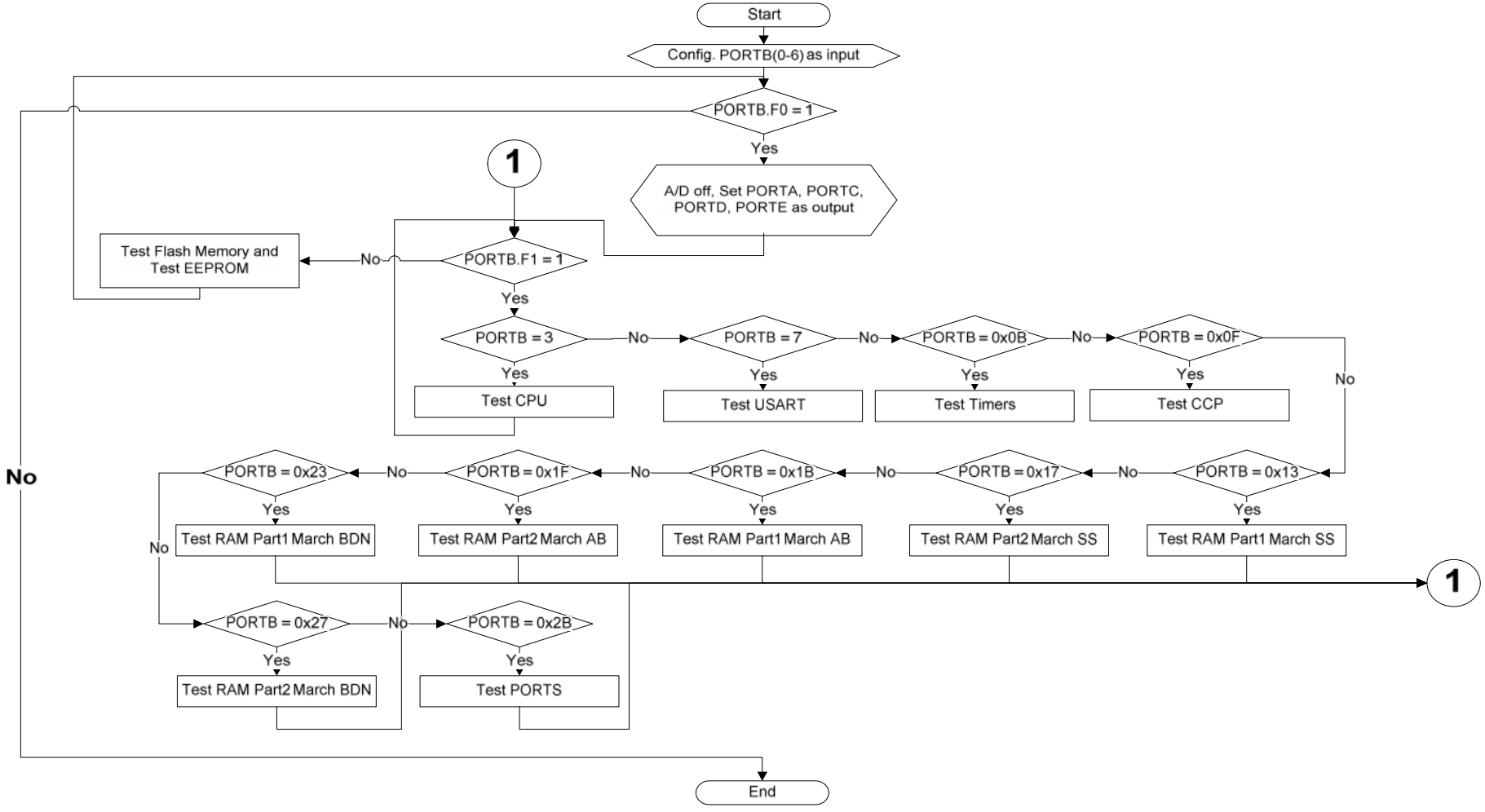
[18] A. J. van de Goor, Z. A.-A. (2000). "Functional Memory Faults: A Formal Notation and a Taxonomy". 18th IEEE VLSI Test Symposium: 281-289.

[19] Mohamed H. El-Mahlawy, Mahmoud S. Hamed, Mohamed H. Abd-El-Zeem, and Issa Yossef, "FPGA Implementation of the BIST IP For SRAM Chips", 6th International Conference of the Electrical Engineering, Military Technical College, Egypt, 27-29 May. (2008).

[20] Mohamed H. El-Mahlawy, "A novel testing method for monostable multivibrators", 5th International Conference of the Electrical Engineering, Military Technical College, Egypt, pp. EM-6-1- EM-6-11, May. (2006).



List (4): *algorithm for the presented testing methodology for PIC18F4X2*



List (5): *algorithm for the presented testing methodology for PIC16F87X*