VOLTaiRE: Low-cost Fault Detection Solutions for VLIW Microprocessors

Smitha Shyam, Sujay Phadke, Benjamin Lui, Hitesh Gupta,
Valeria Bertacco and David Blaauw
University of Michigan
Electrical Engineering and Computer Science Department
Ann Arbor, MI 48109-2122

(smithash, sphadke, blui, guptah, valeria, blaauw)@umich.edu

ABSTRACT

Reliability is emerging as a critical problem in microprocessor design due to aggressive technology scaling and architectural innovations required by ever increasing performance demands. In this paper we present VOLTaiRE, a **V**LIW processor with **O**n**L**ine **T**esting and **RE**configurability, which guarantees increased reliability of microprocessor datapaths in the presence of permanent defects and transistor wearouts. The processor includes a set of novel, distributed online property checkers, which monitor the correctness of the functional units during its normal operation. If any of the units are found to be defective, the processor ceases to use that unit and can reconfigure itself to function solely with the remaining working units by exploiting the built-in parallelism in VLIW architectures. This reconfigured design provides full functionality but degraded performance. We designed our experimental processor using a $0.18\mu m$ TSMC process with semi-custom and synthesized blocks. Experimental results show that the coverage provided by our online checkers is at least 93% for all functional units. Moreover, our post-layout analysis indicates that the area and power overhead associated with the property checkers is less than 7% of the processor core.

1. INTRODUCTION

The continuous race towards smaller and smaller feature sizes, combined with evolving architectural innovations, has made it possible for CMOS technology to maintain the performance growth outlined by Moore's law. However, technology scaling has significantly increased the susceptibility of digital circuits to permanent and transient faults, thereby adversely impacting circuit reliability [1]. Shrinking geometries, lower operating voltages and higher frequencies have resulted in a large increase in the number of occurrences of errors [3]. Consequently, the protection of complex VLSI designs against all possible error sources with minimal overhead has emerged as a primary challenge for semiconductor companies.

Fault events in digital VLSI circuits can be broadly classified into three types - permanent, intermittent and transient. Permanent errors reflect unalterable modifications to the original circuit and arise mainly due to manufacturing imperfections that result in opens or shorts. These faults are commonly modeled as wires being stuck-at a particular logical value. Design errors caused by an erroneous circuit implementation can also result in permanent faults. In contrast, intermittent faults are primarily influenced by time-dependent environmental factors such as temperature and operating voltage. Electromigration and gate-oxide breakdown are two phenomena that result in sporadic errors and can eventually lead to permanent faults [2]. Transient faults, such as soft errors, occur when particle-induced transient effects are generated, due to the interaction of particles with the semiconductor material. Digital noise due to capacitive crosstalk and power supply variations also affect both signal integrity and performance of digital circuits.

Designing an integrated circuit so that it is protected against permanent, transient and intermittent faults is a daunting and time consuming task due to the large number of fault sources, as well as their unpredictable impact on circuit operation. To protect against design errors, functional verification is deployed to an extent that is today a considerable portion of the total design time and effort. Mainstream techniques in this domain are both directed random simulation [12] and formal [13, 14] and semi-formal solutions [15], all aiming at the common goal of achieving the highest level of confidence in the correctness of a design. For manufacturing defects, non-concurrent testing mechanisms are commonly used. However, the necessary equipment is very expensive [4]. Finally, transient errors are inherently non-testable due to their fleeting nature.

All these challenges necessitate the need for fast and efficient online reliability solutions that can ensure the correctness of a design during its entire lifetime, with low cost and performance impact. The main advantage of online techniques is that they can detect failures while the processor is operational; hence they can monitor the correctness of a design after it is deployed in the field. They can be designed to detect and recover from any of the type of faults describe above. Classic examples of such solutions are concurrent error detection (CED) schemes [16] , in addition to the more recent trend of incorporating a small amount of synthesizable assertions in a design with the purpose of detecting unverified potential design errors [8]. The main drawback of currently available online techniques is the associated area and power overhead.

In this paper we detail an online reliability solution, VOLTaiRE, that ensures reliable operation of datapath functional units by incorporating property checkers in a Very Large Instruction Word (VLIW) architecture. The main benefits of our solution are minimal area and power overhead. In addition, even though our solutions are detailed in the context of a VLIW processor, the detection mechanisms can be easily altered to suit other types of processors.

1.1 Contributions of this Paper

VOLTaiRE addresses the reliability concerns discussed above to ensure correct operation of the datapaths on a fixed-point multi-issue VLIW processor. The architecture of VOLTaiRE is innovative in two aspects:

- The design features online property checkers that verify the processor's computations while it is operational. These low-cost distributed property checkers provide run-time identification and diagnosis of silicon defects.

- The microarchitecture can reconfigure itself to respond to failures in the design. To achieve this, VOLTaiRE exploits the multiplicity of the resources in the VLIW processor and uses only the functional units that are still working properly to do the computations.

VOLTaiRE is capable of detecting and correcting most of the permanent faults in the processor datapaths. In addition, VOLTaiRE is also able to detect design errors but it can not correct them. In the event of permanent faults, VOLTaiRE first reconfigures the pipeline and then corrects the erroneous data generated by recomputing the faulty operation. Furthermore, VOLTaiRE differentiates between transient errors caused by single event upsets and permanent faults so that the reconfiguration is performed only in the latter case. While in reconfigured mode, VOLTaiRE functions with a reduced level of performance since one or more of its functional units are disabled. The extent of performance degradation will depend on the workload being executed. In the proposed chip implementation, we obtain an average fault coverage of 94% for the ALUs and 100% for the multipliers while incurring area/power overheads of less than 7% of the original processor core.

This paper is organized as follows. Section 2 discusses the various approaches for online testing and the relative merits of these techniques. An architectural overview of the system in the context of property checkers is presented in Sections 3 and 4. Section 5 proposes the property checkers and the reconfigurability features of VOLTaiRE. Section 6 discusses the heuristics developed for evaluating the coverage provided by the checkers and the results obtained. Section 7 provides chip statistics and floor planning. Finally, we conclude in Section 8.

2. RELATED WORK

In the context of online testing of processors, various concurrent error detection (CED) schemes have been proposed in the literature [16]. All CED schemes incorporate an output characteristic predictor that predicts some special characteristic of the output, and a checker that compares the expected behavior with that of the actual output and generates an error when a mismatch is detected. Duplication is the simplest way of detecting errors. It has been suggested in [16] that a diverse implementation of the logic is a more robust approach compared to plain duplication as it offers protection against multiple failures due to a common source. Other solutions proposed in the direction of online testing are the Berger codes [17], that can detect all unidirectional errors, and Bose-Lin codes [18] that can detect up to t unidirectional errors (known as t-EC). These codes are suitable for the reliability of systems that have large occurrences of one kind of binary numbers as in memories. However, it is not easy to use these codes for online testing of datapaths, as they impose constraints on the way the logic blocks are designed so that only unidirectional faults can occur. As an example, to detect up to 2 unidirectional errors, Bose-Lin codes need a logic block to be inverter-free and that each logic gate has a fanout of less than three.

Parity prediction schemes have been used for ensuring fault-secure datapaths [19]. Here the multipliers and ALUs are constructed using fault-secure adders that act as building blocks. The inherent drawback of both unidirectional codes and parity prediction techniques is that they place heavy restrictions on the implementation of the hardware, resulting in an area overhead that often

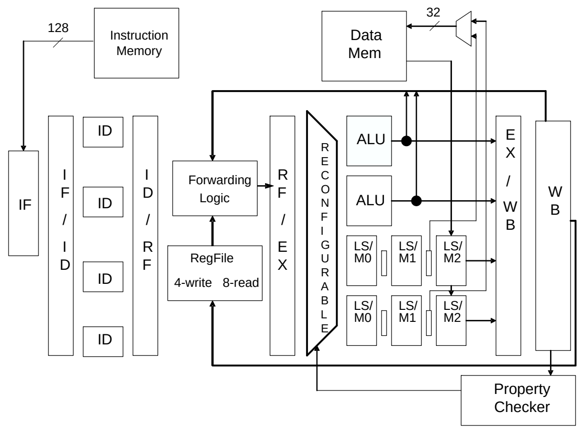

Figure 1: Block Diagram of VOLTaiRE. The processor is based on an Alpha architecture, which we have augmented with detection and correction mechanisms.

compares to, or in some cases is even greater than, that of plain duplication [20]. Further, these hardware restrictions have an additional negative impact on system performance.

A recent trend in hardware design has been exploring "better than worst-case" solutions, that is, design solutions that can relax some of the classic design constraints, such as timing, correctness, voltage, etc., while still producing properly working hardware systems. Some of the solutions proposed in this domain can detect and correct design errors and/or permanent faults. For instance, Razor [6] provides correction of timing-related errors under process variations. However, the detection and correction mechanisms of Razor can only address timing-related errors and can not detect permanent defects or time-dependent-device-breakdown (TDDB). DIVA [7] on the other hand, uses a complete, although streamlined processor to ensure correct operation in face of both design errors and faults, thus consuming a considerable amount of power and area.

VOLTaiRE is different from previous approaches as it can detect and respond to a wide range of defects with minimal area and power overhead. The on-chip property checkers check for permanent errors that have escaped post-silicon detection as well as errors due to device wearouts. Further, VOLTaiRE does not impose any restrictions on the implementation of the normal datapath except for making provisions for reconfigurability. VOLTaiRE does not need to replicate an entire datapath. Instead, it embeds sufficient property checks on-chip to ensure the datapath correctness. Additionally, the distributed nature of the VOLTaiRE checkers provides accurate detection and diagnosis of defects, at area/power overheads less than that of even the most efficient previous techniques (e.g., DIVA). Finally, it takes advantage of the inherent redundacy of VLIW architectures to reconfigure a faulty design rather than relying on an extra simple processor or a redundant unit.

3. VOLTAIRE DESIGN DESCRIPTION

VOLTaiRE is a 4-way, 32-bit fixed point VLIW processor whose instruction set is loosely based on the Alpha instruction set. The block diagram of VOLTaiRE with the various modules and property checks is depicted in Figure 1. We first provide an overview of the architecture and then present the checker modules that ensure the reliability of the datapaths.

Each VLIW instruction is 128-bits long and consists of 4 independent 32-bit instructions. In VLIW architectures, the burden of grouping independent instructions into a single bundle is placed on the compiler. Hence the target micro-architecture is guaranteed that all the incoming instructions in a bundle can be executed in parallel. VOLTaiRE has two identical pipelines for carrying out ALU instructions and two for load/store/multiply (LSM) instructions. Hence, each VLIW instruction bundle can be composed of at most two ALU instructions and two LSM instructions. When a compiler can not completely fill a bundle with independent instructions, NOPs are used as fillers.

Our baseline processor has five pipeline stages: The instruction fetch (IF) stage is responsible for fetching a 128-bit VLIW instruction bundle from memory. The instruction decode (ID) stage decodes 4 independent instructions per cycle. The Register File (RF) stage is involved with register accesses for the instructions. We have an 8-ported-read and 4-ported-write register file to cater to the 4-way pipeline in a single cycle. The 8 read ports are necessary because each instruction needs at most 2 reads and 4 instructions are executed in parallel in every cycle. The 4 write ports are needed to facilitate the 4 instructions retiring and writing their results simultaneously. The execute (EX) stage consists of arithmetic logic units and load/store/multiply units. The multiplier used in the VOLTaiRE processor is 16-bit wide and produces a 32-bit result, as opposed to the ALU and load/store units which are 32-bits wide. ALU instructions take one cycle to execute, while the instructions using the LSM units take three cycles. This is shown in the schematic in Figure 1 where the LSM units are separated by two internal pipeline registers. The execute stage also houses the reconfiguration module. It is responsible for switching the data paths between the 2 ALU and/or the 2 LSM units. Naturally, data paths cannot be interchangeable between an ALU and a LSM unit.

The Write Back (WB) stage writes the results of the EX stage into the register file. Also, the WB stage provides an extra buffer zone for the property checkers to validate the results of the EX stage. As data is not written back to the register file until the property checker validates the data, execution of instructions up to the writeback stage can be seen as speculative.

4. VOLTAIRE FAULT-DETECTION MODEL

From an architectural perspective, we envision VOLTaiRE as a fault-secure reliable VLIW processor. VOLTaiRE can detect and correct errors resulting from silicon defects and wearout faults. In VOLTaiRE, the property checkers continuously monitor the datapaths for errors. The status of each functional unit is stored in the processor status register (PSR). When an error is detected in one of the functional units, the corresponding bit in the status register is set and the instruction is flushed. The entire pipeline is reconfigured and the failing instruction is recomputed using another functional units.

The correct identification of errors is critical. To avoid the corruption of data from soft errors, the status register's bits are updated only if an execution unit fails twice executing the same instruction. That is, the first time a fault in an execution unit is flagged by the property checkers, the pipeline is flushed and the instruction is re-executed. This way, transient faults in the checker or in the processor will not cause to incorrectly update the status registers and mark a unit as faulty, and thus hamper the processor's performance.

It is possible that a few errors occur in the datapaths that escape detection. Our goal is to detect errors within a reasonable timeframe while minimizing the overhead associated with the correctness checks, and imposing no restrictions on the hardware implementation of the processor.

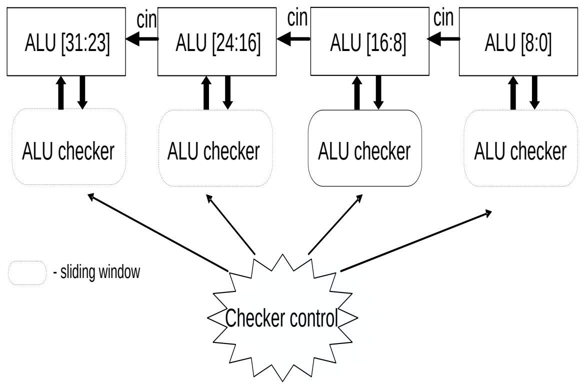

Figure 2: The ALU property Checker uses a sliding window mechanism to evaluate the correct operation of a 32-bit ALU using only a 9-bit mini-ALU.

5. VOLTAIRE RELIABILITY FEATURES

A novel feature of VOLTaiRE is the use of property checkers to verify the correct functionality of datapaths. Properties are conditions that are always satisfied during the normal operation of a processor. For instance, the output of an adder should always be equal to the sum of the operands. In general, a property could be a mathematical formulation of a relationship between outputs and inputs of a logic block, or it could state a specific characteristic relation among certain internal signals, or it could impose constraints on latency, timing or a combination of these. Property checkers can be deployed at many distinct locations in a design: to monitor signals within a logic block, to guarantee correct communication among blocks, and/or check correct functionality at the system level.

On-chip property checkers do not undermine the importance of verification, system level testing or post manufacturing testing. However, by having properties placed on-chip, we can validate the correctness of the design within the input space the user is interested in and, due to the ability to reconfigure our design we can rectify the system when faults occur.

Our property checkers are in-charge of flagging faulty execution units. They check the results of the datapath computations based on arithmetic properties that can be used to verify them. We have developed mini-ALUs for verifying the arithmetic logic units and boundary and residue checkers for the multiplication units.

5.1 ALU Checker

The computations of the 32-bit ALUs of VOLTaiRE are checked using sliding-window 9-bit mini-ALUs. As shown in Figure 2, during each instruction cycle, a mini-ALU checks 9-bits out of the 32-bit result, keeping one bit overlap in each 9-bit window. For instance, during the first cycle the mini-ALU verifies the least significant 9 bits of the ALU result and in the following cycle, it verifies the correctness of bits 8 to 16 of the ALU result, using an overlap of one bit. The checker's window slides up the 32-bit ladder until it reaches the most significant 9 bits (24 to 31), and then, restarts again from the lowest 9-bits. We use this overlap mechanism because it allows us to simplify the checker implementation, since we can avoid realizing the input carry logic in the mini-ALU checker. The same type of ALU checker is also used to validate the address generation logic in the load/store units.

An alternative to our solution could have been the use of residue checkers, which have been widely used for detecting errors with arithmetic operations. However, residue checkers cannot be used for verifying logical operations. Hence, we chose to verify the ALU result in small portions using a mini-ALU. It must be noted that, in comparison to redundancy techniques which duplicate an ALU and incur high area and power penalties, our mini-ALU is much smaller and simpler to verify.

5.2 Multiplier Property Checker

Multiplications, in general, are more computationally intensive than ALU operations. Consequently, our strategy of small windows of verification, used for the ALU checker, is not feasible for the multiplier units. Hence, we have devised two ad-hoc techniques specific for multipliers, namely bounded checker and residual checker.

5.2.1 Boundary Checker

Boundary checking is a novel, simple, yet highly effective method to test the bounds on a given multiplication operation. It is based on the idea of using the nearest powers of two of a multiplication's result to determine the bounds on the product. With this technique, we execute a fast compare of the result of the circuit implementation with that of the checker unit to determine whether the produced result is within the correct bounds.

Our technique starts by first generating upper and lower boundaries for the two 32-bit input operands of a multiplication, say A and B. For simplicity of explanation, let us first consider the case where the two operands are positive. Clearly, there exist integers i and j such that

$$2^i \leq A < 2^{i+1} \tag{1}$$
$$2^j \leq B < 2^{j+1} \tag{2}$$

We then observe that the following two inequalities are valid:

$$A*2^j \leq A*B < A*2^{j+1} \tag{3}$$
$$B*2^i \leq A*B < B*2^{i+1} \tag{4}$$

This pair of inequalities provide bounds on the product $A*B$, that are efficient to compute. For instance, suppose that $A = 5$ and $B = 11$; the Equations (3) and (4) above would evaluate to: $5*8 \leq 55 < 5*16$ and $11*4 \leq 55 < 11*8$. In case one operand, say A, is a negative value, then we can still find a positive integer i such that $-2^{i+1} \leq A < -2^i$, and our inequalities can be derived in the same fashion by just substituting these two boundary values for A. The Equations would become:

$$A*2^{j+1} \leq A*B < A*2^j \tag{5}$$
$$-B*2^{i+1} \leq A*B < -B*2^i \tag{6}$$

A similar reasoning holds when both operands are negative.

The presence of perfect powers of 2 in the checker enables an efficient hardware implementation to determine the bounds required. First, the integers i and j can be calculated by counting the number of significant digits. The four multiply operations required to generate the bounds have 2^n as one of the operands. Hence, they can be replaced by shift operations. The case where we need to multiply an operand by a power of two which is also a negative value can be easily adapted to require simply a partial bit inversion and a shift operation. The inequalities are checked using simple comparators. In order to reduce the performance penalty, we split these operations across different pipeline stages so that the bound-checking operation does not impact the critical path. In particular, the shift operations are completed in the execute stage, while the comparisons are performed in the write-back stage. If an error is flagged by a comparator, we first identify the source multiplier that computed this faulty product. We then flush the instruction and re-compute the multiplication using a different multiplier unit.

Boundary checkers are particularly efficient for multiplications involving small positive integers since they would generate a tight boundary constraint. Fortunately, it has been observed that a major portion of the arithmetic operations occurring in a processor's typical data load involve small operands [11]. Consequently our boundary checker technique has proven to be an effective method to evaluate the correctness of multiplier units.

5.2.2 Residue Checker

In addition to the boundary checkers, we also use arithmetic residue checkers to validate the correct functionality of the multiplier units [9]. Given an n-bit operand X, its residue X_r with respect to r is the result of the operation X modulo r. When applied to multiplication, residue codes adhere to the following property:

$$(X_r * Y_r)_r = (X*Y)_r \tag{7}$$

Moreover, when the value of $r = 2^a - 1$, for a given a of choice, the residue operation is simple to implement in hardware. Hence the name "low-cost residue code". For our checker implementation, we used $a = 2$ and the value of the residue is $r = 3$.

Residue codes can detect most faults in the multiplier except those that manifest as multiples of the residue. As our choice of residue is 3, we can detect each single-bit error in a multiplication result. However, it is possible that errors impacting two of more output bits could go undetected by the residue checker. Note that our fault model focuses only on single faults. However, since we have not imposed any implementation restrictions on the multiplier, a single fault at an internal node could manifest as multiple-bit errors in the output. If a multiple-bit error is such that the difference between the actual and incorrect result is a multiple of the residue, then the error goes undetected. For instance, if the correct result of multiplication were 16 and we obtain an output of 19, the checker would not detect the error since $16_3 = 1 = 19_3$. Fortunately, most errors missed by the residue checker are caught by the boundary checker and the two checkers used in tandem provide a good overall coverage for any internal fault in the multiplier.

Ensuring the correctness of checkers is very important in order to avoid false negatives. As the checkers are trivial compared to the actual functional blocks that they are monitoring they can be functionally verified completely by formal verification tools [14]. In addition, the comparators used in the checkers could be made dual-rail to obtain completely self-checking checkers [10]. The checkers in VOLTaiRE are very small and are physically located away from the execution units to reduce the probability of manufacturing defects occurring in both structures. Furthermore, these checkers do not need to be resilient to soft-error strikes, since we re-compute an erroneous result before flagging a unit as defective.

5.3 Reconfigurability

The status of each functional unit is stored in the processor status register (PSR). When an error is detected in one of the functional units, the corresponding bit in the status register is set and the instruction is flushed. The entire pipeline is reconfigured and the failing instruction is recomputed using one of the other functional units. The ALU instructions are single cycle and if one of them breaks down, we stall the pipeline every other clock cycle to allow the other functional ALU to take on the extra burden. On the other

Operation	Total nets	Observed F	Detected F	Percent
Add st@0	380	343	341	99.42%
Add st@1	380	335	333	99.40%
Xor st@0	380	57	57	100%
Xor st@1	380	39	39	100%
Mul st@0	1138	1136	1136	100%
Mul st@1	1138	1125	1125	100%

Table 1: Property Checker Coverage for Random Inputs.

Operation	Total nets	Observed F	Detected F	Percent
Add st@0	380	252	236	93.65%
Add st@1	380	231	222	96.10%
Mul st@0	1138	918	918	100%
Mul st@1	1138	1118	1118	100%

Table 2: Property Checker Coverage for Typical-case Inputs.

hand, if one of LSM units breaks down, VOLTaiRE suffers an additional 3-cycle penalty. Note that an execution unit is pronounced faulty only after ensuring that the failing instruction produced an erroneous result twice. This circumvents the false negatives that could otherwise be obtained from single event upsets. The datapath is reconfigured and the processor may operate at reduced performance depending on the program workload.

6. VOLTAIRE COVERAGE ANALYSIS

Quantifying the coverage provided by the property checkers is an intricate task when developing an on-line fault detection architecture. We analyzed the VOLTaiRE design for shorts, stuck-at-0 and stuck-at-1 faults. Each of the injected fault was simulated with 1000 test inputs, both random and typical-case data. Note that, not all injected faults are going to result in false computations. We recorded the number of valid faults, the faults caught by the property checkers, and first times these faults were caught. Tables 1, 2 and 3 show the results of the simulations.

Tables 1 and 2 provide the results of coverage for random and typical-case input data, respectively. Here, the "Total nets" column is the total number of nets where faults can occur. The next column lists the number of nets where the presence of faults actually caused an incorrect output. The last two columns report the absolute value and the fraction. The results are presented for both random data and typical-case data extracted from the SPEC2000 benchmark suite. For each functional unit we analyzed the resiliency of both stuck-at-0 (st@0) and stuck-at-1 (st@1) faults. For the ALU unit, the coverage was evaluated on the ADD and XOR instructions. Note that for the faults that were missed in the ALU ADD instructions, the simulation results showed that these inputs had a very low frequency, less than 0.01% probability of appearing in a benchmark.

Furthermore, it can be seen that the ALU property checker, mini-ALU, gives better coverage for random data than typical-case data in the case of ADD instructions, as this checker is based on the property of a sliding window mechanism. This result is expected as typical-case data is biased towards smaller values, whereas random data is unbiased.

The multiplier makes use of two types checks, one of which is favorable to smaller data values. Consequently, we observe that we can achieve 100% coverage for both random and typical-case data. The residue checker used for the multiplier can be used to obtain 100% coverage for the ADD instructions, but these checks cannot verify logical operations.

In Table 3, the multiplier coverage results are further elaborated.

Multiplier operation	Observed faults	Detected faults	Percent detected	Bounded alone	Residue alone	Both checkers
Rand-st@0	381622	379983	99.57%	0.65%	89.62%	9.30%
Rand-st@1	704416	703989	99.93%	0.02%	90.11%	9.80%
Typ-st@0	190520	189986	99.72%	0.70%	59.38%	39.64%
Typ-st@1	887538	887393	99.98%	0.0%	30.93%	69.05%

Table 3: Multiplier Property Checker Coverage.

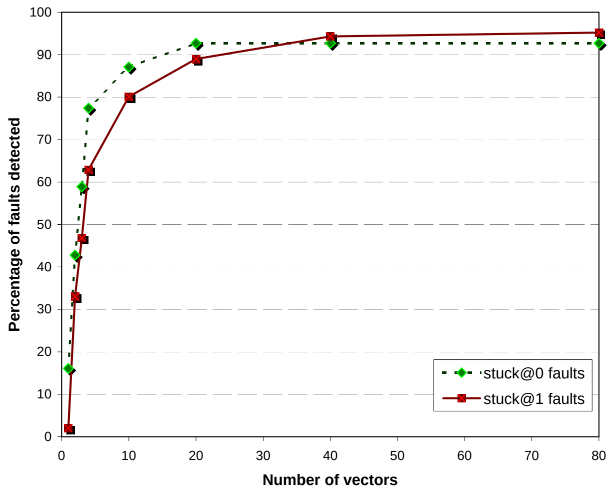

Figure 3: Cumulative plot of the first detection trend for the ALU property checker for typical-case data.

Every fault that can occur in a particular network is simulated with 1000 random and typical-case data. The table reports the number of these observed faults in column "Observed" and the number of faults caught by the checkers in "Detected". It also reports the percentage of the detected faults caught by the bounded checker alone and not by the residue checker in column "Bounded alone", and vice-versa in "Residue alone". Column "Both checkers" gives the percentage of faults detected by both checkers. Even though the contribution of bounded checks is small when compared to residue checks, it is nevertheless needed to obtain improved coverage.

Figure 3 shows the first times the ALU property checkers detected an error in an operation's result. For example, the ALU checker caught 89% of the stuck-at-1 faults and 91% of the stuck-at-0 faults within their 20th appearance of the incorrect propagation to an output value. The corresponding data for the multiplier property checkers is not shown in the plot as they detected all faults within the 3rd appearance of a faulty result.

7. VOLTAIRE CHIP CHARACTERISTICS

The VOLTaiRE 4-wide VLIW prototype was implemented in synthesizable Verilog and it was synthesized for minimum delay using Synopsys's Design Compiler. This produces a structural verilog specification of the processor implemented with Artisan's standard cells in TSMC 0.18um fabrication technology. The design was then placed and routed using Cadence Sedsm, which in turn yields a physical design with wire capacitances. The custom SRAM blocks and several circuits within the chip-to-chip communication module required custom designs and were specified at the schematic level using Virtuoso Schematic Editor and Virtuoso Layout Editor. The design was back-annotated to get a more accurate delay profile and simulated to verify timing and functional correctness. Layouts at

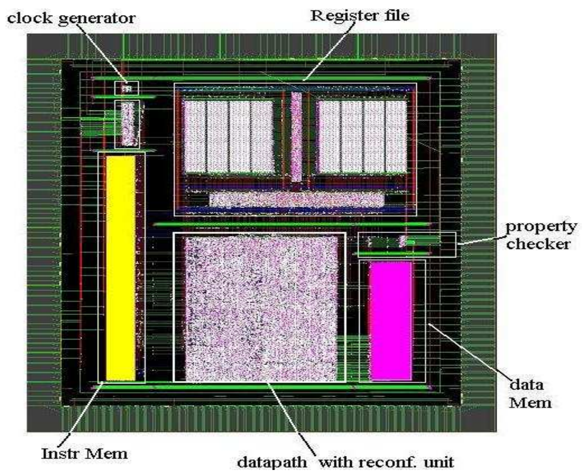

Figure 4: System-level final layout of the processor.

Module	Area(mm^2)	Power(W)
Fetch	0.057	0.029
Decode	0.066	0.052
RF	1.700	0.200
Execute	1.200	1.710
Write	0.224	0.075
AluChk	0.084	0.050
MulChk	0.154	0.089

Table 4: VOLTaiRE absolute area and power consumption by functional unit.

both the block-level and global-level were verified for correctness using Mentor Graphics Calibre DRC and LVS tools.

The layout of VOLTaiRE is shown in Figure 4. During the implementation of VOLTaiRE from architecture to layout, several considerations were necessary. In order to have no latency during the register access stage, we built a semi-custom 8-port read and 4-port write register file. Several clever techniques were used to reduce timing delay during register access. The reconfigurable data path is implemented in the execution stage. To reduce critical path delay, most of the control signals needed for reconfiguration are generated in parallel in the RF stage itself. The checkers are also distributed over the execute and write-back stages for performance reasons. The clock tree at the global level is a first order balanced structure. The chip specifications for VOLTaiRE are:

- operating frequency is 280MHz.
- operating voltage is 1.8 V.
- technology is TSMC 0.18μm.
- total area of the chip is 2.5mm X 2.9mm.
- number of PADS is 120.

In Table 4, the area and power measurements of the property checkers can be compared to the rest of the pipeline stages. Notice that the property checkers form a minimal part of the pipeline. Together they are responsible for 6.9% additional area and 6.3% additional power for the processor core. Note that the execution stage

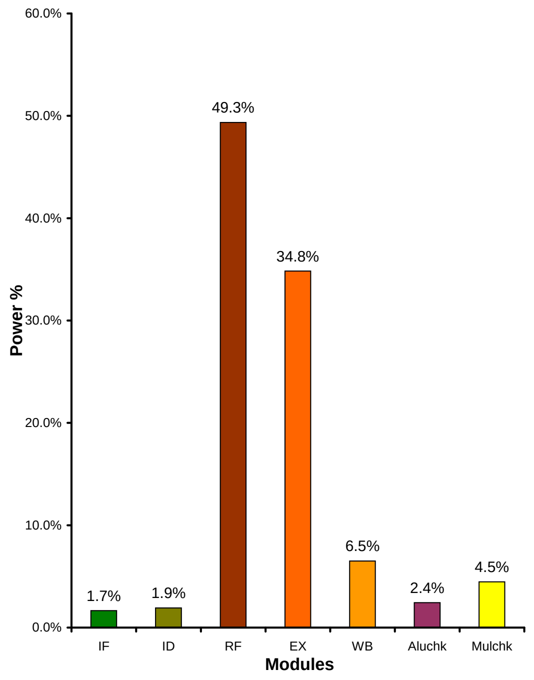

Figure 5: Distribution of the total area of the core.

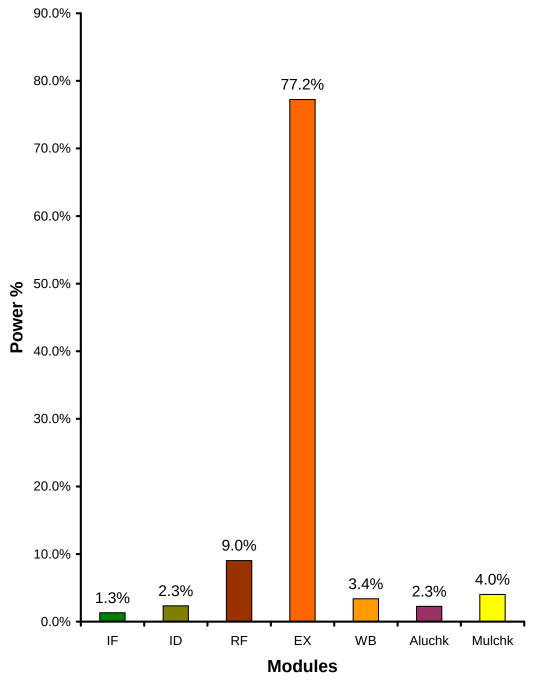

Figure 6: Distribution of the total power of the core.

contributes to the maximum power because of the multiple execution units. Figures 5 and 6 show the contribution of each individual module to the area and power of the processor core, respectively.

8. CONCLUSIONS AND FUTURE WORK

In this paper we present a novel VLIW microprocessor design with a low cost fault detection solution for datapaths. Our solution deploys a series of efficient online property checkers that ensure functional correctness while incurring minimal area and power overhead. When a functional unit fails to operate correctly, our design can reconfigure itself to isolate the faulty unit and continue to be fully operational at a reduced level of performance. Experimental results show that we achieve a high level of test coverage (above 93%) with an area and power penalty of less than 7%.

In future we plan to extend the reconfigurability aspect to all the control and memory units of the design. While checkers for the datapath were designed using simple mathematical properties, checkers for the control unit will require sophisticated analysis.

9. REFERENCES

[1] Critical reliability challenges for the International Technology Roadmap for Semiconductors. International Sematech Tech Transfer 03024377A-TR, 2003.

[2] J. Srinivasan, S. V. Adve, P. Bose and J. A. Rivers. The impact of technology scaling on lifetime reliability. In Proc. of Intl. Conference on Dependable Systems and Networks, June 2004.

[3] C. Constantinescu. Trends and challenges in VLSI circuit reliability. In Proc of IEEE Micro, vol 23,no 4.pp 14-19, July/August 2003.

[4] K. Cheng, S. Dey, M. Rodgers and K. Roy. Test challenges for deep sub-micron technologies. In Proc. of 37th Design Automation Conference, 2000

[5] S. Borkar. Design challenges of technology scaling. In Proc of IEEE Micro July/August 1999.

[6] D. Ernst, N. Kim, S.Das et al. Razor: A low-power pipeline based on circuit-level timing speculation. In Proc of IEEE Micro, Nov 2003.

[7] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design.", In Proc of Dependable Systems and Networks (DSN), 2001.

[8] A. Bayazi and S. Malik. Complementary use of runtime validation and model checking, In Proc of International Conference for Computer Aided Design, 2005.

[9] A. Avizienis. Arithmetic Error Codes: cost and effectiveness studies for application in digital system design, IEEE Trans. Computers, Vol. C-20, No. 11, pp. 1322-1331, Nov. 1971.

[10] E. J. McCluskey. Design techniques for testable embedded error checkers, IEEE Computer, Vol. 23, No. 7, pp. 84-88, July 1990.

[11] T. Austin, V. Bertacco, D. Blaauw and T. Mudge. Opportunities and challenges for better than worst-case design, in Proc of Asia-South Pacific Design Automation Conference, Jan 2005.

[12] J. Bergeron. Writing test benches: functional verification of HDL models. Kluwer Academic Publishers, 2nd edition, 2003.

[13] K. McMillan. Applying SAT methods in unbounded symbolic model checking. In Proc. of Computer Aided Verification Conference, LNCS vol. 2404, July 2002.

[14] A. Biere, A. Cimatti, E. Clarke and Y. Zhu. Symbolic model checking without BDDs. In Proc. of Tools and Algorithms for the Analysis and Construction of Systems, LNCS vol. 1579, 1999.

[15] P. H. Hu, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor and J. Long. Smart simulation using collaborative formal and simulation engines. In Proc of International Conference for Computer Aided Design, 2000.

[16] S. Mitra and E. J. McCluskey. Which concurrent detection scheme to choose? In Proc. of Intl. Test Conference, Oct 2000.

[17] J. M. Berger. A note on an error detection code for asymmetric channels, Information and Control, Mar 1961.

[18] B. Bose and D. J. Lin. Systematic unidirectional error-detecting codes. In the Proc of IEEE Trans on Computers, Nov 1985.

[19] M. Nicolaidis, R. Duarte, S. Manich and J. Figueras. Fault-secure parity prediction arithmetic operators, IEEE Design and Test of Computers, 1997.

[20] D. Das and N. A. Touba. Synthesis of circuits with low-cost concurrent error detection based on Bose-Lin codes. VLSI Test Symposium, May 1998.