AGING AWARE DESIGN OF A MICROPROCESSOR BY DUTY CYCLE BALANCING

BY

ABHINAY RAJ KALAMBUR SABARAJAN

MASTER OF SCIENCE
ELECTRICAL ENGINEERING

State University of New York at Buffalo

Spring 2015

ACKNOWLEDGEMENT

On the occasion of completion and submission of our project I would like to express our deep sense of gratitude to **Dr. Ramalingam Sridhar**, Professor Department of Computer Science Engineering, State University of New York at Buffalo for his kind patronage and fervent support, encouragement which helped me a lot for the completion of this project. He has greatly helped and motivated me to pursue the project by providing me with the necessary knowledge, equipment and facilities required. I am thankful to him for his keen attention shown on me to bring out this project a successful one.

A special thanks to Shixiong Jiang and Pengzhan Yan for their immense help and guidance throughout the duration of the project, by the way of valuable suggestions and ideas. Their patience and dynamism with which they guided my work needs to be mentioned.

I also want to thank my friends and parents, who taught me the value of hard work and provided me enormous support during the whole tenure of my project work. I would like to extend my regards to my fellow students who gave me constant support and suggested ideas on enhancing and completing the project on stipulated time.

ABSTRACT

The degradation of CMOS devices over the lifetime can cause the severe threat to the system performance and reliability at deep submicron semiconductor technologies. The negative bias temperature instability (NBTI) is among the most important sources of the aging mechanisms. Microprocessors fabricated at nanoscale nodes are exposed to accelerated transistor aging due to Bias Temperature Instability and Hot Carrier Injection. As a result, device delays increase over time reducing the Mean Time To Failure (MTTF) of the processor. To address this challenge, many (micro)-architectural techniques target the execution stage of the instruction pipeline, as this one is typically most critical. In this paper, I propose a novel aging-aware instruction set encoding methodology (ArISE) that improves the instruction encoding iteratively using a heuristic algorithm. This Algorithm is based on Duty Cycle balancing which directly relates to the variation in threshold voltages of CMOS transistors.

INTRODUCTION

We know when it's time to get a new car. Our odometer is far into six digits, perhaps the engine is burning lots of oil, or the transmission is growling. Fixing all that might well cost quite a bit more than our ancient vehicle is worth.

But what about our microprocessor? Unlike automobiles, microprocessors don't have convenient little gauges that reflect how much wear and tear they've endured. And wear they do—though we'll probably never notice it. The degradation of their transistors over time leads slowly but surely to decreased switching speeds, and it can even result in outright circuit failures.

We generally don't perceive this deterioration because semiconductor companies always play it very safe—they set the clock-speed rating of their microprocessors so conservatively that almost every one of their products will continue to operate flawlessly throughout its intended lifetime.

Negative-bias temperature instability (NBTI) is a key reliability issue in MOSFETs. NBTI manifests as an increase in the threshold voltage and consequent decrease in drain current and transconductance of a MOSFET. The degradation exhibits logarithmic dependence on time. It is of immediate concern in p-channel MOS devices, since they almost always operate with negative gate-to-source voltage; however, the very same mechanism also affects nMOS transistors when biased in the accumulation regime, i.e. with a negative bias applied to the gate.

In sub-micrometer devices nitrogen is incorporated into the silicon gate oxide to reduce the gate leakage current density and prevent boron penetration. However, incorporating nitrogen enhances NBTI. For new technologies (32 nm and shorter nominal channel lengths), high-K metal gate stacks are used as an alternative to improve the gate current density for a given equivalent oxide thickness (EOT). Even with the introduction of new materials like hafnium oxides, NBTI remains.

It is possible that the interfacial layer composed of nitrided silicon dioxide is responsible for those instabilities. This interfacial layer results from the spontaneous oxidation of the silicon

substrate when the high-K dielectric is deposited. To limit this oxidation, the silicon interface is saturated with N resulting in a very thin and nitrided oxide layer.

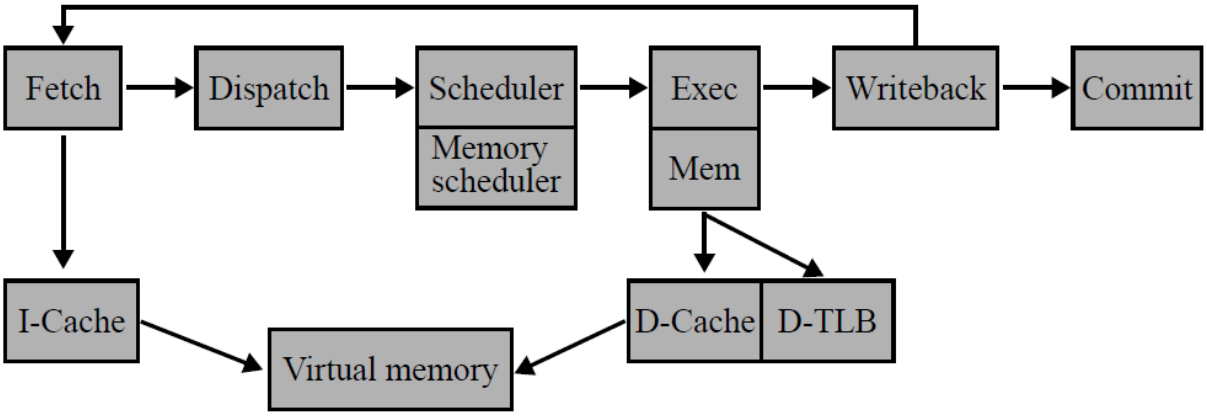
It is commonly accepted that two kinds of trap contribute to NBTI:

- Firstly, interface traps are generated. Those traps cannot be recovered over a reasonable time of operation. Some authors refer to them as permanent traps. Those traps are the same as the one created by channel hot carrier. In the case of NBTI, it is believed that the electric field is able to break Si–H bonds located at the Silicon-oxide interface. H is released in the substrate where it migrates. The remaining dangling bond Si- (Pb center) contribute to the threshold voltage degradation.
- Next, on top of the interface states generation some preexisting traps located in the bulk of the dielectric (and supposedly nitrogen related), are filled with holes coming from the channel of pMOS. Those traps can be emptied when the stress voltage is removed. This Vth degradation can be recovered over time.

The existence of two coexisting mechanisms created a large controversy, with the main controversial point being about the recoverable aspect of interface traps. Some authors suggest that only interface traps are generated and recovered; today this hypothesis is ruled out.[citation needed] The situation is clearer but not completely solved. Some authors suggest that interface traps generation is responsible for the hole trapping in the bulk of dielectrics. A tight coupling between two mechanisms may exist but nothing is demonstrated clearly.

With the introduction of high K metal gates, a new degradation mechanism appeared. The PBTI for positive bias temperature instabilities affects nMOS transistor when positively biased. In this particular case, no interface states are generated and 100% of the Vth degradation may be recovered. Those results suggest that there is no need to have interface state generation to trapped carrier in the bulk of the dielectric.

Nowadays almost all microprocessors ranging from low-power embedded parts to high-performance processors use a pipelined architecture to increase the instruction throughput and by that means the performance. The Basic pipeline architecture of a microprocessor is as shown:



The Main processes that have higher impact on microprocessor aging in this pipeline architecture are those that involve accessing memory, decoding and execution stages. Hence this paper proposes a new algorithm to implement Instruction set encoding that is based on Duty Cycle Balancing. Duty cycle is the proportion of time during which a component, device, or system is operated.

Note: The mapping of instructions to their binary representation is called Instruction Set Encoding.

The Voltage Stress or the change in Threshold voltage of an NMOS/PMOS depends on the duty cycle of the device as per the below relation.

$$\Delta V_{th}(\delta, T, t) = \left(\frac{\sqrt{K_v^2 \cdot \delta \cdot t_m}}{1 - \beta(\delta, T, t)^{1/2n}} \right)^{2n}$$

Thus the voltage stress is directly related to **duty cycle (δ)** and hence more the value of duty cycle ratio, more will be the stress and faster the degradation due to aging. In order to avoid this problem, I am proposing a new algorithm for Instruction Set Encoding design based on Duty Cycle balancing.

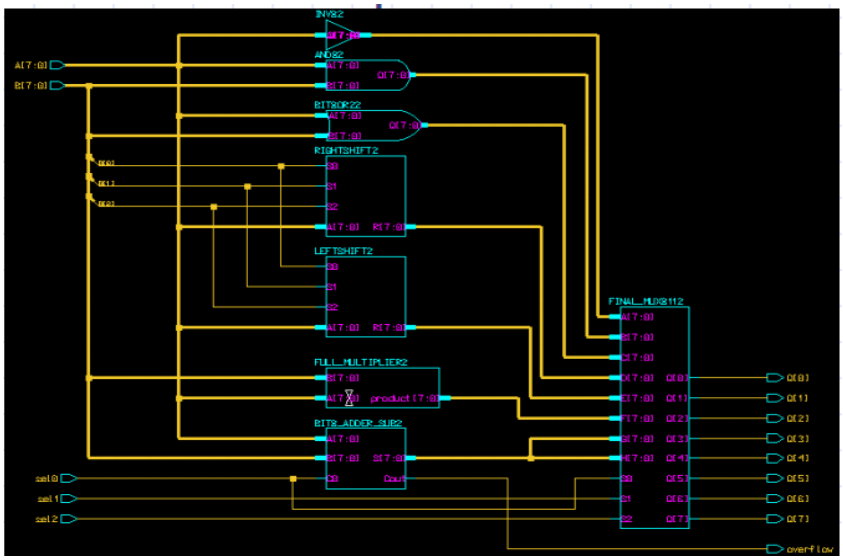
PHASE 1: DUTY CYCLE BALANCING:

To mitigate the NBTI-induced aging on the transistors, there are mainly three types of solutions: a) design customized NBTI-resilient blocks, b) exploit low energy states of the SRAM cells for alleviate the aging effect, and c) balance the duty cycle ratio of the transistor segments. However, all previous solutions targeted at mitigating the NBTI-induced aging effects in general SRAM cell structures, such as caches, register files. There is no scheme targeting at aging-aware design specifically by utilizing the access pattern and lifetime behavior. To counter this, we propose a new scheme i.e. to design a new ISE pattern for the microprocessor based on duty cycle balancing.

The Duty cycle balancing deals with conducting a detailed analysis on the lifetime behaviors of the cache memory, cache lines and the logic unit by dividing them into valid paths and In-valid paths. For the In-valid paths, we propose to bit-flip/complement these paths periodically. For the valid paths, we propose to do the idle-time-based path invalidation first and then apply the bit-flipping/complementing to the invalidated paths.

SIMULATIONS:

Simulations of 8bit ALU using Cadence:



Duty cycle calculations for ALU part:

Valid Paths: Duty cycle matches an average of 75%

Invalid Paths: Duty cycle comes around 39.7%

Cache Memory Simulation using Cacti 6.5:

CACTI is an integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. By integrating all these models together, users can have confidence that tradeoffs between time, power, and area are all based on the same assumptions and, hence, are mutually consistent. CACTI is intended for use by computer architects to better understand the performance tradeoffs inherent in memory system organizations.

CACTI is available in two forms: a web-based version and a C++ source code version. The web interface version first added for CACTI 4.0 is frequently updated with the latest versions and bug fixes, and allows CACTI to be easily accessible to a larger user community. The web interface can be accessed at http://quid.hpl.hp.com:9081/cacti/. The web interface should meet the needs of most CACTI users. For users that need to modify CACTI or integrate it into other tools to support their research, CACTI source code is still available.

Note that since CACTI is continually being upgraded, results for specific cache configurations and technologies may change as new versions are released and bugs are fixed. Please continue using a single version of the source code if consistency is important for your work. We do not plan on keeping previous web versions of CACTI available online.

As technology shrinks, the disparity between transistor and wire delay will grow. The properties of future caches/memories will primarily depend on the characteristics of the interconnection network that connects various sub-modules of a cache/memory. CACTI 6.5 is a significantly enhanced version that primarily focuses on interconnect design. In addition to a new streamlined code base with numerous bug fixes, 6.5 includes the following extensions over 5.3. 1) The ability to model different types of wires, such as RC based wires with different power, delay, and area characteristics and differential low-swing buses. 2) Ability to model Non-Uniform Cache Access (NUCA) for chip multiprocessors that takes into account the effect of network

contention during the design space exploration. 3) Power model for router components. 4) Improved design space exploration that takes into account different wire types and router types (for NUCA). 5) Improved API to specify constraints involving power, delay, area, and bandwidth. 6) Improved analytical models for dominant cache components such as wordlines and bitlines.

C Code snippet to implement cache in cacti6.5:

For Fetch stage:

```
# Cache size
-size (bytes) 64
//-size (bytes) 1048576
//-size (bytes) 2097152
//-size (bytes) 4194304
//-size (bytes) 8388608
//-size (bytes) 16777216
//-size (bytes) 33554432
//-size (bytes) 134217728
//-size (bytes) 67108864
//-size (bytes) 1073741824
# Line size
-block size (bytes) 8
//-block size (bytes) 64
# To model Fully Associative cache, set associativity to zero
//-associativity 0
//-associativity 1
-associativity 4
//-associativity 8
//-associativity 16
-read-write port 1
-exclusive read port 0
-exclusive write port 0
-single ended read ports 0
# Multiple banks connected using a bus
-UCA bank count 1
-technology (u) 0.032
//-technology (u) 0.045
//-technology (u) 0.068
//-technology (u) 0.090
# following three parameters are meaningful only for main memories
-page size (bits) 8192
-burst length 8
-internal prefetch width 8
```

following parameter can have one of five values -- (itrs-hp, itrs-lstp, itrs-lop, lp-dram, comm-dram)
//-Data array cell type - “lp-dram"
-Data array cell type - "itrs-hp"
//-Data array cell type - "itrs-lstp"
//-Data array cell type - "itrs-lop"
following parameter can have one of three values -- (itrs-hp, itrs-lstp, itrs-lop)
//-Data array peripheral type - "itrs-hp"
-Data array peripheral type - "itrs-lstp"
//-Data array peripheral type - "itrs-lop"
following parameter can have one of five values -- (itrs-hp, itrs-lstp, itrs-lop, lp-dram, comm-dram)
-Tag array cell type - "itrs-hp"
//-Tag array cell type - "itrs-lstp"
following parameter can have one of three values -- (itrs-hp, itrs-lstp, itrs-lop)
-Tag array peripheral type - "itrs-hp"
//-Tag array peripheral type - "itrs-lstp"
Bus width include data bits and address bits required by the decoder
-output/input bus width 256
//-output/input bus width 256
// 300-400 in steps of 10
-operating temperature (K) 350
Type of memory - I-cache (with a tag array) or ram (scratch ram similar to a register file)
or main memory (no tag array and every access will happen at a page granularity Ref: CACTI 5.3 report)
-cache type "I-cache"
//-cache type "ram"
//-cache type "main memory"
to model special structure like branch target buffers, directory, etc.
change the tag size parameter
if you want cacti to calculate the tagbits, set the tag size to "default"
-tag size (b) "default"
//-tag size (b) 45
fast - data and tag access happen in parallel
sequential - data array is accessed after accessing the tag array
normal - data array lookup and tag access happen in parallel
final data block is broadcasted in data array h-tree
after getting the signal from the tag array
-access mode (normal, sequential, fast) - "fast"
//-access mode (normal, sequential, fast) - "normal"
//-access mode (normal, sequential, fast) - "sequential"
DESIGN OBJECTIVE for UCA (or banks in NUCA)
-design objective (weight delay, dynamic power, leakage power, cycle time, area) 0:0:0:0:100
Percentage deviation from the minimum value
Ex: A deviation value of 10:1000:1000:1000:1000 will try to find an organization
that compromises at most 10% delay.

```
# NOTE: Try reasonable values for % deviation. Inconsistent deviation
# percentage values will not produce any valid organizations. For example,
# 0:0:100:100:100 will try to identify an organization that has both
# least delay and dynamic power. Since such an organization is not possible, CACTI will
# throw an error. Refer CACTI-6 Technical report for more details
-deviate (delay, dynamic power, leakage power, cycle time, area)
60:100000:100000:100000:1000000
# Objective for NUCA
-NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area)
100:100:0:0:100
-NUCAdeviate (delay, dynamic power, leakage power, cycle time, area)
10:10000:10000:10000:10000
-Cache model (NUCA, UCA) - "UCA"
//-Cache model (NUCA, UCA) - "NUCA"
# In order for CACTI to find the optimal NUCA bank value the following
# variable should be assigned 0.
-NUCA bank count 0
# NOTE: for nuca network frequency is set to a default value of
# 5GHz in time.c. CACTI automatically
# calculates the maximum possible frequency and downgrades this value if necessary
-Wire inside mat - "global"
//-Wire inside mat - "semi-global"
-Wire outside mat - "global"
-Interconnect projection - "conservative"
//-Interconnect projection - "aggressive"
# Contention in network (which is a function of core count and cache level) is one of
# the critical factor used for deciding the optimal bank count value
# core count can be 4, 8, or 16
//-Core count 4
-Core count 8
//-Core count 16
-Cache level (L2/L3) - "L2"
```

Duty cycle calculations for Fetch stage:

Valid path: Duty cycle matches an average of 71%
In-valid Paths: Duty cycle comes around 33.17%

For Writeback stage:

```
-size (bytes) 64
//-size (bytes) 16777216
//-size (bytes) 33554432
//-size (bytes) 134217728
//-size (bytes) 67108864
//-size (bytes) 1073741824
```

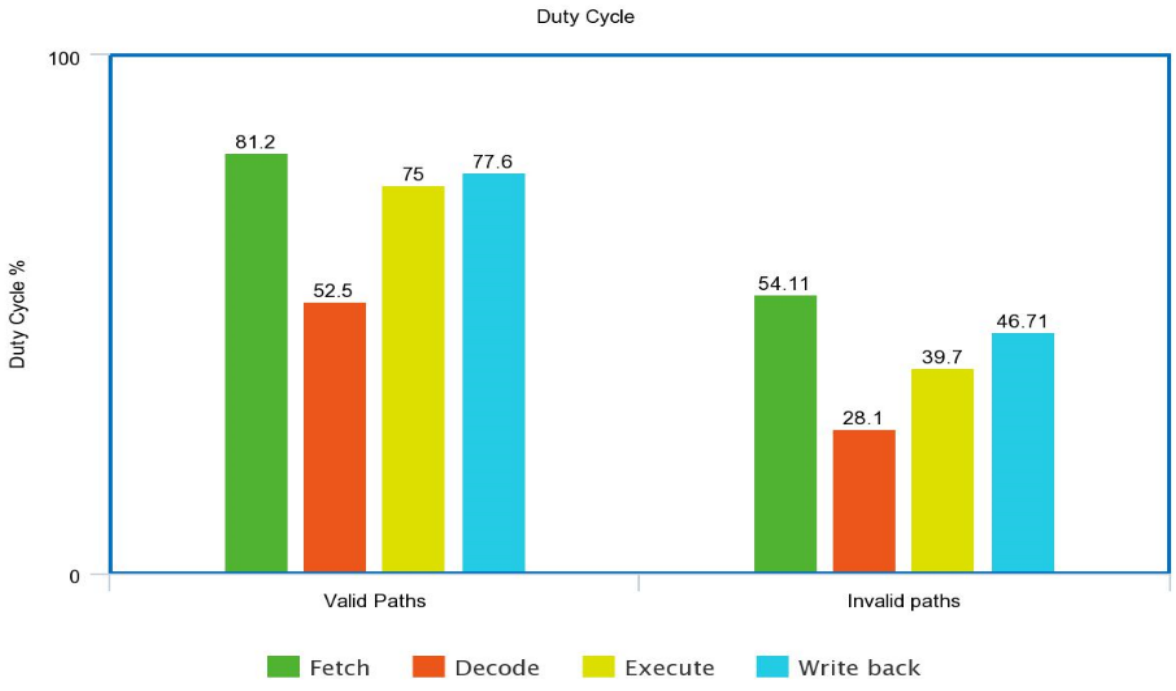
-block size (bytes) 8
-associativity 4
-read-write port 1
-exclusive read port 0
-exclusive write port 0
-single ended read ports 0
-UCA bank count 1
//-technology (u) 0.032
-technology (u) 0.045
//-technology (u) 0.068
//-technology (u) 0.078
following three parameters are meaningful only for main memories
-page size (bits) 8192
-burst length 8
-internal prefetch width 8
following parameter can have one of the five values -- (itrs-hp, itrs-lstp, itrs-lop, lp-dram, comm-dram)
-Data array cell type - “ltrs-hp"
following parameter can have one of the three values -- (itrs-hp, itrs-lstp, itrs-lop)
-Data array peripheral type - "itrs-hp"
following parameter can have one of the five values -- (itrs-hp, itrs-lstp, itrs-lop, lp-dram, comm-dram)
-Tag array cell type - "itrs-hp"
following parameter can have one of the three values -- (itrs-hp, itrs-lstp, itrs-lop)
-Tag array peripheral type - "itrs-hp"
Bus width include data bits and address bits required by the decoder
//-output/input bus width 512
-output/input bus width 64
-operating temperature (K) 350
-cache type “D-cache"
to model special structure like branch target buffers, directory, etc.
change the tag size parameter
if you want cacti to calculate the tagbits, set the tag size to "default"
-tag size (b) "default"
//-tag size (b) 45
fast - data and tag access happen in parallel
sequential - data array is accessed after accessing the tag array
normal - data array lookup and tag access happen in parallel
final data block is broadcasted in data array h-tree
after getting the signal from the tag array
//-access mode (normal, sequential, fast) - "fast"
-access mode (normal, sequential, fast) - "normal"
//-access mode (normal, sequential, fast) - "sequential"
fast - data and tag access happen in parallel
sequential - data array is accessed after accessing the tag array
normal - data array lookup and tag access happen in parallel

final data block is broadcasted in data array h-tree
after getting the signal from the tag array
//-access mode (normal, sequential, fast) - "fast"
-access mode (normal, sequential, fast) - "normal"
//-access mode (normal, sequential, fast) - "sequential"
DESIGN OBJECTIVE for UCA (or banks in NUCA)
//-design objective (weight delay, dynamic power, leakage power, cycle time, area)
100:100:0:0:0
-design objective (weight delay, dynamic power, leakage power, cycle time, area) 0:0:0:100:0
-deviate (delay, dynamic power, leakage power, cycle time, area)
20:100000:100000:100000:1000000
//-deviate (delay, dynamic power, leakage power, cycle time, area)
200:100000:100000:100000:20
-Cache model (NUCA, UCA) - "UCA"
//-Wire signalling (fullswing, lowswing, default) - "default"
-Wire signalling (fullswing, lowswing, default) - "Global_10"
-Wire inside mat - "global"
//-Wire inside mat - "semi-global"
-Wire outside mat - "global"

Duty cycle calculations for writeback stage:

Valid Paths: Duty cycle matches an average of 78.8%%
In-valid Paths: Duty cycle comes around 44.62%

Dutycycle comparison



The values obtained above are done by duty cycle balancing method. Now that we have obtained the values, we need to compute the optimum value of duty cycle for the critical and non-critical paths and using it we must develop an algorithm for generating a new Instruction Set Encoding (ISE) pattern so that the microprocessor functioning provides the desired duty cycle thus reducing the aging process.

Once the new pattern is created, it is modelled using SimpleScalar and the delay models are compared. The algorithm must be an iterative one such that it runs continuously till the ISE pattern for the optimum value of duty cycle calculated is achieved.

Aging aware Instruction Set Encoding (ArISE)

This technique exploits the fact that the instruction set encoding (ISE) has a considerable impact on wearout of the decoding stages, since it affects the input patterns (via the applied opcodes) and the gate-level implementation of these stages, which both influence wearout. To find a good instruction set encoding in terms of MTTF, our approach uses a hierarchical optimization algorithm based on simulated annealing to obtain an aging-aware opcode for each instruction in such a way, that the overall lifetime of the decoding stages is improved. Thereby, only the representing bit patterns are modified, while the opcode length remains unchanged. Together with existing aging mitigation techniques for other stages, this approach can help to significantly improve the overall microprocessor lifetime.

Improving the ISE is a challenging task, since many instruction encodings have to be modified to become aging-aware. For example, in case of the FabScalar microprocessor the instruction set architecture contains 131 instructions that are encoded using 8 bits. This means that there are $(28)! / (28 - 131)! \approx 10297$ encoding possibilities. Since most encodings infer modifications in the gate-level implementation of the stages and affect signal duty cycles as well as switching activities, each encoding requires a new synthesis and simulation flow for aging analysis. Moreover, instructions cannot be considered in- dependently, since duty cycles and switching activities depend not only on the actual instruction, but also on the preceding and subsequent instructions. Therefore, to see the effect of a particular encoding, or even optimizing

the encoding for just one instruction, one has to simulate instruction streams. Due to this complexity an exhaustive method is infeasible. Hence, we use a hierarchical heuristic methodology based on simulated annealing to find an aging-aware ISE based on duty cycle balancing technique.

ArISE implementation using SimpleScalar:

SimpleScalar:

The SimpleScalar tool set is a system software infrastructure used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. Using the SimpleScalar tools, users can build modeling applications that simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. The SimpleScalar tools are used widely for research and instruction, for example, in 2000 more than one third of all papers published in top computer architecture conferences used the SimpleScalar tools to evaluate their designs. In addition to simulators, the SimpleScalar tool set includes performance visualization tools, statistical analysis resources, and debug and verification infrastructure.

SimpleScalar simulators can emulate the Alpha, PISA, ARM, and x86 instruction sets. The tool set includes a machine definition infrastructure that permits most architectural details to be separated from simulator implementations. All of the simulators distributed with the current release of SimpleScalar can run programs from any of the above listed instruction sets. Complex instruction set emulation (e.g., x86) can be implemented with or without microcode, making the SimpleScalar tools particularly useful for modeling CISC instruction sets.

The PISA instruction set (a.k.a. the portable instruction set architecture) is a simple MIPS-like instruction set maintained primarily for instructional use. A GNU GCC-based cross-compiler and pre-built libraries are also available for this target. The PISA target is particularly useful for computer engineering instruction as the tools can be built on a wide range of host platforms, including Linux/x86, Win2000, SPARC Solaris, and others.

Algorithm for ArISE based on duty cycle balancing

ITERATIVE APPROACH

The starting point of the optimization algorithm is a random ISE (here: FabScalar's standard ISE). For this ISE all pipeline stages have to be synthesized and their MTTF values need to be extracted according to the flow (1+2). Afterwards, a simulated annealing (SA) algorithm is invoked (Step 3). As a first step it generates a "neighbor" ISE for the current one (Step 3.1). For this new ISE the overall MTTF considering the decoding stages is estimated (Step 3.2). As neighbor definition we use in this work the following principal. Two ISEs are neighbors if and only if a) both ISEs differ only in one instruction opcode (only possible if there are more binary opcodes than instructions), or b) the second ISE can be derived from the first one by exchanging the opcodes of two instructions. Using this definition it is guaranteed that every possible ISE can be evaluated and that the difference between the new and the old ISE is not that huge that the optimization process is too random. The next step is to evaluate if the new ISE will replace the old one (Step 3.3).

Algorithm:

1. Select a starting instruction set encoding (ISE): ISE old
2. Compute duty cycle
3. While solution is not good enough or number of steps < limit do

3.0. Adjust temperature *T*

3.1. Generate ISE new

3.2. compute duty cycle

3.3. If duty cycle not equal to optimum duty cycle

3.3.1. then GoTo 3.1.

3.3.2. else ISE old = ISE new

ISE best = ISE old */* store best ISE */*

GoTo 3.1.

EndIf

End.

Instruction Set Encoding based on Iterative approach

Basic MIPS ISE –

Eg

ADD – 00100000

MUL – 00011000

DIV – 00011010

Iterative Approach

Encoding using Smaller bits

ADD – 00000010

MUL – 00000011

DIV – 00000110

This model gave an improvement of 10% in delay and MTTF.

Delay values for various ISE models

ISE 1

No of Hits	Delay (ns)
2000	2.2
4000	6.47
6000	15.8
8000	22.91
10000	22.92
12000	22.89

ISE 2

No of Hits	Delay (ns)
2000	1.9
4000	5.12
6000	13.97
8000	19.26
10000	19.26
12000	19.26

ISE 3

No of Hits	Delay (ns)
2000	2.1
4000	3.75
6000	12.65
8000	18.7
10000	18.7
12000	18.7

Algorithm for ArISE based on duty cycle balancing

HIERARCHICAL APPROACH

To improve the efficiency of the optimization process, I have tried to minimize the number of steps until an acceptable solution is found. Therefore, I applied the *hierarchical optimization approach* shown in Fig. 3. First the instructions are classified into different (sub) groups based on their characteristics, e.g. all ALU-instructions are put into the same group (Step 1). Afterwards, at each hierarchy-level, the (sub) groups are ranked according to their aging impact (Step 2). However, as the aging rate strongly depends on the encoding, the real aging rates cannot be used for this ranking. As a matter of fact, instructions affecting the hardware-implementation of the decoder have the biggest aging impact. Therefore, the impact on hardware modifications due to a particular coding scheme is used to form the ranking. If there are several (sub) groups affecting the hardware-implementation, they are ranked depending on their occurrence frequency. The next step is to find the best encoding for each (sub) group, for all hierarchy-levels starting at the coarsest hierarchy-level (Step 3.1). Thereby, at each level, the group ranking determines the optimization order. As a result, an exhaustive technique can be applied if there are only a few (sub) groups at the same hierarchy level with considerable aging impact.

Using this approach, at each hierarchy-level, only the opcode bits corresponding to that level are modified. For example, if there are 16 groups at the coarsest level, only the four most significant opcode bits are modified (i.e. 16! configurations). As we used just 16 instruction groups, each of which contained at most 16 instructions the search space for the entire optimization process became much smaller than the original space ($16!2 \approx 4 \cdot 1026$ vs. 10297), leading to fewer optimization steps.

Algorithm:

1. Partition instructions into groups and subgroups

/*Instruction groups, subgroups inside groups, */ /*instructions insides subgroups, etc.*/

2. Rank each group (and subgroups subsequently)

2.1 Based on their hardware-impact

2.2 If there are groups/subgroups with same ranking then use occurrence frequency to rank these

3. For the coarsest down to the finest hierarchy-level do

For the highest ranked group down to the lowest do

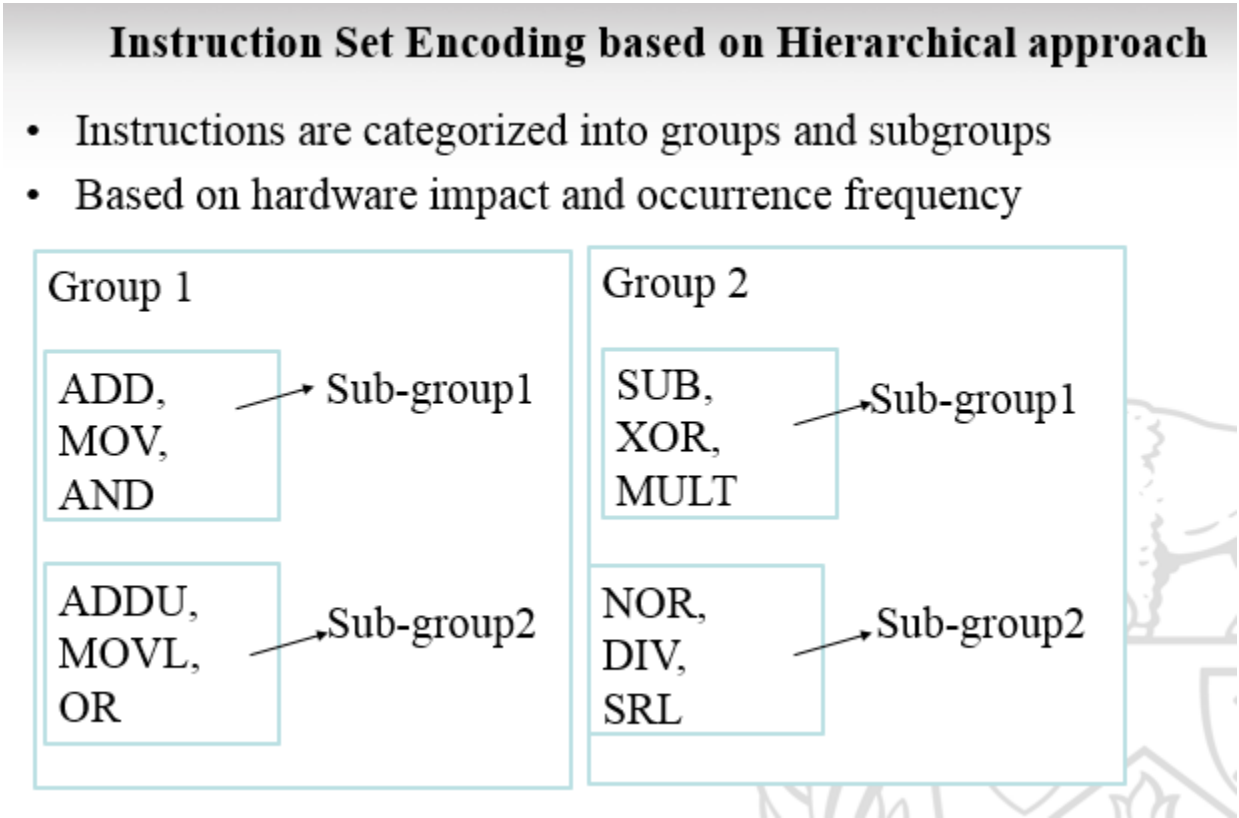
3.1 Find the best encoding for the elements within that group

/*Either exhaustive or with simulated annealing*/

3.2 Stop as soon as duty cycle is satisfactory.

Endfor

Endfor.



Delay values for various ISE models

ISE 1

No of Hits	Delay (ns)
2000	1.95
4000	6.62
6000	10.4
8000	14.11
10000	16.29
12000	16.05

ISE 2

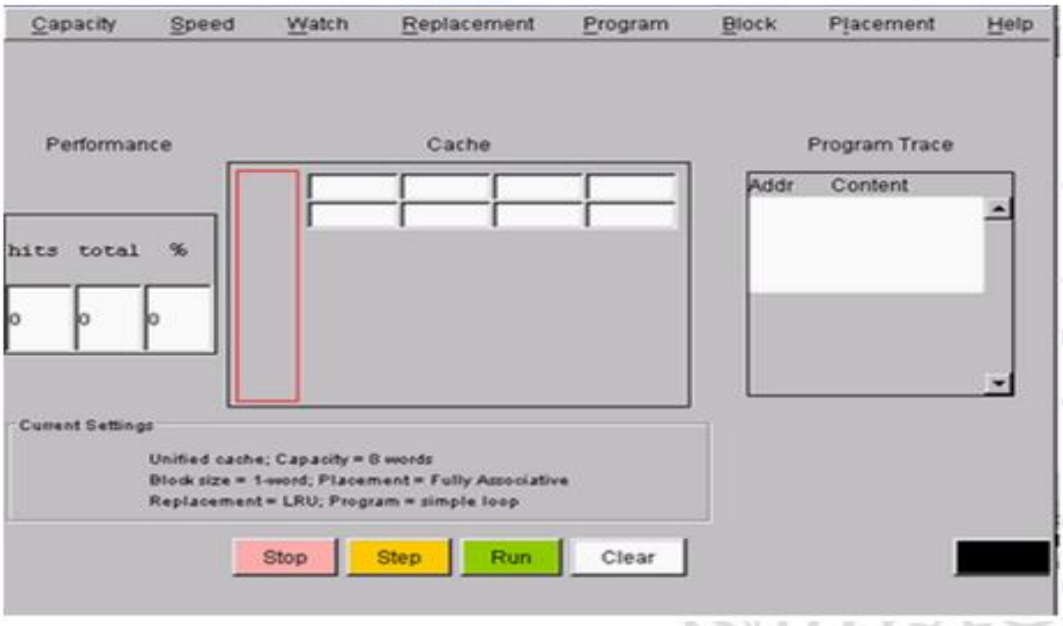
No of Hits	Delay (ns)
2000	1.81
4000	6.1
6000	11.1
8000	13.00
10000	14.38
12000	14.38

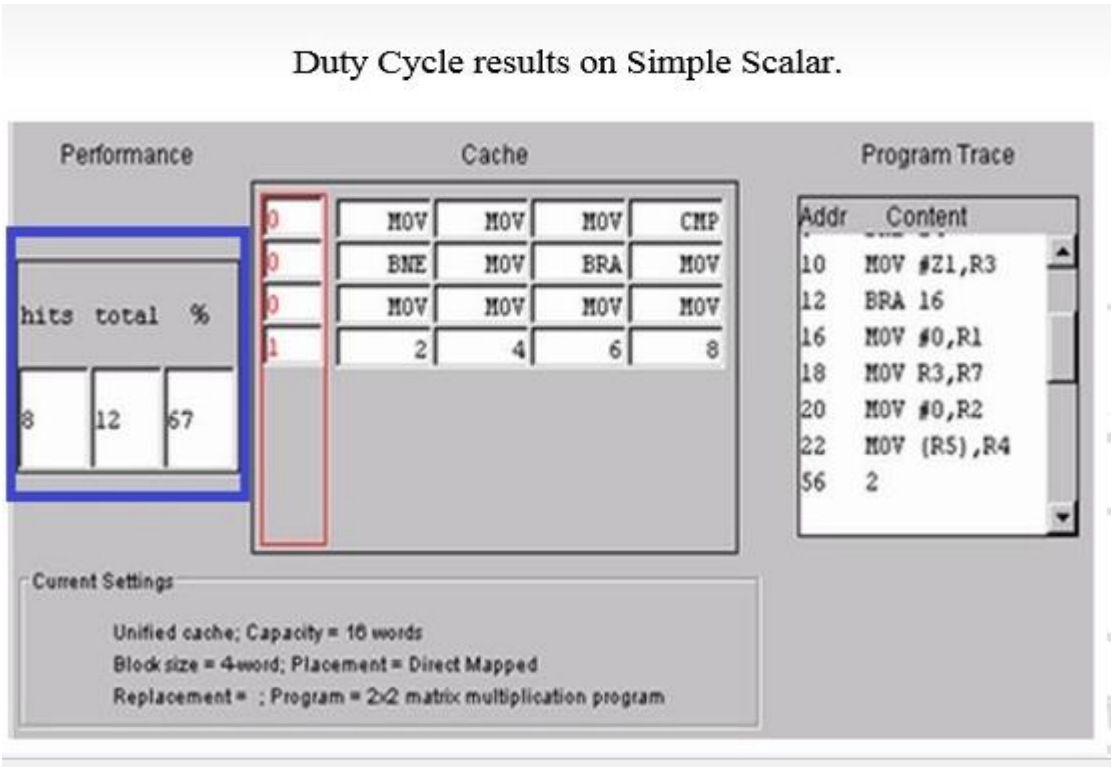
ISE 3

No of Hits	Delay (ns)
2000	1.6
4000	3.75
6000	7.2
8000	11.69
10000	12.31
12000	12.31

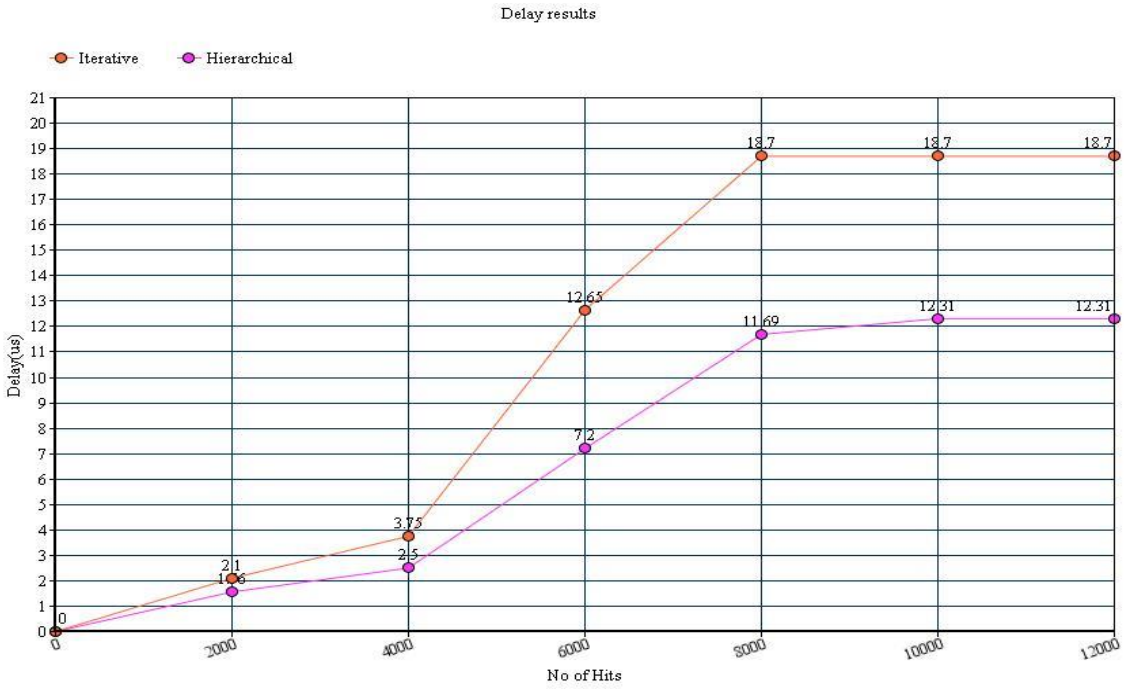
Results of SimpleScalar Simulation

User Interface of Sim Out-of-order simulator





Delay and MTTF model of both ArISE algorithms:



RESULT AND CONCLUSION:

The Optimum value of Duty cycle I got was 64% including valid and invalid paths. By using simplescalar and Hierarchical algorithm, I was able to reach as much as 67%.

Initially, I used the iterative algorithm which improved the delay and MTTF by 10% which was not satisfactory. Later I used the Hierarchical algorithm with which I was able to improve the delay and MTTF by 54%.

With proper grouping and subgrouping of instructions and current device level aging inhibition techniques, the delay and MTTF can be improved as much as to 80%.

REFERENCES

[1] ArISE: Aging-aware instruction set encoding for lifetime improvement by Oboril, Fabian; Tahoori, Mehdi, 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), 2014.

[2] Aging-Aware Instruction Cache Design by Duty Cycle Balancing by Tao Jin; Shuai Wang,

2012 IEEE Computer Society Annual Symposium on VLSI, 2012.

[3] System-Level Modeling And Reliability Analysis Of Microprocessor Systems, Dissertation

Presented by Chang-Chih Chen.

[4] Efficient Instruction Encoding for Automatic Instruction Set Design of Configurable ASIPs,

by Lee, Jong-eun; Choi, Kiyoung; Dutt, Nikil, Proceedings of the 2002 IEEE/ACM international conference on computer-aided design, 11/2002.

[5] The SimpleScalar Tool Set, Version 2.0, by Doug burger and Todd M Austin, SimpleScalar LLC.

[6] Aging-Aware Design of Microprocessor Instruction Pipelines, Fabian Oboril and Mehdi B. Tahoori, Ieee Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 33, No. 5, May 2014.

[7] Aging-Aware Instruction Cache Design by Duty Cycle Balancing, Tao Jin and Shuai Wang, 2012 IEEE Computer Society Annual Symposium on VLSI.

[8] Aging-aware Timing Analysis Considering Combined Effects of NBTI and PBTI, Saman Kiamehr, Farshad Firouzi, Mehdi. B. Tahoori, International Symposium on Quality Electronic Design (ISQED), 2013.

[9] System-level modeling of microprocessor reliability degradation due to BTI and HCI, by Chen, Chang-Chih; Soonyoung Cha; Taizhi Liu; Milor, Linda, 2014 IEEE International Reliability Physics Symposium, 2014.

[10] Aging mitigation in memory arrays using self-controlled bit-flipping technique, by Gebregiorgis, Anteneh; Ebrahimi, Mojtaba; Kiamehr, Saman; Oboril, Fabian; Hamdioui, Said; Tahoori, Mehdi B, The 20th Asia and South Pacific Design Automation Conference, 2015.