GRAPH PROCESSING HARDWARE ACCELERATOR FOR SHORTEST PATH ALGORITHMS IN NANOMETER VERY LARGE-SCALE INTEGRATION INTERCONNECT ROUTING

CH'NG HENG SUN

UNIVERSITI TEKNOLOGI MALAYSIA

GRAPH PROCESSING HARDWARE ACCELERATOR FOR SHORTEST PATH ALGORITHMS IN NANOMETER VERY LARGE-SCALE INTEGRATION INTERCONNECT ROUTING

CH'NG HENG SUN

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Master of Engineering (Electrical)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

MAY 2007

Specially dedicated to
my beloved family

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my deepest gratitude to Professor Dr. Mohamed Khalil bin Haji Mohd Hani for giving me the opportunity to explore new grounds in the computer-aided design of electronic systems without getting lost in the process. His constant encouragement, support and guidance were key to bringing this project to a fruitful completion. I have learnt and gained much in my two years with him, not only in the field of research, but also in the lessons of life.

My sincerest appreciation goes out to all those who have contributed directly and indirectly to the completion of this research and thesis. Of particular mention are lecturer Encik Nasir Shaikh Husin for his sincere guidance and the VLSI-ECAD lab technicians, En. Zulkifli bin Che Embong and En. Khomarudden bin Mohd Khair Juhari, in creating a conducive learning and research environment in the lab.

Many thanks are due to past and present members of our research group at VLSI-ECAD lab. I am especially thankful to my colleagues Hau, Chew, Illiasaak and Shikin for providing a supportive and productive environment during the course of my stay at UTM. At the same time, the constant encouragement and camaraderie shared between all my friends in campus made life in UTM an enriching experience.

Finally, I would like to express my love and appreciation to my family who have shown unrelenting care and support throughout this challenging endevour.

ABSTRACT

Graphs are pervasive data structures in computer science, and algorithms working with them are fundamental to the field. Many challenging problems in Very Large-Scale Integration (VLSI) physical design automation are modeled using graphs. The routing problems in VLSI physical design are, in essence, shortest path problems in special graphs. It has been shown that the performance of a graph-based shortest path algorithm can severely be affected by the performance of its priority queue. This thesis proposes a graph processing hardware accelerator for shortest path algorithms applied in nanometer VLSI interconnect routing problems. A custom Graph Processing Unit (GPU), in which a hardware priority queue accelerator is embedded, designed and prototyped in a Field Programmable Gate Array (FPGA) based hardware platform. The proposed hardware priority queue accelerator is designed to be parameterizable and theoretically cascadable. It is also designed for high performance and it exhibits a run-time complexity for an INSERT (or EXTRACT) queue operation that is constant. In order to utilize the high performance hardware priority queue module, modifications have to be made on the graph-based shortest path algorithm. In hardware, the priority queue size is constrained by the available logic resources. Consequently, this thesis also proposes a hybrid software-hardware priority queue which redirects priority queue entries to software priority queue when the hardware priority queue module exceeds its queue size limit. For design validation and performance test purposes, a computationally expensive VLSI interconnect routing Computer Aided Design (CAD) module is developed. Results of the performance tests on the proposed hardware graph accelerator, graph computations are significantly improved in terms of algorithm complexity and execution speed.

ABSTRAK

Graf adalah struktur data yang meluas dalam sains komputer, dan algoritma yang bekerja dengan mereka adalah teras kepada bidang ini. Kebanyakan masalah yang mencabar dalam bidang automasi rekabentuk fizikal '*Very Large-Scale Integration*' (VLSI) dimodelkan sebagai graf. Banyak masalah penyambungan wayar dalam rekabentuk fizikal VLSI melibatkan masalah mencari-jalan paling pendek dalam graf yang istimewa. Ianya juga telah di tunjukkan bahawa prestasi algoritma mencari-jalan paling pendek berdasarkan graf dipengaruhi oleh prestasi baris gilir keutamaan. Tesis ini mengusulkan perkakasan pemproses graf untuk mempercepatkan perhitungan graf dalam masalah mencari-jalan paling pendek. Unit Pemprosesan Graf (GPU), di mana modul perkakasan pemecut keutamaan giliran dibenamkan dan prototaip dalam perkakasan '*Field Programmable Gate Array*' (FPGA) dapat dibentuk semula. Modul perkakasan pemecut keutamaan giliran tersebut direka supaya mudah diubahsuai, ia berprestasi tinggi dan mampu memberikan kompleksiti masa-lari yang malar bagi setiap tugas SISIPAN atau SARI. Untuk menggunakan perkakasan pemecut keutamaan giliran yang berprestasi tinggi tersebut, pengubahsuaian ke atas algoritma graf juga dilakukan. Dalam perkakasan, saiz baris gilir ketumaan dikekang oleh sumber-sumber logik yang ada. Tesis ini juga mengusulkan pemecut keutamaan giliran hibrid berasaskan perkakasan dan perisian, di mana sisipan ke perkakasan pemecut keutamaan giliran akan ditujukan ke perisian apabila perkakasan pemecut keutamaan giliran tidak mampu untuk menampungnya. Untuk pengesahan rekacipta dan pengujian prestasi, satu modul pengkomputeran VLSI penyambungan wayar '*Computer Aided Design*' (CAD) dibangunkan. Hasil kerja tesis ini menunjukkan bahawa perkakasan pemecut yang diusulkan dapat mempercepatkan penghitungan graf, baik dari segi kerumitan algoritma dan masa perlakuan.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	DECLARATION	ii
	DEDICATION	iii
	ACKNOWLEDGEMENTS	iv
	ABSTRACT	v
	ABSTRAK	vi
	TABLE OF CONTENTS	vii
	LIST OF TABLES	xi
	LIST OF FIGURES	xii
	LIST OF SYMBOLS	xvii
	LIST OF APPENDICES	xviii
1	**INTRODUCTION**	1
	1.1 Background	1
	1.2 Problem Statement	3
	1.3 Objectives	4
	1.4 Scope of Work	5
	1.5 Previous Related Work	6
	1.5.1 Hardware Maze Router and Graph Accelerator	6
	1.5.2 Priority Queue Implementation	8
	1.6 Significance of Research	10
	1.7 Thesis Organization	11
	1.8 Summary	13

2	**THEORY AND RESEARCH BACKGROUND**		14
	2.1	Graph	14
	2.2	Graph-based Shortest Path Algorithm	17
	2.3	Priority Queue	18
	2.4	Priority Queue and Dijkstra's Shortest Path Algorithm	23
	2.5	Modeling of VLSI Interconnect Routing as a Shortest Path Problem	30
	2.6	Summary	33
3	**PRIORITY QUEUE AND GRAPH-BASED SHORTEST PATH PROBLEM – DESCRIPTIONS OF ALGORITHMS**		34
	3.1	Priority Queue and the Insertion Sort Algorithm	34
		3.1.1 Insertion-Sort Priority Queue	35
	3.2	Maze Routing with Buffered Elmore Delay Path Optimization	39
	3.3	Simultaneous Maze Routing and Buffer Insertion (S-RABI) Algorithm	45
		3.3.1 Initial Graph Pruning in S-RABI	45
		3.3.2 Dijkstra's Algorithm applied in S-RABI	47
		3.3.3 S-RABI in maze routing with buffered interconnect delay optimization	49
	3.4	Summary	56
4	**ALGORITHM MODIFICATIONS FOR HARDWARE MAPPING**		57
	4.1	Modification in graph algorithm to remove DECREASE-KEY operation	57
	4.2	Modifications in Dijkstra's and S-RABI algorithm	62
	4.3	Modification of Insertion Sort Priority Queue	68
	4.4	Summary	73

5	**THE GRAPH PROCESSING UNIT**		74
	5.1	Introduction	74
	5.2	System Architecture of Graph Processing Unit (GPU)	76
	5.3	Priority Queue Accelerator Module	78
		5.3.1 Specification and Conceptual Design of hwPQ	79
		5.3.2 Specification and Conceptual Design of Avalon Interface Unit	81
	5.4	hwPQ Device Driver	84
	5.5	Hybrid Hardware-Software Priority Queue (HybridPQ)	87
6	**DESIGN OF PRIORITY QUEUE ACCELERATOR MODULE**		93
	6.1	Hardware Priority Queue Unit (hwPQ)	93
		6.1.1 The design of Processing Element – RTL Design	98
	6.2	Pipelining in hwPQ	102
		6.2.1 Data Hazards in the Pipeline	104
	6.3	Timing Specifications of hwPQ	108
	6.4	Avalon Interface Unit – Design Requirement	113
	6.5	Avalon Interface Unit – RTL Design	114
		6.5.1 Avalon Data Unit	115
		6.5.2 Avalon Control Unit	117
7	**SIMULATION, HARDWARE TEST AND PERFORMANCE EVALUATION**		119
	7.1	Design Verification through Timing Simulation	119
		7.1.1 Simulation of Priority Queue Accelerator Module	119
	7.2	Hardware Test	123
	7.3	Comparison with priority queue software	125

implementation

7.4 Comparison with other priority queue hardware design 128

7.5 Performance Evaluation Platform 130

7.6 Performance of Priority Queue in Graph Computation 132

7.6.1 Worst Case Analysis 134

7.6.2 Practical Case Analysis 139

7.7 Summary 142

8 CONCLUSIONS 145

8.1 Concluding Remarks 145

8.2 Recommendations for Future Work 147

REFERENCES 150

Appendices A - I 158 - 226

LIST OF TABLES

TABLE NO	TITLE	PAGE
2.1	Run-time complexity for each operation among different heap data structures.	30
5.1	Avalon System Bus signal descriptions	82
5.2	Memory-mapped Register descriptions	83
6.1	IO Port Specifications of hwPQ	110
7.1	Set of Test Vectors	120
7.2	Resource Utilization and Performance of hwPQ	125
7.3	Comparison in Run-Time Complexity	126
7.4	Comparison in Number of Processor Cycles	126
7.5	Speed Up Gain by Priority Queue Accelerator Module	126
7.6	Comparison with other hardware implementations	129
7.7	Number of elapsed clock cycles per operation	144
8.1	Features of Hardware Priority Queue Unit (hwPQ)	146

LIST OF FIGURES

FIGURE NO	TITLE	PAGE
1.1	System Architecture	11
2.1	Two representations of an undirected graph	15
2.2	Two representation of a directed graph	15
2.3	A weighted graph	16
2.4	Shortest Path and Shortest Unit Path	17
2.5	Basic Operations of Priority Queue	19
2.6	Simplest way to implement Priority Queue	20
2.7	Priority Queue implemented as array or as heap	21
2.8	Set, Graph, Tree and Heap	22
2.9	Example of Binomial-Heap and Fibonacci-Heap	22
2.10	Function RELAX ( )	23
2.11	Relaxation	23
2.12	Dijkstra’s Shortest Path Algorithm	24
2.13	Illustration of Dijkstra’s algorithm	25
2.14	Illustration of the final execution result	29
2.15	VLSI layout represented in grid-graph	31
2.16	VLSI Routing as shortest unit path problem	31
2.17	Parallel expansion in Lee’s algorithm	32
2.18	VLSI Routing as shortest path (minimum-delay) problem	33
3.1	Insertion-Sort Algorithm	36
3.2	Insertion-Sort Priority Queue Algorithm	37
3.3	Operations in Insertion-Sort Priority Queue	38
3.4	A typical routing grid-graph	39

3.5	Typical maze routing algorithm with buffered delay path optimixation	40
3.6	Elmode Delay Model	41
3.7	Elmore Delay in hop-by-hop maze routing	42
3.8	Elmore Delay for buffer insertion in hop-by-hop maze routing	43
3.9	Graph pruning	46
3.10	Hop-by-hop Dijkstra's Algorithm	48
3.11	Function Cost ( )	50
3.12	Function InsertCandidate ( )	51
3.13	Simltaneous Maze Routing and Buffer Insertion (S-RABI)	53
4.1	DECREASE-KEY and Relaxation	58
4.2	Function DECREASE-KEY ( )	59
4.3	INSERT in Relaxation	60
4.4	EXTRACT in Relaxation	61
4.5	Modifcation rules to remove DECREASE-KEY	61
4.6	Modified Dijkstra's Algorithm – without DECREASE-KEY	62
4.7	Modified InsertCandidate ( )	63
4.8	Modified S-RABI Algorithm	65
4.9	Further optimization to reduce overhead	66
4.10	One-dimensional Systolic Array Architecture	68
4.11	Execution of identical task-cycles for one operation	69
4.12	Series of operations executed in pipeline	70
4.13	Modified Insertion-Sort Priority Queue	71
4.14	Example of INSERT_MOD operation	72
4.15	INSERT_MOD in identical sub-tasks of Compare-and-Right-Shift	76
5.1	NIOS II System Architecture	75
5.2	Different layers of software components in NIOS II System	76

5.3	Top-Level Architecture of Graph Processing Unit	76
5.4	GPU – Software/Hardware System Partitioning	78
5.5	Functional Block Diagram of Priority Queue Accelerator Module	79
5.6	Top-Level Description of hwPQ	80
5.7	Memory-mapped IO of Avalon Slave Peripheral	81
5.8	Functional Block Diagram of Avalon Interface Unit	82
5.9	Programming Model of Priority Queue Accelerator Module	84
5.10	Device driver routine for INSERT operation	85
5.11	Device driver routine for EXTRACT operation	85
5.12	Device driver routine for PEEK operation	86
5.13	Device driver routine for DELETE operation	87
5.14	Software Abstraction Layer of HybridPQ	88
5.15	Functional Block Diagram of HybridPQ	89
5.16	INSERT control mechanism in HybridPQ	90
5.17	EXTRACT control mechanism in HybridPQ	90
5.18	Functions provided in HybridPQ	91
6.1	Top-Level Functional Block Diagram of Priority Queue Accelerator Module	93
6.2	compare and right-shift tasks in an INSERT operation	94
6.3	Left-shift tasks on an EXTRACT operation	95
6.4	Hardware Priority Queue Unit	95
6.5	INSERT operation in systolic array based hwPQ	96
6.6	Execution of identical tasks for one operation	97
6.7	idle and left-shift tasks in EXTRACT	97
6.8	RTL Architecture of Processing Element	98
6.9	Communication between PEs	99
6.10	Behavioral Description of PE	100
6.11	RTL Control Sequence of PE	101

6.12	Series of operations executed in pipeline	102
6.13	Pipelined execution of multiple INSERT	103
6.14	Pipelined execution of multiple EXTRACT	103
6.15	Symbolic representation of PEs in hwPQ	104
6.16	Example of INSERT followed by EXTRACT	105
6.17	Example of INSRT $\rightarrow$ NOP $\rightarrow$ EXTRACT	107
6.18	Several ways to insert idle state	108
6.19	Hardware Priority Queue Unit (hwPQ)	110
6.20	Timing Specification of hwPQ	111
6.21	Communication rule for RESET operation	113
6.22	Communication rule for INSERT operation	113
6.23	Communication rule for EXTRACT operation	114
6.24	Functional Block Diagram of Avalon Interface Unit	115
6.25	Functional Block Diagram of Avalon Data Unit	116
6.26	Behavioral Description of Avalon Data Unit	116
6.27	Functional Block Diagram of Avalon Control Unit	117
6.28	Behavioral Description of Avalon Control Unit	117
6.29	Control Flowchart of Avalon Control Unit	118
6.30	State Diagram of Avalon Control Unit	118
7.1	Simulation of Priority Queue Accelerator Module	121
7.2	Hardware Test Result	124
7.3	Overview of demonstration prototype	131
7.4	GUI of “VLSI Maze Routing DEMO” application	131
7.5	T_{PQ} VS Entire Graph Computation Run-Time	133
7.6	Size of Priority Queue for Entire Graph Computation	133
7.7	Dijkstra’s – Maximum Queue Size VS Graph Size	134
7.8	S-RABI – Maximum Queue Size VS Graph Size	134

7.9	Dijkstra's – Total number of operations VS Graph Size	135
7.10	S-RABI – Total number of operations VS Graph Size	135
7.11	S-RABI (FHPQ): Number of operations VS Graph Size	136
7.12	S-RABI (FHPQ): Total Cycle Elapsed for each operation	137
7.13	Dijkstra's – Speed up Gain of using HybridPQ	137
7.14	S-RABI – Speed up gain of using HybridPQ	138
7.15	S-RABI – FHPQ: Maximum Queue Size VS Graph Size	139
7.16	S-RABI – HybridPQ: Maximum Queue Size VS Graph Size	140
7.17	High Dense – S-RABI: Speed up gain of using HybridPQ	140
7.18	Less Dense – S-RABI: Speed up gain of using HybridPQ	141
7.19	S-RABI – HybridPQ: Speed up gain VS Maximum Queue Size	141
7.20	Dijkstra's – HybridPQ: Speed up Gain VS Maximum Queue Size	142

LIST OF SYMBOLS

API	-	Application Programming Interface
ASIC	-	Application Specific Integrated Circuit
CAD	-	Computer Aided Design
EDA	-	Electronic Design Automation
FPGA	-	Field Programmable Gate Array
GUI	-	Graphical User Interface
HDL	-	Hardware Development Language
IDE	-	Integrated Development Environment
I/O	-	Input/Output
LE	-	Logic Element
MHz	-	Megahertz
PC	-	Personal Computer
PE	-	Processing Element
RAM	-	Random Access Memory
RTL	-	Register Transfer Logic
SoC	-	System-on-Chip
SOPC	-	System-on-Programmable-Chip
UART	-	Universal Asynchronous Receiver Transmitter
UTM	-	Universiti Teknologi Malaysia
VHDL	-	Very High Speed Integrated Circuit Hardware Description Language
VLSI	-	Very Large Scale Integration

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	Numerical Example of Dijkstra’s Algorithm	158
B	Numerical Example of hop-by-hop Dijkstra’s Algorithm	167
C	Numerical Example of S-RABI Algorithm	175
D	Numerical Example of the Insertion Sort Priority Queue Operation	197
E	Introduction to Altera Nios II Development System	203
F	VHDL Source Codes of Priority Queue Accelerator Module	205
G	C Source Code for hwPQ device driver and HybridPQ API	210
H	Sample Graphs for Performance Test and Evaluation	216
I	Design Verification – Simulation Waveform	219

CHAPTER 1

INTRODUCTION

This thesis proposes a graph processing hardware accelerator for shortest path algorithms applied in nanometer VLSI interconnect routing problems. A custom Graph Processing Unit (GPU), in which a hardware priority queue accelerator module is embedded, designed and prototyped on a reconfigurable FPGA-based hardware platform. The hardware priority queue accelerator off-loads and speed up graph-based shortest path computations. For design validation and performance test purposes, a computationally extensive VLSI interconnect routing CAD module (or EDA sub-system) is developed to execute on the proposed GPU. This chapter introduces the background of research, objectives, problem statement, scope of work, previous related works and the significance of this research. The organization of thesis is summarized at the end of the chapter.

1.1 Background

Graphs are pervasive data structures in computer science, and algorithms working with them are fundamental to the field. There are many graph algorithms, and the well-established ones include Depth-First Search, Breadth-First Search, Topological Search, Spanning Tree algorithm, Dijkstra’s algorithm, Bellman-Ford algorithm and Floyd-Warshall algorithm. These graph algorithms are basically shortest path algorithms. For instance, Dijkstra’s algorithm is an extension of the Depth-First Search algorithm except the former solves the shortest path problem on weighted graph, while the latter solve the shortest unit path problem on unweighted graph. Bellman-Ford algorithm and Dijkstra’s algorithm solve single-source shortest path problem, except the former targets graph with negative edges, while the latter is restricted to graph with non-negative edges.

Many interesting problems in VLSI physical design automation are modeled using graphs. Hence, VLSI electronic design automation (EDA) systems are based on the graph algorithms. These algorithms include, among others, Min-Cut and Max-Cut algorithms for logic partitioning and placement, Clock Skew Scheduling algorithm for useful skew clock tree synthesis, Minimum Steiner Tree algorithm and Span Minimum Tree algorithm for critical/global interconnect network synthesis, Maze Routing algorithm for point-to-point interconnect routing, etc. Many routing problems in VLSI physical design are, in essence, shortest path problems in special graphs. Shortest path problems, therefore, play a significant role in global and detailed routing algorithms (Sherwani, 1995).

Real world problems modeled in mathematical set can be mapped into graphs, where elements in the set are represented by vertices, and the relation between any two elements are represented by edges. The run-time complexity and memory-consumption of graph algorithms are expressed in terms of the vertices and edges. A graph searching algorithm can discover much about the graph structure. Searching a graph means systematically following the edges of the graph so as to visit the vertices of graph. Many graph algorithms are organized as simple elaborations of basic graph searching algorithms (Cormen *et al.*, 2001). Hence, the technique of searching in a graph is the heart of these algorithms. In the graph searching process, Priority Queues are used to maintain the tentative search results, which can grow very large as the graph size increases. Consequently, the implementation of these priority queues can significantly affect the run-time and memory consumption of a graph algorithm (Skiena, 1997).

1.2 Problem Statement

According to Moore’s Law, to achieve minimum cost, the number of transistors in an Integrated Circuit (IC) needs to double every 18 months. Achieving minimum cost per transistor entails enormous design effort and high non-recurrent-engineering (NRE) cost. The design complexity grows proportionally to the increase of transistor density, and subsequently, circuit engineers face tremendous design challenges. When physical design moves into nanometer circuit integration range, we would encounter a combinatorial explosion of design issues, involving signal integrity, interconnect delay and lithography, which not only challenge the attempt for effective design automation, but further the need to suppress NRE cost, which in turn increases the demand of EDA (Electronic Design Automation) tools.

Conventional interconnect routing is rather straight-forward, and hence does not pose too great a challenge to the development of algorithms. However, the continual miniaturization of technology has seen the increasing influence of the interconnect delay. According to the simple scaling rule (Bakoglu, 1990), when devices and interconnects are scaled down in all three dimensions by a factor of S, the intrinsic gate delay is reduced by a factor of S but the delay caused by interconnect increases by a factor of S^2. As the device operates at higher speed, the interconnect delay becomes even more significant. As a result, interconnect delay has become the dominating factor affecting system performance. In many system designs targeting 0.35um – 0.5um, as much as 50% to 70% of clock cycles are consumed by interconnect delay. This figure will continue to rise as the feature technology size decreases further (Cong *et al.*, 1996). Consequently, the effect of interconnect delay can no longer be ignored in nanometer VLSI physical design.

Many techniques are employed to reduce interconnect delay; among them, buffer insertion has been shown to be an effective approach (Ginneken, 1990). Hence, in contrast to conventional routing which considers only wires, nanometer VLSI interconnect routing considers both buffer insertion and wire-sizing along the interconnect path, in order to achieve minimum interconnect delay. It is obvious that the complexity of nanometer interconnect routing is greater, and in fact, grows exponentially when multiple buffer choices and wire-sizes (at different metal layers, with different width and depth) are considered as potential interconnect candidates at each point along the interconnect path.

In general, given a post-placement VLSI layout, there are restrictions on where buffers may be inserted. For instance, it may be possible to route wires over a pre-placed macro-cell, but it may not be possible to insert buffers in that region. In this case, the routing has to, not only minimize the interconnect delay, but simultaneously strive for good buffer location, manage buffer density and congestion, and wire sizing. Consequently, many researches have proposed techniques in simultaneous maze routing with buffer insertion and wire sizing to solve the above interconnect routing problem.

A number of interconnect routing algorithms have been proposed, with different strategies for buffer insertion (Chu and Wong, 1997; Chu and Wong, 1998; Chu and Wong, 1999; Dechu *et al.*, 2004; Ginneken, 1990; Lai and Wong, 2002; Jagannathan *et al.*, 2002; Nasir, 2005; Zhou *et al.*, 2000). Most of these algorithms are formulated as graph theoretic shortest path algorithms. Clearly, as many parameters and constraints are involved in VLSI interconnect routing, these algorithms are, essentially, multi-weighted multi-constrained graph search algorithms. In graph search, the solution space and search results are effectively maintained using priority queues. The choice of priority queue implementation, hardware or software, differ significantly on how they affect the run-time and memory consumption of the graph algorithms (Skienna, 1997).

1.3 Objectives

The overall objective of this thesis is to propose the design of a graph processing hardware accelerator for high-speed computation of graph based algorithm. This objective is modularized into the following sub-objectives:

1) To design a Graph Processing Unit (GPU) customized for high-speed computation of graph based shortest path algorithm.

2) To design a priority queue accelerator module to speed up priority queue operations on the above custom GPU.

3) To verify the design and validate the effectiveness of accelerating, via hardware, priority queue operations in a graph algorithm. This is derived from performance validation studies on the application of the proposed GPU executing a compute-intensive VLSI interconnect routing algorithm.

1.4 Scope of Work

1) The Graph Processing Unit (GPU) is implemented on FPGA-based embedded system hardware platform on Altera Stratix II development board.

2) The priority queue accelerator module will have the following features:
   a. It supports the two basic priority queue function: (i) INSERT and (ii) EXTRACT.
   b. It is parameterizable so that the implemented length of priority queue can be adjusted based on available logic resources.
   c. It is cascade-able such that further queue length extension is possible.
   d. It is able to store each queue-entry in 64-bit: 32-bit for priority-value and 32-bit for the associate-identifier.

3) A hybrid hardware-software priority queue is developed. It avoids overflow at hardware priority queue module.

4) A demonstration application prototype is developed to evaluate the design. System validation and performance evaluation are derived by examining the graph based shortest path algorithms on this application prototype. Note that:

a. The test algorithm is called S-RABI for Simultaneous Maze Routing and Buffer Insertion algorithm, proposed by Nasir *et al.* (2006).
b. In order to utilize the hardware priority queue accelerator module effectively, the algorithms have to be modified.

1.5 Previous Related Work

The area of hardware maze router design, generic graph accelerator design, and priority queue has received significant attention over the years. In this section these previous related work are reviewed and summarized.

1.5.1 Hardware Maze Router and Graph Accelerator

Maze routing is the most fundamental algorithm among many other VLSI routing algorithms. Technically speaking, other routing problems can be decomposed into multiple sub-problems and solved with the maze routing algorithm. Many hardware maze routers had been proposed and most the work exploit the inherent parallelism of Lee’s algorithm (Lee, 1961). This includes the Full-Grid Maze Router, independently proposed by (Nestor, 2000; Keshk, 1997; Breuer and Shamsa, 1981). The architecture accelerates Lee’s algorithm using N*N identical processor-elements for worst-case N*N grid-graph, thus huge hardware resources are consumed. Another hardware maze router is the Wave-Front Machine, proposed by Sahni and Won (1987), and Suzuki et al. (1986). The Wave-Front-Machine uses N number of processing-elements and a status map for N*N grid graph.

A more flexible and practical design, the cellular architecture with Raster Pipeline Subarray (RPS) is proposed (Rutenbar, 1984a, 1984b). Applying raster scanning concept, the grid-graph is divided into smaller square regions and floated into RPS. For each square region, RPS updates the status-map. The architecture of RPS is complex but constant for any input size. Systolic Array implementation of

RPS is then proposed (Rutenbar and Atkins, 1988) for better handling of the pipelined data.

The above full-custom maze routers are specifically for maze routing, another approach to accelerate the graph-based shortest path algorithms is via generic graph accelerator. Unweighted graph represented in adjacency-matrix can be mapped into massive parallel hardware architecture where each of the processing units is a simple bit-machine. The computation of bit-wise graph characteristics: reachability, transitive closure, and connected-components can be accelerated. Huelsbergen (2000) had proposed such implementation in FPGA. Besides reachability, transitive closure and connected components, the computation of shortest *unit* path can be accelerated as well. An improved version, Hardware Graph Array (HAGAR) is proposed by Mencer et al. (2002) which uses RAM blocks than mere logic elements in FPGA. The proposed architecture of Huelsbergen (2000) and Mencer (2002) are actually quite similar to Full-Grid Maze Router except the former targets more generic application rather than the specific VLSI maze routing.

In general, most graph problems, however, are weighted. Shortest Path Processor proposed by Nasir and Meador (1995, 1996) can be used to solve weighted-graph problems. It uses square-array analog hardware architecture to direct benefit from the adjacency-matrix representation of graph. The critical challenge of such implementation lies on the accuracy of D/A converter and voltage comparator (both analog) to provide accurate result. An improved version called Loser-Take-All is then proposed, it uses current-comparator instead of voltage-comparator (Nasir and Meador, 1999). Besides that, a digital version is proposed to resolve inaccuracy issues resulted in analog design (Rizal, 1999). Specifically for undirected weighted graph problems, triangle-array is proposed by Nasir *et al.* (2002a, 2002b). The triangle-array saves about half of the logic resources consumed by square-array implementation.

All proposed previous work on hardware maze router and generic graph accelerator primarily explore the inherit parallelism of adjacency-matrix representation in graph. The major problem in such design required huge logic resources, e.g. generic graph accelerator uses Θ (V^2) logic resources for a graph of |V| vertices while maze router uses Θ (V^2) logic resources for a grid-graph of |V * V| vertices (see section 2.1 for definition of 'Θ'). In contrast, grid-graph for VLSI physical design is actually sparse; adjacency-matrix representation is simply a waste besides its inflexibility to support other graph variants.

The hardware maze routers and generic graph accelerators eventually required entire graph input at initial stage, before proceed for shortest unit path computation. On the other hand, nanometer VLSI routing adopts hop-by-hop approach during graph-searching; information of graph vertices is unknown prior to execution. This completely different scenario reflects that the conventional maze routers and generic graph accelerators are not an option.

In addition to that, the hardware maze routers and generic graph accelerators are designed to accelerate elementary graph algorithms, e.g. shortest unit path, transitive closure, connected-components, etc, not only nanometer VLSI routing has evolved into shortest path problem, it has evolved into multi-weight multi-constraint shortest path problem. Certain arithmetic power is needed besides complex data manipulation. This phenomenon leaves no room for the application of the primitive parallel hardware discussed above. New designs of hardware graph accelerators are needed.

1.5.2 Priority Queue Implementation

Due to the wide application of priority queue, much research effort had been made to achieve better priority queue implementations. In general, the research on priority queue can be categorized into: (i) various advanced data structure for priority queue, (ii) specific priority queue data structure with inherent parallelism, targeted Parallel Random Access Machine (PRAM) model, and (iii) full-custom hardware design to accelerate array-based priority queue.

Research in category (i) basically explore the various ‘heap’ structure (a variant of ‘tree’ data structure) to obtain theoretically better run-time complexity of priority queue operations. Binary-Heap, Binomial-Heap and Fibonacci-Heap are some instances of priority queue implementation under this category. Whereas research classified in category (ii) includes, among others, Parallel-Heap, Relaxed-Heap, Sloped-Heap, etc. Basically, priority queue implementation under these two categories is interesting from software/parallel-software point of view; these implementations are capable to provide improvement in term of run-time complexity at the expenses of more memory consumption, but fail to address the severe constant overhead on memory data communication. In short, those heap-like structures are interesting in software but are not adaptable for high speed hardware implementation (Jones, 1986).

Research work in category (iii), full-custom hardware priority queue design is driven by the demand of high-speed applications such as internet network routing and real-time applications. These hardware priority queue can achieve very high throughput and clocking frequency, thus improve the performance of priority queue in both run-time complexity and communication overhead. Works in (iii) includes *Binary Trees of Comparator* (BTC) by Picker and Fellman (1995); the organization of comparators mimics the Binary-Heap. New elements enter BTC through the leaves, the highest priority element is extracted from the root of BTC; therefore constant O(lg *n*) run-time for BTC priority queue operations.

Ioannou (2000) proposed another variant of hardware priority queue, the *Hardware Binary-Heap Priority Queue*. The algorithm maintaining Binary-Heap property is pipelined and executed on custom pipelined processing units, results constant O(*1*) run-time for both INSERT and EXTRACT priority queue operations. Another implementation similar to it but using Binary-Random-Access-Memory (BRAM) is also proposed by Argon (2006). Noted, adding successive layer at binary-tree double the total number of tree-nodes, all these binary-tree based designs suffer from quadratic expansion complexity.

Brown (1988) and Chao (1991), independently propose the implementation of hardware priority queue using First-In-First-Out architecture, called *FIFO Priority Queue*. For *l*-levels of priority, *l* numbers of FIFO arrays is deployed; each stores elements of that priority. This implementation gives constant O(*1*) run-time, besides the FIFO order among elements with same priority is maintained. This implementation inherits the disadvantage as discussed: if the desired priority-level is large, huge number of FIFO arrays is needed. For example, if 32-bit priority-value is desired, then 4,294,967,296 FIFO arrays are needed.

Shift Register and ***Systolic-Shift-Register*** implementation of priority queue (Toda *et al.*, 1995; Moon *et al.*, 2000) has better performance compared to the above designs. The priority level and the implemented worst-case priority queue size can be easily scaled. The designs deploy O(*n*) processing-elements arranged in one dimensional array, for constant O(*1*) INSERT and EXTRACT run-time complexity. The designs has the disadvantage of severe bus loading effect because all processing-elements are connected to the input data bus, which results in low clocking frequency.

1.6 Significance of Research

This research is significant in that it tackles the issue of interconnect delay optimization in VLSI physical design since the interconnect delay now dominates gate delay in nanometer VLSI interconnect routing. Existing maze routers consider interconnects contribute negligible delay, which is now not correct. Nanometer VLSI routing algorithms now has to include strategies to handle interconnect delay optimization problem which include, among others, buffer insertion. Consequently, the algorithms are now more complex in that they are modeled using multi-weighted multi-constrained graphs. These graphs involve searching over millions of nodes, and hence the algorithms are now extremely compute-intensive. The need for hardware acceleration as proposed in this research is clear. The contribution of this research is as follows:

1) A comprehensive design of a 32-bit, parameterizable hardware priority queue accelerator module to accelerate priority queue operations. The module is incorporated into a graph processing unit, GPU. Modifications to the graph algorithms are made such that the proposed design can be applied with other graph-based shortest path algorithms.

2) A hybrid priority queue based on hardware-software co-design is also developed. Such implementation introduces a simple yet efficient control mechanism to avoid overflow in hardware priority queue module.

3) An application demonstration prototype of a graph processing hardware accelerator is developed. It includes the front-end GUI on host to generate sample post-placement layout. Figure 1.1 gives the architecture of the proposed system.

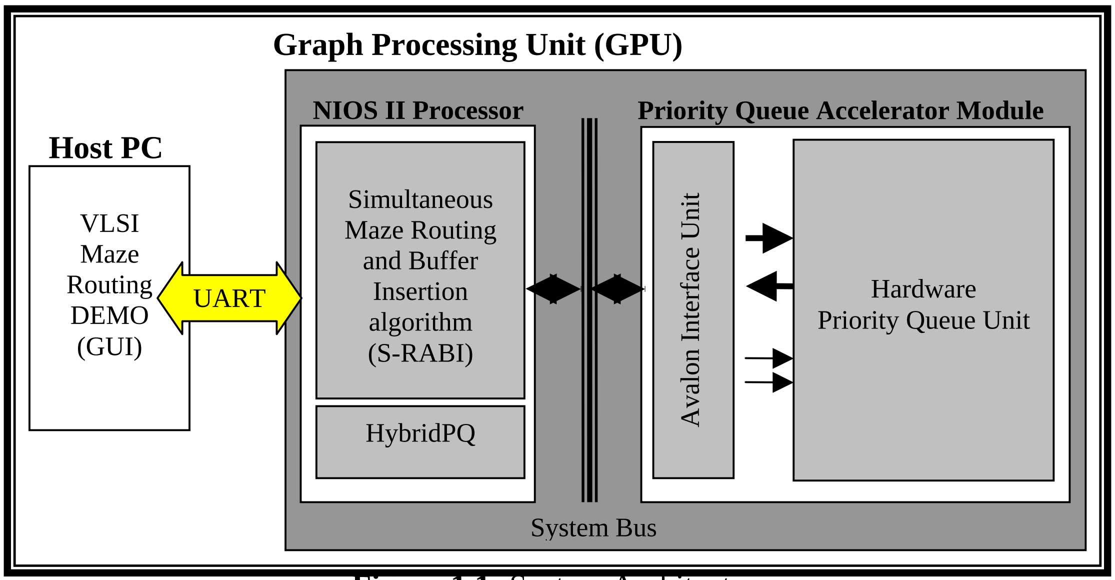
Graph Processing Unit (GPU)
Host PC
VLSI Maze Routing DEMO (GUI)
UART
NIOS II Processor
Simultaneous Maze Routing and Buffer Insertion algorithm (S-RABI)
HybridPQ
Priority Queue Accelerator Module
Avalon Interface Unit
Hardware Priority Queue Unit
System Bus

Figure 1.1: System Architecture

1.7 Thesis Organization

The work in this thesis is conveniently organized into eight chapters. This first chapter presents the motivation and research objectives and follows through with research scope, previous related works, research contribution, before concluding with thesis organization.

The second chapter provides brief summaries of the background literature and theory reviewed prior to engaging the mentioned scope of work. Several topics related to this research are reviewed to give an overall picture of the background knowledge involved.

Chapter Three discusses the priority queue algorithm which leads to our hardware design. Next, the Simultaneous Maze Routing and Buffer Insertion (S-RABI) algorithm applied in nanometer VLSI routing module is presented. It entails the two underlying algorithms which form the S-RABI algorithm.

Chapter Four presents the necessary algorithmic modification on the S-RABI algorithm in order to benefit from the limited but fast operation of hardware priority queue. Next the architecture chosen for the implementation of hardware priority queue accelerator is described; followed by the necessary modifications on the priority queue algorithm for better hardware implementation.

Chapter Five explains the design of the Graph Processing Unit. First the top-level description of GPU is given; followed by each of its sub-components: the NIOS II processor, the system bus, the bus interface and the priority queue accelerator module. Also in this chapter, the development of device driver and HybridPQ is discussed.

Chapter Six delivers the detailed description on the design of priority queue accelerator module. This includes the Hardware Priority Queue Unit and the required bus interface module as per required by our target implementation platform.

Chapter Seven describes the simulation and hardware test that are performed on individual sub-modules, modules and the system for design verification and system validation. Performance evaluations of the designed priority queue accelerator module are discussed and comparisons with other implementations are made. This chapter also illustrates the top-level architecture of nanometer VLSI routing module developed to be executable on GPU. Further by detail analysis on the performance of graph algorithm with the presence of priority queue accelerator module.

In the final chapter of the thesis, the research work is summarized and deliverables of the research are stated. Suggestion for potential extensions and improvements to the design is also given.

1.8 Summary

In this chapter, an introduction was given on the background and motivation of the research. The need for a hardware implementation of priority queue module to accelerate graph algorithm, particularly state-of-the-art nanometer VLSI interconnect routing is discussed. Based on it, several scope of project was identified and set to achieve the desired implementation. The following chapter will discuss the literature relevant to the theory and research background.

REFERENCES

Altera Corporation (2003a). *Introduction to Quartus II.* Altera Corporation.

Altera Corporation (2003b). *SOPC Builder Data Sheet*. Altera Corporation.

Altera Corporation (2004a). *Nios II Hardware Development Tutorial*. Altera Corporation.

Altera Corporation (2004b). *Nios II Processor Reference Handbook*. Altera Corporation.

Altera Corporation (2004c). *Nios II Software Developer's Handbook*. Altera Corporation.

Altera Corporation (2005a). *Avalon Interface Specification*. Altera Corporation.

Alpert, C. J., Hu, J., Sapatnekar, S. S. and Villarrubia, P. G. (2001). A Practical Methodology for Early Buffer and Wire Resource Allocation. *IEEE/ACM Design Automation Conference, Las Vegas.* 2001. Nevada, United States: IEEE/ACM*,* 189-195.

Alpert, C. J., Hrkic, M. and Quay, S. T. (2004). A Fast Algorithm for Identifying Good Buffer Insertion Candidates Locations. *ACM International Symposium on Physical Design (ISPD'04)*. Phoenix, Arizona, USA: ACM, 47-52.

Argon, J. A. (2006). *Real-Time Scheduling Support for Hybrid CPU/FPGA SoCs*. University of Kansas, United States of America: Master Degree Thesis.

Auletta, V., Das, S. K., Vivo, A. D., Pinotto, M. C., Scarano, V. (2002). Optimal Tree Access by Elementary and Composite Templates in Parallel Memory Systems. *IEEE Transactions on Parallel and Distributed Systems*. 13(4): 399-411.

Bakoglu, H. B. (1990). *Circuits, Interconnects, and Packaging for VLSI*. Reading MA: Addison-Wesley.

Bhagwan, R., Lin, B. (2000a). Fast and Scalable Priority Queue Architecture for High-Speed Network Switches. *IEEE Annual Conference on Computer Communication (INFOCOM 2000)* Tel Aviv, Israel: IEEE, vol. 2, 538-547.

Bhagwan, R., Lin, B. (2000b). Design of a High-Speed Packet Switch with Fine-Grained Quality-of-Service Guarantees. *IEEE International Conference on Communication (ICC 2000).* New Orleans, USA: IEEE, vol. 3, 1430-1434.

Breuer, M. and Shamsa, K. (1981). A Hardware Router. *Journal of Digital Systems*. 4(4): 393-408.

Brodal, G. S., Zaroliagis, C. D. and Traff, J. L. (1997). A Parallel Priority Data Structure with Applications. *The 11th International Parallel Processing Symposium*, Geneva, Switzerland. 689-693.

Brown, R. (1988). Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem. *Communications of the ACM*. October 1988. 31(10): 1220-1227.

Chao, J. (1991). A Novel Architecture for Queue Management in the ATM Network. *IEEE Journal on Selected Areas in Communication.* 9(7): 1110-1118.

Chu, C. C. N., Wong, D. F. (1997). A new approach to buffer insertion and wire sizing. *Proceeding IEEE International Conference on Computer Aided Design 1997*. San Jose, California: IEEE, 614-621.

Chu, C. C. N., Wong, D. F. (1998). A Polynomial Time Optimal Algorithm for Simultaneous Buffer and Wire Sizing. *Proc. Design Automation & Test 1998*. Europe. 479-485.

Chu, C. C. N., Wong, D. F. (1999). A Quadratic Programming Approach to Simultaneous Buffer Insertion/Sizing and Wire Sizing. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*. 18(6): 787-798.

Cong, J., Kong, T. and Pan, D. Z. (1999). Buffer Block Planning for Interconnect-Driven Floorplanning. *IEEE/ACM International Conference on Computer Aided Design 1997*. San Jose, California: IEEE, 358-363.

Cong, J., Lei, H., Koh, C-K., Madden, P. H., (1996). Performance Optimization of VLSI Interconnect Layout. Technical Report, Dept. of Computer Science, University of California, L.A., 1-99.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001) *Introduction to Algorithms*. 2nd Edition. The MIT Press, McGraw-Hill Book Company.

Das, S. K., Sarkar, F. and Pinotti, M.C. (1996a). Distributed Priority Queues on Hypercube Architectures. *IEEE, Proceedings of the 16-th International Conference on Distributed Computing Systems 1996*. Hong Kong: IEEE, 620-627.

Das, S. K., Sarkar, F. and Pinotti, M.C. (1996b). Optimal and Load Balanced Mapping of Parallel Priority Queues on Hypercubes. *IEEE Transactions on Parallel and Distributed Systems.* 555-564.

Driscoll, J. R., Gabow, H. N., Shrairman, R. and Tarjan, R. E. (1998). Relaxed Heaps: An Alternative to Fibonacci-Heaps with Applications to Parallel Computation. *Communications of the ACM.* 31(11): 1343-1354.

Dechu, S., Shen, Z. C., Chu, C. C. N. (2004). An Efficient Routing Tree Construction Algorithm with Buffer Insertion, Wire Sizing and Obstacle Consideration. *Proceedings of the ASP-DAC 2004*. Yokohama, Japan.

Elmore, W. C. (1948). The transient response of dampled linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics.* 19(1): 55-63.

Ginneken, L. P. P. P. V. (1990). Buffer Placement in Distributed RC-Tree Networks for Minimal Elmore Delay. *Proc. International Symposium of Circuits and Systems 1990*. 865-868.

Gupta, A. K. and Phoutiou, A. G. (1994). Load Balanced Priority Queue Implementations on Distributed Memory Machine. *ACM - Lecture Notes in Computer Science, July 1994*. vol. 817, pp. 689-700.

Huelsbergen, L. (2000). A Representation for Dynamic Graphs in Reconfigurable Hardware and its Application to Fundamental Graph Algorithms. *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays 2000*. Monterey, CA, USA:ACM, 105-115.

Ioannou, A. D. (2000). *An ASIC Core for Pipelined Heap Management to Support Scheduling in High Speed Networks. Technical Report FORTH-ICS/TR-278 October 2000.* Computer Architecture and VLSI Systems Laboratory (CRAV), Institute of Computer Science (ICS), Foundation for Science and Technology – Hellas (FORTH), University of Crete, Greece. Master Degree Thesis.

Ioannou, A. D. and Katevenis, M. (2001). Pipelined Heap (Priority Queue) Management for Advanced Scheduling in High-Speed Networks. *IEEE International Conference on Communications (ICC 2001)*. Helsinki, Finland: IEEE, vol. 7, 2043-2047.

Jagannathan, A., Hur, S-W. and Lillis, J. (2002). A Fast Algorithm for Context-Aware Buffer Insertion. *ACM Trans. On Design Automation of Electronic Systems, January 2002*. 7(1): 173-188.

JohnsonBaugh, R. and Schaefer, M. (2004). *ALGORITHMS*, Pearson Prentice Hall, 2004.

Jones, D. (1986). An Empirical Comparison of Priority-Queue and Event-Set Implmentations. *Commununication of the ACM, April 1986*. 29(4): 300-311.

Khalil M, Koay K H, (1999). VHDL Module Generator: A Rapid-prototyping Design Entry Tool for Digital ASICs. *Jurnal Teknologi UTM*, December. 31:45-61.

Keshk, H., Mori, S., Nakashima, H., Tomita, S. (1996). Amon2: A parallel wire routing algorithm on a torus network parallel computer. *Proceedings of the 10th international conference on Supercomputing, January1996*. 197-204.
Kuiper, F. A. and Mieghem, P. V. (2004a). Concepts of Exact QoS Routing Algorithms. *ACM/IEEE Trans. on Computer Networking (TON), 2004*. 12(5): 851-864.

Kuiper, F. A. and Mieghem, P. V. (2004b). Quality-of-Service Routing in the Internet: Theory, Complexity, and Algorithms. Delft University of Technology, Netherlands: PhD Thesis.

Kung. H.T. (1980). The Structure of Parallel Algorithm. *Advances in Computers.* 19: 65-112. Academic Press, Inc.

Lai, M. and Wong, D. F. (2002). Maze routing with buffer insertion and wire sizing. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Oct 2002*. 21: 1205-1209.

Lavoie, P. and Savaria, Y. (1994). A Systolic Architecture for Fast Stack Sequential Decoders. *IEEE Transaction on Communication, Feb./Mar./Apr. 1994*. 42(2/3/4): 324-334.

Lee, C. Y., (1961). An Algorithm for Path Connections and Its Applications. *IRE Transactions on Electronic Computers*, 1961.

Leiserson, C.E. (1979) Systolic Priority Queue. *Proceeding Caltech Conference of VLSI.* Jan. 1979. Caltech, Pasadena, California. 200-214.

Meador, J. L. (1995) Spatiotemporal Neural Networks for Shortest Path Optimization. *Proc. IEEE International Symposium on Circuits and Systems (ISCAS95)*. Seattle, Washington, USA. II801-II804.

Mencer, O., Huang, Z. and Huelsbergen, L. (2002). HAGAR: Multi-Context Hardware Graph Accelerators. *12th International Conference of Field Programmable Logic and Applications 2002*. France.

Moon, S.W., Rexford, J., Shin, K.G. (2002). Scalable Hardware Priority Queue Architectures for High-Speed Packet Swicthes. *IEEE Transaction on Computers, Nov. 2000*. 49(11).

Nasir, S., Meador, J. L. (1995). Mixed Signal Neural Circuits for Shortest Path Computation. *Proc. IEEE Conference on Signals, System and Computers 1995*. California, USA. II876-II880.

Nasir, S., Meador, J. L. (1996). Spatiotemporal Neural Networks for Link-State Routing Protocols. *Proc. IEEE International Symposium on Circuits and Systems (ISCAS96)*. Atlanta, Georgia. III547-III550.

Nasir, S., Meador, J. L. (1999). A High Precision Current Copying Loser-Take-All Circuit. *Proc. World Engineering Congress (WEC99)*. Malaysia. EE177-179.

Nasir, S., Khalil, M., Teoh, G. S. (2002a). Implementation of Recurrent Neural Network for Shortest Path Calculation in Network Routing. *Proc. IEEE International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN), 2002, Manila, Philipines.* 313-317.

Nasir, S., Khalil, M., Teoh, G. S. (2002b). Design and Implementation of a Shortest Path Processor for Network Routing. *Proc. 2nd World Engineering Congress (WEC '02), Malaysia*. EE175-179.

Nasir, S., Khalil, M. (2005). Multi-Constrained Routing Algorithm for Minimizing Interconnect Wire Delay. *Universiti Teknologi Malaysia*: Ph.D. Research Proposal.

Nasir, S., Khalil, M., Ch'ng, H. S. (2006). Simultaneous Maze Routing and Buffer Insertion, *VLSI-ECAD Research Laboratory, Universiti Teknologi Malaysia*: Technical Report VLSI-ECAD-TR-NSH-021-06.

Nestor, J. A. (2002). A New Look at Hardware Maze Routing. *Proceedings of the 12th ACM Great Lakes Symposium on VLSI*. April 18-19, 2002. New York, USA. 142-147.

Picker, D. and Fellman, R. (1995). A VLSI Priority Packet Queue with Inheritance and Overwrite. *IEEE Transaction on Very Large Scale Integration Systems, June 1995*. 3(2): 245-252.

Prasad, S. and Deo, N. (1992). Parallel Heap: Improved and Simplified. *Proc. IEEE 6th International Parallel Processing Symposium 1992*. California, USA: IEEE, 448-451.

Prasad, S. and Sawart, S.I. (1995). Parallel Heap: A Practical Priority Queue for Fine-to-Medium-Grained Applications on Small Multiprocessors. *Proc.7th IEEE Symposium on Parallel and Distributed Processing 1995*. Santa Barbara, CA: IEEE, 328-335

Ranade, A., Cheng, S., Deprit, E., Jones, J. and Shih, S. (1994). Parallelism and Locality in Priority Queues. *Proceeding 6th IEEE Symposium on Parallel and Distributed Processing*. Oct 1994. Dallas. (99): 490-496.

Rizal, K. G. (1999). *Shortest Path Processor Using FPGA*. Universiti Teknologi Malaysia: Bachelor Degree Thesis.

Rutenbar, R. A. (1984a). A Class of Cellular Computer Architectures to Support Physical Design Automation. Univ. of Michigan, Computing Res. Lab.: Ph.D. dissertation, CRL-TR-35-84.

Rutenbar, R. A., Mudge, T. N. and Atkins, D. E. (1984b). A Class of Cellular Architectures to Support Physical Design Automation, *IEEE Trans. Computer-Aided Design, Oct. 1984*. vol. CAD-3: 264-278.

Rutenbar, R. A. and Atkins, D. E. (1988). Systolic Routing Hardware: Performance Evaluation and Optimization. *IEEE Transaction on Computer-Aided Design, Mar. 1988*. vol. 7, 397-410.

Sahni, S. and Won, Y. (1987). A Hardware Accelerator for Maze Routing. *Proceeding on Design Automation Conference 1987*. Miami, Florida:ACM/IEEE, 800-806.

Saxena, P., Menezes, N., Cocchini, P. and Kirkpatrick, D. A. (2003). The Scaling Challenge: Can Correct-by-Construction Design Help?. *Proceeding International Symposium on Physical Design 2003*. San Diego, CA:ACM/SIGDA 51-58.

Seido, A. I. A., Nowak, B. and Chu, C. (2004). Fitted Elmore Delay, A Simple and Accurate Model. *IEEE Trans. on VLSI, July 2004*. 12(7): 691-696.

Sherwani, N. (1995). *Algorithms for VLSI Physical Design Automation*, 2nd Edition. Intel Corporation: Kluwer Academic Publishers, Toppan Company (S) Pte. Ltd.

Skiena, S. S. (1997). *The Algorithm Design Manual*. New York: Springer-Verlag.

Suzuki, K., Matsunaga, Y., Tachibana, M. and Ohtsuki, T. (1986). A hardware maze router with application to interactive rip-up and reroute, *IEEE Trans. Computer-Aided Design*. Oct. 1986. vol. 5, 466-476.

Tommiska, M. and Skytt, J. (2001). Dijkstra's Shortest Path Routing Algorithm in Reconfigurable Hardware. *11th International Conference of Field Programmable Logic and Applications*. Monterey, CA. 653-657.

Toda, K., Nishida, K., Takahashi, E., Michell, N. and Tamaguchi, Y. (1995). Design and Implementation of a Priority Forwarding Router Chip for Real-Time Interconnect Networks. *Int. J. Mini and Microcomputers 1995*. 17(1): 42-51.

Wolf, W. (2002). *Modern VLSI Design: System-on-Chip Design, 3/E*, Chapter 3, Prentice Hall.

Zhang, W. and Korf, R. E. (1992). Parallel Heap Operations on EREW PRAM: Summary of Results. *Proc. 6th IEEE International Parallel Processing Symposium*, 1992. Beverly Hills, CA, USA:IEEE, 315-318.

Zhou, H., Wong, D. F., Liu, I-M. and Aziz, A. (2000). Simultaneous Routing and Buffer Insertion with Restrictions on Buffer Locations. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, July 2000*. vol. 19, 819-824.