

**ACCELERATED VERILOG SIMULATOR USING APPLICATION
SPECIFIC MICROPROCESSOR**

TAN TZE SIN

UNIVERSITI SAINS MALAYSIA

2017

**ACCELERATED VERILOG SIMULATOR USING APPLICATION
SPECIFIC MICROPROCESSOR**

by

TAN TZE SIN

**Thesis submitted in fulfilment of
the requirements for the degree
of Doctor of Philosophy**

May 2017

ACKNOWLEDGEMENTS

I would like to express my gratitude to Associate Professor Dr Bakhtiar Affendi Bin Rosdi for willing to supervise my work in this research program. His experience has proven valuable in guiding the success of this endeavor. Through countless meetings and discussions over the years, he supported and encouraged the progress of this project.

Being a collaborative research program of Universiti Sains Malaysia (USM) and Altera (now part of Intel), I like to thank the management team in Altera for their understandings and supports in developing my skill set through this opportunity. My thanks go to the management team including Dato' Dr. Mohamad Sofi, Man On, Siew Ling, Vincent, and others. This research was funded in part by Malaysian Ministry of Higher Education (MOHE) via MyBrain15 program. I would like to thank MOHE for offering me the opportunity to participate in Industry PhD program.

My warmest feeling goes to my wife, Chain Yun, for her willingness to bear with me over the loss times. To my children, your smiles reminded me to move forward during the toughest time. Not to forget my beloved parents, brother, and sister for their encouragements. My appreciation goes to friends, colleagues, and many others who have been supportive over the duration. Thank you all!

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	ii
TABLE OF CONTENTS	iii
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF PLATES	xiii
LIST OF ABBREVIATIONS	xiv
LIST OF SYMBOLS	xviii
ABSTRAK	xix
ABSTRACT	xxi
CHAPTER ONE : INTRODUCTION	
1.1 Background	1
1.2 Problem Statement and Motivation	5
1.3 Objectives of Studies	8
1.4 Scope of Research	8
1.5 Thesis Contribution	9
1.6 Thesis Outlines	9
CHAPTER TWO : LITERATURE REVIEW	
2.1 Introduction	11
2.2 Verilog HDL	11
2.3 Logic Simulator	14
2.4 Software-Based Simulator	16
2.4.1 Interpreter	16
2.4.2 Compiled-code Simulator	17

2.4.3	Gate Level Simulator	18
2.4.4	Parallel Computing	19
2.5	Hardware-Assisted Logic Simulator	21
2.5.1	FPGA Synthesis	22
2.5.2	Emulator	23
2.6	Discussion – Simulator Types	25
2.7	Application Specific Microprocessor	28
2.8	FPGA as Accelerator Platform	31
2.9	Summary	34

CHAPTER THREE : THEORY AND METHODOLOGY

3.1	Introduction	35
3.2	Theory: Adaptation of Simulation Semantics	35
3.2.1	Program Flow	38
3.2.2	Event List	41
3.2.3	Instruction Segment	42
3.2.4	Data Map	43
3.2.5	Netlist Map	43
3.2.6	Simulation in Action	45
3.3	Development Methodology	46
3.3.1	Selecting a Hardware Platform	48
3.3.2	Developing the Prototype System	50
3.4	Summary	57

CHAPTER FOUR : VERILOG COMPILER DESIGN

4.1	Introduction	58
4.2	Compiler Requirements	58

4.3	Developing Verilog Compiler	59
4.3.1	Command Interpreter	63
4.3.2	Pre-processor	64
4.3.3	Verilog Parser	65
4.3.4	Creating Netlist Map	66
4.3.5	Operation Code (Opcode) Mapper	67
4.4	Coding of <i>VCompiler</i>	70
4.5	Validation of Verilog Compiler	71
4.6	Summary	75

CHAPTER FIVE : VerCPU HARDWARE SYSTEM DESIGN

5.1	Introduction	77
5.2	Design Methodology	77
5.3	Main Components in <i>VerCPU</i> System	82
5.3.1	Scheduler and Arbiter	83
5.3.2	Time Keeper	85
5.3.3	Event List Manager	85
5.3.4	Memory Manager	88
5.3.5	<i>VerCPU</i> Core	97
	5.3.5(a) Native Code Processor (NCP)	99
	5.3.5(b) Netlist Manager	105
5.4	Peripheral Components to Support <i>VerCPU</i> System	106
5.5	Firmware	107
5.6	<i>VerCPU</i> Verification	108
5.7	Summary	110

CHAPTER SIX : RESULTS AND DISCUSSIONS

6.1	Validation of <i>VerCPU</i>	114
6.1.1	Functional Validation	115
6.1.2	Performance Comparison	116
6.1.3	Timing Critical Paths	124
6.2	Discussion	125
6.2.1	Architecture	128
6.2.2	Benchmarking Result	129
6.2.3	Roadmap to Enhancements	132
6.2.3(a)	Parallel Processing	132
6.2.3(b)	Cache Memory	133
6.2.3(c)	Pipelining	135
6.2.3(d)	Instruction Optimization	138
6.2.3(e)	Event List Managing Algorithm	139
6.2.3(f)	Capacity and Reliability	139
6.3	Summary	140

CHAPTER SEVEN : CONCLUSION AND FUTURE WORKS

7.1	Future Works	143
-----	--------------	-----

REFERENCES	146
-------------------	-----

APPENDICES

Appendix A :	FUNCTIONAL VALIDATION TEST CASES
Appendix B :	TOP LEVEL RTL OF VerCPU SYSTEM
Appendix C :	RTL OF EVENT LIST MANAGER STATE MACHINE
Appendix D :	RTL OF VerCPU CORE

Appendix E : RTL OF NETLIST MANAGER

Appendix F : RTL OF LOGIC UNIT

LIST OF PUBLICATIONS

LIST OF TABLES

		Page
Table 2.1	Summary of logic simulator types	26
Table 2.2	Speed comparison of logic simulator types	27
Table 2.3	Feature comparison of logic simulator types	28
Table 2.4	Major Verilog simulator types and their weaknesses	34
Table 3.1	<i>VerCPU</i> System prototype with incrementally added features	57
Table 4.1	Native Code Processor (NCP) instruction categories	68
Table 5.1	Function of signals connected to <i>VerCPU</i> System	78
Table 5.2	Description of cache tag manager states	84
Table 5.3	Description of states in Event List Manager	88
Table 5.4	Description of cache tag manager states	95
Table 5.5	Native Code Processor (NCP) instruction categories	100
Table 5.6	Description of states in Netlist Manager	106
Table 6.1	FPGA resource utilization in different <i>VerCPU</i> variants	113
Table 6.2	Functionality validation test cases	115
Table 6.3	Selected test cases used to benchmark performance	121
Table 6.4	Clock cycles taken to complete simulation test cases on different <i>VerCPU</i> variants	122
Table 6.5	Normalized speed comparing <i>VerCPU</i> against Synopsys VCS ^{®1}	123
Table 6.6	Bandwidth of memory access occupied by Event List accesses in <i>VerCPU</i> System	123
Table 7.1	Feature comparison of <i>VerCPU</i> and other logic simulator types	143

LIST OF FIGURES

		Page
Figure 1.1	Typical digital VLSI design flow (Palnitkar, 2003)	2
Figure 1.2	Typical setup of test bench driven digital logic simulator	4
Figure 1.3	Simulation time as design getting complex across process nodes	5
Figure 2.1	Different Verilog abstraction levels	13
Figure 2.2	Logic simulator classification	15
Figure 2.3	Interpreter driven software-based logic simulator	17
Figure 2.4	Compiled-code software-based logic simulator	18
Figure 2.5	GLN software-based logic simulator	19
Figure 2.6	Parallel computing software-based logic simulator	21
Figure 2.7	FPGA synthesis simulator	23
Figure 2.8	Emulator	24
Figure 2.9	Speed and hardware architecture flexibility in solving a problem	30
Figure 2.10	Generalized compiled-code generation steps using C Language compiler as example	31
Figure 2.11	Logical view of FPGA programmable elements	32
Figure 2.12	Throughput improvement running Bing application using FPGA assisted servers (Putnam et al., 2014)	33
Figure 3.1	State variable update during simulation	36
Figure 3.2	Event driven program flow that triggers idling <i>VerCPU</i> in action to achieve fine-grained simulation parallelism	39
Figure 3.3	Data flow of across VerCPU data blocks	40
Figure 3.4	Event entry	41
Figure 3.5	Native Code Processor (NCP) OP code	42
Figure 3.6	Data node	43
Figure 3.7	Netlist Map entries showing connectivity of nodes	44

Figure 3.8	Netlist node	45
Figure 3.9	Simulation in action with using compiled data blocks	46
Figure 3.10	Development timeline of major components in this research	46
Figure 3.11	User point of view of the proposed simulator system and components developed in these research color shaded	47
Figure 3.12	Typical FPGA synthesis flow and storage of configuration bitstream in the end system	49
Figure 3.13	Relationship of hardware and software components involved in the development of a functional <i>VerCPU</i> System	51
Figure 3.14	Assembly of <i>VerCPU</i> System and the inputs	52
Figure 3.15	Block diagram of components on Altera Cyclone V development board. (Altera Corporation, 2013)	54
Figure 3.16	GUI of Quartus during compilation of a project	56
Figure 3.17	QSys showing <i>VerCPU</i> assembled into an embedded NIOS II System	56
Figure 4.1	Memory management classes for small persistent memory blocks	61
Figure 4.2	Adaptation of software development flow for <i>VCompiler</i>	62
Figure 4.3	Program flow of <i>VCompiler</i> broken down into steps	63
Figure 4.4	A compilation process using 4-bit binary counter as example	69
Figure 4.5	Hierarchy and object classes created for <i>VCompiler</i>	72
Figure 4.6	Flow of compiled-code simulation modeling <i>VerCPU</i> execution	74
Figure 4.7	Verification example with waveform captured from simulation	75
Figure 5.1	Top-level view of modularly designed <i>VerCPU</i> System for instantiation in generic embedded system	78
Figure 5.2	Scope of <i>VerCPU</i> Intellectual Property (IP) that was designed in this research	80
Figure 5.3	Design hierarchy of main sub-systems in <i>VerCPU</i>	82
Figure 5.4	Arbiter event dispatch to the first idling core	84
Figure 5.5	State diagram of Arbiter that dispatches event to <i>VerCPU</i> core	85

Figure 5.6	State diagram of the Event List Manager	86
Figure 5.7	Event list arrangement by updating the pointers	87
Figure 5.8	Data path of cache memory. The example on top is a two-way set associative cache having 32-bit address bus, 32 tag entries, and data entry made up of 32 bytes each	91
Figure 5.9	Block diagram of cache memory architecture	92
Figure 5.10	Cascading tag processor	94
Figure 5.11	Cache tag RAM processor states	95
Figure 5.12	Updates from multiple <i>VerCPU</i> cores can corrupt data integrity	98
Figure 5.13	Block diagram in Arbiter that prevents event re-entry to a statement that is under evaluation	99
Figure 5.14	Execution stages in NCP and data paths	101
Figure 5.15	Full adder implementation with 0, 1, X, and Z	102
Figure 5.16	Function call argument passing requires Mutex lock	103
Figure 5.17	Sequence of Mutex lock request in action from NCP 1 for a system with three NCPs.	104
Figure 5.18	State diagram of Netlist Manager	106
Figure 5.19	Main function loop in firmware	109
Figure 6.1	Compilation statistics of <i>VerCPU</i> System of VERSYS_CACHE variant	112
Figure 6.2	Block diagram of ALM utilized to generate two combinatorial outputs shown in blue	113
Figure 6.3	Screenshot of <i>VClientSW</i> in action. One as display console, while the other as simulator control	114
Figure 6.4	Main sequence of activities in Event Manager and NCP upon processing instruction ' <i>assign a = b</i> '. The events are numbered from 1 to 8	117
Figure 6.5	Main sequence of connected node updates and new event spawning. Order of operations is numbered from 1 to 6	118
Figure 6.6	L1 cache system implemented in <i>VerCPU</i> System prototype	120
Figure 6.7	Setup time slack histogram	125

Figure 6.8	Details of data and clock path that are used to arrive at slack calculation	126
Figure 6.9	Graphical view of timing critical path from timing waveform, physical distance, and circuit level representation	127
Figure 6.10	Simulation time taken by Event List accesses	129
Figure 6.11	Normalized simulation speed comparing single core <i>VerCPU</i> with parallel processing and cache	131
Figure 6.12	More efficient use of memory bus bandwidth by having multi-core (VERSYS_MCORE) and cache (VERSYS_CACHE) in <i>VerCPU</i> System	134
Figure 6.13	A 64-bit adder that can be implemented using two stage pipelined 32-bit adders, thereby doubling the clock frequency	137

LIST OF PLATES

	Page
Plate 3.1	
Photo of Altera Cyclone V development board. (Altera Corporation, 2013)	55

LIST OF ABBREVIATIONS

ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-Set Processor
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self Test
CAD	Computer Aided Design
CISC	Complex Instruction Set Computing
CPU	Central Processing Unit
DDR	Double Data Rate
DFT	Design For Test
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
ECC	Error Correction Code
EDA	Electronic Design Automation
FF	Flip-flop

FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
GLN	Gate Level Netlist
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HBM	High Bandwidth Memory
HDD	Hard Disk Drive
HDL	Hardware Description Language
IC	Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IO	Input Output
IP	Intellectual Property
LAN	Local Area Network
LE	Logic Element
LSB	Least Significant Bit
MAC	Media Access Controller
MOHE	Malaysian Ministry of Higher Education

MSB	Most Significant Bit
Mutex	Mutual Exclusive
NCP	Native Code Processor
Opcode	Operation Code
OS	Operating System
OSI	Open Systems Interconnection
PAL	Programmable Array Logic
PC	Personal Computer
PDES	Parallel Discrete Event Simulator
PHY	Physical Layer
PLA	Programmable Logic Array
PLL	Phase Locked Loop
RAM	Random Access Memory
RHS	Right Hand Side
RTL	Register Transfer Level
SDF	Standard Delay Format
SDRAM	Synchronous Dynamic Random Access Memory
SEU	Single Event Upset
SIMD	Single Instruction Multiple Data

SoC	System On-Chip
SRAM	Static Random Access Memory
SPICE	Simulation Program with Integrated Circuit Emphasis
SRAM	Static Random Access Memory
SSD	Solid State Drive
STA	Static Timing Analysis
Tcl	Tool Command Language
TCP	Transmission Control Protocol
UDP	User Defined Primitives
USB	Universal Serial Bus
USM	Universiti Sains Malaysia
UVM	Universal Verification Methodology
VCD	Value Change Dump
VHDL	Very High-Speed Description Language
VLSI	Very Large Scale Integration

LIST OF SYMBOLS

$Chit$	Cache hit rate
$\in \{ \}$	Element in the set
$f()$	Simulation transfer function
i_{last}	Last iteration in the simulation time step
$m_{i,t}$	State variable of simulation model where i is event index and t is time
N_{core}	Number of <i>VerCPU</i> core
$Ntran$	Number of transaction
$n_{i,t}$	Input stimuli to simulation model where i is event index and t is time
Σ	Summation
T_{1core}	Simulation time for single core prototype (VERSYS_1CORE)
T_{cache}	Simulation time for cached quad-core prototype (VERSYS_CACHE)
T_{ch}	Cache access latency
T_{multi}	Simulation time for quad-core prototype (VERSYS_MCORE)
T_{mem}	Memory access latency
T_{proc}	Transaction processing time
t_{last}	Last simulation time

PENYELAKU VERILOG TERPECUT MENGGUNAKAN MIKRO-PEMROSES APLIKASI KHUSUS

ABSTRAK

Simulasi logik merupakan satu langkah penting dalam pembangunan Litar Bersepadu Skala Sangat Besar (VLSI IC). Kemajuan Bahasa Takrifan Perkakasan (HDL) menjadikan Verilog salah satu pengantara dominan yang digunakan untuk pemodelan litar berdigit dan kes ujiannya. Pembekal Perisian Rekabentuk Elektronik (EDA) ada menawarkan perisian dan juga perkakasan untuk tujuan simulasi. Walau bagaimanapun, perisian simulasi adalah perlahan manakala simulasi bantuan perkakasan tidak menawarkan cirian simulasi sebagai mana yang ditetapkan dalam Verilog. Dalam projek penyelidikan ini, sebuah perkakasan simulasi yang berupaya menjalankan simulasi Verilog dicadangkan iaitu Sistem *VerCPU* untuk menangani kelemahan pelantar sedia ada. Pemproses *VerCPU* merupakan mikro-pemproses aplikasi khusus yang direka untuk menjalankan simulasi Verilog. Pemproses ini melakukan hitungan data Verilog dalam bentuk asalnya di samping menyokong aliran program acara selari untuk mencapai kelajuan yang unggul. Sebagai pelantar simulasi yang berasaskan kod, cirian keluaran dibekalkan dengan memberikan keputusan dan penelitian yang sama seperti perisian simulasi. Sebuah sistem prototaip yang lengkap berfungsi, *VerCPU* dibina di atas papan pembangunan Litar Aturcara Dalam Bidang (FPGA). Sistem ini berjaya disahkan dan dibandingkan dengan sebuah perisian simulasi sedia ada iaitu Synopsys VCS[®]. Sistem *VerCPU* berupaya mencapai kelajuan sehingga 6 kali ganda walaupun dengan hanya menggunakan teknik-teknik asas untuk mempercepatkan penilaian. Fungsi sistem ini sudah disahkan berkesan sebagai cara

simulasi Verilog alternatif yang berupaya memenuhi keperluan simulasi masa depan.

ACCELERATED VERILOG SIMULATOR USING APPLICATION SPECIFIC MICROPROCESSOR

ABSTRACT

Logic simulation is an important step in Very Large Scale Integration (VLSI) IC development. Advancement in Hardware Description Language (HDL) has made Verilog a widely adopted language used to model digital circuit and verification test bench. Electronic Design Automation (EDA) vendor provides software and hardware-assisted approaches to carry out simulations. However, software-based simulator is slow whereas hardware-assisted simulator does not offer the same simulation fidelity stipulated in Verilog. In this research project, a hardware-assisted Verilog simulator, *VerCPU* System, was proposed to address shortcomings in existing platforms. The simulator core is a custom designed application specific microprocessor specifically adapted to handle Verilog simulation. The microprocessor computes Verilog data in its native form while supporting event-driven parallelism directly to achieve speed supremacy. Being a compiled-code simulator, simulation fidelity compliancy is retained to offer the same result and visibility like the software-based solution. A functional system, *VerCPU*, was developed and prototyped on a Field Programmable Gate Array (FPGA) development board. This system was successfully verified and benchmarked against a software-based compiled-code simulator, i.e. Synopsys VCS[®]. *VerCPU* System can already achieve up to 6 times speed superiority with basic speed improvement techniques applied. The simulator had proven to be a viable alternate Verilog simulator to meet future simulation needs.

CHAPTER ONE

INTRODUCTION

1.1 Background

Very Large Scale Integration (VLSI) design complexity has grown exponentially since early 1970s. Circuit with billions of transistors in an Integrated Circuit (IC) has started to become the norm (Arden et al., 2010). Engineering cost has grown exponentially as fabrication process node shrinks. Hence, iterations of design change must be minimized. Manual design and analysis of circuit are no longer practical. Instead, a large proportion of design activities are assisted by Computer Aided Design (CAD) tools. Thorough verifications are carried out before a design is handed over for fabrication. Early circuit design was schematic entry. This has become impractical as circuit complexity increases. Text-based Hardware Description Language (HDL) was developed in mid-1980s (Cavanagh, 2007). HDL is now the dominant design entry for multi-million gate digital VLSI. The language is capable of modeling a circuit at different abstraction levels aligned with design maturity at different development stages.

A typical digital IC design flow is summarized in Figure 1.1. Some of the steps are iterative until convergence is achieved. Specification of feature set is the first step. High level behavioral model is suitable at this exploration stage where the focus being functional transformation algorithm of inputs into outputs. Viability and competency of the design are studied. Model is created based on the specifications to select the optimum design parameters that can offer the required feature set and performance. This is the earliest step where HDL can be applied to represent a circuit

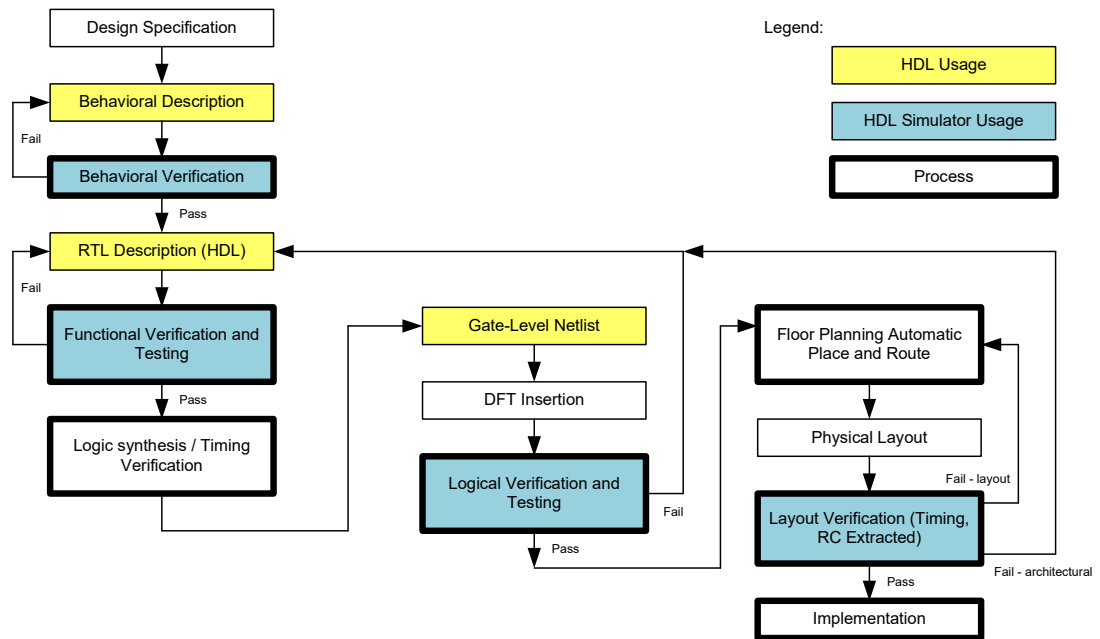


Figure 1.1: Typical digital VLSI design flow (Palnitkar, 2003)

model. The model is expressed in behavioral code and simulated on HDL compliant simulator. Once key parameters are finalized, the design is partitioned into manageable sub-systems for Register Transfer Level (RTL) coding. The codes are restricted to synthesizable Verilog content offered by the chosen technology. At this stage, data flow through circuit level implementation of the design is coded.

Next, RTL is synthesized into connections of primitive components offered by the process technology. The primitive components include but not limited to logic gate, flip-flop, memory, Phase Locked Loop (PLL), etc. Transformation is carried out by automated synthesis tools. Timing constraints are provided to the tool in order to optimize the circuit for required performance. This is accomplished through physical layout aware component placement and driver strength adjustment. Output from synthesis process is a circuit model with structural connected primitive components known as Gate Level Netlist (GLN).