Selecting Power-Optimal SBST Routines for On-Line Processor Testing

A. Merentitis[1], N. Kranitis[1], A. Paschalis[1], D. Gizopoulos[2]

[1] *Department of Informatics & Telecommunications, University of Athens, Greece*
[2] *Department of Informatics, University of Piraeus, Greece*
{amer, nkran, paschali}@di.uoa.gr, dgizop@unipi.gr

Abstract

Software-Based Self-Test (SBST) has emerged as an effective strategy for on-line testing of processors integrated in non-safety critical embedded system applications. Among the most popular applications falling in this category are the various mobile devices. However, in-field testing of processors integrated in mobile devices has the extra requirement of minimum energy consumption, since these devices are operating on battery.

In this paper initially we present the parameters contributing to energy consumption in order to set the scope of the problem and make a qualitative analysis about the efficiency of the SBST routines from a low energy perspective. Then we propose a power evaluation framework based on a combination of tools from the testing and computer architecture technical areas. Utilizing this framework we evaluate for the first time the most effective structural SBST strategies in terms of energy consumption for three representative processor configurations, showing the validity of our analysis. Finally, the SBST strategies are combined and an appropriate strategy for each processor component is selected, in order to minimize overall energy consumption during on-line testing up to an order of magnitude between the best and worst case combination of test routines.

1. Introduction

New types of defects appearing in deep submicron technologies require at-speed testing in order to achieve high test quality. In the multimillion gate SoC era, design and test engineers also face signal integrity problems and serious power consumption and overheating concerns, especially when the circuit has to be placed in special test modes [1]. The problem becomes even more complicated for on-line testing, that aims at detecting and/or correcting operational faults. Redundancy techniques (hardware redundancy, information redundancy, time redundancy and software redundancy) are either high cost solutions or impose serious performance overhead [2]. *Software-Based Self-Test* (*SBST*) has been proposed [3]-[10] as a low-cost solution for on-line testing of processors integrated in non-safety critical embedded system applications, that can effectively address all the previous challenges. It is using existing processor resources for test pattern generation and application, with no hardware or performance overhead. Additionally, peak power consumption for the test routine is lower than that caused by hardware DFT approaches, since the processor is operating in normal mode. However, energy consumption of the test routines is also critical, especially for in-field testing of processors integrated in mobile devices, in order to maximize battery life and to avoid long term reliability problems.

SBST techniques which are *functional* in nature and use random instruction sequences, operations and operands, have been proposed in [3], [4]. Such techniques have low test development cost due to their high abstraction level, but they also achieve medium to high fault coverage with a large number of instruction sequences. Thus, the derived test program is large and requires excessive test-execution time. Additionally, long fault-simulation time is required for fault grading. Therefore, functional-based SBST techniques are unsuitable under strict energy constraints.

SBST techniques *structural* in nature, have been proposed [5]-[9] as promising techniques for efficient testing of embedded processors. Test patterns are generated targeting structural faults for a specific fault model. After that, the test patterns are transformed to self-test routines which are used to apply test patterns to the inputs of the Circuit Under Test (CUT) and collect test responses from the outputs of the CUT. All self-test routines together constitute a test program. It is important to note that the test program should comply with stringent requirements in code size, data size, and execution time to be suitable for on-line testing.

The routines considered in this paper for low energy SBST are following the two most effective structural test pattern generation (TPG) strategies. The first TPG strategy is based on deterministic automatic TPG (ATPG) and is usually applied to combinational components, where instruction-imposed constraint ATPG is feasible. The second TPG strategy is based on high-RTL deterministic TPG that exploits inherent regularities of the most critical to test processor components like ALUs, shifters, multipliers, multiplexers and register files. This TPG strategy is high-level, since the derived test patterns are independent of gate-level implementation and constitute test sets of constant or linear size. Although several structural SBST variations have been compared in terms of test effectiveness in the past, low energy consumption was only recently taken into consideration and an evaluation from this perspective does not exist yet.

In this paper, for the first time we evaluate the previously mentioned structural SBST strategies, targeting low energy testing of embedded processor cores. Only routines that are proved in terms of test efficiency are considered and from these only those that comply with the requirements of on-line testing defined in [8] are selected for the evaluation. Taking these routines as a basis, five questions relevant to low power SBST are addressed:

- Under which conditions is it efficient from an energy perspective to deploy Multiple Input Signature Register (MISR) test response compaction implemented as an assembly routine and when is it preferable to use direct comparison with the expected values instead?
- Assuming that high-RTL TPG is used as in [7], [8], [9] is it preferable from an energy view point to use compact loops or immediate coding style?
- Assuming that constrained ATPG is deployed and then the patterns are mapped to processor instructions as in [6], [10], is it preferable from an energy point of view to set the operands using a coding style with immediate instructions or a coding style that favors test data fetching?
- Are constrained ATPG or high-RTL deterministic patterns preferable for low energy SBST and under which conditions?
- What is the impact of performance aiding mechanisms from a strict energy efficiency testability point of view?

After addressing the previous questions, a qualitative analysis regarding the efficiency of SBST routines from the low energy point of view is presented. Additionally, an architectural-level power evaluation framework is utilized in order to validate the theoretical analysis for three representative processor configurations, which cover a wide range of embedded processors and microprocessors. The first is typical of a basic MIPS-like CPU and is ideal to study the energy consumption of each test strategy in a barebones datapath. The second configuration is similar, but also incorporates a simple branch predictor, as well as level-one (L1) instruction and data caches and is typical of a MIPS-based embedded processor. Finally, the last configuration has also a unified second-level (L2) cache and a more sophisticated branch predictor and is typical of a high performance microprocessor. Moreover, the results of the architectural-level simulator for the simple configuration are also validated using a commercial gate-level power reporting tool. Experimental results show that the energy savings achieved by selecting the most efficient combination between those test routines that already comply with the requirements of on-line testing can be as high as 90.3 %, depending also from the processor configuration.

It should be noted at this point that a novel approach utilizing instruction rescheduling in order to minimize power consumption was introduced in [10]. However, the methodology was applied on a simple MIPS-like processor without parallel multiplier or pipeline logic, thus direct comparisons with the processor configurations used in this paper are not feasible.

The rest of the paper is organized as follows. A theoretical analysis of the parameters contributing to power consumption is presented in section 2. Section 3 delineates the proposed power evaluation framework. The structural test strategies evaluated in this contribution are outlined in section 4. Results for total energy consumption, validation of the correctness of theoretical analysis and selection of the most energy efficient test routines corresponding to the abovementioned strategies are provided in Section 5. Finally, section 6 concludes the paper.

2. Theoretical Analysis

A brief theoretical analysis of the parameters contributing to power consumption is necessary in order to set the scope of the problem and identify the tradeoffs offered by each strategy. Power consumption in CMOS circuits can be either static or dynamic. Leakage current or other current drawn continuously from the power supply causes static power dissipation. Dynamic dissipation occurs during output switching due to the short-circuit current and charging/discharging of load capacitance. The importance of static power consumption increases as dimensions scale down, however current CMOS technologies are dominated by dynamic power consumption. For a single node the latter can be approximated by the following mathematic formula:

$$P = C_L \cdot S \cdot V_{dd}^2 \cdot f_{CLK},$$

where C_L is the equivalent load capacitance, V_{dd} is the power supply voltage, S is the number of node switches and f_{CLK} is the operating frequency. Additionally, energy consumption for a period T equals $E = P_{av} \cdot T$ and $T = N \cdot \tau$, where P_{av} is the average power consumption over period a T, N is the total number of execution cycles and τ is the clock period. It is apparent from the previous formulas that, for a given circuit and a given CMOS technology, energy consumption can be reduced if the test program is executed in a small number of cycles and average power consumption is low. However, these two factors cannot always be optimized simultaneously. For example, it is known (e.g. [11]), that as a general rule, in order to achieve low average power consumption it is important to reduce the number of instructions causing high switching activity and dominant among these are the load and store instructions. In particular, [12] shows that every cycle required for a load instruction is expected to cause higher switching activity than a cycle of an immediate instruction, resulting in increased average power consumption P_{av}. However, in order to replace load and store instructions usually a number of other instructions are required, which can result in increased execution cycles.

Specifically, during SBST a common way to reduce memory accesses is to utilize a test response compaction routine and store a single signature. Assembly code snippets (the parts that are repeated for every test response) for an effective compaction scheme (software MISR) that can be used for this purpose as well as the direct comparison alternative, are presented in Figure 1.

```
sll  $24,$23,0x001f
sra  $25,$24,0x001f
and  $25,$25,$22
xor  $23,$23,$25
srl  $23,$23,0x0001
addu $23,$23,$24
xor  $23,$23,$17
```

```
lw   $15,0($21)
xor  $10,$15,$17
bnez $10,Error
```

Figure 1: Assembly for MISR and direct comparison

Each individual response is incorporated in a common signature and at the end of the execution of the test routine the register holding the signature (*$23* in Figure 1) is compared with the expected signature value. Alternatively to MISR compaction, direct comparison with the expected values can be deployed. Specifically, after a test response is generated the corresponding expected value is loaded from a predefined memory position in a register (*$15* in Figure 1) and a *xor* instruction having the two values as inputs is used to examine if the generated result is correct. Finally, a branch not equal zero (*bnez)* instruction receiving the outcome of the *xor* is used to differentiate the correct and erroneous conditions, by altering the flow of execution when an error has occurred. Additionally to test response compaction, a coding style using immediate commands to set the operands also reduces memory accesses, at the cost of doubling the number of instructions required for setting the operand.

Taking into account performance aiding mechanisms like caches and branch predictors further complicates the problem of analytically deriving an optimal SBST strategy from energy viewpoint, because it results in speculative execution of instructions and the number of required stalls (i.e. for a load instruction) becomes variable. The parameters that impact the number of cycles required by the load instructions and thus define whether load instruction should be avoided are the following:

- Cache size and associativity
- Cache replacement algorithm
- Cache load latency
- Memory load latency

In general, if in the specific processor ISA load instructions require on average a number of cycles greater than the number of cycles (less than ten in most architectures) required for executing the entire MISR compaction routine, as can be the case for high-end processors, the compaction is efficient from an energy perspective, otherwise it is better to proceed in direct comparison of the produced values with the expected results. Moreover, the use of compact loops increases spatial and temporal locality, so it can be induced that loop-based routines will be more efficient in number of cycles as well as power consumption if caches are available. Finally, branch prediction schemes will not affect routines that do not use loops and are expected to further improve the efficiency of loop-based test routines.

3. Power Evaluation Framework

The power evaluation framework utilized in this paper is comprised by a combination of tools from the testing and computer architecture technical areas. Specifically, the validation of the test routines is based on a commercial fault simulator and the energy metrics provided in the experimental section are generated by a widely used power simulator, Wattch. The Wattch framework [13] is an architectural-level simulator for estimating power consumption. It can accurately model four main categories of processor elements (array structures, memories, combinational logic and wires, clock networks). According to its developers, it operates three orders of magnitude faster than industrial layout-level tools and yet experimental validation of the generated reports for a wide range of applications (including commercial processors like Alpha 21264 and MIPS R10000) shows that the estimated power consumption is within 10-13% of the values reported by the tools operating at the post layout netlist. Additionally, the Wattch framework implements a MIPS-like ISA with a five stage pipeline, so it has a core very similar to the processor we used for validating the test routines. It is based on the SimpleScalar toolset, thus is extendable and parameterizable, both essential requirements for a comprehensive evaluation of SBST strategies in different processor configurations. This last advantage, coupled with the fact that the power models incorporated by Wattch are general as well as accurate [15], meaning that the results are valid for a wide range of processors, constitute the use of it ideal for evaluating the SBST strategies from an energy point of view. Finally, the flexibility offered by a configurable architectural-level simulator allows us to further validate the results, using commercial gate-level power reporting tools, when the gate-level netlist of a processor similar to a configuration supported by Wattch is available.

In order to derive which test routines are more efficient from an energy perspective for a specific processor, the processor specification is utilized to select the appropriate high-RTL test routines from an existing library of self-test routines and also to set the basic parameters of the Wattch configuration (cache sizes, cache and memory latency, etc). The gate-level netlist produced from the synthesis tool is used by the constrained ATPG strategy to generate patterns fine-tuned to the specific netlsist. The SBST routines of either strategy (high-RTL TPG or constrained ATPG) are also validated by fault simulation on this netlist. In case that fault coverage is adequate, the routines are compiled by the SimpleScalar GCC-port and the resulting executable is run by Wattch to generate the power consumption reports. The framework used for generating power results is presented in Figure 2.

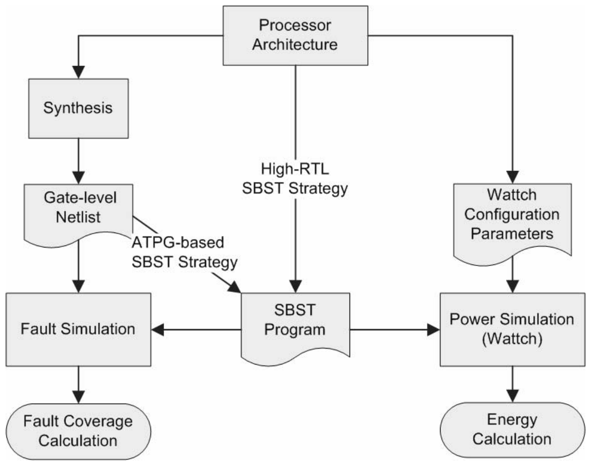

Figure 2: The power evaluation framework

In this paper three different configurations, covering a wide range of processor benchmarks, from simple embedded processors to high performance microprocessors are considered. Detailed characteristics for the simulated processor configurations are presented in Table 1.

	Simple	Moderate	Advanced
Branch Miss Penalty	3 cycles	3 cycles	3 cycles
Decode Width	1	1	1
Issue Width	1	1	1
Memory Ports	2	2	2
L1 Data Cache	none	32 KB	32 KB
L1 Instruction Cache	none	32 KB	32 KB
L2 Unified Cache	none	none	256 KB
L1 D-Cache Latency	-	1 cycle	2 cycles
L1 I-Cache Latency	-	1 cycle	2 cycles
L2 Cache Latency	-	-	6 cycles
Memory Latency	2 cycles	8 cycles	18 cycles
Integer ALUs	1	1	1
Floating Point ALUs	1	1	1
L/S Queue Size	2	2	2
Register Unit Size	32	32	32

Table 1: Simulated Processor Configurations

The first configuration is typical of a basic MIPS-like CPU, with an integer multiplier, an ALU and a register file consisting of 32 physical registers that are 32 bits wide. Some minor changes were performed to the code of the simulator in order to support this very basic configuration, since level one cache is normally included by default. This simple configuration is ideal to study the energy consumption of each test strategy in a barebones datapath. Moreover, this configuration is similar (with the exception that it incorporates FP ALU and FP multiplier) to the benchmark used for validating the test routines. This similarity allows us to utilize it to study the relative behaviour of the test routines using a commercial gate-level power reporting tool in addition to the Wattch simulator and thus obtain a higher degree of confidence for our conclusions. The second configuration also incorporates a simple branch predictor, as well as separate level one instruction and data caches and is typical of MIPS-based embedded processor. Finally, the third configuration also supports unified L2 cache and a more sophisticated branch predictor and is typical of a high performance microprocessor. For all power measurements the most comprehensive model supported by Wattch is used (cc3), which assumes clock gating with linearly scaled power for each unit according to port usage and 10% of the base power when the unit is not accessed.

In order to strengthen our confidence for the accuracy of the results provided by the power evaluation framework, we also used a traditional commercial gate-level power reporting tool to generate energy metrics. Only the simple processor configuration was considered, which closely resembles the available processor benchmark. The results are not directly comparable in absolute values with those generated by Wattch, because the latter incorporates some additional hardware (like FP ALU and FP multiplier, etc) that does not exist in our processor benchmark. Thus the target of this process is to verify the relative behavior of the test routines and see if this behavior is in compliance with the results provided by Wattch. The generated energy metrics for direct comparison and MISR compaction are depicted in Table 2. It is apparent that relative behaviour of the test routines for the various alternative coding styles and pattern deployment techniques matches the behaviour predicted by the Wattch simulations.

	Component & Test Strategy	Simple Configuration	
		Direct comparison	MISR compaction
high-RTL	Reg. File Imm.	✓ 2,798	18,067
	ALU Loop	✓ 3,048	6,467
	ALU Imm.	3,879	8,145
	MULT Loop	9,998	20,452
	MULT Imm.	9,657	18,417
const ATPG	Reg. File Imm.	31,563	100,215
	ALU Imm.	3,553	7,372
	MULT Imm.	✓ 4,838	7,897
	MULT Fetch	7,522	9,173

Table 2: Energy metrics obtained by a gate-level tool

4. SBST Strategies

Following the description of the methodology, a short outline of the SBST strategies that are evaluated in this contribution is necessary. It is already mentioned that both strategies are structural and can guarantee high test coverage within a reasonable number of cycles, thus a direct comparison is feasible. Moreover, it should be noted that enhancing the test routines with code targeting pipeline [9], [14], exception and control logic would increase total fault coverage to the levels reported in [9], however this code was developed using a purely high-RTL TPG while constrained ATPG is not feasible, thus there is no point to include these test routines in the evaluation.

The processor components considered for the evaluation of test strategies from the low energy testing point of view are the parallel multiplier, the register file and the ALU. These components usually dominate the datapath area and are also characterized by the highest switching activity, thus are the main sources of energy consumption in the processor datapath.

The gate-level independent deterministic test set introduced in [7] results in near complete stuck-at fault coverage for the entire register file component with minimum patterns. As an alternative to high-RTL based TPG, constraint-based ATPG can also be deployed for testing the register file. However, the register file is a sequential circuit that cannot be directly targeted by combinational ATPG, so the modules that are actually targeted are the two n-to-1 multiplexers implementing the read ports (each of these multiplexers is *m* bit wide, where *m* is the register size in bits). We can easily derive that for a register file with 32 registers which are 32-bits wide the size of each test vector is $5 + 32 \cdot 32 = 1029$ bits, where the five first bits are the selection signal of the multiplexer. Moreover, setting the appropriate register values for each pattern requires 32 *lui* and 32 *ori* instructions. Application of these test strategies results in almost complete test coverage of the register file for both. However the number of cycles required for execution of the test routine is much less for the high-RTL based TPG and the difference increases when a more advanced processor model is used.

The ALU is comprised by a number of arithmetic units and logic arrays that are characterized by inherent regularity and can thus be effectively tested by high-RTL TPG as described in [7]. Since it is a combinational circuit, constrained ATPG fine-tuned to the gate-level can be applied alternatively to the abovementioned high-RTL deterministic method. Application of either test strategy results in almost complete test coverage of the ALU for both.

The parallel multiplier is a combinational unit and the surrounding logic does not impose any architectural constraints, so applying ATPG is straightforward process. The number of test patterns generated by a commercial ATPG tool was 82, with high effort compression active. Then these patterns were mapped to proper processor instructions and were applied at the processor level. Alternatively to ATPG, 256 patterns exploiting the inherent regularity of multiplier units were generated, as described in [7]. This is a gate-level independent high-RTL method and has also the benefit of being able to generate and deploy the patterns in an algorithmic manner, by using compact loops. In order to evaluate the efficiency of the loops from a minimum power consumption point of view both the loop-based coding style and a coding style deploying immediate instructions were utilized.

Test coverage results for a fully pipelined processor benchmark [9] implementing the MIPS-I ISA are presented in Table 3 for constrained ATPG and high-RTL TPG, using either direct comparison with the expected values or MISR compaction and a single comparison with the expected signature in the end. Gate-count for the processor benchmark used for validating the test routines is 26,432 and fault simulation reports 85,587 uncolapsed stuck-at faults. Additionally, it should be noted that the results would be similar for any MIPS-like processor benchmark, both in regard to fault coverage but also in regard to the required number of test patterns, as shown in [7], [9].

Component	Direct Comparison		MISR Compaction	
	const. ATPG	high-RTL	const. ATPG	high-RTL
Register File	99.9	99.9	99.9	99.9
ALU	99.1	98.7	98.8	98.4
Multiplier	99.7	99.0	99.3	98.3
Processor	91.5	91.2	90.1	89.4

Table 3: Test coverage for structural SBST strategies

5. Experimental results

In this section the SBST strategies outlined previously are evaluated in terms of energy efficiency. It is important to underline that, in all the cases where constrained ATPG is used, high effort compression is activated, in order to reduce the number of test patterns and the resulting number of cycles required for execution of the self-test routine. The results reported by the simulator for the simple processor configuration with direct comparison and MISR compaction are depicted in Table 4 and 5, respectively. The first column shows the targeted functional component and the SBST strategy used. Specifically, the high-RTL strategy can utilize either immediate instructions or, in the case of the ALU and multiplier, also compact loops for test pattern generation and deployment. In a similar way, constrained ATPG can use either immediate instructions or data fetching in order to set the operands of the test instructions. The second column is the number of cycles required by the SBST routine and the third is total energy at processor level, reported in nanojoules.

	Component & Test Strategy	Cycles	Energy (nJ)
high-RTL	Register File Immediate	1,182	✓ 5,204
	ALU Loop	1,640	✓ 6,337
	ALU Immediate	1,594	6,536
	MULT Loop	4,134	18,059
	MULT Immediate	3,661	16,365
const ATPG	Register File Immediate	10,496	54,492
	ALU Immediate	1,347	6,672
	MULT Immediate	1,825	✓ 8,631
	MULT Fetch	3,541	14,536

Table 4: Simple configuration, direct comparison

	Component & Test Strategy	Cycles	Energy (nJ)
high-RTL	Register File Immediate	6,806	25,607
	ALU Loop	2,680	9,866
	ALU Immediate	2,597	9,885
	MULT Loop	8,229	31,007
	MULT Immediate	7,484	29,112
const. ATPG	Register File Immediate	26,872	131,110
	ALU Immediate	2,643	11,023
	MULT Immediate	2,870	11,779
	MULT Fetch	3,946	17,596

Table 5: Simple configuration, MISR compaction

5.1 Selecting MISR compaction or direct comparison

In the case of simple processor model, experimental results show that direct comparison is more efficient than MISR compaction. Specifically, MISR compaction decreases the average energy per cycle that can be calculated by dividing total energy with number of cycles; however the increase in cycles introduced by the compaction scheme outweighs that benefit. *The fact that MISR compaction is not energy efficient for the simple processor configuration is in compliance with the theoretical analysis, since even worst case load instructions require considerably fewer cycles than those introduced by execution of the compaction routine*. In the case of moderate processor model, worst case load instructions require up to eight cycles, causing the latency of the load instructions to be on average around five to six cycles, according also to cache utilization by other programs during the in-field testing. This results in a marginal situation, since MISR compaction requires seven cycles. Thus, most routines are again better with direct comparison but there are a few exceptions to this rule (Table 6). In the case of advanced processor model load instructions have latency of at least two cycles and in some cases six (from the L2 cache). However, the case of a complete miss (load the expected value from memory) is infrequent for small programs like the test routines considered in this paper, so on average we can expect load latency a little greater than six cycles, resulting again in a marginal situation.

5.2 Selecting coding style

Another important question is what coding style is preferable from an energy perspective in combination with constrained ATPG. In order to answer that question both coding styles are used in the case of the parallel multiplier unit. It is interesting that in the simple model constrained ATPG with a coding style using immediate instructions for setting the operands is preferable; however the moderate model is more efficiently tested by a coding style that uses operand fetching. Again, this result is expected since the stalls required for *every* load instruction (due to the absence of caches) to fetch the data from the memory in the simple model constitutes the immediate coding style preferable for it. In the moderate model the existence of data cache reduces considerably the frequency of stalls and changes the outcome in favour of the coding style that uses load instructions for setting the operands. Finally, in the advanced model the outcome changes again in favour of the immediate coding style, due to the increased cache and memory latency.

In case that a high-RTL strategy utilizing component regularities is used, it is also important to determine the most efficient coding style. Specifically, the question that needs to be answered is whether it has any meaning to avoid using compact loops for generation and application of the patterns, in order to reduce the number of cycles and consequently the energy consumption. The results provided in Table 4 and Table 5 for the ALU and multiplier components show that this technique is meaningful only for the multiplier in the simple configuration, if the increase in the size of the test program is an acceptable tradeoff. However, the corresponding results from Table 6 show that *avoiding the use of loops has no meaning for the moderate and the advanced processor models, since not only the size of the test program is increased but also energy consumption is significantly higher*. The previous observation can be justified if we consider that compact loops result in increased spatial and temporal locality and can better exploit the available caches. This effect is apparent from the fact that although the code without loops has reduced instructions, the required number of cycles is higher.

5.3 Selecting the most efficient test routines

Moreover, it is important to determine the best combination of test routines from an energy efficiency point of view. It is obvious that three separate cases need to be considered, one for each of the three processor configurations. In every case, the best component test strategy is highlighted on Tables 4, 5 and 6 with a tick symbol. For the simple model the combination that achieves the lowest energy consumption includes the high-RTL test routine for the register file and the constrained ATPG test routine using immediate coding style to set the operand values for the parallel multiplier. The ALU can be tested by either the high-RTL or constrained

	Component & Test Strategy	Moderate Configuration				Advanced Configuration			
		Direct comparison		MISR compaction		Direct comparison		MISR compaction	
		Cycles	Energy (nJ)	Cycles	Energy (nJ)	Cycles	Energy (nJ)	Cycles	Energy (nJ)
high-RTL	Register File Immediate	8,538	✓27,892	16,717	64,556	8,540	✓28,251	16,725	65,198
	ALU Loop	4,120	✓15,397	4,611	18,869	4,119	✓15,604	4,587	19,721
	ALU Immediate	5,118	18,345	5,472	21,781	5,126	18,592	5,434	22,913
	MULT Loop	7,971	38,689	15,341	75,587	8,009	39,902	15,362	81,234
	MULT Immediate	25,219	82,907	24,145	84,611	25,221	84,735	23,884	87,060
const. ATPG	Register File Immediate	91,257	301,676	186,461	627,304	91,259	305,822	185,647	632,497
	ALU Immediate	9,371	31,992	9,468	34,372	9,373	32,849	9,510	34,650
	MULT Immediate	11,817	40,182	10,734	38,414	11,819	40,976	10,744	✓38,582
	MULT Fetch	4,681	22,506	4,693	✓22,419	10,872	41,027	10,931	40,805

Table 6: Test strategies for the Moderate & Advanced processor configurations

ATPG test routines, with the former being slightly better in terms of energy consumption. Low energy testing of the parallel multiplier in the moderate and advanced model also favors constrained ATPG, utilizing data fetch operations for setting the operands for the moderate configuration and an immediate coding style for the advanced configuration. On the other hand, the register file and the ALU can be tested more efficiently if high-RTL test routines are used for both. Finally, in order to evaluate the energy savings achieved by selecting the most efficient routines, we created one program comprised by the best and another by the worst test routines for each processor configuration. Simulation of these two opposite cases shows that the energy savings can be as high as 87.9 % for the simple, 90.3 % for the moderate and 89.5 % for the advanced configuration.

6. Conclusions

In this paper we initially presented a qualitative analysis regarding the efficiency of the SBST routines from a low energy perspective. Moreover, for the first time we evaluated various SBST approaches in terms of energy consumption, using a framework based on the combination of commercial testing tools with a widely accepted architectural-level simulator. Additionally, we utilized that framework as well as commercial gate-level power reporting tools to verify our theoretical analysis. The derived conclusions are valid for a wide range of embedded processors, because the power evaluation framework is based on a generic micro-architectural simulator. Finally, we combined the available SBST methodologies in order to choose an appropriate test strategy for each processor component so as to minimize overall energy consumption during in-field testing.

References

[1] M. Nicolaidis, Y. Zorian, "On-line Testing for VLSI – A Compendium of approaches", in *Journal of Electronic Testing: Theory and Applications (JETTA)*, Vol. 12, No. 1-2, 1998, pp 7-20.

[2] H. Al-Assad, B. T. Murray, and J. P. Hayes, "Online BIST for embedded systems," *IEEE Design & Test of Computers*, vol. 15, no. 4, pp. 17–24, Oct.–Dec. 1998.

[3] K. Batcher, C. Papachristou, "Instruction randomization self test for processor cores", in Proc. Of the *VLSI Test Symposium (VTS) 1999*, pp. 34-40.

[4] P. Parvathala, K. Maneparambil, W. Lindsay, "FRITS – A Microprocessor Functional BIST Method", in Proc. Of the *International Test Conference (ITC) 2002*, pp. 590-598.

[5] F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero, "Fully Automatic Test Program Generation for Microprocessor Cores", in Proc. of the *Design Automation & Test in Europe (DATE) 2003*, pp.1006-1011.

[6] L. Chen, S. Ravi, A. Raghunathan, S. Dey, "A Scalable Software-Based Self-Testing Methodology for Programmable Processors", in Proc. Of the *Design Automation Conference (DAC) 2003*, pp. 548-553.

[7] N. Kranitis, A. Paschalis, D. Gizopoulos, G. Xenoulis, "Software-Based Self-Testing of Embedded Processors", *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 461-475, April 2005.

[8] A. Paschalis, D. Gizopoulos, "Effective software-based self-test strategies for on-line periodic testing of embedded processors", *IEEE Transactions on CAD*, Vol. 24, no.1, pp. 88 –99, Jan. 2005.

[9] N. Kranitis, A. Merentitis, N. Laoutaris, G. Theodorou, A. Paschalis, D. Gizopoulos, C. Halatsis, "Optimal periodic testing of intermittent faults in embedded pipeline processor applications", in Proc. Of the *Design, Automation and Test in Europe (DATE), 2006*, pp. 65-71.

[10] J. Zhou, H.-J. Wunderlich, "Software-Based Self-Test of Processors under Power Constraints" in Proc. Of the *Design, Automation and Test in Europe (DATE), 2006*, pp. 430-435.

[11] V. Tiwari, S. Malik, A. Wolf, "Compilation Techniques for Low Energy: An Overview", in Proc. Of *IEEE Symposium on Low Power Electronics (ISLPED), 1994*

[12] J. T. Russell, M. F. Jacome, "Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors", in Proc. of *Computer Design: VLSI in Computers and Processors (ICCD),* 1998, pp. 328-333.

[13] D. Brooks, V. Tiwari, M. Martonosi, "Wattch: a framework for architectural level power analysis and optimizations", in Proc. Of *International Symposium on High-Performance Computer Architecture (HPCA), 2000*, pp. 83- 94.

[14] M. Psarakis, D. Gizopoulos, M. Hatzimihail, A. Paschalis, A. Ragjunathan, S. Ravi, "Systematic Software-Based Self-Test for Pipelined Processors", *Design Automation Conference, 2006 (DAC)*, pp. 393-398.

[15] D. Brooks, P. Bose, M. Martonosi, "Power-performance simulation: design and validation strategies", in Proc. Of the ACM SIGMETRICS, vol. 31, pp. 13-18, March 2004.