International Journal of Natural and Engineering Sciences 7 (3): 36-42, 2013
ISSN: 1307-1149, E-ISSN: 2146-0086, www.nobel.gen.tr

A Power Efficient Register File Architecture in Embedded Processors

Leila REIKHTECHI[1] Mahdi FAZELI[2*] Ahmad PATOOGHY[2]
[1]Islamic Azad University of Broujerd, Broujerd, IRAN
[2]Iran University of Science and Technology, Tehran, IRAN

*Corresponding author
E-mail: m_fazeli@iust.ac.ir

Received: July 07, 2013
Accepted: December 10, 2013

Abstract

Reliability and energy efficiency are usually two opposing factors that are both critical in embedded processors. In this paper we present a register file-partitioningalgorithm, which accounts for both the AVF and access rate. Using this algorithm we divide the register file into protected and unprotected regions. We show that this algorithm can reduce the AVF by 65% while simultaneously reducing the energy consumption by 13% on an ARM processor for selected test benches.

Keywords: Embedded Systems, Fault Tolerant, Low Power Design, Register File

INTRODUCTION

Embedded systems are extensively used in various areas of communication, networking and multimedia. In these systems, power efficiency and reliability has risen to be the most concerning issues. Due to their ever-decreasing size, transistors power density has increased to such levels that it prevents the design from working at their maximum clock frequency. They have also become highly susceptible to soft errors [10]. It has been observed that the majority of the errors manifested in architectural state of a processor in both combinational and sequential logic are caused by faults in register file [3]. Research has also shown that the register file is one of the most power hungry segments. For example, in Motorola M.CORE architecture, register file consumes 42% of the data-path power [14], and up to 25% of the total processor power [1]. Therefore, it's essential for any fault-tolerant mechanism to have minimum power overhead. This need is further amplified by the fact that soft error rate increases exponentially with temperature [7].

Three methods have been traditionally used to protect register files. Hardware based methods use ECC, parity or duplication among other methods to protect register file in whole or in part [6][3]. Compiler based techniques try to use various compiler based methods to reduce vulnerability duration. [Yan05] proposes an instruction scheduling that tries to reduce the distance between loads and stores in a bid to lower the register file vulnerability. [7] Reduces the vulnerability of registers by temporarily writing live variables to protected memory. Hybrid methods try to combine the above methods. [7] Uses compiler to change register assignments at the program level separately for each function [7] . They eliminate the overhead of hardware decision and improve the quality of decisions using compile-time analysis. Their Methodology althoughselectively protects the register file and tries to minimize the power overhead, still fails to completely eliminate it. Register file partitioning has been extensively recommended as a method to decrease power consumption [12] [2] [4]. [12] Used profiling and statistical application analysis to achieve 40% energy saving on dual-bank configuration and a further 15% -20% on multi-bank register files. These approaches address the high power consumption of register files but do not consider reliability as an issue.

Our work was motivated by the increasing need in today's technology for energy efficient and reliable register files. There is a definite gap for techniques that can increase reliability by an acceptable margin without enforcing a high-energy overhead or even decreasing the total power consumption. Our technique uses register partitioning as a way to improve reliability and power consumption. We introduce a register partitioning technique, which considers both reliability and power consumption as factors for register partitioning and can be tweaked to suit our particular needs or limitations. We show in our experiment on an ARM processor for MiBench test bench that our proposed technique decreases the vulnerability of register file by 64% -79% while simultaneously reducing the energy consumption by 11% -16%.

The remainder of the paper is organized as follows. Section II introduces various other works that has been done on both reliability and power efficiency of register files and presents our motivation for conducting this research. In Section III and IV we formulate our technique and show how it works through a small-scale sample. Section V and VI outlines the experimental environment and provides the experimental results with analysis. We finally discuss future works and draw conclusion in section VII and VIII.

Preliminaries

A. Power Consumption in a Protected Register File A typical two-port register file, shown in figure 1, has several components that dissipate power. Most important components are decoder, bit lines, word lines, sense amplifier, control circuitry and memory cells [8].

We model the dynamic power consumption of the register file as $Ptotal = Paccess \times A$, where A is the total access countand *Paccess* is the average power required for a single read or write operation.

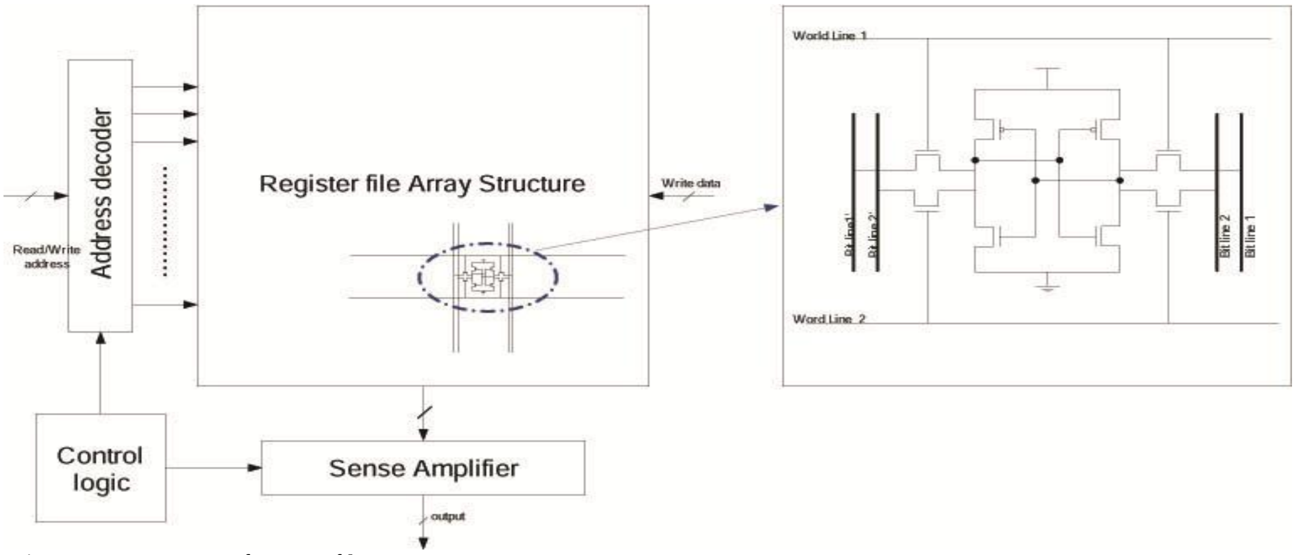

Fig. 1. Basic structure of register file.

For a partitioned register file with two partitions, however, accessing power for read or write operations can be derivedas:

$$Ptotal = (P1 \times A1 + P2 \times A2 + P12 \times A12) \quad (1)$$

In this model, *P*1, *P*2 and *P*12 are power of accessing partition 1, partition 2 and cross accessing partitions and *A*1, *A*2 and *A*12 are the total instruction count of each access. Values of *P*1 and *P*2 depend on register file structure and size. If partition 2 is bigger than partition 1, because of large bit-line capacitance of partition 2 as compared to partition 1, we would have $P1 < P2 < P12$ and vise versa. Due to the bit line length, as shown in in Figure 2, the total energy of accessing register file is almost linearly dependent on the register file size. These results are extracted from circuit-level register file simulations using SPICE and are normalized against the un-partitioned configuration.

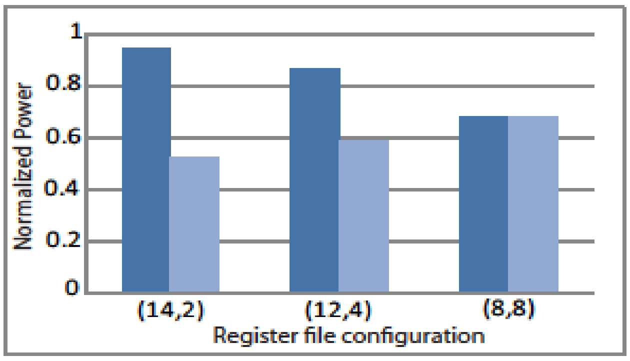

Fig. 2. partitioned register file power usage

B. Register File Reliability Measurement

Architectural Vulnerability Factor is the factor we use forreliability measurement. It is the probability that a fault inthat particular structure will result in an error [11]. Thevulnerability of a register is defined as the sum of lifetimeof variables assigned to it. The lifetime of a variable is fromits definition until its last use and represents the time when valid data is present in the register. Transient fault to the register occurring during that time period therefore destroys data integrity and can manifest itself into an error. Thus given the same transient fault rate, vulnerability can be used to predict the soft error rate. The vulnerability of a register file is simply the sum of vulnerability of all registers. During our work, we consider the average AVF of the registers in the unprotected region as the AVF of the register banks.

In order to estimate the reliability of the proposed registerfile architecture, the Architectural Vulnerability Factor is employed. The AVF of a part is the probability that a fault results in an error. To measure the AVF of a register we should first extract the fraction of time in which the register is vulnerable to faults so called ACE time (Architecturally Correct Execution)[11]. If a fault occurs in each bit of a register in its ACE time, it will produce an error. In contrast, the un-ACE time of a register is the fraction of time in which a faulty bit in the register will not result in an error. Finally, the AVF of a register is the percentage of time in which the register is in its ACE time.

Figure 3 and 4 represents a typical register ACE and Un-ACE time. As shown in figure 3, when a write operation is performed to the register, it enters its ACE time until the last read operation from the register. The time duration between the last read from the register and the next

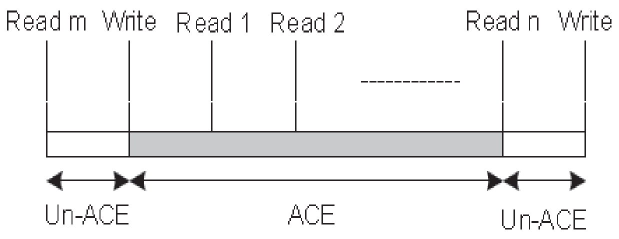

Fig. 3. ACE time of a register

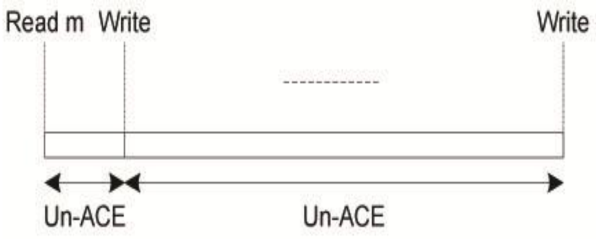

Fig. 4. Un-ACE Time of a register

write to theregister is considered as the Un-ACE time. If there is no read operation between two consecutive write operations (Figure 4), the register is always in its Un-ACE time. It means that a value is written to a register but never used.

The Proposed Partitioning Technique

Our goal is to divide the register file into two parts and to protect one of them against faults such that the critical registers are protected as much as possible against soft errors while somewhat maintaining the power reduction achieved through partitioning. We define critical registers as those, which have the highest AVF so our goal is to put the registers, which have the highest AVF in the protected region. This approach while giving us the highest reliability may not result in optimal power saving. To reach the optimal power saving our partitioning technique should put the registers with highest access rate in the smaller partition and also minimize the cross access between the two partitions. To achieve both of these goals, our partitioning technique uses a cost function. Our cost uses αas a variable which can be tweaked to provide more AVF protection or more power saving. Our cost function

$$\alpha AVF(R) + (1-\alpha)(Power(R) + CrossPower(R)) \qquad (2)$$

provides us with the necessary tool to design an algorithm that reaches a convincingly accurate greedy answer to the NP-Complete partitioning problem. The $Power(R)$ +$CrossPower(R)$ is the original cost function that is usedfor register partitioning with only the energy consumption inmind. It uses two standard factors for comparing file register'spower consumption. $Power(R)$ is the power used by register$R$ in various read and write accesses and $CrossPower(R)$,which can be calculated for each of the protected and unprotectedparts, compensates for the extra power used when register$R$ is simultaneously used with another register from thatregister partition. By adding the $AVF(R)$ into the formula,we account the AVF of register as a factor for calculating thecost and thus as a factor for selecting which registers to put inthe protected section. In our experiment on ARMv9 processorusing MiBench test bench, we have used various values for $\alpha$to observe the balance of these two factors.

We offer two algorithms for partitioning the register bank:

A. Static Greedy

Algorithm 1 is a static greedy algorithm. It uses our costfunction to find the appropriate registers, which should beselected for the protected region. In this algorithm, we first put all the registers in the unprotected region. Our algorithmthen chooses the register with maximum AVF and puts itinto the protected region. After that it compares the crossusage power of the registers with the ones already present inthe protected region, AVF and power usage of each registerand selects a predefined number of registers that best suit outreliability and energy constraints.

Algorithm 1: Static Greedy Algorithm

```
1 for $i$ = 0 to Size(Register Bank) do
2     Assign $R_i$ to Unprotected Region;
3 end
4 Assign $R_i$ With Maximum AVF to Protected Region;
5 for $i$ = 1 to Size(Protected Region) do
6     Find $R_j$ in Unprotected Region With
      Max($\alpha AVF(R_j) + (1-\alpha)(Power(R_j) + CrossPowerProtected(R_j))$);
7     Assign $R_j$ to Protected Region;
8 end
```

B. Dynamic Greedy

The dynamic greedy algorithm [Algorithm 2] is inspiredfrom the classic MinCut algorithm. Compared to the staticgreedy algorithm our dynamic greedy algorithm can alsocalculate appropriate partitioning size. Our algorithm beginsby assigning the Register with maximum AVF to the protectedsection and the one with minimum AVF to the unprotectedregion. Our algorithm then decides for all of the other registersin the register bank where to be located. At each step, wefind the register with the highest AVF and cross access withprotected region (The best register to be protected) and theregister with least AVF and most cross access with unprotectedpartition (The worse register to be protected). We compare theweight of these two registers and assign one of them to theprotected or unprotected section. The algorithm then finds theclosest partitioning size from the possible cuts and move extraregisters that may exist to the other part according the theirweight. Nevertheless this algorithm is more precise in drawingthe line between the protected and unprotected region, it worksmuch slower.

There are two methods that can be used for assigning registersto the correct partition. A hardware approach needs morecomplex hardware that will cause in more energy usage. Thesoftware approach needs to be implemented on the compileror be done as post-compile processing.

Algorithm2: Dynamic Greedy Algorithm

```
1 Assign $R_i$ With Maximum AVF to Protected Region
2 Assign $R_i$ With Minimum AVF to Unprotected
  Region;
3 for $i$ = 2 to Size(Register Bank) do
4     Find $R_j$ With $W_Rj$=Max($\alpha AVF(R_j) + (1-\alpha)$
      $(Power(R_i) + CrossPowerProtected(R_i))$);
5     Find $R_j$ With $WR_j$=Max($\alpha(1-AVF(R_j))+(1-\alpha)$
      $(Power(R_j) + CrossPowerUnprotected(R_j))$);
6     if $W_R^i > W_{Rj}$ then
7         Assign $R_i$ to Protected Region;
8     else
9         Assign $R_i$ to Unprotected Region
10    end
11 end
12 $(P;Q)$ = Select nearest partition from
   (2; 14), (4; 12), (8; 8);
13 Move extra Registers $R_i$ to other partition in $(P,Q)$;
```

Example Of Algorithm

In this section we present how our algorithm works ona small 5 register RF as proof of concept. We present therelationship between these registers in a complete graph shownin Figure 5. In this graph each node represents a register. Eachregister has a number and a percentile written in it, which arerespectively the number of unaccompanied accesses (accessesthat involve no other register beside them) and the AVF of the register. Each node is colored according to it's assignment. "Dark Green" and "Red" represents assignment to Protected and Unprotected partition respectively. "Light Green" and"Orange" arealsoused to display the register currently getting assigned tothe Protected or Unprotected region. Each edge in this graph represent the number of times both of these registersis accessed simultaneously.

A. Static Greedy

In our static greedy algorithm we choose to divide our register file into 3-2 (Protected-Unprotected) partitions. Our algorithm first finds the register with maximum AVF (R1 with

%94) and moves it to the protected region. (Fig. 6.A) After this stage, our static greedy algorithm in each step will find the register that will maximize our cost function and moves that register to the protected region. (Fig. 6) The cost function of each unassigned node is calculated as

$$\alpha \times AVF(R_x) + (1-\alpha)(R_x + (\Sigma\, E_{xy}\, |R_y 2\ Protected)$$

Using this method our protected algorithm will choose R3 and R2 and moves them to the protected region before terminating.

The finishing result of this algorithm can be seen in (Fig. 6.D)

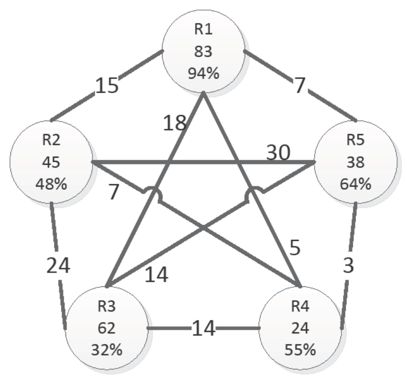

Fig. 5. Sample Relation Between Registers

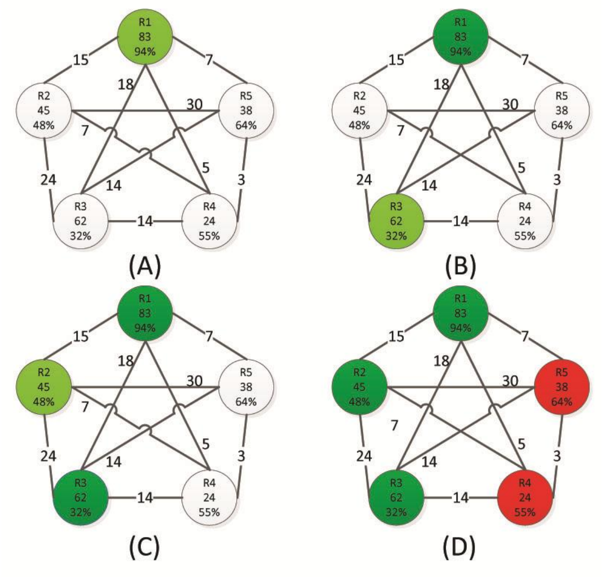

Fig. 6. Static Greedy Algorithm

B. Dynamic Greedy

In our static greedy algorithm We first move the register with maximum AVF to the protected region and the registerwith the minimum AVF to the unprotected region. (Fig. 7.A) We then calculate our cost function for all the against both the protected and unprotected region atevery turn and assign the register with maximum cost functionto the appropriate partition. (Fig. 6) It can be seen that unlikethe static greedy algorithm

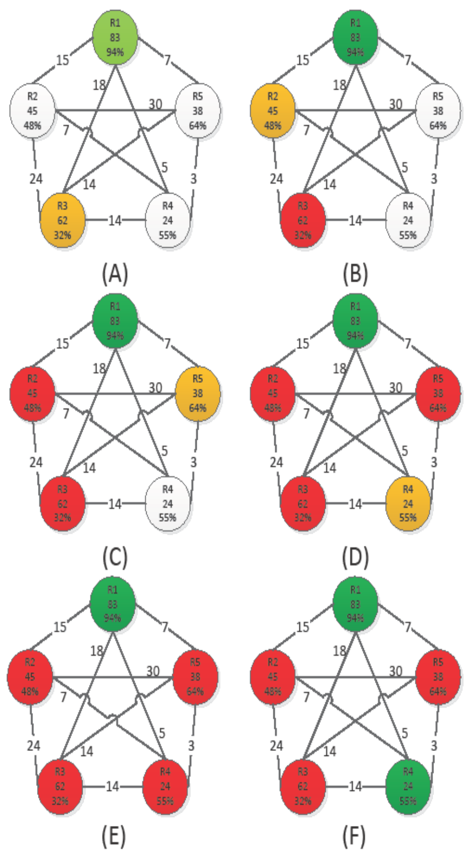

Fig. 7. Dynamic Greedy Algorithm

which divided the register bank into3-2 partitioning that was requested of it; our dynamic greedygoes onto a 1-4 partitioning (Fig. 6.E) before rounding it tothe closet acceptable partitioning (2-3) as the best partitioning. (Fig. 6.F)

Experimental System

Various protective hardware measures can be taken to eliminatethe AVF in a register. Each of these methods have poweroverhead, which could limit their efficiency and usability.

Four methods have been extensively used for protection ofthe register files. The first method is parity, which poses limited energy,overhead but is limited as it provides no errorprotection mechanism [9]. ECC on the other hand can correcterrors but it doubles the energy consumption of the register file[9]. [Memik05] uses duplication to protect the register file andencounters an on average 15% of power overhead. The finalmethod that is used for protecting the register file is the useof hardened memory cells such as rSRAM [13]. In rSRAMtwo symmetrical stacked capacitors are used to increase the minimum amount of charge required to flip the logic state [13]. In our study we use this technology for registers used in our protected region and consider a 20% power overhead for them. In our proposed partitioning techniques, we first profile the workload programs to extract information about register file access and AVF. To obtain execution profile of applications, we use a Verilog HDL model of ARM v9 processor. As a test bench for our method, we use some tests from the well-known MiBench benchmark, an embedded benchmark suite [5]. These benchmarks are listed on table I along with their description. All of them are cross-compiled for ARM processor with GNU-GCC compiler.

We simulate this benchmarks on our processor model for one million instructions. The resulted profiles contain a timing accurate register file access statistics along with register values. We analyzed these profiles to obtain statistical data from applications such as register access and cross access information; then we applied our algorithm to find the optimum register assignment. To compute power consumption of our register file, we developed a complete circuit-level netlist of register file. The netlist was simulated using Synopsys HSIM for 0:18_*m* TSMC CMOS process. Our register file structure, capacitance and transistor sizing are similar to the two-port register file IP-cores distributed by foundries/IP-core design houses at 0:18_*m* technology node. We used this information in our power estimation tool to estimate total power consumption of register file. Our power estimation tool can accurately estimate power consumption of different structures of partitioned register file. In this paper, our register file structure consists of two partitions with variable size (2*;* 4*;* 8 for smaller partition and 8*;* 12*;* 14 for bigger one).

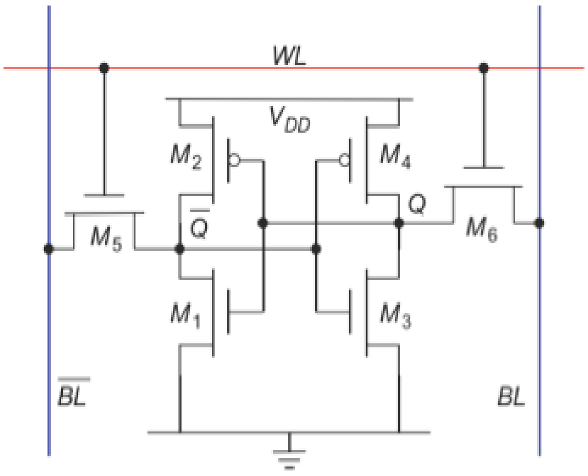

Fig. 8. An rSRAM cell

Table I Mibench Benchmarks

benchmark	description
qsort	quick sort algorithm for sorting words
basicmath	basic mathematical operation
fft	fast Fourier transform
bitcnts	counting bits
stringsearch	searching in strings

RESULTS

We simulated five of the MiBench tests for one million instructions on LEONv9 processor. We tested our technique for various amounts of *α*and different partitioning approaches. As sampled in figures 10 and 9, it was universally observed that the 8-8 partitioning achieves better power saving and reliability for every value αagainst the 4-12 and 2-14 partitioning approaches For this reason we primarily focus on the 8-8 partitioning in the upcoming observations.

Table II Register Selection For Protected Region On Various Values Ofalpha

α	Selected Registers	Power Reduction %	AVF Reduction %
0	0 1 2 3 4 11 12 14	18.82	45.58
0.1	0 1 2 3 4 11 12 13	19.21	48.61
0.2	0 1 2 3 4 8 11 12	18.27	51.73
0.3	0 1 2 3 4 8 11 12	18.27	51.73
0.4	0 1 4 6 8 9 11 12	13.98	63.47
0.5	0 1 4 6 8 9 11 12	13.98	63.47
0.6	0 1 4 6 8 9 11 12	13.98	63.47
0.7	0 4 6 7 8 9 11 12	13.29	63.47
0.8	0 4 6 7 8 9 11 12	13.29	63.47
0.9	0 4 6 7 8 9 11 12	13.29	63.47
1	0 4 5 6 7 8 9 11	12.43	64.95

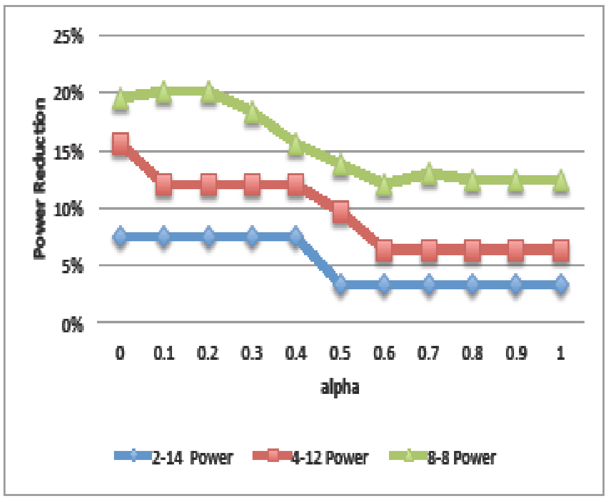

Fig. 9. Power Reduction for various partitioning

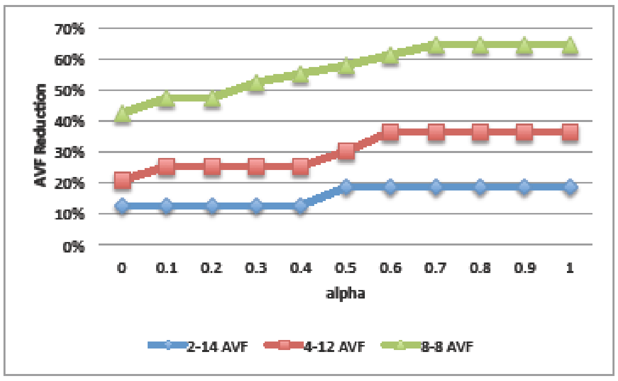

Fig. 10. AVF Reduction for various partitioning

Our results show that our technique can greatly decrease the AVF while simultaneously providing power saving. Figure 12 showcases our AVF reduction and 11 shows the percentage of our energy saving on various test benches. It’s notable that variable $\alpha$is working in our algorithm as expected. Increase in αresults in an overall reduction in power saving but increases the reliability of our system. Thus this algorithm can provide the user with diverse choices in terms of energy saving and reliability depending on the needs and priorities. Table II also shows which registers are chosen to be in our protected region depending on various values for αin string search test. It displays how changing the αvalue will noticeably change the registers that need our protection and how these changes effect the power and reliability of our model.

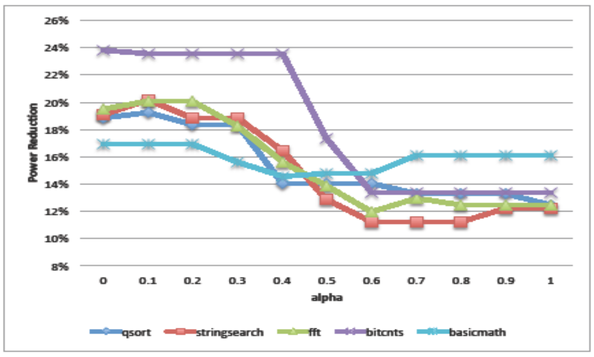

Fig. 11. Power reduction results

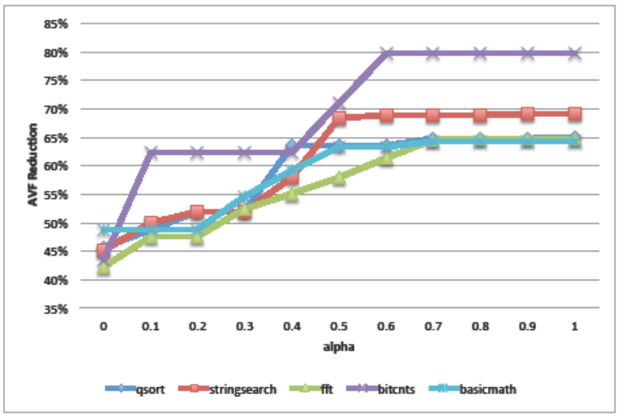

Fig. 12. AVF Reduction results

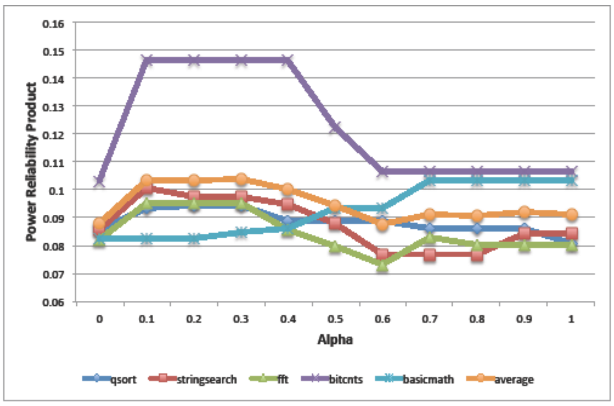

Fig. 13. PRP for various αvalues

These results while providing us with a versatile tool to achieve our requirements, bring forward the question: Which αvalue is better rounded and provides the best overall results on both power saving and reliability? For measuring this factor we present the Power-Reliability Product (PRP), which we define as:

$$PRP = Power\ Reduction \times AVF\ Reduction \quad (3)$$

Figure 13 shows the PRP of our technique for various test benches. It can be seen that the PRP value for α= 0 and α= 1 is roughly equal in most of the test benches showing that prioritizing either of the two factors achieve the same overall results. Our observation is that choosing the registers for partitioning solely on the basis of AVF proves to provide exceptional reliability increase while giving us an acceptable power reduction. It can be stated that sacrificing Reliability for power in our technique is generally not advised unless there are hard power limitations that can be achieved for a smaller value of α.

FUTURE WORKS

In future works we intend to apply our technique to multibank register file partitioning. Multi-bank register file partitioning has been shown to achieve 10% -15% more power reduction compared to the dual bank. Using more than two register banks also gives us the chance to implement different protection methods for different banks and will give us much more flexibility to optimize power usage against vulnerability. In fact, we can use different fault-tolerant techniques to protect the partitions taking into account their vulnerability. This means that for partitions having higher level of vulnerability, we can employ more powerful protection techniques. This provides an attractive trade-off between the reliability obtained and the amount of overhead imposed.

CONCLUSION

In this paper we presented a novel register file partitioning technique. Our technique differs from other method in its approach to partitioning as not only a power saving technique but also as a protection mechanism. With this approach, our partitioning divided the register file into protected and unprotected regions. We showed in our experiment on ARMv9 processor using MiBench that our technique can gain an AVF reduction of 64 -79% while simultaneously achieving 11 -16% power saving.

REFERENCES

[1] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based dynamic voltage scheduling using program checkpoints. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 168, Washington, DC, USA, 2002. IEEE Computer Society.

[2] Rajeev Balasubramonian, Sandhya Dwarkadas, and Divid H. Albonesi.

Reducing the complexity of the register file in dynamic superscalar processors. *Microarchitecture, IEEE/ACM International Symposium on*, 0:037, 2001.

[3] Jason A. Blome, Shantanu Gupta, Shuguang Feng, and Scott Mahlke. Cost-efficient soft error protection for embedded microprocessors. In *CASES '06: Proceedings of the 2006 international conference on Compilers,*

architecture and synthesis for embedded systems, pages 421–431, New York, NY, USA, 2006. ACM.

[4] Jos´e-Lorenzo Cruz, Antonio Gonz´alez, Mateo Valero, and Nigel P. Topham. Multiple-banked register file architectures. *SIGARCH Comput. Archit. News*, 28(2):316–325, 2000.

[5] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. pages 3 – 14, dec. 2001.

[6] Mallik Kandala, Wei Zhang, and Laurence T. Yang. An area-efficient approach to improving register file reliability against transient errors. *Advanced Information Networking and Applications Workshops, International Conference on*, 1:798–803, 2007.

[7] Jongeun Lee and Aviral Shrivastava. Compiler-managed register file protection for energy-efficient soft error reduction. In *ASP-DAC '09: Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 618–623, Piscataway, NJ, USA, 2009. IEEE Press.

[8] Xue mei Zhao and Yi zheng Ye. Structure configuration of low power register file using energy model. In *ASIC, 2002. Proceedings. 2002 IEEE Asia-Pacific Conference on*, pages 41–44, 2002.

[9] Gokhan Memik, Mahmut T. Kandemir, and Ozcan Ozturk. Increasing register file immunity to transient errors. *Design, Automation and Test in Europe Conference and Exhibition*, 1:586–591, 2005.

[10] S. Mitra, Ming Zhang, N. Seifert, T.M. Mak, and Kee Sup Kim. Built-in soft error resilience for robust system design. pages 1 –6, may. 2007.

[11] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K.

Reinhardt, and Todd Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. *Microarchitecture, IEEE/ACM International Symposium on*, 0:29, 2003.

[12] Rakesh Nalluri, Rohan Garg, and Preeti Ranjan Panda. Customization of register file banking architecture for low power. In *VLSID '07: Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference*, pages 239–244, Washington,

DC, USA, 2007. IEEE Computer Society.

[13] P. Roche, F. Jacquet, C. Caillat, and J.-P. Schoellkopf. An alpha immune and ultra low neutron ser high density sram. pages 671 – 672, apr. 2004.

[14] Jeff Scott, Jeff Scott, Lea Hwang Lee, Lea Hwang Lee, John Arends, John Arends, Bill Moyer, and Bill Moyer. Designing the low-power m*core architecture. 1998.