Configurable Fault-Tolerance for a Configurable VLIW Processor

Fakhar Anjam and Stephan Wong

Computer Engineering Laboratory,
Delft University of Technology,
Mekelweg 4, 2628CD, Delft, The Netherlands
Email: {F.Anjam, J.S.S.M.Wong}@tudelft.nl

Abstract. This paper presents the design and implementation of configurable fault-tolerance techniques for a configurable VLIW processor. The processor can be configured for 2, 4, or 8 issue-slots with different types of execution functional units (FUs), and its instruction set architecture (ISA) is based on the VEX ISA. Separate techniques are employed to protect different modules of the processor from single event upsets (SEU) errors. Parity checking is utilized to detect errors in the instruction and data memories and the general register file (GR), while triple modular redundancy (TMR) approach is employed for all the synchronous flip-flops (FFs). At design-time, a user can choose between the standard non fault-tolerant design, a fault-tolerant design where the fault tolerance is permanently enabled, and a fault-tolerant design where the fault tolerance can be enabled and disabled at run-time. These options enable a user to trade-off between hardware resources, performance, and power consumption. A simulation based technique is utilized for testing purposes. The processor is implemented in a Xilinx Virtex-6 FPGA as well as synthesized to a 90 nm ASIC technology. Compared to the permanently enabled fault-tolerance, in scenarios, where fault-tolerance is not required at some point in time, considerable power savings (up to 25.93% for the FPGA and 70.22% for the ASIC) can be achieved by disabling the fault-tolerance at run-time.

Keywords: Softcore, VLIW processor, SEU error, Configurable fault-tolerance

1 Introduction

Very long instruction word (VLIW) processors exploit instruction level parallelism (ILP) by means of a compiler. A VLIW processor can execute multiple operations (a long instruction) per cycle to increase performance [5]. When the data path of a processor gets larger and complex, the probability of errors (such as radiation-induced soft errors) also increases. Because VLIW processors can provide high performance at low power, they are gaining wide-spread utilization not only in general-purpose embedded systems but also in safety-critical

systems such as biomedical, space, military, communication, industrial, and automotive systems. Therefore, it is important to employ fault-tolerant techniques in these processors for guaranteeing high reliability and dependability of the safety-critical systems. Run-time detection plays an important role in dependable systems, where it is needed that the computed data is either correct or an error signal is generated whenever there is a possible error.

In this paper, we present configurable fault-tolerant techniques for a softcore VLIW processor (ρ-VEX) [21]. The processor is parameterized and different parameters such as the issue-width, the number and types of different FUs, number of registers, memory buses, and latencies for the FUs can be chosen at design-time. The processor is implemented in VHDL and the fault-tolerance techniques are implemented at hardware level. The processor employs different fault-tolerance techniques such as parity checking and TMR to increase the reliability and dependability of the system. The processor is implemented in a Xilinx Virtex-6 FPGA as well as synthesized to a 90 nm ASIC technology.

Apart from the general parameters such as the issue-width, number of FUs, etc., the fault-tolerance is also configurable. At design-time, a user can choose to implement a processor with no fault-tolerance, a processor with the fault-tolerance permanently enabled, or run-time configurable. The permanently enabled and the run-time configurable designs consume almost similar dynamic power. The advantage of the latter design is that the fault-tolerance can be disabled at run-time, resulting in reduced dynamic power consumption (25.93%, 12.43%, and 5.55% for the FPGA and 65.92%, 67.30%, and 70.22% for the ASIC for the 2-, 4-, and 8-issue processors, respectively). The fault-tolerance can be enabled/disabled by executing an instruction on the processor. For applications which can tolerate some bit flips such as audio/video decoding, the fault-tolerance can be disabled at run-time to reduce the dynamic power consumption. On the other hand, applications which are susceptible to even a single bit flip such as sending/receiving DTMF tones on a mobile device, can enable the fault-tolerance at run-time. The configurable processor provides a trade-off for hardware resources, performance, and power consumption.

The remainder of the paper is organized as follows. Related work is discussed in Section 2. Section 3 briefly introduces our configurable ρ-VEX softcore VLIW processor. The fault-tolerant design of the ρ-VEX processor is presented in Section 4. Experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

Recently, fault-tolerance for microprocessor systems is gaining increasing importance. Transient errors are considered as the main source of errors in processor systems. Different on-line detection and mitigation techniques are proposed for detecting and correcting transient error faults. These techniques are mainly based on the redundancy approaches. In these techniques, instructions are replicated and computed, and then the results of the original and the duplicated

instructions are compared to check for errors. Mainly, there are two approaches for redundancy; software-based and hardware-based.

A software-based redundancy approach utilizes a compiler to generate duplicate/triplicate instructions. This approach increases code size and power consumption and reduces performance [13]. The advantage is that no hardware modification is needed. Compiler-based software redundancy schemes with the effect of increased code size and performance degradation are presented in [3][10]. Similar techniques to detect errors in VLIW and superscalar processors are discussed in [9][2][14]. A software method to detect transient and common-mode faults in statically-scheduled VLIW processor in presented in [18].

A hardware-based redundancy approach requires changes to the architecture and additional hardware for managing replication, re-computation, and comparing results to detect errors. The advantage is that there is no need to change the code or the compiler, and that there is little or no performance degradation and no code size overhead. At the hardware level, one solution is to replicate the complete processor system, and then implement a majority voter to select between the three results [11][20]. In that case, there is no need to change the processor architecture, with the disadvantage that a fine-grain control over instruction-level checking is not possible. Another solution is to modify the architecture, implement additional FUs and other control hardware units to perform the execution of replicated instructions [7][15][16]. A technique that utilizes additional FUs to detect and correct transient errors generated in combinational logic is presented in [4]. The author in [8] triplicates the sequential elements in the processor to detect and correct SEU errors. Recently, hybrid approaches (software and hardware) for error detection and correction were presented in [4][17].

3 The Configurable ρ-VEX VLIW Processor

The VEX ISA, developed by the Hewlett-Packard (HP) and STMicroelectronics [6] is a 32-bit clustered VLIW ISA that is scalable and customizable to individual application domains. The ISA is loosely modeled on the ISA of the HP/ST Lx (ST200) family of VLIW embedded cores [5]. Based on *trace scheduling*, the VEX C compiler is a parameterized ISO/C89 compiler. A flexible programmable machine model determines the target architecture, which is provided as input to the compiler. The compiler reads a machine configuration file and schedules the code according to it. A VEX software toolchain including the VEX C compiler and the VEX simulator is made freely available by HP Laboratories [1].

The ρ-VEX is a configurable (design-time) open-source softcore VLIW processor [21]. The ISA is based on the VEX ISA [6]. Different parameters of the processor such as the issue-width, the number and type of different FUs, supported instructions, memory-bandwidth, register file size etc., can be chosen at design time. The processor is a 5-stage pipelined processor consisting of *fetch*, *decode*, *execute0*, *execute1/memory*, and *writeback* stages. Regular functional units include arithmetic logic units ($ALUs$), multiplier units ($MULs$), branch or control unit ($CTRL$), and load/store or memory unit (MEM). The fetch stage

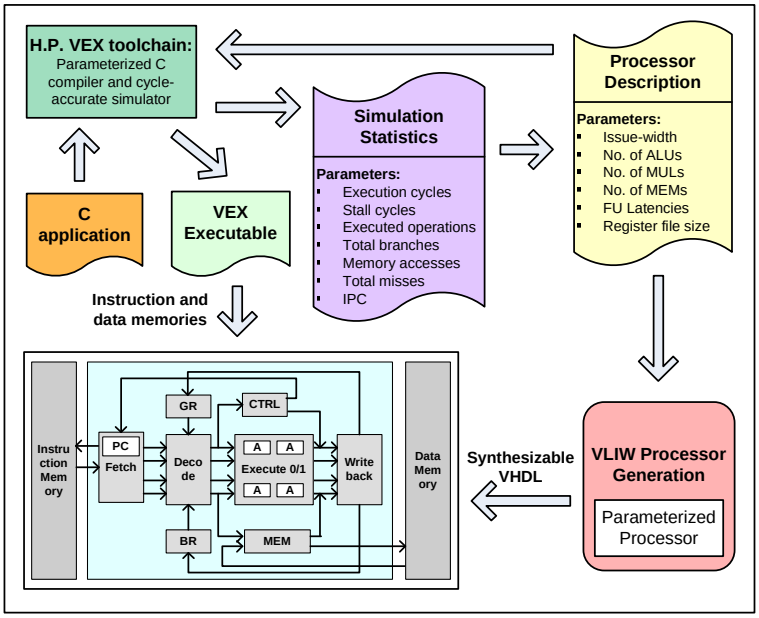


Fig. 1. Methodology to generate a ρ-VEX VLIW processor.

fetches a VLIW instruction from the attached instruction memory, and splits it into syllables that are passed on to the decode stage. Here, instructions are decoded and register contents used as operands are fetched from the GR register file. Branch operations take place in the CTRL unit located in the decode stage. The ALU/MUL operations take place in either the execute0 (1-cycle latency) and execute1 (2-cycle latency) stages. Load/store operations takes place in the MEM units in the execute1/memory stage. All write activities are performed in the writeback stage to ensure that all targets are written back at the same time. The processor has a 64×32-bit multiported general register file (GR) and an 8×1-bit multiported branch register file (*BR*). The number of GR and BR registers can be changed at design time (maximum 64 for GR and 8 for BR). Figure 1 depicts the methodology to generate a ρ-VEX processor. The user can profile an application to determine a suitable processor, and then quickly implement it along with the executable (instruction and data memories). The only difference between the FPGA and ASIC implementations is that, for FPGA, the GR register file is implemented with dual-port synchronous RAMs (DP_RAMs), while for ASIC it is implemented with flip-flops (FFs).

4 The Fault-Tolerant ρ-VEX VLIW Processor

Single event upset (SEU) effects a memory cell or FF. It is a bit flip caused by a charged particle. The noise induced by a radiation when exceeds the threshold

voltage, a bit flip may occur. Due to wire process shrinking, the threshold voltage is decreasing, and hence, electronic systems are becoming more susceptible to SEUs. When an SEU occurs in a memory (storage or configuration), it is called as *permanent error*. When it occurs in a flip-flop, it is referred to as a *transient error*. To recover from the permanent error, reconfiguration or re-loading of the configuration data to the configuration memory is required. For a memory used as a general storage (e.g., instruction memory), the permanent error could be checked and corrected by parity checking and some error correcting code (ECC). TMR technique is used widely to recover from a transient error. When TMR mitigation techniques are adopted, the same circuit is triplicated and a majority voter is implemented between the three computed results. In this way, a single fault occurring in one part of the TMR circuit is protected as the result is obtained from the other two circuits.

For this paper, we consider SEU errors that occur due to a direct hit in a flip-flop or a memory element used as a general storage (instruction and data memories, and the GR register file). We do not consider the FPGA configuration memory, and assume that it is protected by other techniques. According to [12], the probability that SEU errors in combinational logic can propagate to a register on a clock is very low, therefore, we do not consider such permanent SEU errors in combinational logic. The ρ-VEX processor utilizes two types of sequential cells for its implementation: synchronous DP_RAMs for instruction/data memories and the GR register file (in case of FPGA implementation), and synchronous FFs used for other storage such as general registers, pipeline registers, state machines, and status/control functions. We employ different SEU protection techniques for the DP_RAMs and FFs. The hardwired DP_RAMs in the Xilinx and Altera FPGAs provide an extra bit per byte of data which can be used as a parity bit. Hence, for a 32-bit word, up to 4 parity bits are available and can be used without increasing the number of DP_RAMs. In case of an ASIC, additional area is required to implement parity bits in instruction and data memories. Following, we discuss different modules of the fault-tolerant ρ-VEX processor which utilize different error protection techniques.

4.1 Instruction Memory

For the ρ-VEX processor, each operation called syllable is encoded in a 32-bit word. Multiple syllables are combined to make a long instruction which is executed every clock cycle. The instruction width for a 2-, 4-, or 8-issue ρ-VEX processor is 64-bit, 128-bit, and 256-bit, respectively. Our design provides configurable number of parity bits (1, 2, and 4) per 32-bit instruction (syllable). Hence, for every 8 bits of instruction, a parity bit is available. The parity bits are statically calculated by *XOR* operations in the assembler tool and stored along with the instructions in the dedicated parity bits of the memory. Instructions are read and passed through the fetch stage to the decode stage. The parity bits are checked in the decode stage in parallel with instruction decoding to minimize the timing overhead. If a parity error is detected for an instruction, the decode and the fetch stages are flushed, and the pipeline is halted. The correct instruction

can then be copied from the higher level memory (Flash card, on-board memory, etc.) to the local instruction memory, and the pipeline can then be restarted.

4.2 Data Memory

The data width of the ρ-VEX processor is 32-bit whatever the issue-width may be. The data memory is implemented with DP_RAMs. Additional bits are utilized as parity bits. Because the ISA has memory operations that can operate on words, half-words, and bytes, therefore, we utilized 1 parity bit per byte of the data. Initially, parity bits are generated statically in the assembler tool and placed along with data in the external memory. During initialization, the data and the parity bits are copied from the external memory to the local data memory. During a store operation, the parity bits are calculated and written to the data memory together with the new data. The parity bits are generated in the MEM unit which resides in the execute0 stage. During a load operation, a data word is read from the data memory along with the parity bits. The parity of the data word is checked in the writeback stage before writing the word to the GR register file. If there is a parity error, a data error trap is generated and the pipeline is halted. The simplest method to recover from this error is to reload the whole data memory for the program from the external memory and start the program from the beginning. Other complex error recovery methods such as roll back to the instruction which modified the data location may also be considered but implementing such methods are out of scope of this paper.

4.3 GR Register File (FPGA Implementation)

The 2-, 4-, and 8-issue ρ-VEX processors require 64×32-bit GR register files with 2-write-4-read (2W4R) ports, 4W8R ports, and 8W16R ports, respectively. The hardware resource requirement for these register files grows large when implemented with an FPGA's configurable look-up tables (LUTs), therefore, the register files are implemented with 18 Kbits DP_RAMs. Each DP_RAM is configured in simple dual port (SDP) mode with 1W1R port, and for multiple ports, DP_RAMs are organized into multiple banks and data is duplicated across various DP_RAMs. In this design, the number of write ports defines the number of banks and the number of read ports defines the number of DP_RAMs per bank. Figure 2 depicts the GR register file for a 2-issue ρ-VEX processor with 2 banks each having 4 DP_RAMs. The *direction table* is a small register table having the same number of ports as the original register file. For the 2W4R, 4W8R, and 8W16R ports register files, the direction table is 64×1-bit, 64×2-bit, and 64×3-bit, respectively. The direction table is implemented with FFs, and to provide SEU error protection, TMR approach is utilized as discussed in Section 4.4. Each FF is triplicated and a majority voter is implemented. Hence, an SEU error in a single FF can be tolerated.

Each write port is associated with a bank and all the DP_RAMs in a bank are simultaneously updated. Each DP_RAM is organized in a 32-bit wide aspect ratio and parity bits are design-time configurable (1, 2, or 4 for each 32-bit word).

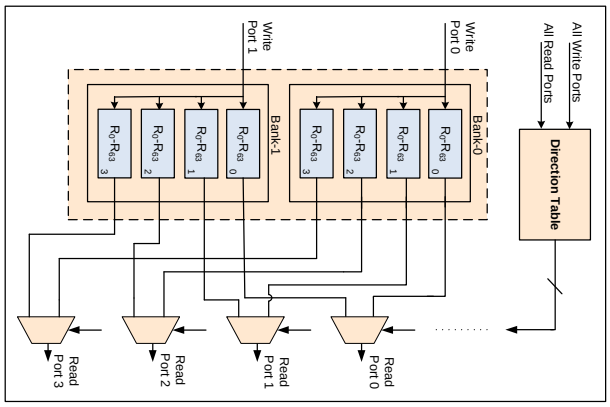


Fig. 2. 64×32-bit 2W4R ports GR register file for 2-issue ρ-VEX processor implemented with FPGA's DP_RAMs.

The parity bits are generated in the writeback stage and written together with the data. The register data is accessed in the decode stage but the parity check is done in the execute0 stage to avoid the timing overhead. If a parity error is detected on the read data on a register file port, the pipeline is flushed and the error correction procedure is started. We implemented a simple mechanism to correct the corrupted data. For each write port, the written data is already duplicated in multiple DP_RAMs each associated with a read port (4, 8, and 16 DP_RAMs for 2-, 4-, and 8-issue processors, respectively). When a parity error is detected in a data on a read port, the same data is read on another port from a different DP_RAM in the same bank. The parity for this data is also checked. If the parity is correct, it is assumed that this data is correct. This data is then written to all the DP_RAMs in the bank where the corrupted data was present in a DP_RAM. The pipeline is then restarted at the point of the failing instruction and normal execution resumes. Currently, we check only one neighbor DP_RAM for the correct data instead of all the DP_RAMs in a bank to simplify the design. If a data word cannot be corrected in this way (e.g., if the same location in all the DP_RAMs in a bank is corrupted at the same time), an unrecoverable error trap is generated.

4.4 TMR Approach for Flip-Flops

In the ρ-VEX processor, flip-flops are used for different purposes such as data holding registers, status registers, pipelines latches/registers, state machine registers, etc. The VEX ISA specifies a 1-bit 8-element multiported branch register file (BR) for a multi-issue VLIW processor. For 2-, 4-, and 8-issue ρ-VEX processors, the ISA requires BR register files with 2W2R ports, 4W4R ports, and 8W8R ports, respectively. In case of ASIC implementation, the GR, BR, and link register (LR) files for the processors are implemented with FFs. TMR approach is utilized to protect against the SEU errors in all the FFs used in the

processors. Each FF is triplicated and a majority voter is implemented for it, and hence, an SEU error in a single FF can be tolerated. Because the FFs are continuously clocked, any SEU error can be removed within one clock cycle with the output of the voter providing the correct (glitch-free) value. The robustness of the TMR scheme can further be increased by providing a separate clock tree for each lane of the TMR FFs. In this way, all of data errors resulting from an SEU hit on one clock tree can be tolerated and automatically corrected on the next clock edge.

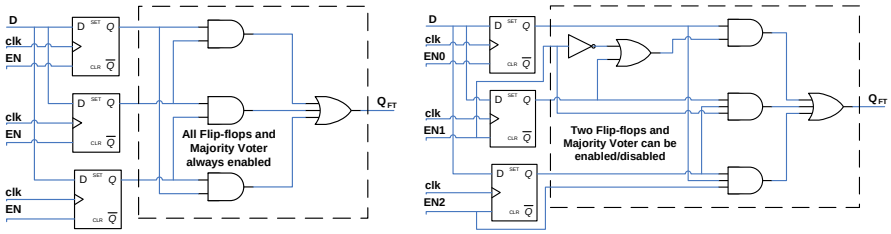
4.5 Working of the Configurable Fault-Tolerant System

We implemented fault-tolerance techniques that can be configured to be enabled or disabled at run-time. The user can specify to include or not include the fault-tolerance in the processors at design-time. Additional to this, the user can specify at design-time to implement a fault-tolerant design of a processor in which the fault-tolerance is permanently enabled. Figure 3(a) depicts the TMR scheme and the majority voter for the permanently enabled fault-tolerant design. If an application requires that fault-tolerance should always be enabled, this design has the advantage of requiring less hardware resources, consuming less dynamic power, and running at higher clock frequency.

On the other hand, if an application requires fault-tolerance only at specific instances of time (e.g., to execute certain specific modules or when the device has to be used in an increased radiations environment) and does not require fault-tolerance at other times, the system should be able to turn off the fault-tolerance circuit to avoid consuming additional dynamic power due to triplication of the FFs. In our case, at design-time, the user can also specify to implement a processor in which the fault-tolerant circuit can be enabled and disabled at run-time. The fault-tolerance can be enabled and disabled be executing an instruction on the processor and this control can be given to the user or to a program. Figure 3(b) depicts the TMR scheme and the majority voter for the run-time enabled/disabled fault-tolerant design. In this case, the additional two FFs and the majority voter can be enabled/disabled by controlling the *EN1* and *EN2* signals. This design slightly increases the hardware resources and the critical path. The advantage is that dynamic power consumption can be reduced at run-time if an application does not require fault-tolerance at some point in time.

4.6 Test Methodology

To test our fault-tolerant design, we utilized the simulation-based fault-injection method presented in [19]. The advantage of this method is that it does not require any additional hardware setup and allows fast and easy implementation of the fault injection platform but limits the number of experiments due to its high computational requirements and long simulation time. Using the VHDL description of the ρ-VEX processor, we can perform realistic emulation of the faults and detailed monitoring of the system. We utilized *ModelSim simulation tool (version 64-bit SE 6.6e)* to perform fault injection and record the results.



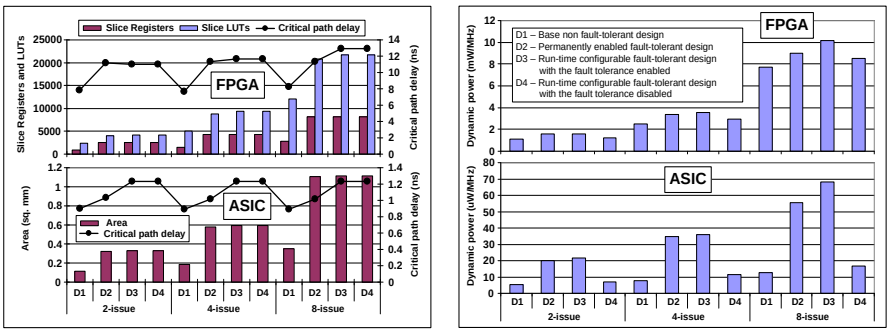
(a) Permanently enabled scheme. (b) Run-time enabled/disabled scheme.

Fig. 3. Two approaches used for TMR.

We have written special non-synthesizable routines that generate faults in different regions of the processor at different clock edges, and then record the results. Single bit errors are induced in the pipeline registers and other sequential elements of the processors. Injecting errors in FFs does not require stalling a processor and the execution can continue as normal. To inject errors in the GR register file (FPGA implementation) of a processor or the instruction or data memory requires stalling the processor. We injected 3000 single bit errors in the 2-, 4- and 8-issue ρ-VEX processors running *matrix multiplication* and *sorting* applications. All the errors in the TMR structure (FFs) were corrected. Errors in the instruction and data memories were detected and the processor was stopped to correct these errors. All of the correctable errors in the GR register file (FPGA implementation) were corrected and the non-correctable errors generated a trap halting the processor execution.

5 Experimental Results

Figure 4 depicts the hardware resources, critical path delay, and dynamic power consumption results for the base and the fault-tolerant ρ-VEX processors without instruction and data memories. For the FPGA implementation, we utilized the *Xilinx ISE (version 13.3)* and the Virtex-6 *XC6VLX240T-1FF1156* FPGA, whereas for the ASIC, we utilized the *Synopsis Design Compiler (version G-2012.06-SP2)* and targeted a 90 nm technology. The GR and BR register files in all cases are 64×32-bit and 8×1-bit, respectively. The 2-, 4-, and 8-issue cores have 2, 4, and 8 $ALUs$, and 2, 2, and 4 $MULs$, respectively. Each type of core has a single load/store (MEM) unit. In Fig. 4, *D1* and *D2* represent the base non fault-tolerant and the permanently enabled fault-tolerant designs, respectively. *D3* and *D4* (both having same area in terms of hardware resources) represent the processor design in which fault-tolerance can be enabled/disabled at run-time. *D3* represents the fault-tolerance enabled scenario, while *D4* represents the fault-tolerance disabled scenario. The parity bits are design-time configurable, i.e., 1, 2, or 4 bits per 32-bit of word. The results presented in Fig. 4 are for 4 bits of parity per 32-bit word, i.e., 1 bit per byte of data.



(a) Resources/area & critical path delay. (b) Dynamic power consumption.

Fig. 4. Hardware resource utilization, critical path delay, and dynamic power consumption for the ρ-VEX VLIW processors for the Xilinx Virtex-6 FPGA and 90 nm ASIC technology. In addition to the mentioned resources, the 2-, 4-, and 8-issue cores utilize 4, 16, and 64 RAMB36s for GR register files, and 4, 4, and 8 DSP48E1s modules, respectively, in the FPGA implementation.

As can be observed from Fig. 4(a), adding fault-tolerance to a processor requires more hardware resources especially the FFs (which are triplicated for TMR approach) and the additional logic gates for implementing majority voters. For the FPGA implementation, the number of DP_RAMs for the GR register files and instruction and data memories remain the same because we utilize the available extra parity bits in the DP_RAMs. For the ASIC, the area required for implementing additional parity bits for instruction and data memories increases. In terms of bits increase, it is 1, 2, or 4 bits per 32-bit of word for instruction and data memories depending upon the desired number of parity bits. Designs *D3/D4* utilize slightly more hardware resources and runs at less frequency compared to *D2*. The additional logic gates utilized for majority voters in *D3/D4* may be accommodated in the already utilized LUTs (FPGA implementation), therefore, the critical path delay remains almost the same as that for *D2*. It is also to note that the critical path delay in FPGAs is also dependent upon the placement and routing congestion. In case of the ASIC, the increase in the critical path delay can be clearly observed when moving from *D1* to *D2* to *D3/D4* due to adding additional logic gates in the path (majority voters).

Instead of measuring the absolute power consumption for certain applications, we measure the average dynamic power at typical operating conditions utilizing 10% switching activities, as presented in Fig. 4(b). We utilized the Xilinx *XPower Analyzer* tool and the *Synopsis Design Compiler* to measure the power consumption for the *XC6VLX240T-1FF1156* FPGA and the 90 nm technology, respectively. As can be observed from the figure, implementing fault-tolerance in the processors increases the dynamic power consumption due to increased hardware resources. Designs *D2* and *D3* (fault-tolerance enabled) consume almost similar dynamic power, while *D4* (fault-tolerance disabled) consumes considerably less power compared to *D2*. In case of the FPGA implementation, the *D4*

consumes 25.93%, 12.43%, and 5.55% less dynamic power compared to the 2-, 4-, and 8-issue *D2* designs, respectively. In the larger issue-width cores, the GR register files require increased number of DP_RAMs due to increased number of ports. In FPGAs, DP_RAMs contribute more to the dynamic power compared to FFs, therefore, for the 8-issue processors in our case, the dynamic power consumption does not reduce considerably when moving from *D2* to *D4*. This effect is not visible in the ASIC results, as the GR register files are implemented with FFs, not DP_RAMs. For the ASIC implementation, the *D4* consumes 65.92%, 67.30%, and 70.22% less dynamic power compared to the 2-, 4-, and 8-issue *D2* designs, respectively. This is considerable power reduction, and if fault-tolerance is not required at some point in time, it can be turned off to reduce the dynamic power consumption.

6 Conclusions

In this paper, we presented configurable fault-tolerance techniques for the ρ-VEX softcore VLIW processor. The issue-width of the processor can be configured to be 2-issue, 4-issue, or 8-issue with different mix of functional units. The fault-tolerance designs can detect and correct SEU errors. The designs are implemented in a Xilinx Virtex-6 FPGA, as well as synthesized to a 90 nm ASIC technology. Parity checking is utilized to detect errors in the instruction and data memories, and the general register files (FPGA implementation). For all other sequential elements, the TMR technique with majority voting is implemented. Different designs for fault-tolerance scheme such as permanently enabled at design-time or with run-time options for enabling and disabling, were presented. These options enable a user to trade-off between hardware resources, performance, and power consumption. When fault-tolerance is not required at some point in time, disabling the fault-tolerance at run-time results in considerable dynamic power reduction (up to 25.93% for the FPGA and 70.22% for the ASIC) compared to the permanently enabled fault-tolerant design.

References

1. H.P. Labs. VEX Toolchain. http://www.hpl.hp.com/downloads/vex/
2. Blough, D., Nicolaun, A.: Fault Tolerance in Super-scalar and VLIW Processors. In: IEEE Workshop on Fault Tolerant Parallel and Distributed Systems. pp. 193–200 (1992)
3. Bolchini, C.: A Software Methodology for Detecting Hardware Faults in VLIW Data Paths. IEEE Transactions on Reliability 52(4), pp. 458–468 (2003)
4. Chen, Y., Leo, K.: Reliable Data Path Design of VLIW Processor Cores with Comprehensive Error-coverage Assessment. Elsevier Journal of Microprocessors and Microsystems 34, pp. 49–61 (2010)
5. Faraboschi, P., Brown, G., Fisher, J., Desoli, G., Homewood, F.: Lx: A Technology Platform for Customizable VLIW Embedded Processing. In: International Symposium on Computer Architecture. pp. 203–213 (2000)

6. Fisher, J., Faraboschi, P., Young, C.: Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools. Morgan Kaufmann (2005)
7. Franklin, M.: A Study of Time Redundant Fault Tolerance Techniques for Superscalar Processors. In: International Workshop on Defect and Fault Tolerance in VLSI Systems. pp. 207–215 (1995)
8. Gaisler, J.: A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture. In: International Conference on Dependable Systems and Networks. pp. 409–415 (2002)
9. Holm, J., Banerjee, P.: Low Cost Concurrent Error Detection in a VLIW Architecture using Replicated Instructions. In: International Conference on Parallel Processing. pp. 192–195 (1992)
10. Hu, J., Li, F., Degalahal, V., Kandemir, M., Vijaykrishnan, N., Irwin, M.: Compiler-Directed Instruction Duplication for Soft Error Detection. In: Design, Automation and Test in Europe Conference and Exhibition. pp. 1056–1057 (2005)
11. Ichinomiya, Y., Tanoue, S., Ishida, T., Amagasaki, M., Kuga, M., Sueyoshi, T.: Memory Sharing Approach for TMR Softcore Processor. In: International Conference on Applied Reconfigurable Computing. pp. 268–274 (2009)
12. Liden, P., Dahlgren, P., Johansson, R., Karlsson, J.: On Latching Probability of Particles Induced Transients in Combinational Networks. In: International Symposium on Fault-Tolerant Computing. pp. 340–349 (1994)
13. Nickle, J., Soman, A.: REESE: A Method of Soft Error Detection in Microprocessors. In: International Conference on Dependable Systems and Networks. pp. 401–410 (2001)
14. Oh, N., Shirvani, P., McCluskey, E.: Error Detection by Duplicated Instructions in Super-scalar Processors. IEEE Transactions on Reliability 51(1), pp. 63–75 (2002)
15. Rashid, F., Saluja, K., Ramanathan, P.: Fault Tolerance Through Re-execution in Multiscalar Architecture. In: International Conference on Dependable Systems and Networks. pp. 482–491 (2000)
16. Sato, T., Arita, I.: Evaluating Low-cost Fault-tolerance Mechanism for Microprocessors on Multimedia Applications. In: Pacific Rim International Symposium On Dependable Computing. pp. 225–232 (2001)
17. Scholzel, M., Mulleri, S.: Combining Hardware- and Software-Based Self-Repair Methods for Statistically Scheduled Data Paths. In: International Workshop on Defect and Fault Tolerance in VLSI Systems. pp. 90–98 (2010)
18. Sterpone, L., Sabena, D., Campagna, S., Reorda, M.: Fault Injection Analysis of Transient Faults in Clustered VLIW Processors. In: IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems. pp. 207–212 (2011)
19. Touloupis, E., Flint, J., Chouliaras, V., Ward, D.: Study of the Effects of SEU-Induced Faults on a Pipeline-Protected Microprocessor. IEEE Transactions on Computers 56(12), pp. 1585–1596 (2007)
20. Vasudevan, V., Waldeck, P., Mehta, H., Bergmann, N.: Implementation of Triple Modular Redundant FPGA based Safety Critical System for Reliable Software Execution. In: Australian Workshop on Safety Related Programmable Systems. pp. 113–119 (2006)
21. Wong, S., Anjam, F.: The Delft Reconfigurable VLIW Processor. In: International Conference on Advanced Computing and Communications. pp. 242–251 (2009)