A

DISSERTATION REPORT

on

DYNAMIC THRESHOLD PREDICTION TO MITIGATE VOLTAGE DROOP IN x86 AVX ISA PROCESSORS

by

Shri Bhushan Singh

2017PEV5146

Submitted in partial fulfilment for the award of degree of

MASTER OF TECHNOLOGY

in ***VLSI Design***

Internal Supervisor
Dr. C. Periasamy
Asst. Professor
Dept. of ECE
MNIT Jaipur

External Supervisor
Rohit Jindal
Engineering Manager
Intel India Pvt. Ltd.
Bengaluru

Department of Electronics and Communication Engineering

Malaviya National Institute of Technology, Jaipur, 2019

CERTIFICATE

Department of Electronics and Communication Engineering
Malaviya National Institute of Technology, Jaipur

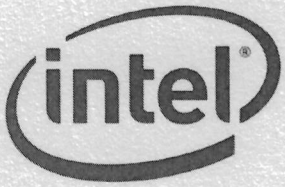
This is to certify that thesis entitled "**Dynamic Threshold Prediction to Mitigate Voltage Droop in x86 AVX ISA Processors**" which is submitted by **Shri Bhushan Singh (2017PEV5146)** in partial fulfilment of requirement for the degree of Master of Technology in VLSI Design submitted to **Malaviya National Institute of Technology Jaipur** is a record of students own work carried out under my supervision. The matter in this report has not been submitted to any other university or institution for the award of any degree.

Date

Place

Dr. C. Periasamy
(Project Supervisor)
Assistant Professor
Dept. of Electronics and
Communication Engineering
MNIT Jaipur 302017

CERTIFICATE



This is to certify that the Dissertation Report on "**Dynamic Threshold Prediction to Mitigate Voltage Droop in x86 AVX ISA Processors**" by **Shri Bhushan Singh** (2017PEV5146) is completed under my supervision and guidance, hence approved for submission in partial fulfilment of the requirement for degree of **Master of Technology** in **VLSI Design** to the department of Electronics and Communication Engineering, **Malaviya National Institute of Technology, Jaipur** in the academic session 2018-2019 for the full time post-graduation program of session 2017-2019.The matter in this report has not been submitted to any other university or institution for the award of any degree.

Date: 10th July 2019

Rohit Jindal
Engineering Manager
Intel India Pvt. Ltd., Bengaluru

Declaration

I, Shri Bhushan Singh, hereby declare that this thesis submission titled as Dynamic Threshold Prediction to Mitigate Voltage Droop in x86 AVX ISA Processors, is my own work and that, to the best of my knowledge and belief.

It contains no material previously published or written by another person, nor material which to be substantial extent has been accepted for the award of any other degree by university or other institute of higher learning.

Wherever I used data (Theories, results) from other sources, credit has been made to that sources by citing them (to the best of my knowledge).Due care has been taken in writing the thesis, errors and omissions are regretted.

Shri Bhushan Singh

ID: 2017PEV5146

Acknowledgement

I would like to thank all peoples who have helped me in this project, directly or indirectly. I am especially grateful to my supervisor ***Dr. C. Periasamy*** *(Assistant Professor, Dept. of ECE, MNIT Jaipur) for his invaluable guidance during my project work, encouragement to explore parallel paths and freedom to pursue my ideas. My association with him has been a great learning experience. He made it possible for me to discuss with a number of people and work in different areas.*

I express my sincere gratitude to ***Mr. Rohit Jindal*** *(Engineering Manager, Intel) and* ***Kartheek Domakuntla*** *(Core Verification Lead, Intel) for the support and guidance in this research work.*

I express my sincere gratitude to Professor ***D. Boolchandani*** *(Head of Department)for his support and guidance. Many thanks to the committee members for their valuable comments and guidance in this research work.*

I would also like to thank Ministry of HRD, Government of India for its support to me to pursue my Masters in VLSI Design from Malaviya National of Technology, Jaipur.

-Shri Bhushan Singh

List of Abbreviations

AVX- Advanced vector extension

ISA- Instruction set architecture

SIMD- Single Instruction Multiple Data

FMA- Fused Multiply Addition

OOO – Out of Order Unit

EXE- Execution Unit

MEU- Memory Execution Unit

CPU- Central Processing Unit

Ld- Load

St- Store

Vec_Uop's – Vector Micro Operations

Non_Vec_Uop's – Non-Vector Micro Operations

Uop- Micro-Operation

FE- Front End

MSID- Micro Sequence Instruction Decoder

IFU - Instruction Fetch Unit

IQ- Instruction Queue

FPU- Floating Point Unit

FIFO- First In First Out

IPC - Instructions Per Cycle

RISC- Reduced Instruction Set Computer

CISC- Complex Instruction Set Computer

RAT- Register Alias Table

ROB- Reorder Buffer

RS- Reservation Station

Alloc- Allocation Unit

PRF- Physical Register File

Disp- Dispatch Block

CTE- Cluster Test Environment

RAW- Read after Write

WAW- Write after Write

WAR- Write after Read

Th/Thr- Threshold

EC- Energy Cost

FSDB- file system data base

List of Tables

List of Figures

Abstract

Over last few decades, there were exponential advances in the processor. In general 64-bit processors do computation using small data storage registers, but due to the need to process large amount of data, in modern processors vector instructions have been added. These vector instructions operate on larger size of registers (128 bit, 256bit and 512bit), thereby consuming large amount of power on chip. Due to their large power consumption nature these instructions e.g. SIMD (single instruction multiple data) and FMA (fused multiply accumulate) are called heavy instructions. These are very commonly used in multimedia applications like image processing and deep learning. When these instructions are being processed by processors large amount of current will be drawn leading to sudden drop of on-chip voltage which is called voltage droop.

One such case occur in processors when there is random and frequent variation in incoming instruction load, causing di/dt variations ultimately leading to frequent on-chip voltage level changes, which can cause circuit fault and ultimately chip burn also. To avoid voltage droop analog detectors have been placed which monitor the voltage level continuously and prevent the voltage from going beyond the level by stopping further execution of heavy instructions. But the problem with the analog detectors are that they are not proactive, once they detect the voltage around the threshold level only then they stop the heavy load execution. So to overcome this problem dynamic threshold prediction has been done which will detect the instructions based on the predicate in the reservation station and based the ratio of heavy to light instructions (instruction ratio) an optimum threshold level will be determined which will not fully block the execution dispatch ports. Light instructions will not be blocked from being executed while heavy instructions will wait until proper power level is available. It has been shown that this instruction based predicted dynamic threshold level minimizes the dispatch port blockage thus boosting processor performance by saving cycles. Further it has been shown how three level of threshold is better that single and fixed threshold.

Contents

Chapter: 1
INTRODUCTION

1.1 Introduction and Motivation

Over the past six decades, there were exponential advances in semiconductor fabrication technology, as foretold by Moore's law. These advances in turn have enabled the design of powerful processors residing in the computing devices that pervade our lives today. Our quest for fast and energy efficient devices has led us to processor designs that are among humankind's most complex creations.

Processor architects have been incorporating more and more features and operating modes into their designs over the years. The prevailing processor core design trends today utilize complicated execution structures for out of order execution and advanced power management techniques. In addition, designers are incorporating more and more application specific accelerators into their designs to squeeze out more performance under always tighter energy and power constraints. These accelerators reside on a chip alongside the processor cores, a memory subsystem, and peripheral components, all connected via an onchip interconnect. Each of these individual components are complex on their own, with several design blocks, power management features, and operating modes.

Due to increasing demand of large data processing in domains like multimedia and deep learning fields, new instructions have been added in processors, these newly added instructions like SIMD(single instruction multiple data) and FMA(fused multiply addition) require large registers because they perform similar kind of operation on large data set, thereby consuming large amount of power. These are called AVX instructions (advanced vector instructions). These AVX instructions operate on large registers of 128, 256 or 512-bit. These AVX instructions speed application because they can process more data per instruction. On the other hand simple processes like text editing uses smaller registers and consume less power.

Voltage droop is major phenomenon in modern IC circuits and CPU's operating at higher frequencies and lower technology nodes. When the incoming instructions are varying too frequently, the logic circuit draw high switching current when the load is high while in low load case it draw very less current, resulting in undesirable voltage droop phenomenon. As a consequence of this irregular and frequent current change in

circuits operating at high very high frequencies, the voltage droop causes delay and can lead to faults in circuit operation.

For this dissertation purposes, the voltage droop occurring in the CPU has been explained and methods to avoid the loss due to the droop has been proposed and verified.

1.2 Objective

The purpose of this project is to avoid droop effect in CPU proactively. Analog detectors can detect the voltage level and switch off certain parts of circuit which require large amount of power. But this method is not proactive as only after detection of voltage level around the threshold value, it stops the execution flow, thereby affecting the performance due to multiple cycle execution blockage.

So to avoid this situation we can design a module which will calculate the power needed in the upcoming cycles and will send a request to the PCU (power control unit) even before the execution of the instruction, thus proactively avoiding both droop effect and also boosting the performance by reducing the number of blocked cycles. And to show this various concepts like cycle energy cost and port dispatch block etc has been discussed.

1.3 Thesis Organisation

This thesis is organized in total 7 chapters including this introduction chapter. Following the introduction, chapter 2 gives the basic idea about intel cpu architecture, An overview of the art of verification its (will give a glimpse of the related works through providing some literature analysis on this topic)

In chapter 3, there is discussion about voltage droop effect in general.

In chapter 4, there is detailed discussion about power consuming instructions and non power consuming instructions. There is brief discussion about droop phenomenon happening in processors.

In chapter 5, various solutions to mitigate voltage droop problem in processors are discussed and advantages and shortcomings of these methods has been discussed.

In chapter 6, describes the proposed dynamic threshold prediction to mitigate the voltage droop phenomenon. Some calculations related to this are done. There are some tables which are showing the comparison of proposed design and the old design.

Finally in chapter 7, conclusion of whole project and future work which can be done after this work is explained.

Chapter: 2
LITERATURE REVIEW

This chapter provides some basic information and literature survey which is required in the next chapters. In the first section, there is a general overview of Intel x86 Architecture and out of order based processor, its advantages and detailed structure of Out of Order unit and the next section provides basics of instruction set architecture. The next section introduces energy consumption at various abstraction layers in processors. And in the next section, there is an overview of the importance of the functional validation aspect of design.

2.1 Intel CPU Architecture (x86 uArch)

It is a very common myth that Intel x86-64 processors are power consuming due to their CICS architecture (complex instruction set).But Intel processors are not fully based on CISC architecture it is a mixed combination of both CISC and RISC.[27]

2.1.1 RISC Vs CISC

	CISC	**RISC**	**uCode CISC &OOO**
Memory req	Smaller	Larger	Smaller
Decode	Complex	Simple	Complex
Registers	Fewer	More	More
Clock speed	Slower	Faster	Faster
Inst. Complexity	Complex	Simple	Simple

Table 2.1. Comparison RISC vs. CISC [28]

So to increase the speed and build a very fast RISC machine

Translate from IA (CISC) to RISC breaking the complex instructions in to simpler micro-operations(called uops - micro-operation).This work is done internally by the out of order and micro instruction translation engine.

In the next page there are figures showing the two types of execution.

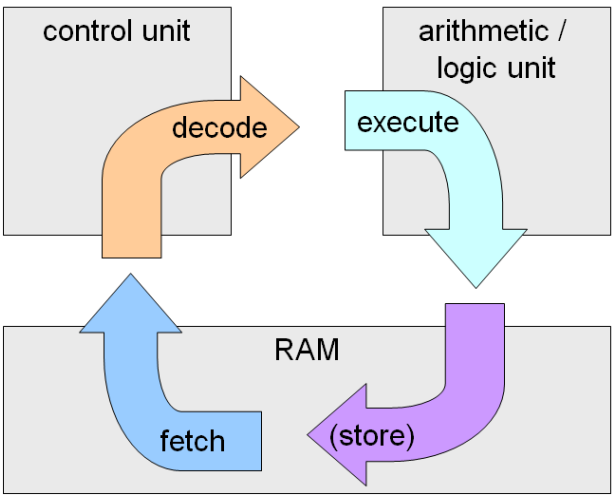


Figure 2.1: Simple Processor Execution Phases[27]

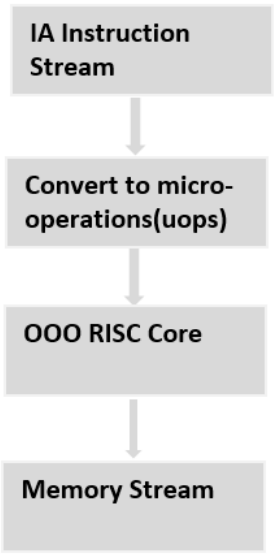


Figure 2.2: uCode CISC & OOO [9]

2.1.2 Intel core x86 architecture

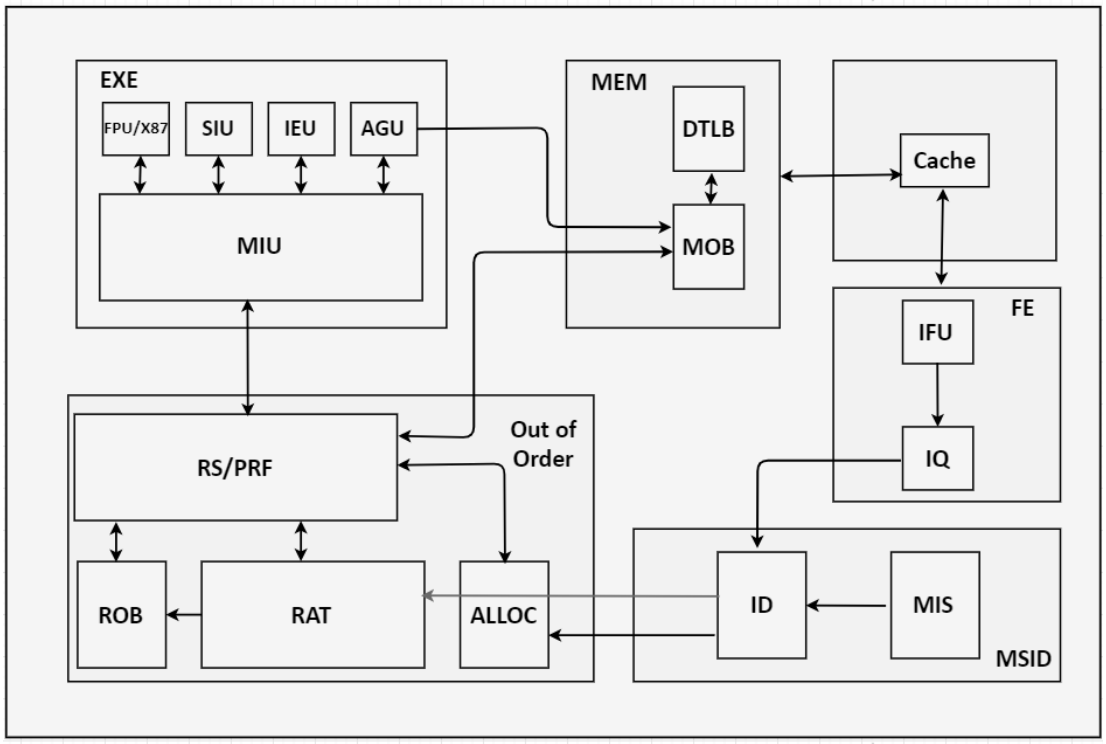


Figure 2.3: Intel Core (CPU) overview

Intel x86 core consists of mainly five units-

i- **FE/MSID**-(Front End/ Micro Sequence Instruction Decide)- This unit is responsible for decode and fetch of instructions from the memory and converting the complex instructions into simpler micro-operations and sending the uops to out of order unit through instruction queue.[11]

ii- **Out of Order (OOO)**- This is the most complex block in core. This block is responsible for sending the uops to execution unit in out of order fashion based on the resource availability of resources and in the end in order retirement of the instructions.[9]

iii- **Execution Unit**- This is the most stable unit, it gets all the data from out of order and memory unit and execute the uops in the corresponding function block and send them back to ooo and memory.[15]

iv- **MEU**- Memory Execution Unit- this unit is responsible for providing operand and other necessary data to various units in the core.

v- **MLC**- Mid Level Cache- this block is responsible for holding speculative data which may be required in the upcoming cycles.[23]

2.1.3 Working principle of Out of Order based Processor

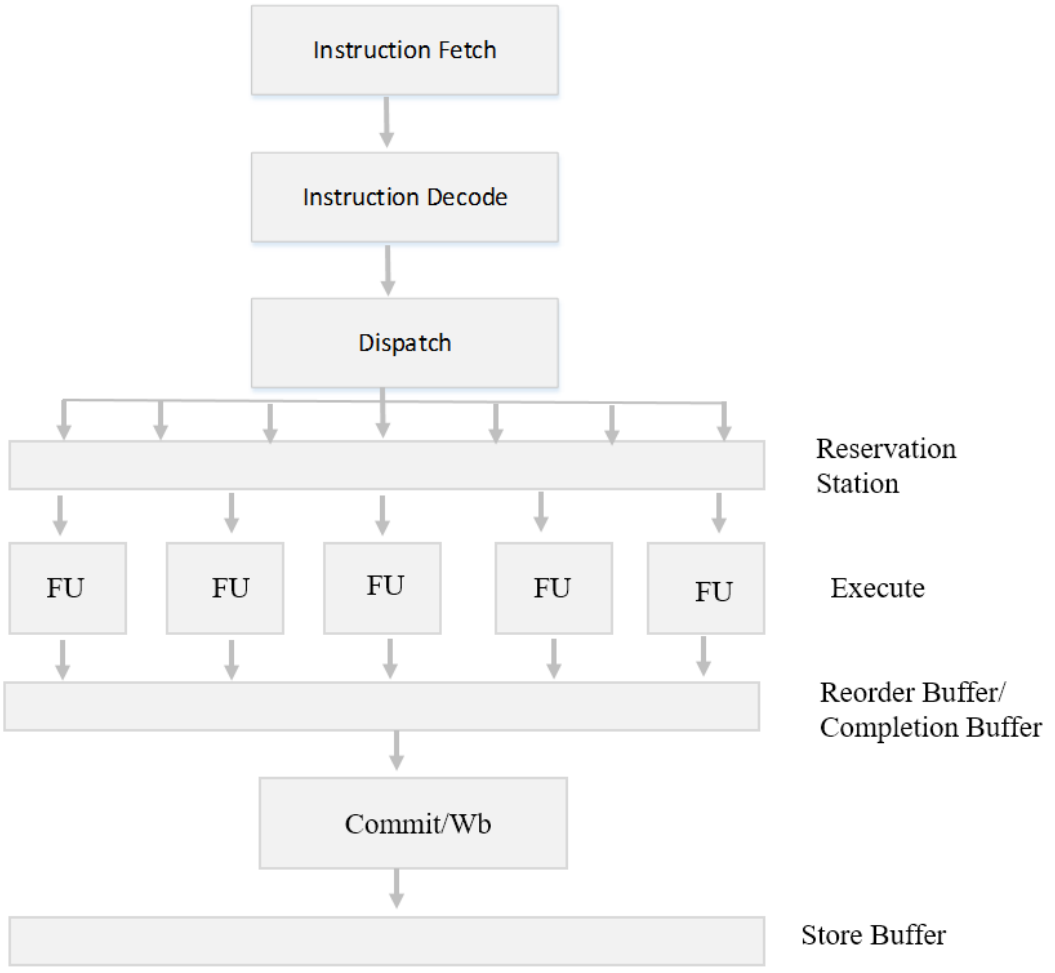


Figure2.4: Out of Order Execution General Overview [18]

2.2 Overview of Out of Order Unit

2.2.1 Out of Order Unit Structure

The Out-of-Order (OOO) unit is responsible for dispatching uops for execution out of program order, based on source availability, and for marshalling them back into program order for retirement.

Allocation

The OOO cluster receives uops from the Front-End (FE).These uops are allocated pipelined resources as per their requirement. The uop allocation is stalled if the required resources are not available. The sources and the destination registers of the uops are renamed to the Reorder Buffer (ROB) entries, which serves as a register file for reading and writing uops results.

Reservation Station

Next the uops are written into the Reservation Stations (RS). The RS first checks for sources location – the sources may be in the RS (not produced and therefore still tracked by the RS table) or in the Physical Register File (PRF). The uops sources are tagged with the appropriate connection to the producing uops that are still in the RS or marked as valid if the sources are in the PRF. If the source values are not available from the PRF the uops will wait in the RS for them to become available. These values become available when the uops producing them execute (from bypass), or when they write their results back into the PRF. In all these case, the tracking scheme in the RS will indicate that a source is valid for all its consumers in the RS. The value can be used when in one of the bypass levels or when it is available for the PRF. The uops with the available sources are dispatched from the RS for execution to the Execution Cluster according to a FIFO scheme that is part of the dispatch logic. For loads and stores the relevant information is sent to the Memory Cluster on dispatch.

Re-Order Buffer (ROB)

After execution, uops write their results: the data and arithmetic flags are written back into the PRF and fault data and valid information are written back into the ROB. The executed uops are retired in the original program order from the ROB. If uops executed correctly and no event needed to be handled then the uops are retired successfully. Otherwise, the pipeline is cleared and appropriate event handler is invoked to service the event.

2.2.2 Performance gain due to branch prediction and out of order

A simple example to show the benefits of out of order processing and branch prediction:

A	:	loop:	x0 ← [x1][0]	(data read)
B	:		x0 ← x0*x2	(multiply)
C	:		x1 ← x0	(writeback)
D	:		x1 ← [x1][1]	(pointer increment)
E	:	entrance:	if([x] is != nil) goto loop	(branch prediction)

Suppose we have 2 stage pipelined Cache and 4 stage pipelined floating point execution.

Case I - *Without out of order and branch prediction* – as shown into the figure it takes total 12 cycles to complete one iteration, without branch prediction and out of order execution.

Stage/Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
fetch	A1	B1	C1	D1	E1								
issue		A1	B1	C1	D1	E1							
execute1			A1		B1				C1	D1		E1	
execute2				A1		B1				C1	D1		
execute3							B1						
execute4								B1					

Figure 2.5: 12 cycles per iteration (without out of order and branch prediction)

Case II - *With out of order and without branch prediction* – it takes 8 cycles for the example thus making it faster than the first case.

Stage/Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
fetch	A1	B1	C1	D1	E1								
issue		A1	B1	C1	D1	E1							
execute1			A1		B1	D1		E1	C1				
execute2				A1		B1	D1			C1			
execute3							B1						
execute4								B1					

8 cycles

Figure 2.6: 8 cycles per iteration (with out of order and no branch prediction)

Case III – With out of order and branch prediction – it takes only 5 cycle to complete one iteration, so it makes the processor very fast.

Stage/ Cycle	**1**	**2**	**3**		**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
fetch	A1	B1	C1		D1	E1	A2	B2	C2	D2	E2	A3	B3	C3	D3	E3	A4	B4	C4
issue		A1	B1		C1	D1	E1	A2	B2	C2	D2	E2	A3	B3	C3	D3	E3	A4	B4
execute1			A1			B1	D1		A2	C1	B2	D2		A3	C2	B3	D3		
									E1					E2					
execute2					A1		B1	D1		A2		B2	D2		A3		B3	D3	
execute3								B1					B2					B3	
execute4									B1					B2					B3

5 cycles

Figure 2.7: 5 cycles per iteration (out of order with branch prediction)

2.3 Energy Consumption Estimation at different levels of Abstraction Layer

There are various ways to look any digital design or processor in general, the same system and its behavior can be described from different perspective at different of level of abstraction. Similarly, in a processor the energy consumption can be estimated at different levels of abstraction. In the following section an overview of energy

consumption estimation at different abstraction level is provided, with the advantages and disadvantages at various abstraction layers.

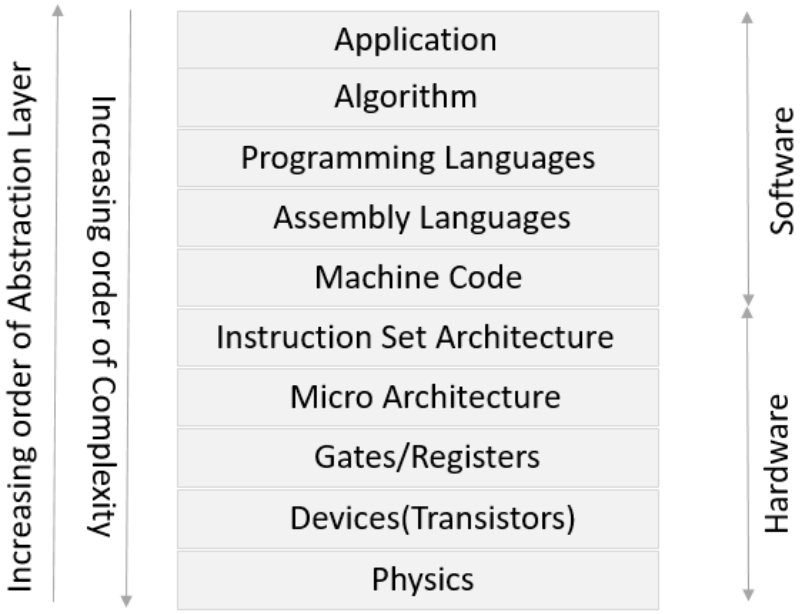


Figure 2.8: Abstraction Layers of a Processor

2.3.1 Energy Consumption at Source-Code Abstraction Level

The top most or highest level of abstraction level is source code level in which energy consumption analysis can be done. Source code level is the easiest way to estimate the energy consumption because source codes are easily available and other information needed related to processor like memory transfer and memory accesses and number of logical or arithmetic operations performed can be roughly estimated through any high level language like C++ or java. But this not the best way to analyze the energy consumption of any hardware, as this is the top most level of abstraction and it's difficult to establish a direct relationship between the operation happening at hardware level and source code. There are different kind of compilers available in market and each of these are specific to the programming language and to improve the compilation performance these compilers often target a specific type of hardware and moreover compilers are often designed optimized e.g. instruction scheduling and unrolling of loops to decrease compilation time. And avoiding the loop and execution path is difficult during energy cost estimation. Very less research in the area of energy cost

estimation at source code level has been done. In [23] before estimation of energy cost path information has been analysis has been done.

2.3.2 Energy Consumption at Functional Level

A processor or any digital system is a combination of various functional blocks. Estimation of the energy consumption of these individual blocks can be performed and the summation of these individual block energy consumption will result in the total energy consumption of the system. The energy consumption at this level is generally estimated through either direct measurement or through simulation. But while calculating and modelling the total energy computation of the processor several factors need to been taken care like on-chip data transfer between the functional blocks and clock speed (frequency) and the number of functional unit blocks available to process same operation. And these factors are not static and the actual value can be only determined after task execution. Therefore this model of energy consumption can be applied to any processor and the complexity is also moderate at this level. In [22] energy consumption model has been developed which is a hybrid model of the two levels functional level and instruction level.

2.3.3 Energy Consumption at Instruction-Level

Instruction Set Architecture (ISA) is the mid-level in the abstraction layer of a processor. Since every instruction is related to some operation or functional block in a processor. There are many models available which estimate the energy consumption at instruction level. The estimation of energy consumption at instruction level basically done using either any instruction level simulator or by analyzing the assembly code of the. For each and every instruction estimation of energy cost is estimated. These estimations can be done into two fashions with taking design details into consideration and without taking design details into consideration. For more accurate modelling of energy cost design details like pipelining, cache miss/hit and other things should be taken care of. Estimation of base and scale energy cost and from these total energy cost of instructions has been calculated in Chapter 4.[8]

2.3.4 Energy Consumption at RTL (Register Transfer Level)

RT-level abstraction level refers to digital signal flow between hardware registers, memory, bus, combinational logic devices and logic unit etc. VHDL (VHSIC Hardware Description Language), System Verilog , VHDL,Verilog, and SystemC [14] are, fairly

often, used to describe RT abstraction level. RT-level energy estimation is used to quickly predict the total switching activity in logic design compared to the simulation speed of gate-level estimation. In this technique, switching activity profiles, circuit modules or control signals are integrated to construct macro-module which can be parameterized in terms of the supply voltage level, the internal organization of the processor and the input bus width etc.. In general, the RT-level energy estimation can be classified into two types. The first type is to use a simulator to extract actual parameters for macro-model equation. The second type is based on the static analysis of the structure of the circuit so that parameters for macro-model equation could be extracted.

2.3.5 Energy Consumption at Gate-Level

Gate-level describes the operation of the circuit in terms of a structural interconnection of boolean logic gates such as nor, add, and xor etc.. The design of gate-level is represented as a netlist, which describes the connectivity of an electronic design. Gate-level simulation (GLS) is often used at the early design stage. The simulation is performed to measure switching activity of every net of circuit. In order to perform simulation, technological information which provides timing, power information of the processor and Gate-level netlist that contains all of the logic and delays of the final system should be provided to the simulator. The drawbacks of this method are that the simulation is time consuming and Gate-level netlist is confidential for most of manufacturers.[30]

2.3.6 Transistor-Level

As the name suggests, transistor level describes the operating behavior of circuit elements such as transistor, capacitors, inductors and resistors. With the rapid progress of semiconductor technology, the transistor count on a single chip has reached 2,000,000,000 in latest ARM and 1,400,000,000 latest Intel i7. In this level, transistor is often modeled as a device with only two states, “on” and “off”. In “off” state the transistor is modeled as an open circuit, while in the “on” state, the transistor is modeled by a linear resistance [14]. Transistor level power estimation employs simulation to track the current drawn from the power supply. Only a few studies have been made at this level, because the simulation is time consuming and it requires the knowledge of the circuit layout.

Verification of a Design

2.4 Functional Verification

Functional verification can expose functional logic errors in the hardware designs which are described in behavioral model, register transfer level model, gate level model, or switch level model. Functional errors are introduced due to various factors including careless coding, misinterpretation of the specification, microarchitecture design complexity, corner cases, and so on. If any functional bug is found in a chip already fabricated, the error needs to be corrected and the modified version of the design needs to be fabricated again, which is very expensive. In the worst case, bug fixing after delivery to customers will entail a very costly replacement as well as re-fabrication expenses. For example, in 1994 Intel's Pentium processor had a functional error called FDIV bug and the company had to spend a staggering cost to replace the faulty processors [21][8]

2.5 CTE Methodology

One of the fundamental decisions that Intel took early in the core processor development program was to develop Cluster Test Environments (CTEs) and maintain them for the life of the project. There is a CTE for each of the clusters into which the processor design is logically subdivided. These CTEs are groupings of logically related units (e.g. all the execution units of the machine constitute one CTE) surrounded by code that emulates the interfaces to adjacent units outside of the cluster and provides an environment for creating and running tests and checking results. The CTEs took a good deal of effort to develop and maintain, and were themselves a source of a significant number of bugs. However, they provided a number of key advantages: First and foremost, they provided controllability that was otherwise lacking at the full-chip level. An out of order, speculative execution engine like the processor is inherently difficult to control at the instruction set architecture level. Assembly language instructions (macroinstructions) are broken down by the machine into sequences of microinstructions that may be executed in any order relative to one another and to microinstructions from other preceding or following macroinstructions. Trying to produce precise micro architectural behavior from macroinstruction sequences is like pushing on a piece of string. This problem is particularly acute for the back end of the machine the memory and bus clusters which lie beyond the out-of-order section of the microarchitecture pipeline. CTEs allowed to provoke specific micro architectural

behavior on demand. Second, CTEs allowed to make significant strides in early verification of the processor Structural Register Transfer Level (SRTL) even before a full-chip model was available. Integrating and debugging all the logic and microcode needed to produce even a minimally functional full-chip model was a major undertaking. Because of the CTEs, testing could be started as soon as there was released code in a particular unit, long before trials at the full-chip level. Even after a full-chip model, the CTEs essentially decoupled validation of individual unit features from the health of the fullchip model.

Cluster test environment (CTE) methodology is composed of stimulus generation, checkers and coverage for Intel processor cores. There is a CTE for each of the clusters into which the processor design. These CTEs are groupings of logically related units (e.g. all the execution units of the machine constitute one CTE) surrounded by code that emulates the interfaces to adjacent units outside of the cluster and provides an environment for creating and running tests and checking results.

i- Testbench Environment
ii- Reference Model
iii- Checker
iv- Coverage
v- Monitor
vi- Driver/sequencer/injection randomization

Validation Phases of Specman:

i- Elaborate
ii- Model Build
iii- Connect
iv- Model Run
v- Post run

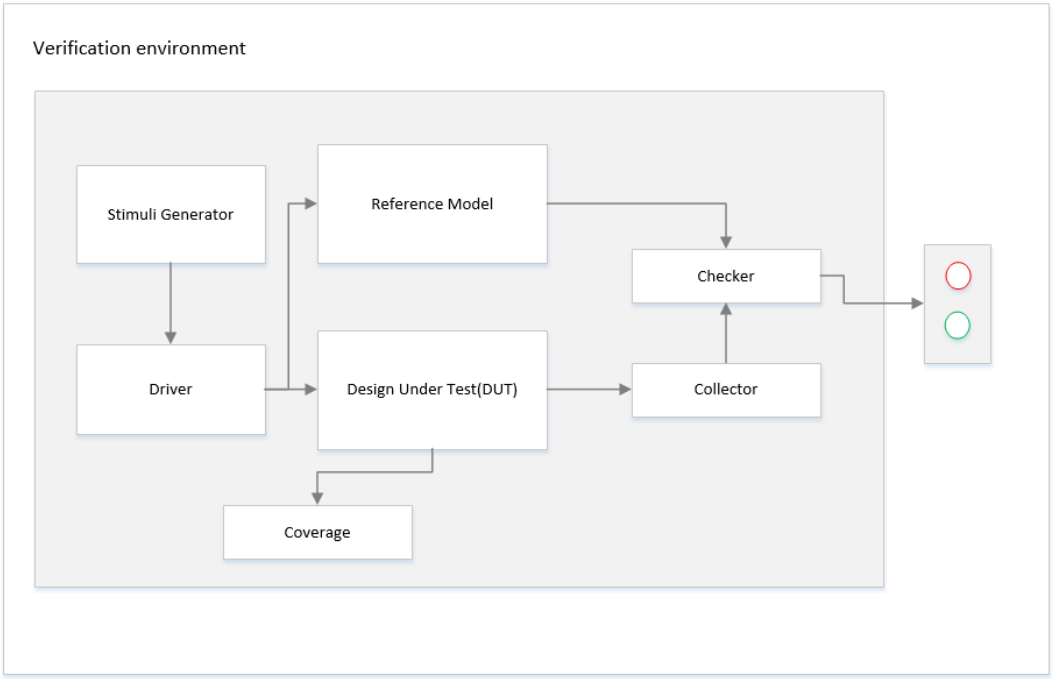


Figure 2.9: General Verification Environment

Chapter: 3

OVERVIEW OF DROOP PHENOMENON

In Electrical system, a sudden large increase in Current will cause a drop in the voltage. The size of the drop depends on a lot of parameters including: the size of the current increase, the duration of the increase, the capacitance and resistance of the system and the power supply.

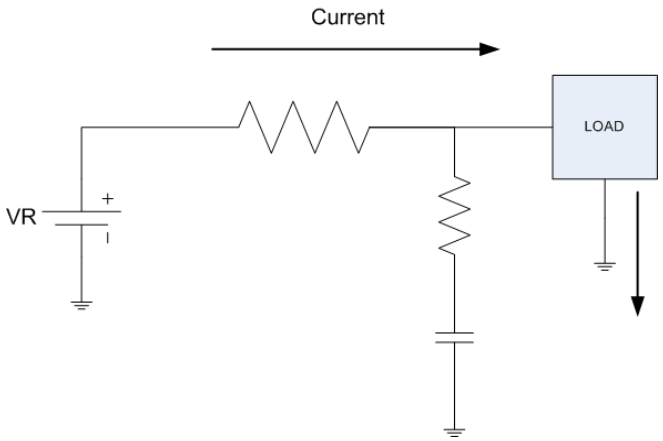


Figure 3.1: Voltage Droop in Circuit

3.1 Voltage Droop

Voltage droop is major phenomenon in modern IC circuits and CPU's operating at higher frequencies and lower technology nodes. When the logic circuit due to high load draw high switching current, undesirable voltage droop occurs.

As a consequence of this irregular and frequent current change in circuits operating at high very high frequencies due to irregular load, the voltage droop causes delay and can lead to faults in circuit operation.

To tackle this problem guardbands are used (extra timing (cycle) addition), but this is the most conservative way to tackle the problem, because the guardbands are usually calculated keeping in mind the worst case scenario, so it can lead to significant power and performance loss in circuits operating at very high frequencies and modern CPU's where both power and performance are the major factor.

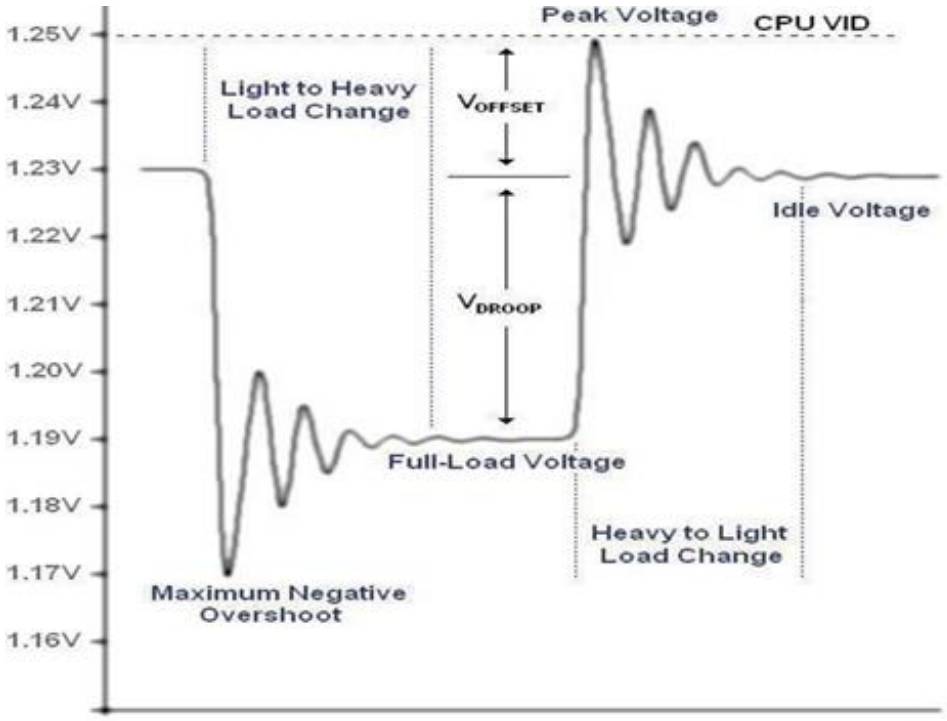


Figure 3.2: Voltage Droop in Processor Due to Load Change [31]

There is significant voltage drop in CPU when it goes from an idle state to a load state. This sudden and large voltage drop is called **voltage droop**.

3.2 Voltage Droop is Designed to Help

Voltage droop is an inbuilt in feature of Intel Processors and it is designed to make sure the voltage level never goes over beyond the specified level what has been set in the BIOS. This is necessary because there is always an overshoot whenever there is a voltage change due to irregular load change(variations), and without voltage droop, the overshoot could make the voltage higher than what has been set in the BIOS (for a split second), it may damage the CPU.

Chapter: 4
POWER CONSUMING UOP's
NON-POWER CONSUMING UOP's
AND VOLTAGE DROOP IN CPU

In general processors do computation using small data storage registers.64-bit registers are frequently used On 64-bit processors. But due to the need to large/huge data processing in most of the modern processors vector instructions have been added and these vector instructions operate on larger size of registers (128-bit, 256-bit, 512-bit). Intel's latest processors support AVX-512 instructions (advanced vector extension instructions). These AVX instructions operate on large registers of 128, 256 or 512-bit. These AVX instructions speed some application because they can process more data per instruction.

Some of these instructions use a lot of power and generate lot of heat. To keep power usage within bounds, Intel reduces the frequency of the cores dynamically. This frequency reduction (throttling) happens in any case when the processor uses too much power or becomes too hot. However, there are also deterministic frequency reductions based specifically on which instructions you use and on how many cores are active (downclocking). Indeed, when any 512-bit instruction is used, there is a moderate reduction in speed, and if a core uses the heaviest of these instructions in a sustained way, the core may run much slower. Furthermore, the slowdown is usually worse when more cores use these new instructions. In the worst case, you might be running at half the advertised frequency and thus your whole application could run slower.

There are heavy and light instructions. Heavy instructions are those involving floating point operations or integer multiplications (since these execute on the floating point unit). Light instructions include integer operations other than multiplication, logical operations, data shuffling (such as vpermw and vpermd) and so forth. Heavy instructions are common in deep learning, numerical analysis, high performance computing, and some cryptography (i.e., multiplication-based hashing). Light instructions tend to dominate in text processing, fast compression routines, vectorized implementations of library routines such as memcpyin C or System.arrayCopy in Java, and so forth.

Intel cores can run in one of three modes: license 0 (L0) is the fastest (and is associated with the turbo frequencies, license 1 (L1) is slower and license 2 (L2) is the slowest. To get into license 2, you need sustained use of heavy 512-bit instructions, means approximately one such instruction every cycle. Otherwise, any other 512-bit instructions will move the core to L1.

The downclocking is determined on a per-core basis based on the license and the total number of active cores, on the same CPU socket, irrespective of the license of the other cores. That is, to determine the frequency of core under downclocking, you need only to know its license (determined by the type of instructions it runs) and count the number of cores where code is running. Thus you cannot downclock other cores on the same socket, other than the sibling logical core when hyperthreading is used, merely by running heavy and sustained AVX-512 instructions on one core.

4.1 Grouping the Instructions Based on Their Minimum and Maximum Energy Consumption

The energy consumption is the lowest when there are no RAW dependencies between instructions and the highest when there are. The groups we have created for this purpose are:

• Simple Integer: Simple integer instructions are integer arithmetic and logic instructions besides multiplications and divisions and also all register movement, and compare and test instructions for integers.

• Simple float/double: These are all float and double additions and subtractions along with register movement and compare.

• Multiplication: Multiplications for integer, float and double operands.

• Division: Divisions for integer, float and double operands.

• Load: Loads for integer, float and double operands for different cache level access.

• Store: Stores for integer, float and double operands for different cache level access.

4.2 Instructions Consuming Large Amount of Power

Following are the major vector instruction which require large amount of power in execution:

SIMD – Single Instruction Multiple Data

This an extension of x86 architecture to increase the performance when exactly same type of operations needs to be performed on multiple set of data objects. Its size can be

of 128, 256, 512 bits. Example- digital signal processing (DSP) and graphics and multimedia processing.

SIMD is applicable to common tasks such as adjusting the contrast in a digital image or adjusting the volume of digital audio. Modern CPU's include SIMD instruction to improve multimedia uses performance.

Applications where SIMD can be used is – when the same value needs to be added to a large number of data set or points, or a common operation needs to performed in any application. Like changing the contrast of image. Each pixel of the image consists of three values for brightness red, green and blue. To change the brightness, first the R, G and B values are read from memory, the required values are added to the corresponding components, and the resulting value is written back to destination.

FMA- Fused Multiply Addition (Multiply accumulate operation)

This has 3 operands, fused uop is very useful where addition and multiplication is required simultaneously to get the final value

$$a = b.c + d$$

It is also be of 128, 256, 512 bits.[29]

Load Uop's –

High priority has been given to load uops because in reservation station uops waits for the operands to be available (this increases the IPC gain).

Store Uop's -

Store uops have higher energy cost relative to load uops because, cache system is prioritized for load uops. Storage to farthest cache doesn't happen immediately, because we may require the data in later cycles by other ups as source, so store takes more cycle than load uops.

4.3 Voltage Droop in CPU

One such case of sudden voltage drop can occur in modern CPU's due to irregular current variation happening due to the incoming instruction's (uop) of varying data type and size. This is a major concern after the addition of AVX and other power hungry instructions, which require large amount of current, causing the voltage to go below the specified level.

4.4. Energy per Instruction (EPI)

It is a measure of the amount of energy expended by a microprocessor for each instruction that the microprocessor executes. There are various factors that affect a microprocessor's EPI. Energy per instruction (EPI) is a measure of the power efficiency of a microprocessor. It records the average amount of energy expended per instruction processed by the microprocessor. EPI is measured in Joules/instruction. EPI is related to other commonly used power-efficiency metrics performance/watt and MIPS/watt. Specifically, EPI is the reciprocal of IPS/watt. This relationship is shown in the following equation:

(Joules/Instruction) = (Joules/Second)/(Instructions/Second) = Watt/IPS

Now we consider the EPI of a practical microprocessor. EPI is a function of several factors:

1) Design (microarchitecture, logic, circuits, and layout)
2) Process technology
3) Environment (supply voltage)

4.4.1 Instruction (uop) Energy Cost:

Energy cost of uops :

Average energy cost of uop's depends on its type and size:

Which is basically dependent on following factors:

i- Number of cycles required by the uop to complete the execution

ii- Resources required for the specific uop

iii- And other factors like number of dispatch ports available, cache miss/hit for that particular type of uop.

Type of uops:

A- Executable uops
   - i- X86 uops
   - ii- SIMD/AVX
   - iii- FMA

B- Load/store uops
   - i- Load
   - ii- Store

Size of uops:

i- 128

ii- 256

iii- 512

UOP	No. of Cycles required	Dispatch ports available
FMA	4/6	0/1/5(3)
SIMD	1/3	0/1(2)
Load	-	2/3(2)
Store	-	7/8(2)

Table 4.1: Number of Cycles Required by Uops and Available Dispatch Ports

So basically uop energy cost is :

Uop Energy Cost = aA+bB+cC

where a,b,c are weightage and A= number of cycles, B= resource factor, C= number of dispatch ports available.

4.4.2 Energy Cost Calculation

To calculate the total energy cost of a particular type of uop we need the base and scale values

4.4.2.1 Base Energy Cost

For each type of uops type we have some base value of energy cost which is calculated using instruction level modelling and simulation of the design.

4.4.2.2 Scaling Factor:

Each type of uop under consideration can be of either 128 bit, 256 bit or 512 bit, so corresponding scaling to the base energy cost of that particular type based on data size of uops need to done to obtain total energy cost.

4.4.2.3 Total energy cost of uop

Total energy cost of uop is a linear function of base energy cost and scale factor, it can be obtained by following formula.

Uop energy cost = base energy cost +multiplication factor(shift)* scale

UOP	*Base value*	*Scale Value*	*Left Shift(Mul. Fact.)*	*Energy Cost*
X86/INT	*	*	*	*
SIMD_128	1	1	1	1+1*1=2
SIMD_256			2	1+1*2=3
SIMD_512			3	1+1*4=5
FMA_128	2	3	1	2+1*3=5
FMA_256			2	2+2*3=8
FMA_512			3	2+4*3=14
LOAD_128	1	1	0	1+0*1=1
LOAD_256			1	1+1*1=2
LOAD_512			2	1+2*1=3
STORE_128	1	3	0	1+0*3=1
STORE_256			1	1+1*3=4
STORE_512			2	1+2*3=7

Table 4.2: Base, Scale and Multiplication Factor of Various Uops

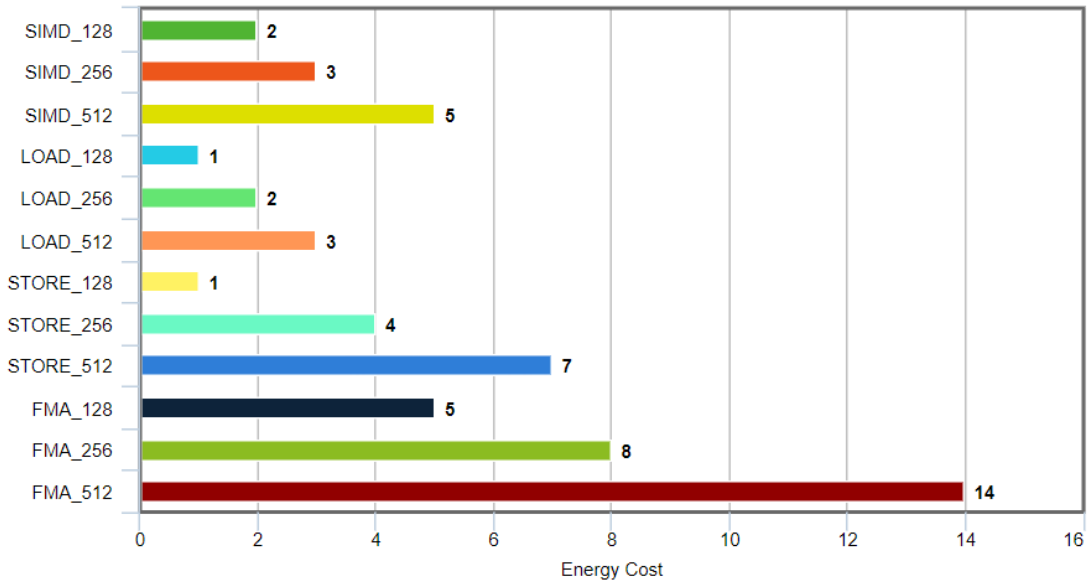


Figure 4.1: Energy Cost of Various Data Type and Data Size uops

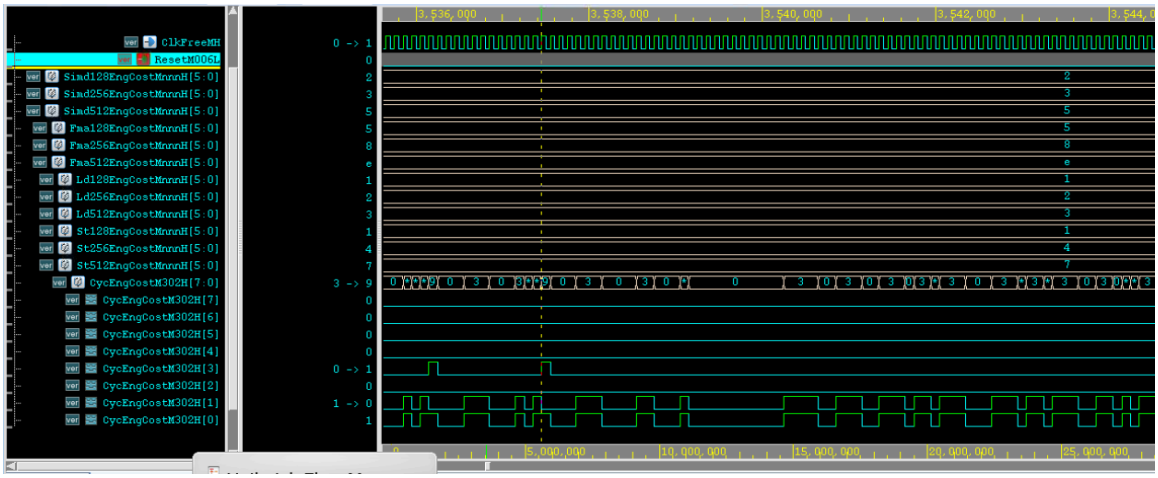


Figure 4.2: Individual uop's Energy Cost and Cycle Energy Cost

4.5 Cycle Energy Cost

The total energy consumed in a cycle is the sum of the energy cost of individual uops dispatched in that particular cycle. As a uop can be dispatched through pre-defined ports only, so prior to dispatch that ports should not be gated.

Cycle Energy Cost =

(Number of Dispatched uop to Execution Unit Ports in the Cycle)*(Energy Cost of the uop)

+ (Number of load/store uops)*(Energy Cost)

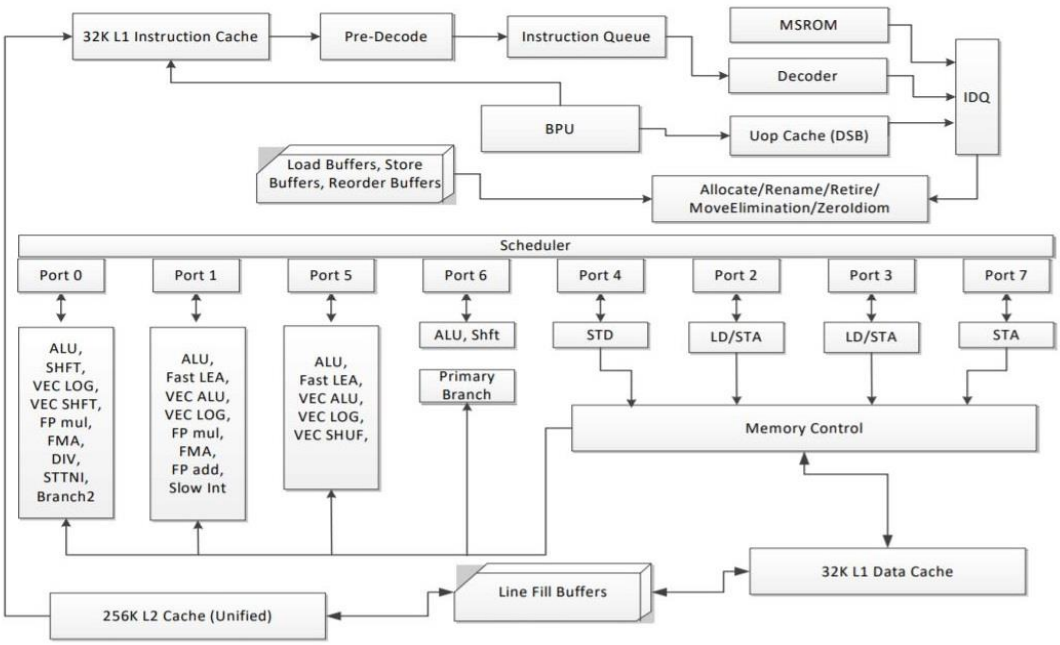


Figure 4.3: Dispatch Ports of a Processor

4.5.1 Calculating Cycle Energy Cost for Randomly Incoming Uops

In this section for randomly incoming stream of uops, from individual port energy cost cycle energy cost has been calculated.

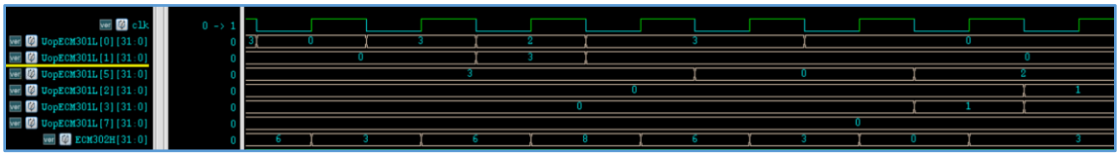


Figure 4.4: Cycle Energy Cost(Randomly incoming uops of various energy cost on different dispatch ports and total energy cost of particular cycle)

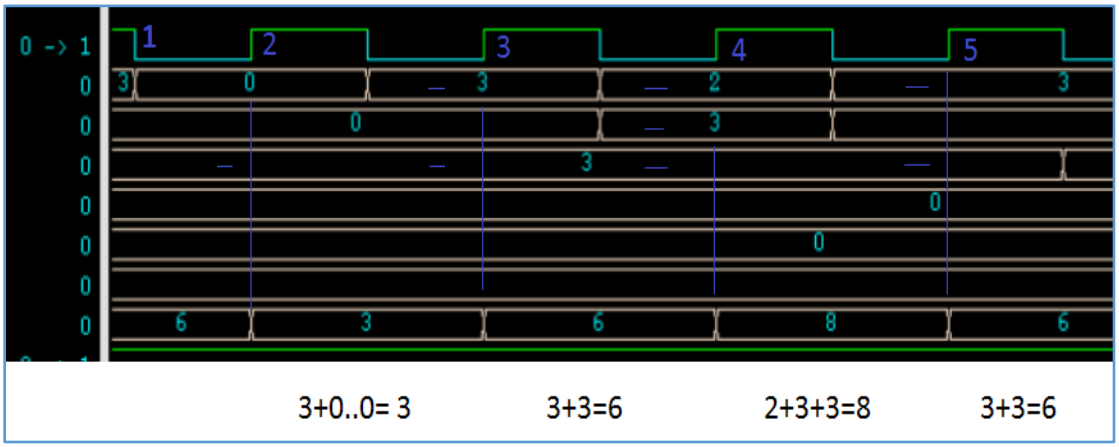


Figure 4.5: Cycle Energy Cost Calculation

Cycle/Port	1	2	3	4	5
Port0	0	3	2	3	-
Port1	0	0	3	0	-
Port5	3	3	3	3	-
Port2	0	0	0	0	-
Port3	0	0	0	0	-
Port7	0	0	0	0	-
Cycle Energy Cost	-	3	6	8	6

Table 4.3: Cycle Energy Cost Calculation

4.6 Variation in Cycle Energy Cost Cause of Droop in CPU

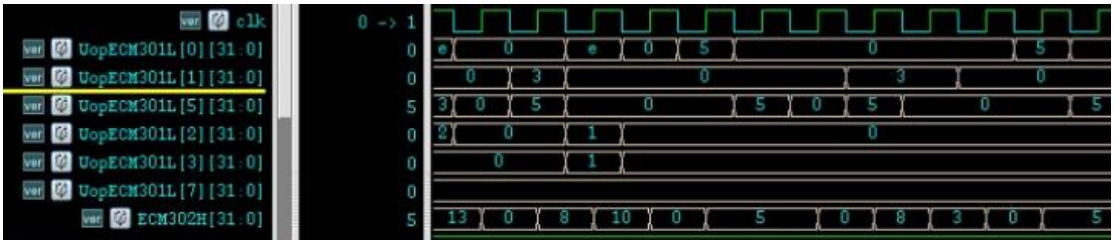


Figure 4.6: Energy Cost vs Time For randomly Incoming Uops

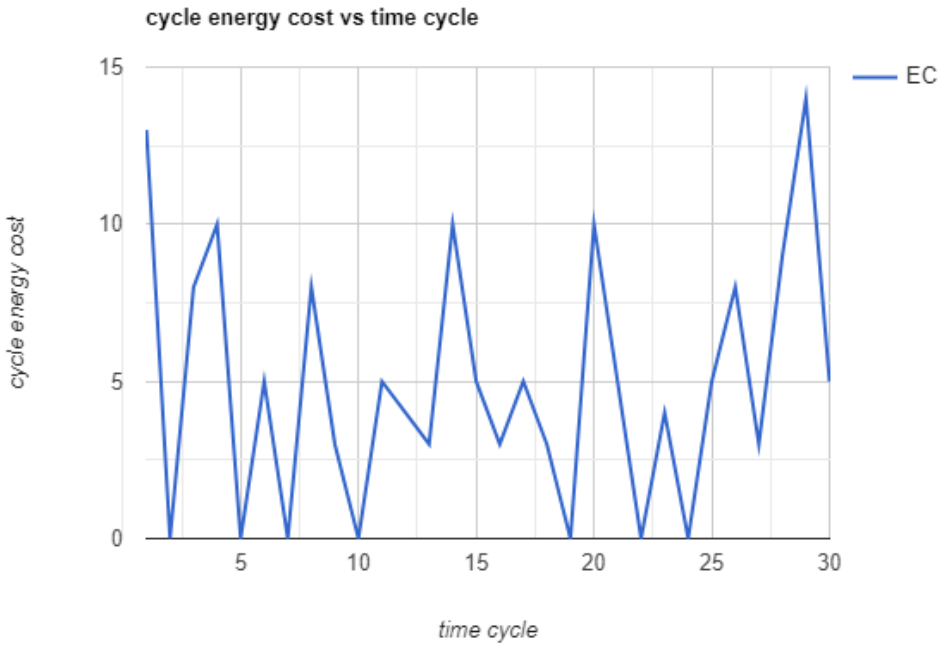


Figure 4.7: Variation in Cycle Energy Cost with Time

As we can see that the energy cost is varying rapidly over the cycles, this is the reason for voltage droop.

4.6.1 Using FIFO to Calculate Average Value of Energy Cost Over a Period of Cycles

Figure 4.8: FIFO

Droop Calculation:

It has three parts

1- DC value

2- AC part

   i- Positive part

   ii- Negative part

DC (average) part calculation:

Dc value is the normalized value of energy cost over a number of cycles (depends on the FIFO size).

AC part

i- Positive part- This is the energy cost of the present cycle which will be get added to the FIFO(pushed to FIFO).

ii- Negative part- This is the energy cost of the oldest entry of the FIFO which will be get subtracted from the FIFO(dropped out of the FIFO).

Total (instantaneous) Energy Cost = DC Part + AC Positive Part - AC Negative Part

Example

i- Overshoot Case-

Let us take the FIFO size = 30

And energy cost of last 30 cycles are (FIFO entries) : <30 entries of FIFO>

DC part = Average of total sum the elements of FIFO = $\sum FIFO_i/30$

Let us take the dc value -12**.**

AC positive part = present cycle energy cost

Let us take this value = 18

AC negative part = oldest cycle energy cost

Let us take this value = 10

So **Total (instantaneous) Energy Cost =** DC Part + AC sPositive Part - AC Negative Part

= 12.xx+18-10

= 20.xx(overshoot case)

ii- Undershoot Case-

DC part = $\sum$ FIFO(i)/30

Let us take the dc value -12**.**

AC positive part = present cycle energy cost

Let us take this value = 8

AC negative part = oldest cycle energy cost

Let us take this value = 16

So **Total (instantaneous) Energy Cost =** DC Part + AC Positive_Part - AC Negative_Part

= 12.xx+8-16

= 4.xx(undershoot case)

So we have overshoot and undershoot cases when there is sudden and continuous change in the data type and size of the uops, this causes rapid di/dt ratio(current) fluctuations(variation). This rapid variations in current can cause irregular voltage drop and may result in unnecessary heating of the chip.

This problem is more severe when the incoming uops data size is varying too often.

E.g. – if in the present cycle energy cost is 20, in the next cycle it becomes 40 and in the next to next cycle it becomes 10. Thus the current variation is happening rapidly, this will lead to voltage droop happening too frequently causing circuit faults.

Chapter: 5
SOLUTION TO DROOP PROBLEM

Solution to the Droop Problem

To prevent this voltage droop following methods have been there. First is analog detection and the other is fixed value of voltage threshold.

5.1 Placing Analog Detectors

To detect the instantaneous voltage level and prevent the voltage from going beyond the certain limits analog detectors can be placed which detect the sudden voltage drop(droop) happening due to instruction load variation and stop the execution of the upcoming instructions, thus avoiding the voltage droop problem in processors.

Problem with Analog Detectors

The main problem with analog detectors is when any voltage droop is detected in processors, it will block the execution of all instructions irrespective of the instruction data type and size, so it leads to full hang scenario. Once voltage droop is detected, the execution of heavy instructions stops until there is enough power available.

As this method is based on detecting the voltage droop after the execution of instructions.

And moreover the instruction flow need to be stopped, this can be done through blocking the dispatch ports (ports used by heavy instruction) which will ultimately block the flow of other non power consuming instructions also, leading to performance loss.

5.2 Threshold Level

To avoid this we may limit the di/dt to some predetermined value or threshold level. The value of threshold depends on the maximum allowed current variation. The optimum value of threshold depends on various factors. The value should be like this that should not be too less, because if threshold level is too low then the blockage probability is too high leading to too often dispatch port block and performance loss. On the other hand if the threshold level is high then most of the time all gates and stacks are open leading to power dissipation and heating. Optimum value is determined by PCU (power control unit)

5.2.1 Single (Fixed) Threshold

Based on various factors like maximum available energy in a cycle and lowest possible voltage for proper operation an optimum value of threshold is decided, if the energy cost sum exceeds this threshold then dispatch will be blocked until more power become available.

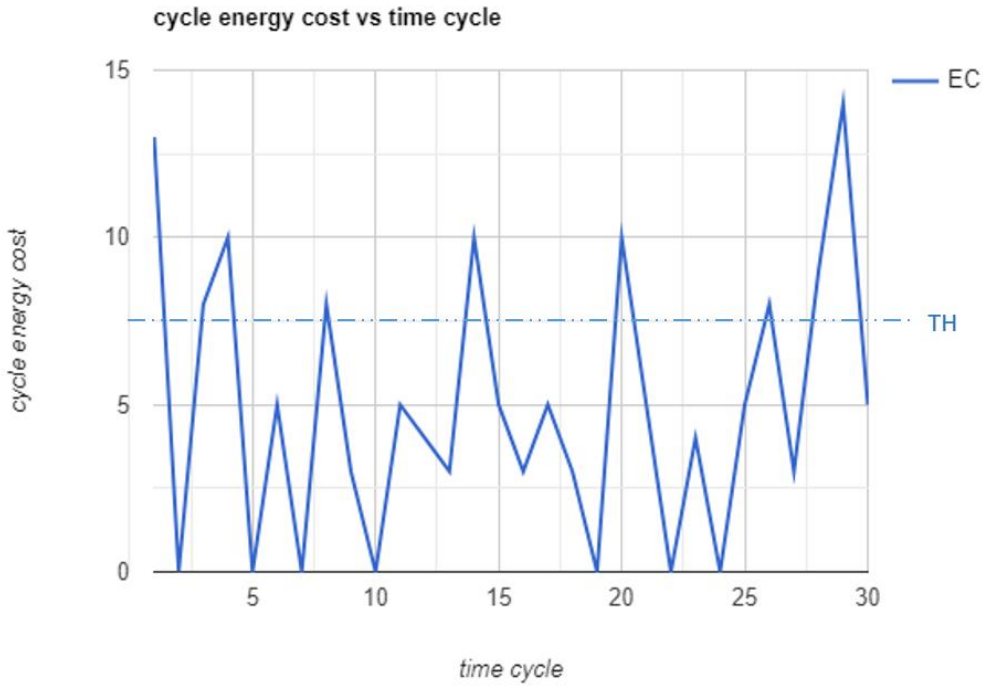


Figure 5.1: Single Level Threshold (Fixed Threshold)

5.2.2 Calculating Blockage Percentage for single and fixed value of threshold:

In this section we will be calculating the execution and memory dispatch port block percentage for randomly incoming uops and threshold level of 128.

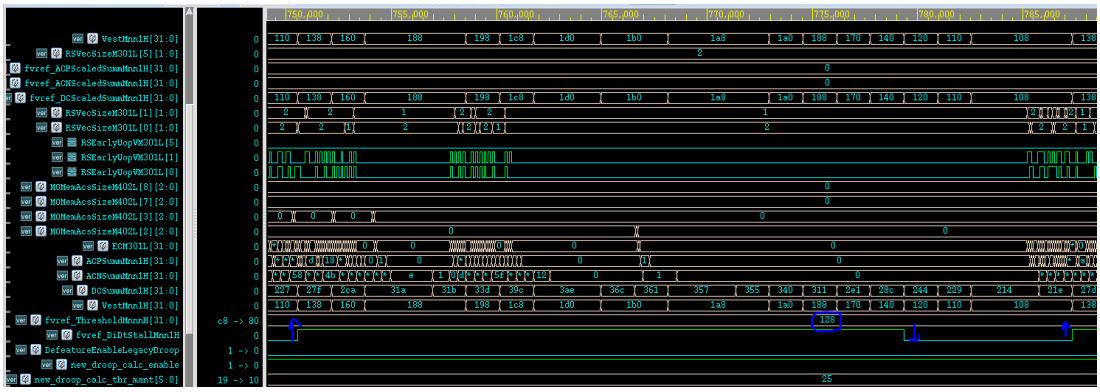


Figure 5.2 (a)

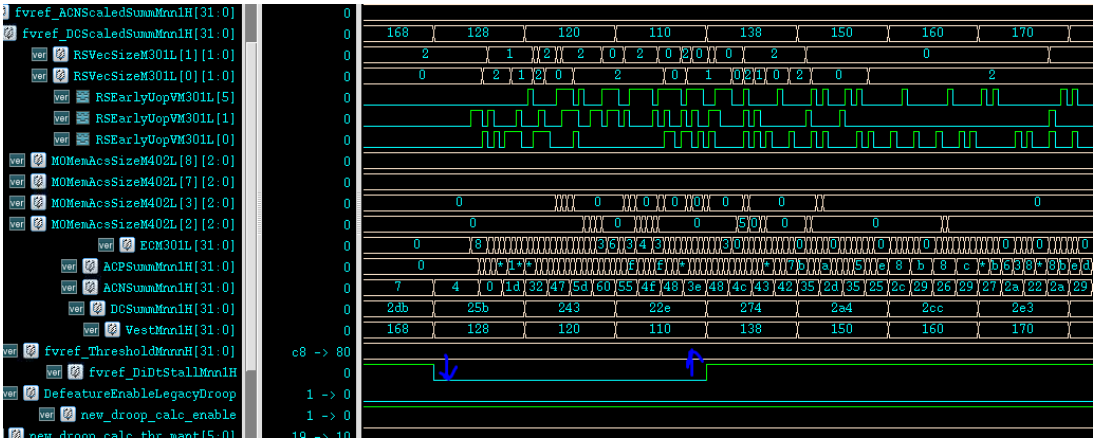


Figure 5.2 (b)

Figure 5.2 (c)

Figure 5.2 (d)

Figure 5.2 a, b, c, d showing di/dt stall for different values of threshold (when energy cost exceeds the threshold level)

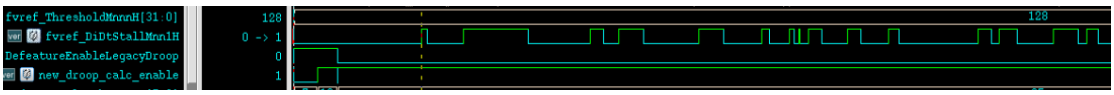


Figure 5.3: di/dt stall (block) for randomly incoming uops (instructions) and for single (fixed) value of threshold of 128

End cycle	Start Cycle	Cycle Difference	End cycle	Start Cycle	Cycle Difference	End cycle	Start Cycle	Cycle Difference
318550	0	318550	1513750	1486550	27200	2873750	2825750	48000
334550	318550	16000	1712150	1513750	198400	2979350	2873750	105600
424150	334550	89600	1745750	1712150	33600	2980950	2979350	1600
587350	424150	163200	1764950	1745750	19200	2982550	2980950	1600
740950	587350	153600	1809750	1764950	44800	3038550	2982550	56000
774550	740950	33600	1900950	1809750	91200	3177750	3038550	139200
873750	774550	99200	1961750	1900950	60800	3188950	3177750	11200
1012950	873750	139200	1984150	1961750	22400	3192150	3188950	3200
1169750	1012950	156800	2011350	1984150	27200	3216150	3192150	24000
1190550	1169750	20800	2044950	2011350	33600	3220950	3216150	4800
1238550	1190550	48000	2081750	2044950	36800	3233750	3220950	12800
1251350	1238550	12800	2110550	2081750	28800	3288150	3233750	54400
1268950	1251350	17600	2147350	2110550	36800	3326550	3288150	38400
1286550	1268950	17600	2382550	2147350	235200	3347350	3326550	20800
1320150	1286550	33600	2435350	2382550	52800	3406550	3347350	59200
1385750	1320150	65600	2435350	2382550	52800	3552150	3406550	145600
1419350	1385750	33600	2652950	2435350	217600	**unblock**	**1918550**	Total
1486550	1419350	67200	2825750	2652950	172800	**blocked**	**1633600**	3552150

Table 5.1: Blocked and Unblocked Cycles for Fixed Threshold

So for this particular test (with randomized incoming instructions and single threshold) Blockage percentage can be calculated by

$$\text{blockage percentage} = \frac{\text{number of cycles dispatch is blocked}}{\text{total number of cycles (test ran)}}$$

$$= \frac{1633600}{3552150} \text{ x } 100 = 45.98\%$$

→ So the dispatch is blocked for approximately half of the total running time cycle, affecting the processor performance drastically.

→ So if the fixed threshold level to lower some value then the blockage percentage will increase, more performance loss.

→ if the threshold value is set to some higher value then it requires higher power licenses all the time leading to power loss(and this become more severe when the incoming instructions (uops) are light or rare heavy instructions).

Chapter: 6
INSTRUCTION AWARE THRESHOLD VARITION

6.1 Disadvantages of Single and Fixed Threshold Level

So as per the result in the previous chapter, the instruction dispatch is blocked for approximately half of the total running time cycle, affecting the processor performance drastically. If the fixed threshold level is decreased to lower some value then the blockage percentage will increase, more performance loss. And if the threshold value is set to some higher value then it requires higher power licenses all the time leading to power loss, this become more severe when the incoming instructions (uops) are light or mixed with rare heavy instructions.

6.2 Proposed Method

This method is based on detecting the data type and size of instructions present in the reservation station and based on that and available power of a particular cycle calculating(dynamically predicting) the optimum threshold level thus preventing further execution of heavy instructions smartly, even before the execution of instruction happens itself and not affecting the non-power consuming instructions.

6.2.1 Model for Calculating Dynamic Threshold Level Based on Instruction Data Type and Data Size

Concept of Uop(Instruction) Weightage Ratio

To calculate the dynamically varying optimum threshold level based on uop data type and size we need a model which will detect the data type and size for every incoming uop and will calculate the weightage (percentage).Based on the ratio(weightage) of light and heavy load instructions sitting in the reservation station which are yet to be dispatched a threshold level is determined and request is send to the PCU to increase or decrease the power license. Only upon getting the grant the proper license the heavy uops can be dispatched, till then they have to wait in the reservation station. For this we need multiple counters to count of which data type of uop and how many uops are present of particular type in the reservation Station.

6.2.2 Droop Control Block

The droop control module is present in out of order unit. The following block diagram shows its connection with other modules in the core. It is connected with reservation station, reorder buffer and register alias table and also with dispatch ports.

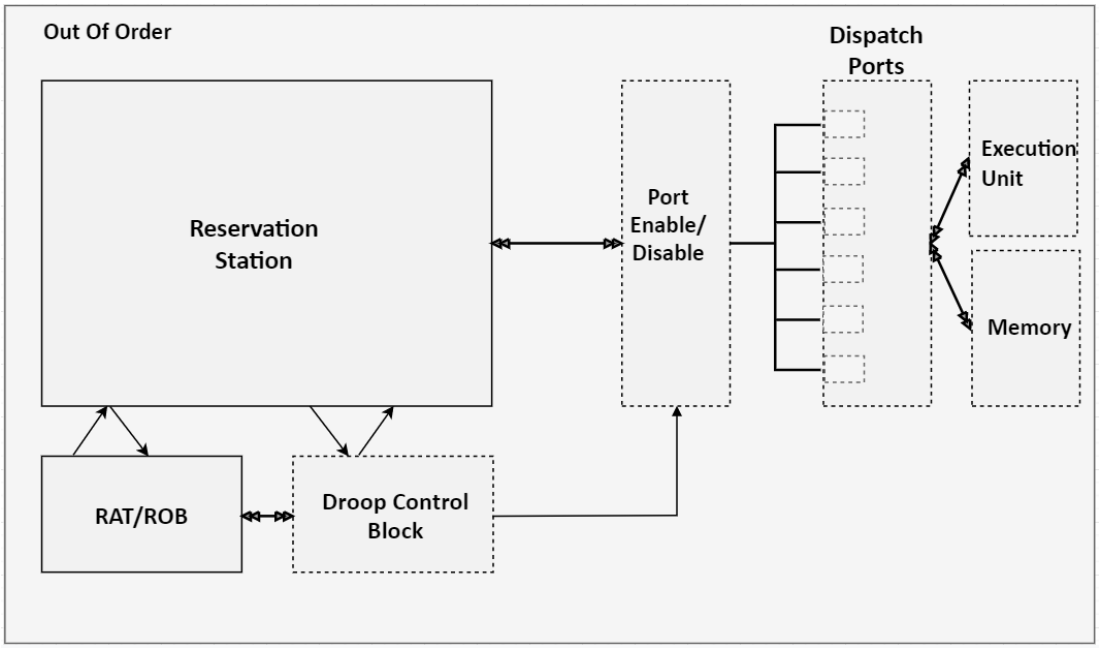


Figure 6.1: Block Diagram Model for Uop Weightage Calculation and Droop Block

6.2.3 DUT Parts

In the following various parts of the droop control logic block has been shown.

It consists of mainly following parts-

FIFO to calculate the sum of the energy cost for a number of cycles.

Droop control logic (threshold) which calculates the weightage of the uop and chooses one threshold based on that.

Current threshold level block and running energy cost sum are compared and based on the logic value ports are enabled or disabled.

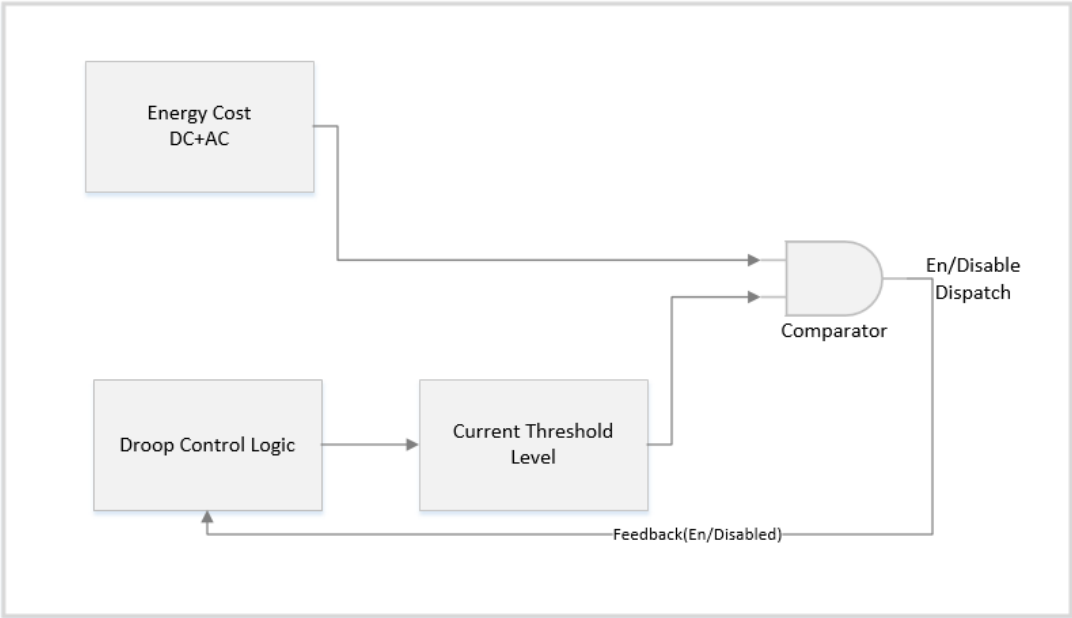


Figure 6.2: Droop Logic Block Diagram

6.3 Calculation Part: Blockage Part and Performance

In this design there are multiple levels of threshold and based on the uop weightage one of them will be selected.

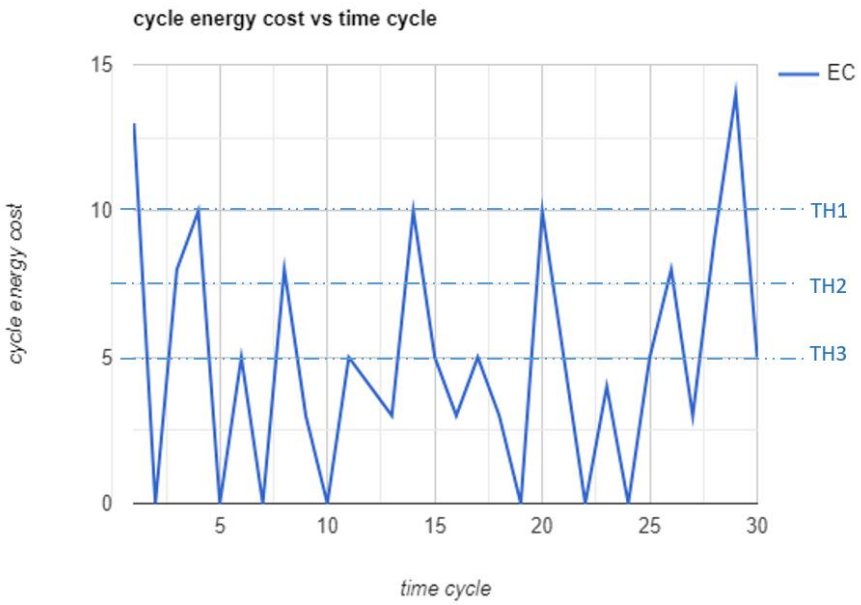


Figure 6.3: Choosing among Multiple Thresholds Based on Instruction Weightage

For the same testcase which generates same scenario we have tested the design which has new droop control logic module.

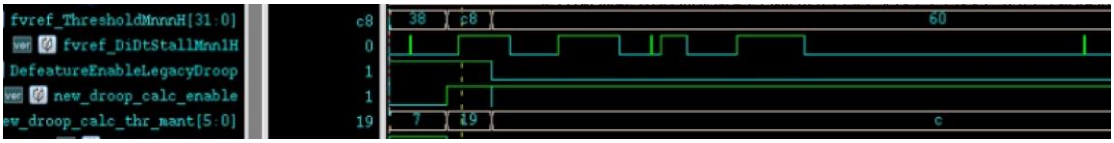


Figure 6.4: Dynamic Threshold variation 38→c8→60 as the energy cost is increasing the value of threshold is also increasing (only after getting grant from PCU). Case of threshold level stuck at 60, because of not availability of power at that time

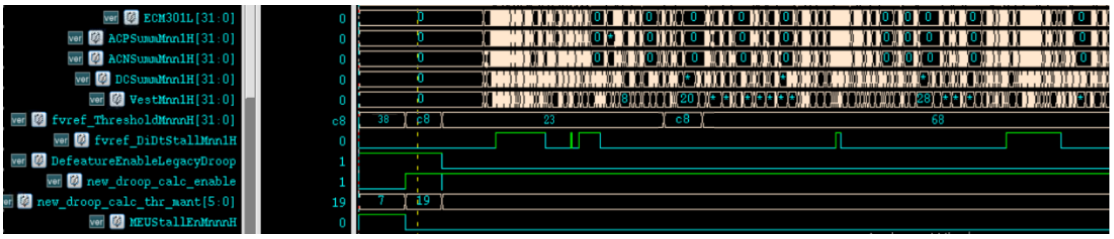


Figure 6.5: Dynamic Threshold Prediction with Less Blockage as Compared to Fixed Threshold

6.3.1 Blockage Percentage for Dynamic Threshold with Multilevel Threshold for Same Directed Test Case

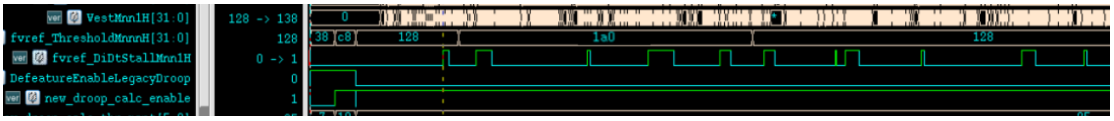


Figure 6.6: Blockage Calculation for Dynamic Threshold

End cycle	Start Cycle	Cycle Difference	End cycle	Start Cycle	Cycle Difference	End cycle	Start Cycle	Cycle Difference
318550	0	318550	1320150	1286550	33600	2836450	2665650	170800
334550	318550	16000	1385750	1320150	65600	2879350	2836450	42900
393850	334550	59300	1471650	1385750	85900	3048850	2879350	169500
436750	393850	42900	1481950	1471650	10300	3075650	3048850	26800
659450	436750	222700	1716750	1481950	234800	3108950	3075650	33300
667350	659450	7900	1738950	1716750	22200	3122150	3108950	13200
810250	667350	142900	2044950	1738950	306000			
873750	810250	63500	2081750	2044950	36800			
996350	873750	122600	2110550	2081750	28800			
1021550	996350	25200	2147350	2110550	36800			
1089150	1021550	67600	2312650	2147350	165300			
1118550	1089150	29400	2337550	2312650	24900			
1268950	1118550	150400	2652950	2337550	315400	**unblock**	**2627450**	Total
1286550	1268950	17600	2665650	2652950	12700	**blocked**	**494700**	3122150

Table 6.1: Dispatch Blocked Cycles for Dynamic Threshold

So for this particular test (with randomized incoming instructions and dynamic threshold)
Blockage percentage is given by

$$\text{blockage percentage} = \frac{\text{number of cycles dispatch is blocked}}{\text{total number of cycles (test ran)}}$$

$$= \frac{494700}{3122150} \text{ x } 100 = 15.84\%$$

So we can see that there is very less dispatch block for this dynamically predicted threshold as compared to the fixed threshold condition.

6.4 Comparison

Single (Fixed) Threshold Vs. Dynamic Threshold

The design has been tested for 20000 tests and no functional bug has been detected either by random testcase or directed test .And from among these 20k tests, for randomly chosen 40 tests the comparison between these two modes have been done and it has been proved that there is significant performance improvement when there is frequent load variation.

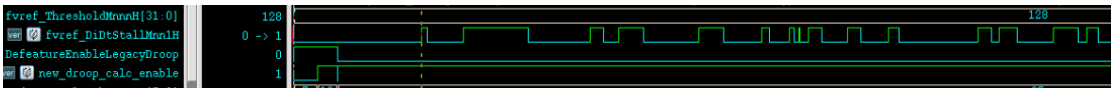


Figure 6.7(a) Blockage for Single threshold

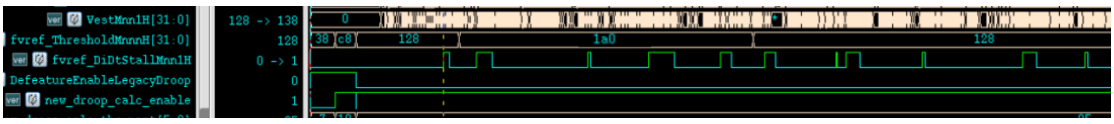


Figure 6.7(b) Blockage for Dynamic threshold

For directed test cases, with varying uop ratio from light to heavy, the graph has been plotted. For uop ratio near to zero it is seen that the blockage percentage for both are same as the energy consumption is less and there is no problem for both. As the heavy instructions starts mixing with light instructions the difference in performance is significant. As the uop weightage ratio starts increasing (more and more heavy instructions only with very less light instructions), again both starts showing same blockage probability as in this case there is no use of prediction.

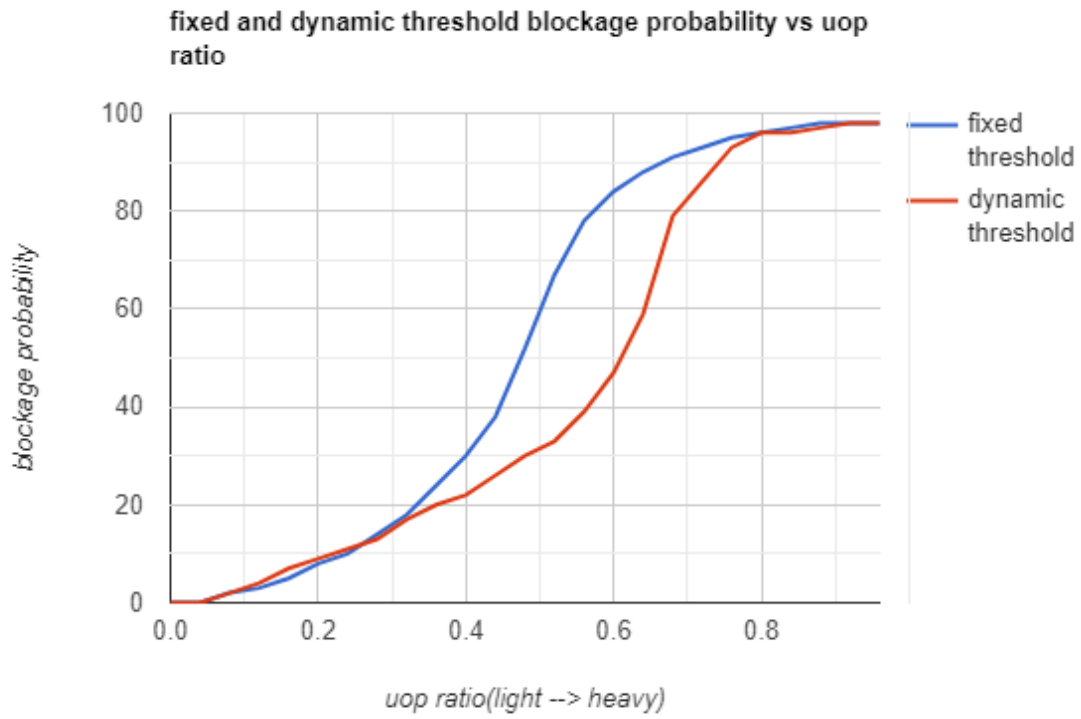


Figure 6.8: Fixed and Dynamic Threshold Blockage Percentage vs Uop Ratio

6.5 Validation Part

6.5.1 Out of order verification environment

To verify that the design is working properly as per the design specification, verification environment was created using CTE methodology (in specman or e language).

It consists of following parts. Each part was coded in specman language using CTE methodology

i- Testbench Environment

ii- Reference Model

iii- Checker

iv- Coverage

v- Monitor

vi- Driver/sequencer/injection randomization

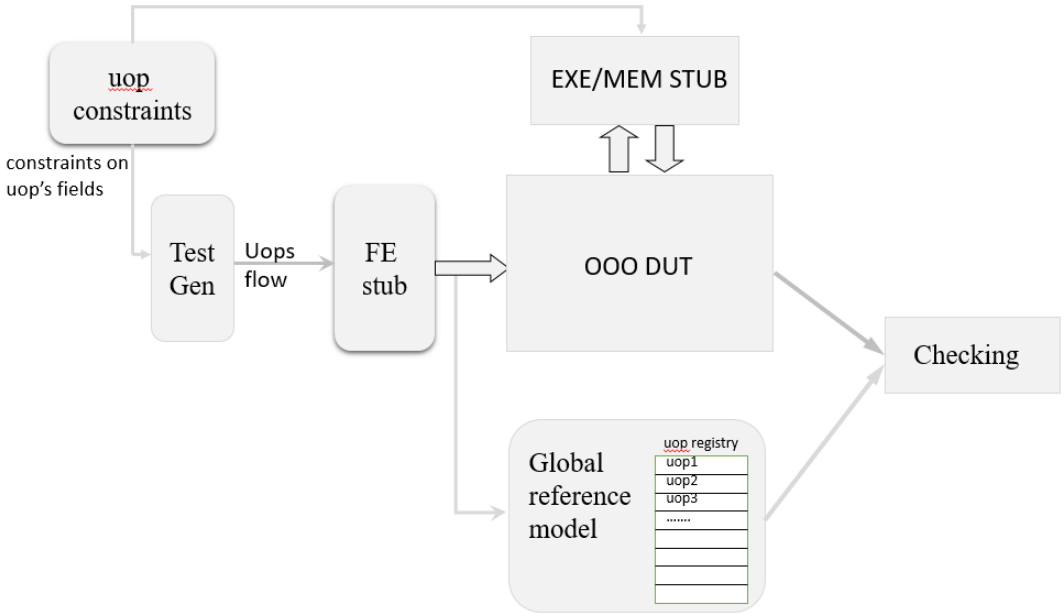


Figure 6.9: Out of Order Unit Verification Environment

A brief overview of these parts and coverage result

Random Injection (Sequencer and Driver)

In the sequencer part tescases to generate random uops of various data type and size and to emulate the threshold variation within the specified limits was coded. This help in rigorously testing the design and finding corner cases.

Checker

Assertion based checkers have been implemented which compare the value between CTE and RTL for various variables like uop energy cost for each port and cycle energy cost and for port dispatch block etc.

```
`ASSERTC_FORBIDDEN(EC_P0, (UopEngCostCTE[0] != UopECM[0]), droop_en, `ERR_MSG("Port 0 EC diff"));
`ASSERTC_FORBIDDEN(EC_P1, (UopEngCostCTE[1] != UopECM[1]), droop_en, `ERR_MSG("Port 1 EC diff"));
`ASSERTC_FORBIDDEN(EC_P5, (UopEngCostCTE[5] != UopECM[5]), droop_en, `ERR_MSG("Port 5 EC diff"));
`ASSERTC_FORBIDDEN(EC_P2, (UopEngCostCTE[2] != UopECM[2]), droop_en, `ERR_MSG("Port 2 EC diff"));
`ASSERTC_FORBIDDEN(EC_P3, (UopEngCostCTE[3] != UopECM[3]), droop_en, `ERR_MSG("Port 3 EC diff"));
`ASSERTC_FORBIDDEN(EC_P7, (UopEngCostCTE[7] != UopECM[7]), droop_en, `ERR_MSG("Port 7 EC diff"));
```

```
// Check EC calculation per port

`ASSERTC_FORBIDDEN(EC_P0, (UopEngCostCTE[0] != UopECM[0]), droop_en, `ERR_MSG("Port 0 EC diff"));
`ASSERTC_FORBIDDEN(EC_P1, (UopEngCostCTE[1] != UopECM[1]), droop_en, `ERR_MSG("Port 1 EC diff"));
`ASSERTC_FORBIDDEN(EC_P5, (UopEngCostCTE[5] != UopECM[5]), droop_en, `ERR_MSG("Port 5 EC diff"));
`ASSERTC_FORBIDDEN(EC_P2, (UopEngCostCTE[2] != UopECM[2]), droop_en, `ERR_MSG("Port 2 EC diff"));
`ASSERTC_FORBIDDEN(EC_P3, (UopEngCostCTE[3] != UopECM[3]), droop_en, `ERR_MSG("Port 3 EC diff"));
`ASSERTC_FORBIDDEN(EC_P7, (UopEngCostCTE[7] != UopECM[7]), droop_en, `ERR_MSG("Port 7 EC diff"));
```

Figure 6.10: Assertion for Port Energy Cost

```
//Port Block
  `ASSERTC_FORBIDDEN(Block_P0, (BlockL[0] != ref_BlockL[0]), 0, `ERR_MSG("Port 0 block RTL = %d && CTE = %d",BlockL[0],ref_BlockL[0]));
  `ASSERTC_FORBIDDEN(Block_P1, (BlockL[1] != ref_BlockL[1]), 0, `ERR_MSG("Port 1 block RTL = %d && CTE = %d",BlockL[1],ref_BlockL[1]));
  `ASSERTC_FORBIDDEN(Block_P5, (BlockL[5] != ref_BlockL[5]), 0, `ERR_MSG("Port 5 block RTL = %d && CTE = %d",BlockL[5],ref_BlockL[5]));
  `ASSERTC_FORBIDDEN(Block_MEM,(MemBlockH != ref_MemBlockH), 0, `ERR_MSG("MEM block RTL = %d && CTE = %d"   ,MemBlockH,ref_MemBlockH));
```

Figure 6.11: Assertion for Dispatch Block to Execution Unit and Memory Ports

Coverage

Dispatch block covergroups:

To verify that the design behavior is in accordance with the specifications when energy cost is exceeding above specified threshold level, uop dispatch to ports are being blocked, we ran several tests and found that expected coverpoints are hitting.

i- **Covergroup-** Execution_port_dispatch_block covergroup

Coverpoints- 1- Port_0_dispatch_block

2- Port_1_dispatch_block

3- Port_5_dispatch_block

4/5- Cross_coverage_0_1_5_dispatch_block

ii- **Covergroup-** Memory_port_dispatch_block covergroup

Coverpoints(items) 1- Port_2_dispatch_block

2- Port_3_dispatch_block

3- Port_7_dispatch_block

4/5- Cross_coverage_2_3_7_dispatch_block

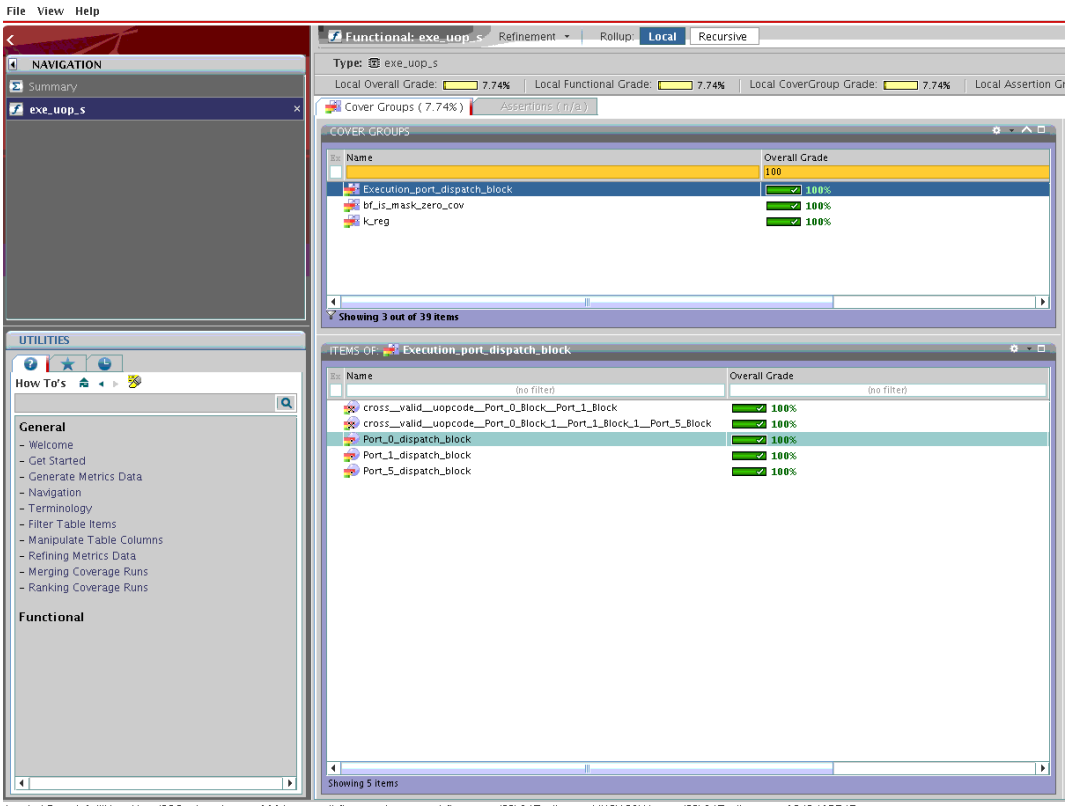


Figure 6.12: Execution Port Dispatch Block Covergroup

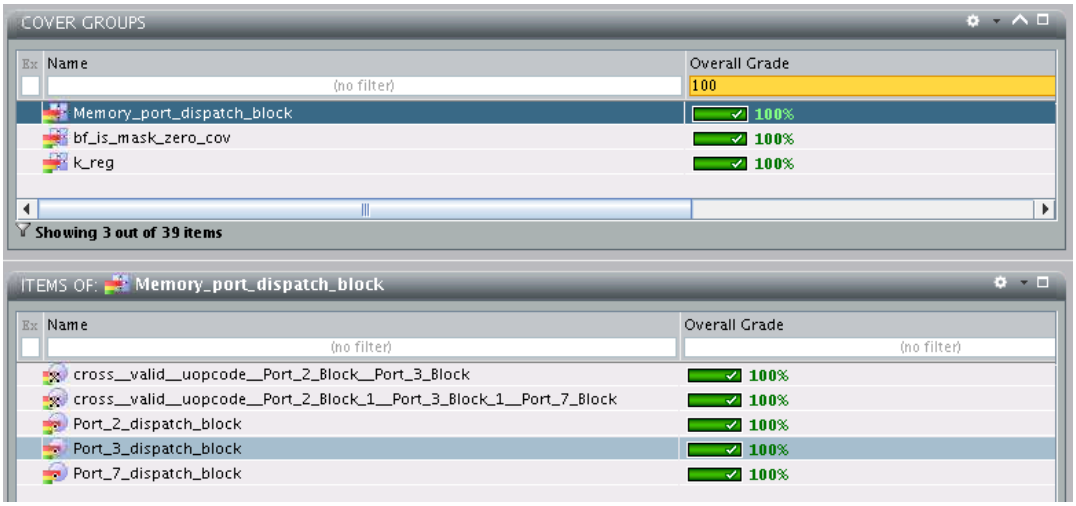


Figure 6.13: Memory Port Dispatch Block Covergroup

Apart from the above cte implementation of coverage, port coverage was also implemented in system verilog (in the design).

```
// Coverage

FPV_coverage_DroopCalcBlock_P0 : `COVER( 1'b1 ##10 $rose(Block_P0) , posedge clk , Reset );
FPV_coverage_DroopCalcBlock_P1 : `COVER( 1'b1 ##10 $rose(Block_P1) , posedge clk , Reset );
FPV_coverage_DroopCalcBlock_P5 : `COVER( 1'b1 ##10 $rose(Block_P5) , posedge clk , Reset );
FPV_coverage_DroopCalcBlock_Mem:`COVER( 1'b1 ##10 $rose(Block_MemH) , posedge clk , Reset );
```

Chapter: 7
CONCLUSION AND FUTURE WORK

7.1 Conclusion

The voltage droop occurring in the CPU has been explained and methods to minimize the loss due to the droop has been proposed and verified. So we have seen that changing the threshold level dynamically based on instruction (uop) data type and size helps in reducing the dispatch blockage and improved performance.

The functional verification of the feature has done using CTE methodology which is simulation based methodology. To get more confidence, formal verification can be perform for the same.

7.2 Problem and Future Aspects

In cases when there are light instructions only or rare heavy instructions there is no improvement at all. On the other hand when the instruction flow is flooded with heavy instructions and no light instructions at all, in this case also there is no effect. So future work can be done in this area for more improvement of performance.

References

[1] Tao Wang, Student Member, IEEE, Chun Zhang, Jinjun Xiong, Member, IEEE, Pei-Wen Luo, Liang-Chia Cheng, and Yiyu Shi "On the Optimal Threshold Voltage Computation of On-Chip Noise Sensors", IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 35, NO. 10, OCTOBER 2016

[2] Fangming Ye, Farshad Firouzi, Yang Yang, Krishnendu Chakrabarty, and Mehdi B. Tahoori, "On-Chip Voltage-Droop Prediction Using Support-Vector Machines", 2014 IEEE 32nd VLSI Test Symposium (VTS), 978-1-4799-2611-4/14/$31.00 ©2014 IEEE

[3] Shih-Yao Lin , Yen-Chun Fang , Yu-Ching Li , Yu-Cheng Liu , Tsung-Shan Yang , Shang-Chien Lin ,Chien-Mo Li, Eric Jia-Wei Fang," IR Drop Prediction of ECO-Revised Circuits Using Machine Learning ",2018 IEEE 36th VLSI Test Symposium (VTS),IEEE Computer Society, 978-1-5386-3774-6/18/$31.00 ©2018 IEEE

[4] Yu-Cheng Liu1 , Cheng-Yu Han1 , Shih-Yao Lin1 , James Chien-Mo Li1," PSN-aware circuit test timing prediction using machine learning", IET Computers & Digital Techniques, IET Comput. Digit. Tech., 2017, Vol. 11 Iss. 2, pp. 60-67

[5] Russ Joseph David Brooksy Margaret Martonosi, "Control Techniques to Eliminate Voltage Emergencies in High Performance Processors", IEEE Computer Society Proceedings of the The Ninth International Symposium on High-Performance Computer Architecture (HPCA-9'03) 1530-0897/02 $17.00 © 2002 IEEE

[6] Virendra Singh, Nihar Hage, Rohini Gulve, Masahiro Fujita, "On Testing of Superscalar Processors in Functional Mode for Delay Faults", IEEE Computer Society, 2380-6923/16 $31.00 © 2016 IEEE DOI 10.1109/VLSID.2017.58 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems

[7] Matthew Travers, "CPU Power Consumption Experiments and Results Analysis of Intel i7-4820K", Technical Report Series NCL-EEE-MICRO-TR-2015-197

[8] Sheayun Lee, Andreas Ermedahl, Sang Lyul Min. "An Accurate Instruction-Level Energy Consumption Model for Embedded RISC Processors", Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems - LCTES '01, 2001

[9] G. Shen ; N. Patkar ; H. Ando ; D. Chang ; C. Chen ; Chien Chen ; F. Chen ; P. Forssell, **"**A 64b 4-issue out-of-order execution RISC processor**",** Proceedings ISSCC '95 - International Solid-State Circuits Conference, 10.1109/ISSCC.1995.535508 Publisher: IEEE

[10] Indradeep Ghosh, Sekar, K. Boppana, v. Fujitsu Labs., America Inc., Sunnyvale, CA , (2002) "Design for verification at the register transfer level"- Design Automation Conference, Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design

[11] B. Solomon ; A. Mendelson ; R. Ronen ; D. Orenstien ; Y. Almog, "Micro-operation cache: a power aware frontend for variable instruction length ISA" , IEEE Transactions on Very Large Scale Integration (VLSI) Systems (Volume: 11 , Issue: 5 , Oct. 2003)

[12] Eduard Cerny. Synopsys, Inc. Marlborough, USA , Dmitry Korchemny Intel Corp , (2007) "Using SystemVerilog Assertions for Creating Property-Based Checkers"

[13] Allon Adir, Eli Almog, Laurent Fournier, Eitan Marcus, Michal Rimon, Michael Vinov, and Avi Ziv, IBM Research Lab, Haifa, (2004) "Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification"

[14] Hao Shen Yuzhuo Fu Sch. of Microelectron., Shanghai Jiao Tong Univ., China, (2005) "Priority directed test generation for functional verification using neural networks", Design Automation Conference, Proceedings of the ASPDAC 2005.

[15] Sapumal B. Wijeratne ; Nanda Siddaiah ; Sanu K. Mathew ; Mark A. Anders, "A 9-GHz 65-nm Intel® Pentium 4 Processor Integer Execution Unit" , IEEE Journal of Solid-State Circuits (Volume: 42 , Issue: 1 , Jan. 2007)

[16] Pei-Jun Ma, Yong Jiang, Kang Li, Jiang-Yi Shi "Functional Verification of Network Processor" 978-1-4577-0321-8/11/$26.00 ©2011 IEEE

[17] Bentley, B., Intel Corp., Hillsboro, OR, USA, (2002) "High level validation of next-generation microprocessors"- High-Level Design Validation and Test Workshop, 2002. Seventh IEEE International**,** 27-29 Oct. 2002

[18] Mostafa I. Soliman,"A VLIW architecture for executing multi-scalar/vector instructions on unified datapath" 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)

[19] Prabhat Mishra, Dutt, N.; Krishnamurthy, N.; Ababir, M.S., (2004) "A topdown methodology for microprocessor validation"- Design & Test of Computers, IEEE, Volume 21, Issue 2, Mar-Apr 2004

[20] Alon Gluska , (2003) "Coverage-Oriented Verification of Banias" – IEEEDesign Automation Conference, Proceedings, 2-6 June 2003

[21] Janick Bergeron, Writing testbenches: functional verification of HDL models, Kluwer Academic Publishers, Norwell, MA, 2000

[22] Kim, Doo-Hwan, and Jang-Eui Hong. "ESUMLEAF: a framework to develop an energyefficient design model for embedded software", Software & Systems Modeling, 2015.

[22] Bob Bentley, Intel Corporation "Validating the Intel@ Pentium@ 4 Microprocessor" 2001 International Conference on Dependable Systems and Networks

[23] Swadhesh Kumar ; P K Singh, "A study of recent advances in cache memories", 2014 International Conference on Contemporary Computing and Informatics (IC3I)

[24] Pawel Gepner, "USING AVX2 INSTRUCTION SET TO INCREASE PERFORMANCE OF HIGH PERFORMANCE COMPUTING CODE", Computing and Informatics, Vol. 36, 2017, 1001–1018, doi: 10.4149/cai 2017 5 1001

[25] Thomas Jakobs, Gudula Runger, "On the energy consumption of Load/Store AVX instructions", Proceedings of the Federated Conference on Computer Science and Information Systems pp. 319–327 DOI: 10.15439/2018F28 ISSN 2300-5963 ACSIS, Vol. 15

[26] Vijayalakshmi Saravanan, Senthil Kumar Chandran, Sasikumar Punnekkat, D. P. Kothari, "A Study on Factors Influencing Power Consumption in Multithreaded and Multicore CPUs", WSEAS TRANSACTIONS on COMPUTERS, ISSN: 1109-2750, Issue 3, Volume 10, March 2011

[27] Intel® 64 and IA-32 Architectures Software Developer's Manuals, https://software.intel.com/en-us/articles/intel-sdm#combined

[28] Emily Blem, Jaikrishnan Menon, Karthikeyan Sankaralingam, "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86architectures",https://ieeexplore.ieee.org/xpl/conhome/6518038/proceeding, IEEE Xplore: 03 June 2013INSPEC Accession Number: 13539004 DOI: 10.1109/HPCA.2013.6522302

[29] https://lemire.me/blog/2018/09/07/avx-512-when-and-how-to-use-these-new-instructions/

[30] "Power Estimation and Optimization Methodologies for VLIW-Based Embedded Systems", Springer Nature, 2004

[31] https://www.masterslair.com/vdroop-and-load-line-calibration-is-vdroop-really-bad

the

ORIGINALITY REPORT

14%	11%	7%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.dac.com Internet Source	3%
2	www.intel.com.az Internet Source	1%
3	www.ics.forth.gr Internet Source	1%
4	Submitted to Malaviya National Institute of Technology Student Paper	1%
5	download.intel.com Internet Source	1%
6	www.masterslair.com Internet Source	1%
7	epdf.tips Internet Source	1%
8	en.wikipedia.org Internet Source	1%
9	Mingsong Chen, Xiaoke Qin, Heon-Mo Koo,	

	Prabhat Mishra. "System-Level Validation", Springer Nature, 2013 Publication	1%
10	www.studymode.com Internet Source	<1%
11	Submitted to Engineers Australia Student Paper	<1%
12	Submitted to University of Liverpool Student Paper	<1%
13	Submitted to Colorado Technical University Online Student Paper	<1%
14	geometra.descriptiva.es.wikimiki.org Internet Source	<1%
15	Ming-Yi Sum, Shi-Yu Huang, Chia-Chien Weng, Kai-Shuang Chang. "Accurate RT-level power estimation using up-down encoding", The 2004 IEEE Asia-Pacific Conference on Circuits and Systems, 2004. Proceedings., 2004 Publication	<1%
16	dspace.uiu.ac.bd Internet Source	<1%
17	www.projectsparadise.com Internet Source	<1%
18	"Introduction", Functional Verification Coverage	

	Measurement and Analysis, 2004 Publication	<1%
19	ethesis.nitrkl.ac.in Internet Source	<1%
20	www.lib.kobe-u.ac.jp Internet Source	<1%
21	Submitted to Cranfield University Student Paper	<1%
22	www.primidi.com Internet Source	<1%
23	annaunivnotes.files.wordpress.com Internet Source	<1%
24	Sheayun Lee. "An Accurate Instruction-Level Energy Consumption Model for Embedded RISC Processors", ACM SIGPLAN Notices, 8/1/2001 Publication	<1%
25	es.scribd.com Internet Source	<1%
26	mmediego.blogspot.com Internet Source	<1%
27	archive.org Internet Source	<1%
28	Sheayun Lee, Andreas Ermedahl, Sang Lyul	