ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Rapid Area-Time Estimation of C-Based Applications for Hardware/Software Partitioning and Design Exploration

LIEU MY CHUONG

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Master of Engineering

2009

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Acknowledgements

I believe that I am so indebted to many people during my endeavor of completing my M.Eng Degree by Research. First and foremost, I would like to thank Professor Thambipillai Srikanthan for providing the opportunity to research in the field of Embedded Computing. His visions and experiences in the field of high-performance Embedded Processing have greatly motivated me and I am grateful for his guide and support throughout the course of the research.

Secondly, I'd like to express my sincere gratitude towards Mr Lam Siew Kei. I have no word to describe how much I appreciate his help which contributed significantly to the research. He has constructively challenged my findings to strengthen my dissertation and conference papers. He has dedicatedly helped me develop and organize my thesis so as to best convey our contributions … Without him, this thesis and research would never ever be in shape. With a Korean-style bow: "Thank you very much, bro!"

Although Dr Ling Keck Voon and Ms Wong Wai Ee from DSI are not involved directly in this project, their help and support in the last projects have laid a solid foundation for me in the field of FPGA. Therefore, I'd like to take this opportunity to acknowledge their indirect contribution to my achievements.

I also want to thank Jeremiah and Ms Nah who have helped me set up lab machineries to carry out my experiments. Thank you Peter Duong, Victor Nguyen, Sunita and Ramea! Daily chats and discussions with you guys were really fun and helpful! Besides, I'd like to thank Mr Gia Bao for the *financial rescue* last year :D. Thank you all !!!

Of course, most important supports and motivations come from my folks. Dad, Mom and my little sister. Thank you for your kind words (*jia you jia you*), good care and delicious food :p.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Abstract

FPGAs (Field-Programmable Gate Arrays) have become an attractive solution to meet the technological and market uncertainties in embedded processing. The advent of reconfigurable systems that incorporates microprocessor and FPGA on the same platform necessitates efficient hardware-software partitioning strategies in emerging design flows. However, commercially available design flows do not enable designers to make design explorations for effective hardware-software partitioning.

In this thesis, we propose a framework to realize the rapid estimation of hardware area-time trade-off measures that can facilitate in the FPGA porting of C-codes segments. The proposed framework relies on the Trimaran compiler infrastructure for its advanced scheduling schemes to expose the inherent parallelism in C-based algorithms. The original VLIW machine model employed in Trimaran was modified to incorporate heterogeneous functional units that are well suited for efficient FPGA implementation. In addition, we have proposed an architecture template in order to assist in the estimation of the data-path and control-path of the hardware generated to accelerate the C-code segment.

The proposed data-path estimation technique relies on bit-width analysis and a hybrid approach for the estimation of the area-time measures. The hybrid approach employs pre-characterized parameters and analytical models for area-time estimation. Our experiments show that the bit-width optimization technique employed leads to area reduction of up to 3 times.

We have also proposed two methods to improve the estimation of the delay measures by taking into account the post implementation interconnect delay between the hardware components by addressing the slack distribution of all the combinatorial paths and floor-planning considerations to predict the placement of the hardware components on the FPGA.

The area-time estimation of the control unit based on one-hot encoding scheme examines both the next-state decoding logic and control signal decoding logic provides an average error of only about 10% when compared to commercial tools.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

A hardware generator has been developed to automatically generate synthesizable RTL codes for a given C code segment. Results show that our RTL implementations are comparable to those generated using the commercial Altera C2H RTL generator. Finally, our proposed framework outperforms the commercial tool for some cases and, unlike existing commercial tools such as Impulse-C or Celoxica Handel C, the proposed framework is capable of accelerating the area-time trade-off analysis to facilitate the porting of C-codes to commercial FPGA architectures.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE OF CONTENTS

Table of Contents 1
List of Abbreviations 3
List of Figures 5
Lists of Tables 7
CHAPTER 1 Introduction 8
1.1. Research Objectives 9
1.2. Research Overview 10
1.3. Thesis Organization 14
CHAPTER 2 Background & Literature Review 15
2.1. High-Performance Embedded System Design Challenges and Trends 16
2.1.1. Embedded systems 16
2.1.2. Embedded applications 17
2.1.3. Market requirements 17
2.1.4. Hardware technologies 17
2.1.5. Design Methodologies 19
2.2. High-level Estimation Process 22
2.2.1. Background 22
2.2.2. System-level estimation 23
2.2.3. High-level Synthesis 25
2.2.4. Hardware estimation 26
2.3. Commercial Tools for Automatic RTL Generation 28
2.4. Low-level estimation for logic synthesis and place and route 29
2.5. Summary 30
CHAPTER 3 Compiler assisted approach for extracting inherent parallelism 34
3.1. Design Approaches and Challenges of HWAs 35
3.2. Trimaran Infrastructure 37
3.2.1. Trimaran compiler overview 37
3.2.2. VLIW architecture 39
3.2.3. Important Trimaran capabilities 39
3.2.4. Inefficiencies of VLIW architecture for hardware implementation 47
3.3. Proposed Architecture Template for area-time estimation 48
3.3.1. h-VLIW architecture 48
3.3.2. RTL Models of the h-VLIW architecture 50
3.4. Mapping Trimaran's IR 53
3.5. Summary 59
CHAPTER 4 Control Path Area-Time Estimation 60
4.1. Overview of FSM and Encoding Schemes 61
4.1.1. Common Encoding Schemes 61
4.2. Constructing FSM for Area-Time Estimation 64
4.2.1. NSDL (Next State Decoder Logic) 64
4.2.2. CSDL (Control Signal Decoding Logic) 68
4.3. Estimation of FSM 69
4.3.1. Estimation of NSDL 69
4.3.2. Estimation of CSDL 72
4.3.3. Estimation of Total Number of Slices for the Control Unit 76
4.4. Evaluation of the proposed models 77
4.5. Summary 83
CHAPTER 5 Data-path Area-Time Estimation 84

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

5.1. Characterizing the RTL Components and Interconnect....85
5.1.1. Characterizing the RTL components....86
5.1.2. Charactering the Interconnect Delay....88
5.2. Area Estimation....92
5.3. Delay Estimation....95
5.3.1. Static Estimation Delay for Delay-Paths with Non-Pipelined CFUs....96
5.3.2. Static Estimation Delay for Delay Paths with Pipelined CFUs....96
5.3.3. Estimation delay caused by CFU fan-out....97
5.4. Evaluation of the proposed approaches....98
5.4.1. Area Estimation....98
5.4.2. Delay Estimation....101
5.5. Summary....105
CHAPTER 6 Dynamic delay estimation process....106
6.1. Delay Estimation Based on Slack Distribution (SLK)....107
6.1.1. Overview of Approach....107
6.1.2. Determining the value of Λ....108
6.1.3. Evaluation of the SLK approach....110
6.2. Delay Estimation using Simplified Floor-Planning Method (SFP)....113
6.2.1. Overview of Proposed Approach....114
6.2.2. Simplified Floor-planning Algorithm....115
6.2.3. Performance Evalutations....118
6.3. Comparison Between the SLK and SFP Approaches....121
6.4. Summary....122
CHAPTER 7 Logic Optimizations for High-Level Estimation....123
7.1. Bit-width Analysis and Constant Propagation....124
7.1.1. Bit-width analysis....124
7.1.2. Constant Propagation....125
7.2. Bit-width Aware Area-Time Estimation of RTL Components....127
7.2.1. Estimation Model for the Adder/Subtractor....129
7.2.2. Estimating area-time of Embedded Multipliers....131
7.2.3. Estimating Equal Comparator....136
7.2.4. Estimating LT/GT (Less Than/Greater Than) Comparator....139
7.2.5. Estimating area-time of Multiplexers....142
7.3. Summary....145
CHAPTER 8 Performance evaluation of entire area-time estimation framework....146
8.1. Complete framework....148
8.2. Result Analysis for Xilinx and Altera FPGAs....153
8.2.1. Experimental Setup....153
8.2.2. Estimation with Xilinx FPGA(Spartan3-4 and Virtex2Pro-6)....156
8.2.3. Experiments for Altera FPGA (CYCLONEIIC8)....169
8.2.4. Comparison of the Execution Time....173
8.3. Comparison with Altera C2H....174
8.4. Summary....176
CHAPTER 9 Conclusions and future work....178
9.1. Conclusions....178
9.2. Future Works....181
Bibliography....183

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

LIST OF ABBREVIATIONS

CIT	Control Information Table
CLB	Configurable Logic Block
FF	Flip-Flops
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HWA	Hardware Accelerator
LC	Logic Cell
LUT	Look-up Table
RTL	Register Transfer Level
TTM	Time to Market
NRE	Non Recurring Engineering
VLIW	Very Long Instruction Word
EDA	Electronic Design Automation
ESL	Electronic System Level
PCB	Printed Circuit Board
ESL	Electronic System Level
IR	Intermediate Representation
ILP	Instruction Level Parallelism
LLP	Loop Level Parallelism

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TLP	Thread Level Parallelism
CFU	Configurable Functional Unit
DFG	Data Flow Graph
PAR	Place and Route
NSDL	Next State Decoding Logic
CSDL	Control Signal Decoding Logic
h-VLIW	Hybrid Very Long Instruction Word
UUC	Unit Under Characterization
SLK	Slack-based Method
SFP	Simplified Floorplan –based Method

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

LIST OF FIGURES

Figure 1.1 CRATE Framework....12
Figure 1.2 Snapshot of CRATE which has been integrated into Trimaran's GUI....13
Figure 2.1 Factors that influences the design of embedded systems....17
Figure 2.2 Characteristic spectrum of various computing platforms....19
Figure 2.3 Evolution of embedded computing platforms and design methodologies..20
Figure 2.4 General co-design flow with focus on hardware development....22
Figure 2.5 Typical high-level estimation flow....24
Figure 2.6 Two-level methodology of Carlo....28
Figure 3.1 FIR filter: software and hardware implementation....36
Figure 3.2 Simple DFG....36
Figure 3.3 Trimaran compiler infrastructure....39
Figure 3.4 VLIW architecture....40
Figure 3.5 Super-block scheduling....41
Figure 3.6 Eliminate Branch using If conversion to form hyper-blocks....42
Figure 3.7 Meld scheduling....43
Figure 3.8 Example of Calling functions....43
Figure 3.9 Example of Loop Unrolling....44
Figure 3.10 Illustration of the concept of loop pipelining....45
Figure 3.11 Illustration of Custom instruction substitution....45
Figure 3.12 Schedules with different number of functional units....47
Figure 3.13 Bus-based architecture and mux-based architecture....50
Figure 3.14 Two realizations for the same algorithm (b) pipelined datapath (c) data-path with shared resources....50
Figure 3.15 Architecture Template with heterogeneous Functional Units (h-VLIW)..51
Figure 3.16 General view of accelerator at RTL....52
Figure 3.17 RTL Model of Customized Functional Units (CFU)....52
Figure 3.18 RTL model of multiplexer....53
Figure 3.19 RTL model of register....53
Figure 3.20 Mapping DFG to the architecture template....55
Figure 3.21 Example of proposed resource binding strategy....56
Figure 3.22 Circuits with and without resource sharing....57
Figure 3.23 Operation binding algorithm....58
Figure 4.1 General overview of a Finite State Machine....62
Figure 4.2 Cascaded LUTs for the decoding logic of binary and gray encoding scheme....65
Figure 4.3 An example CFG....65
Figure 4.4 Internal Data Structure for FSM construction....68
Figure 4.5 RTL codes of NSDL....68
Figure 4.6 Illustration of 4 Registers feeding one CFU via Mux....69
Figure 4.7 Decoder for multiplexer's select signals in VHDL....70
Figure 4.8 An example FSM....71
Figure 4.9 Implementation of the example FSM....71
Figure 4.10 Tree of 4-input OR-gates....73
Figure 4.11 implementation of OR-tree using Xilinx carry chain....74
Figure 4.12 Forming OR-tree by cascading LUTs....74
Figure 4.13 Algorithm to compute LUT and depth of OR-tree....75
Figure 4.14 Implementation of an OR-tree using carry chain....76

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Figure 4.15 Implementation details of OR-tree using carry chain 77
Figure 4.16 LUT-FF packing pattern 78
Figure 4.17 Synthesis option settings 80
Figure 4.18 Setting up the decoder circuit for verification with the estimation technique 80
Figure 5.1 Data-path with typical connections 84
Figure 5.2 Inserting input-output registers to characterize RTL components 86
Figure 5.3 Schematic capture of a 16-bit adder that is used for characterization 87
Figure 5.4 Timing report of Xilinx Synthesis Tool for the 16-bit adder 87
Figure 5.5 Flip-flop circuit for interconnect characterization 89
Figure 5.6 Placed and Routed circuit for interconnect characterization and net delay report 90
Figure 5.7 Cases for detection of LUTs-FFs packing 92
Figure 5.8 Algorithm for area estimation of data-path 94
Figure 5.9 Model of a delay path 95
Figure 5.10 Estimation model for Delay-path with Non-pipelined CFUs 96
Figure 5.11 Estimation model for Delay path with pipelined CFUs 96
Figure 5.12 Placement of components with fan-out effect 97
Figure 5.13 3 samples of dummy circuits for evaluating the area estimation 99
Figure 5.14 Area estimation results 100
Figure 5.15 Errors compared to Post-PAR delay of Simple model and Synthesis Tool (Virtex2Pro-6) 103
Figure 5.16 Errors compared to Post-PAR delay of Simple model and Synthesis Tool (Spartan3-4) 104
Figure 6.1 Estimation error (Predictability) versus λ 109
Figure 6.2 Estimation errors for Virtex2Pro Family 111
Figure 6.3 Estimation error (Predictability) versus λ for Spartan3-4 FPGA 111
Figure 6.4 Estimation errors for Spartan3-4 family 113
Figure 6.5 Model for delay estimation using simplied floorplanning 114
Figure 6.6 Sample circuit and simplified representation 116
Figure 6.7 Regular spaced grid for logic group placement 116
Figure 6.8 Simplified Floorplanning Procedure 117
Figure 6.9 Delay estimation errors for experiments with Virtex2pro-6 118
Figure 6.10 Delay estimation errors for experiments with Spartan3-4 119
Figure 6.11 Comparison of 3 techniques for Virtex2Pro 121
Figure 6.12 Comparison of 3 techniques for Spartan3 121
Figure 7.1 Transfer functions for bit-width analysis (source: [51]) 125
Figure 7.2 Illustration of the concept of constant propagation 125
Figure 7.3 Bit-Information Inferrence Procedure 127
Figure 7.5 Altera Full-adder 129
Figure 7.4 Xilinx Full-adder 129
Figure 7.6 Implementation an adder on FPGA 129
Figure 7.7 Partial products and adder tree 131
Figure 7.8 2-stage adders 132
Figure 7.9 Summing Partial Product for multiplication 133
Figure 7.10 Implementation of large multiplier with multiple embedded multipliers 133
Figure 7.11 Comparison of actual and estimated values for Multipliers on the Xilinx device 135
Figure 7.12 Comparison of actual and estimated values for Multiplier on Altera 135
Figure 7.13 Equal comparator circuit 136

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Figure 7.14 Equal comparator circuit 138
Figure 7.15 Greater than comparator circuit 141
Figure 7.16 Mapping of 4-to-1 multiplexer on Altera FPGA LUT 143
Figure 7.17 Implementation of Multiplexers and area estimation algorithm 144
Figure 8.1 Comparing proposed approach to a common design flow 147
Figure 8.2 Proposed framework implemented in CRATE 148
Figure 8.3 Example of IR output 149
Figure 8.4 CRATE's report 152
Figure 8.5 Illustration of IO register packing 154
Figure 8.6 Illustration of fidelity 155
Figure 8.7 Area estimation for Virtex2Pro-6 156
Figure 8.8 Area estimation for Spartan3-4 156
Figure 8.9 Area estimation for Virtex4LX-12 158
Figure 8.10 Comparison of SLICEs before and after bit-width optimization 160
Figure 8.11 Comparison of LUTs before and after bit-width optimization 160
Figure 8.12 Delay estimation for Virtex2Pro-6 161
Figure 8.13 Delay estimation for Spartan3-4 161
Figure 8.14 Distribution of Embedded Multipliers on Virtex and Spartan 162
Figure 8.15 Delay estimation for Virtex4LX-12 163
Figure 8.16 Area estimation for Altera CycloneII 170
Figure 8.17 Delay estimation for Altera CycloneII 171
Figure 8.18 Comparison on Area-time Product of CRATE's RTL and C2H's RTL 175

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

LISTS OF TABLES

Table 3-1 Control Information Table (CIT) 53
Table 4-1 Different encoding schemes 61
Table 4-2 An example of a large FSM 63
Table 4-3 An example of Control Information Table of a Multiplexer 69
Table 4-4 Next-state decoder 70
Table 4-5 Area-Time Estimation of 10 controllers for Xilinx Spartan3-4 82
Table 5-1 Area-Time Information Of Rtl Components For Different Fpga Families 88
Table 5-2 Characterized Interconnect Delay for Three Xilinx FPGAs 91
Table 5-3 Experimental Results for Area estimation 100
Table 5-4 Embedded Functions used in Experiment 102
Table 5-5 experimental results for estimation of critical path using simple model 103
Table 6-1 Experiments result with Virtex2pro family 110
Table 6-2 Estimation results for Spartan3-4 Family 112
Table 6-3 Delay estimation results FOR VIRTEX II p-6 Family (XC2VP70-6FF1517) 120
Table 6-4 Delay estimation results FOR SPARTAN3-4 Family 120

Table 7-1 Bit Information Inference Rule 126
Table 7-2 Area-time of adders for Xilinx Spartan3-4 130
Table 7-3 Area-time of adders for Altera Cyclone II C8 130
Table 7-4 Estimation of Embedded Multipliers for Xilinx Spartan3-4 133
Table 7-5 EstImation of embedded multipliers for altera cycloneii-8 134
Table 7-6 Estimation of Equal comparator fir Altera Cyclone II-8 138
Table 7-7 Estimation of equal comparator for Xilinx Spartan3-4 139
Table 7-8 Estimation of greater than comparator for Altera cyclone II-8 141
Table 7-9 Estimation of greater than comparator for Xilinx Spartan 3-4 141
Table 7-10 Example multiplexer area-time information stored in database (xilinx Spartan3-4) 142
Table 8-1 Embedded Functions used in final experiments 153
Table 8-2 Area estimation with and without control path 159
Table 8-3 Effect of Floor-planning process 164
Table 8-4 Comparison on Estimation time and execution time 173
Table 8-5 Comparison between C2H's RTL and CRATE's RTL 175

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 1 INTRODUCTION

This thesis addresses the problem of high-level area-time estimation of Field Programmable Gate Arrays (FPGAs) in high-level SoC design flow. The term *area* refers to hardware resources on FPGA such as Slices, Look-Up Tables (LUTs), Flip-flops (FFs) and embedded multipliers that are required to perform computations of an application. *Time* refers to the maximum clock speed at which the application can run on the FPGA. The urgent need for efficient high-level FPGA estimation is driven by two factors: 1) the emergence of heterogeneous System-on-a-Chip (SoC) platforms that exploits the flexibility of reconfigurable logic and 2) the migration of Electronic Design Automation (EDA) tools towards system-level design to handle the growing complexity of embedded systems.

After almost two decades of technology evolution, FPGAs have now become an attractive solution to meet the technological and market challenges in embedded processing [1]. Increasingly, designs are migrating from the traditional microprocessor-ASIC (Application-Specific Integrated Circuit) based platforms to SoC which integrate single or multiple microprocessor cores with application specific and/or reconfigurable logic (e.g. FPGA) [1][2][3][4]. This trend is driven by both economic factors (i.e. stringent TTM (Time-To-Market) pressure and market uncertainties) and technological advances (i.e. fabrication process and EDA tools). Recent FPGA architectures have also been coined as "System on a Reconfigurable Chip" with the incorporation of a large variety of hard-IP cores in the devices. Examples of such embedded hard-IP cores include the PowerPC and DSP Extreme Blocks in the Xilinx devices [5]. With such a wide variety of processing options, the design space has grown unmanageable for traditional design methodologies. This has led to the advancement of system-level design methodologies that will play a central role in the design success of current and future embedded products.

The key task for system-level design is to identify the best architecture configuration that meets the conflicting design constraints (cost, performance, power, time) of embedded systems. An architecture configuration is defined as a combination of hardware-software (HW-SW) components which function together to perform the desired tasks. In order to identify suitable candidates for hardware realization,

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

designers must be able to efficiently estimate the quantitative outcome of mapping the application portions to hardware or software. These estimates should include cost (e.g. Slices/LUTs/FFs/gates/μm^2 for HW or memory space for SW) and performance (e.g. minimum clock period or clock cycles) measures. In addition, hardware estimation must be performed on the high-level representation of the application without undergoing time-consuming RTL synthesis and implementation. In the remainder of this report, *high-level applications* will refer to applications that are specified using high-level programming languages (e.g. C, Matlab, etc.), and *high-level estimation* refers to the hardware estimation of high-level applications.

While high-level estimation methodologies for ASIC design are well established [6] and have been employed in almost all ASIC-based EDA tools (e.g. Catapult C [7] and Synfora [8]), high level estimation of FPGA-based implementation remains a challenging research area. Till date, the most widely used FPGA-based EDA tools (e.g. Xilinx [9] and Altera [10]) do not incorporate high-level estimation. The major difficulty in achieving good results in FPGA-based high-level estimation is due to 1) the fine-grained programmable logic blocks of commercial FPGAs that incorporate diverse processing elements (Look-up-table, Flip-flops, XOR gates), and 2) the unpredictable routing delay. Related works in the academia often ignore the physical implementation effects during high-level estimation, which leads to unreliable results. The challenges in FPGA-based high-level estimation and the shortcomings of existing research work in this area serve as motivations for this project.

1.1. Research Objectives

This project aims to devise an efficient high-level area-time estimation strategy to facilitate effective design space exploration of FPGA-based implementations. The outcome of this project will be a tool that can assist the design space exploration process to evaluate the area-time trade-offs of an application. It is envisaged that this tool can be used to provide effective hardware-software partitioning decisions for current and future heterogeneous SoC platforms that are equipped with FPGAs.

The proposed high-level estimation strategy should possess the following benefits:

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

- **Reliability**: The reliability of estimation will be quantified in terms of accuracy and fidelity. Accuracy is a measure of how close the estimation is compared to actual value. Fidelity refers to the consistency of estimation for different benchmarks.
- **Scalability**: The quality of the estimation results must be maintained for applications of different sizes and complexity. The applications used in the experiments must therefore consist of applications with diverse characteristics.
- **Fast**: The estimation must have a short runtime for rapid exploration of large (few hundreds of design points) design space. In particular, the proposed method should be several orders of magnitude faster than the implementation runtime of commercially available FPGA tools.
- **Portability**: The proposed method must provide for an easy extension to different FPGA devices with minimum modification. In particular, the portability of the proposed method should be demonstrated on widely used state-of-the-art FPGA architectures from different FPGA vendors (i.e. Xilinx and Altera).

1.2. Research Overview

We have successfully developed a complete frame-work Figure 1.1 for high-level area-time estimation of C-functions on the Linux platform. The tool (**CRATE** – CHiPES Rapid Area Time Estimator), which has been developed using C++ and Tcl/Tk GUI programming language accepts ANSI C functions as inputs, invokes a compiler to generate the Intermediate Representation (IR) of the functions, performs area-time estimation of the IR, and reports the results for analysis. ANSI C language has been chosen as the input for high-level estimation because it is widely used in embedded applications. CRATE utilizes the open source Trimaran compiler infrastructure [11] to generate the IR for high level area-time estimation. Trimaran is chosen as the front-end compiler in the framework as it is widely used in the research community, incorporates advanced techniques for code optimization and scheduling, and is well-documented. An architecture template that is similar to the one proposed in [12] has been used. In order to map the high level application to the architecture

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

template, the Trimaran Machine Library is augmented with heterogeneous functional units for FPGA implementation. Although the framework comprises of various stages as depicted in Figure 1.1, this thesis focuses mainly on area-time estimation of the control path and data path. These are the most important tasks in the frame-work that differentiate CRATE from the existing works.

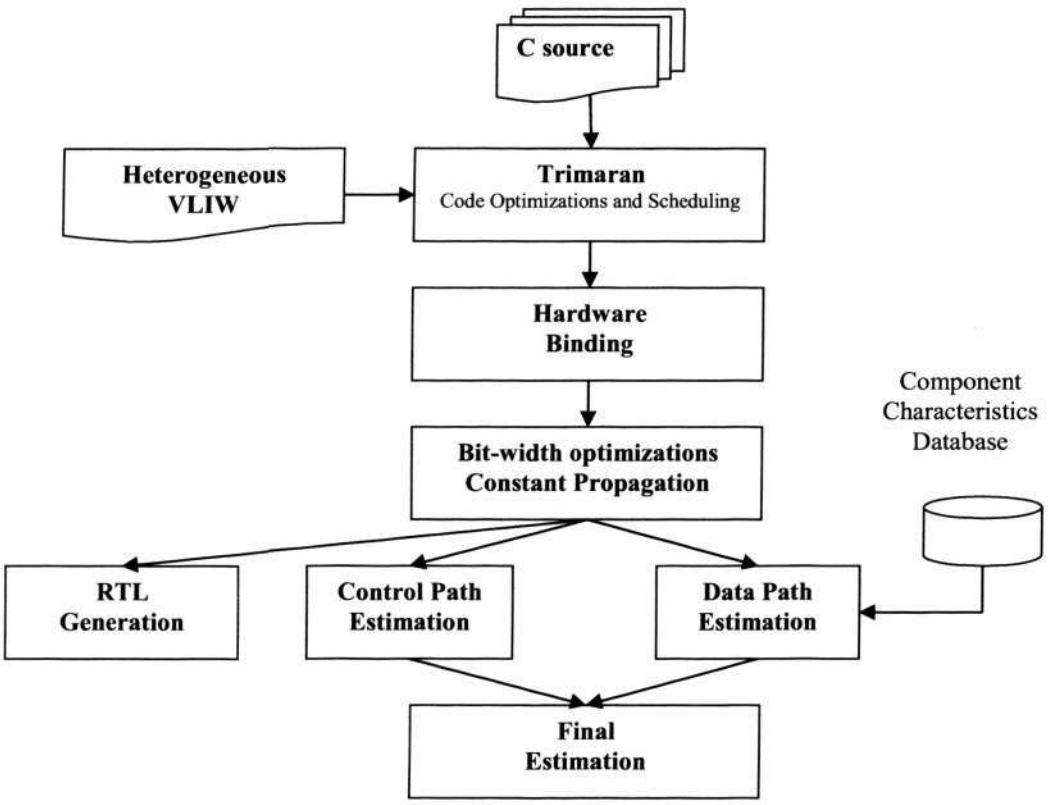


Figure 1.1 CRATE Framework

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

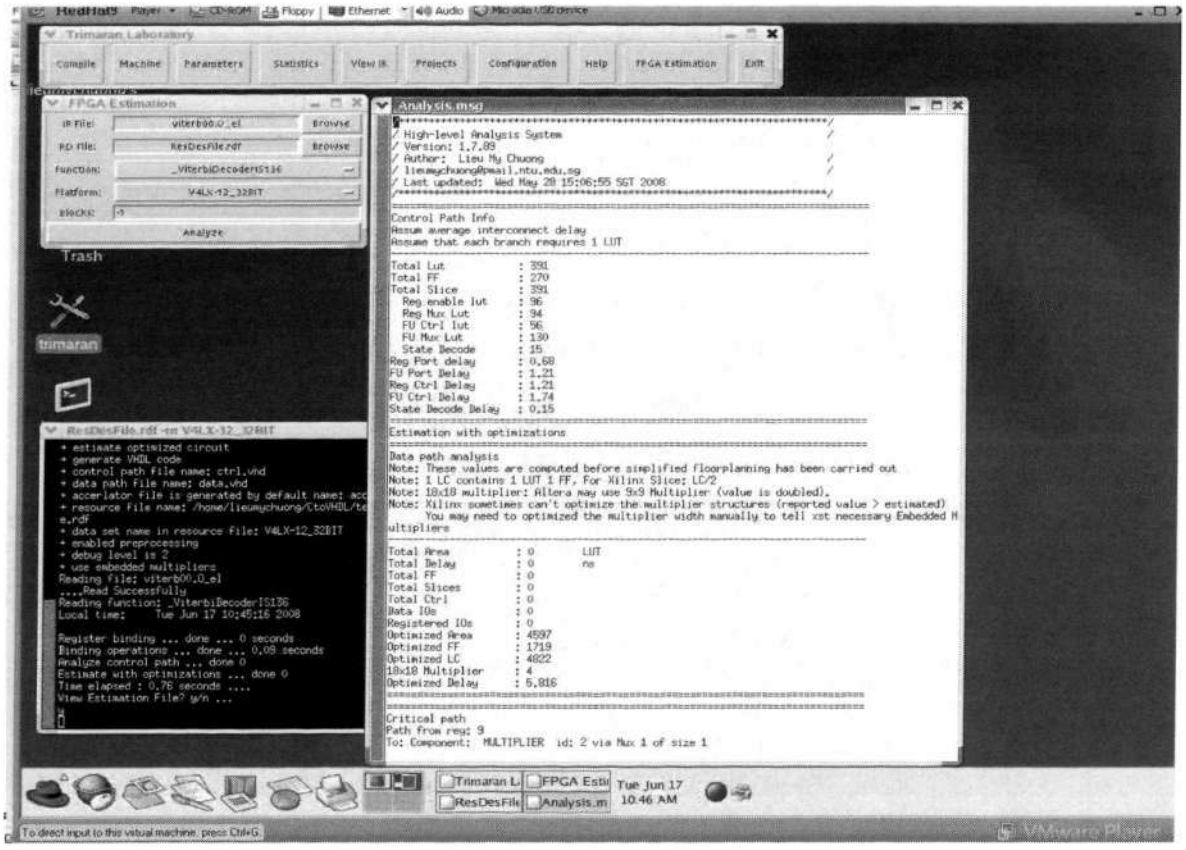


Figure 1.2 Snapshot of CRATE which has been integrated into Trimaran's GUI

The main contribution of this project is a complete framework for high-level FPGA estimation that incorporates the following features:

- **Area Estimation:** CRATE performs reliable FPGA area estimation by taking into account data-width and resource sharing optimizations of the high-level functions. Unlike previous work that often report the estimated area measures as number of LUTs (Look-Up Tables) and FFs (Flip Flops), the proposed method reports the estimated area in terms of number of Slices, which is a more realistic measure of FPGA area utilization. The estimated Slice usage of the proposed method can achieve an average accuracy of 93% and 80% for medium sized functions and large functions respectively.
- **Delay Estimation:** The proposed delay estimation technique incorporates a basic delay model that relies on a set of pre-characterized parameters of the hardware components in the architecture template. In addition, two

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

methods for interconnect delay estimation has been proposed. The first method takes into account the slack distribution of the data-paths. The second method performs a simplified floor-planning process to get a quick approximation of the placement distance between the components. In contrast to existing works, the proposed methods for delay estimation take into account the post place-and-route delays in the estimation process. These novel techniques have been accepted for publication in [13][14]. The proposed method produces results that are within 10% of actual post-place and route values. These results are comparable to the synthesis results of a commercial FPGA tool and can be achieved by the proposed method in significantly shorter amount of time (about 3000 times).

- **Controller's Effect on Final Circuit**: The proposed area-time estimation method first segregates the high-level functions into control and data-path portions to be mapped onto the architecture template. The one-hot encoding scheme is adopted for control path estimation. As circuits for data-intensive processing tend to have large sequential state-machines, the one hot encoding scheme will be suitable for reducing the state encoding logic and improve performance. The area-time measures for the control and data-path are estimated separately and the results are combined to produce the final area-time results.
- **Device Independence:** The proposed method can be easily extended for high-level area-time estimation across different FPGA architectures. In particular, the proposed method has been evaluated on FPGA devices from Xilinx and Altera vendors. The portability of the proposed method has been demonstrated with consistent estimation results obtained for the different devices.
- **Automated HDL Generator:** The proposed framework automatically generates the Hardware Description Language (HDL) of the high-level C-functions. When compared to the Altera C2H tool [10] the HDL generated by the proposed framework leads to higher performance in almost all cases with comparable area utilization. In addition, the proposed hardware generation process is significantly faster than the C2H tool.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

1.3. Thesis Organization

The remainder of the report is organized as follows: Chapter 2 first describes the challenges and trends in embedded system design, which has led to the emergence of reconfigurable systems and high-level design methodologies. A review of existing work in high level estimation of FPGAs is then presented to highlight their limitations. In Chapter 3, we discuss the motivation for adopting Trimaran in the proposed framework, and the modifications made to the compiler infrastructure to enable efficient high-level estimation. We also present the architecture template that has been adopted for high level estimation. Strategies for mapping the high-level application onto the architecture template will also be discussed. We proceed to describe the proposed area-time estimation strategy for the control path in Chapter 4. In particular, we describe methods for constructing the Finite State Machine (FSM) based on the one-hot encoding scheme from the high-level application. The proposed data-path area-time estimation strategies will be presented in Chapter 5 and Chapter 6. In Chapter 5, we present an approach for estimating the data-path which relies on pre-characterized area-time parameters of a set of commonly used components. We define this approach as *static delay estimation* as it does not predict the post implementation interconnect delay of FPGA implementation. In Chapter 6, we introduce two methods for *dynamic delay estimation* to estimate the post-implementation delay. The first method relies on the pre-characterized area-time parameters and an analytical model that is based on slack distribution. The second method, which has been incorporated in CRATE, relies on the pre-characterized area-time parameters and a simplified floor-planning procedure. Chapter 7 presents bit-width optimization strategies that have been incorporated into the proposed high level estimation techniques to achieve better estimation accuracy. We have presented bit-width aware estimation models for commonly used hardware components such as adders, multipliers, comparators and multiplexers for the Xilinx and Altera devices. Chapter 8 presents experimental results to demonstrate the reliability, speed, portability and scalability of the proposed estimation techniques. Finally, Chapter 9 provides the conclusion and future work of this thesis.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 2 BACKGROUND & LITERATURE REVIEW

This chapter begins by describing the challenges and trends in embedded system design, which has led to the emergence of reconfigurable systems that integrates microprocessors and FPGA on the same chip. We will show a general co-design flow for reconfigurable systems to highlight the important stages such as high-level estimation, automatic RTL code generation and low-level FPGA estimation.

A review of existing work in high level estimation of FPGAs is then presented. One of the major limitations in the existing works is that the post implementation effects (i.e. post-place and route area and delay) are often not considered in high level estimation leading to unreliable estimation results. We then discuss a range of commercial tools that can automatically generate RTL codes from the high-level design specifications. It is noteworthy that almost all of these commercial tools are incapable of performing high-level estimation. This chapter ends with a discussion on existing works in low level FPGA estimation, which attempted to estimate the post-place and route area and delay from the input gate-level net-list. This final section aims to provide some insights on how the design characteristics (e.g. size, shape, fan-in and fan-out) affect the post-place and route delay. This has led to the development of a more reliable high-level estimation strategy in this thesis.

The proposed approach in this thesis bridges the two research areas in high-level and low-level estimation to enable reliable high-level area-time FPGA estimation that takes into account the post implementation effects, in order to facilitate efficient design exploration of reconfigurable systems.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

2.1. High-Performance Embedded System Design Challenges and Trends

This section describes the evolution of embedded product development to highlight the crucial role of high-level estimation in modern design methodologies. A definition of embedded system will first be provided, and this is followed by a description of embedded system design considerations. Next, emerging design methodologies for embedded systems will be discussed.

2.1.1. *Embedded systems*

The following is an informal definition of embedded system that was extracted from Wikipedia: "A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function" [15]. Embedded systems are found in almost all modern electronic devices (e.g. microwave oven, credit card reader, router, broad band modem, etc). One of the main reasons that have led to the wide-spread deployment of embedded systems is the ease of programmability in the underlying architecture. As emerging applications become more complex in order to meet the volatile consumer demands, the employment of dedicated hardwired devices is no longer an economical solution. Instead, microprocessors and microcontrollers are favored as they lead to rapid design, debug and verification. Figure 2.1 depicts the factors affecting embedded system design.

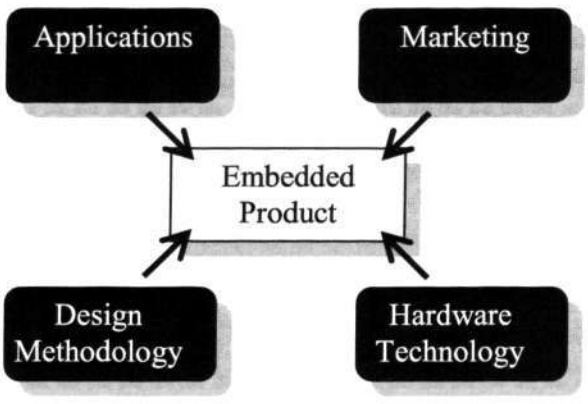


Figure 2.1 Factors that influences the design of embedded systems

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

2.1.2. *Embedded applications*

Embedded applications are becoming increasingly complex. They are no longer restricted to controllers of product lines in factories but are now capable of performing compute-intensive processing tasks such as playing MP3 (e.g. iPod), demodulating FM radio, decoding/encoding MPEG2 images (digital camera), etc. Such applications are often executed on mobile computing devices, which are fast becoming an integral part of our lifestyle. The complexity of emerging applications coupled with the need to deploy them on portable devices has given rise to the stringent non-functional requirements such as performance, area and power. These requirements are often conflicting, hence further increasing the difficulty in the design of embedded systems.

2.1.3. *Market requirements*

In today's globally competitive economy, the product that appears first has a market share advantage. Hence, TTM pressure imposes on the embedded product to be "softer" in order to facilitate easy debugging, verification and upgrade. However, the incorporation of flexible architectures (i.e. microprocessors and/or microcontrollers) leads to a reduction in performance and power efficiency. It is evident that a successful embedded system design methodology must be capable of tackling both the technological and market challenges that are faced by existing and future embedded products.

2.1.4. *Hardware technologies*

Hardware technologies play an important role in the differentiation of embedded products. Hardware technologies refer to the process technology (currently 45nm) and various computing platforms (e.g. ASIC, micro processors (μP), micro-Controller (μC), FPGA, DSP, etc). Figure 2.2 illustrates the characteristic spectrum of various computing platforms.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

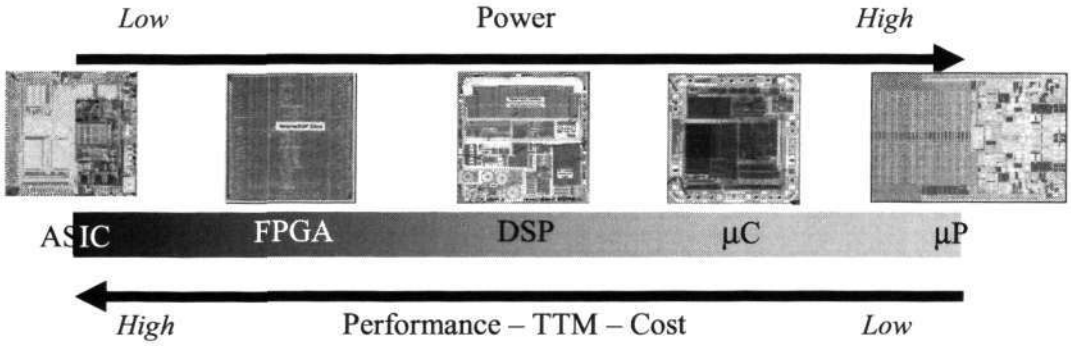


Figure 2.2 Characteristic spectrum of various computing platforms

ASIC (Application Specific Integrated Circuit) refers to a hardwired IC (Integrated Circuit), which performs a dedicated task. The high degree of customization enables ASIC to achieve high performance and power efficiency, at the expense of cost and flexibility. ASIC also suffers from long and risky development time. Hence, while ASIC can sufficiently meet the technological challenges of embedded systems, it is becoming more unpopular due to its economical infeasibility.

DSP (Digital Signal Processors) and µC are domain-specific IC. DSP are commonly employed for data transformation applications while µC are employed in control-intensive applications. In order to increase the computational efficiency, these computing platforms incorporate domain-specific hardware cores. However unlike the µP, the flexibility of these platforms is restricted to their respective domains.

FPGAs are programmable hardware that consists of reconfigurable logic elements, which are interconnected by programmable routing resources [16]. These platforms are capable of meeting the TTM pressure, while outperforming DSP, µC and µP in terms of performance and power efficiency by exploiting concurrency in the applications. The programmability of FPGA coupled with its ability to cater for hardware customization provides a good-trade-off between pure hardware-based and software-based computing systems. To this end, FPGA technology has been gaining attention and increasingly becoming the preferred computing platform in modern high-performance electronic products. The industry is now seeing the emergence of FPGA-based companies that specializes in developing product solutions using high

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

performance and ultra-low power FPGA, for example SiliconBlue [17] which targets the handheld market.

2.1.5. *Design Methodologies*

Gajski in [18] has highlighted that "innovation in design methodologies provide the most plausible way forward to cater to the imposing factors on embedded system design". Gajski reviewed the evolution of Digital IC design methodology from schematic capture, behavioral description to system/architecture level. As design methodology advances, engineers are able to develop more complicated computing platforms as shown in Figure 2.3.

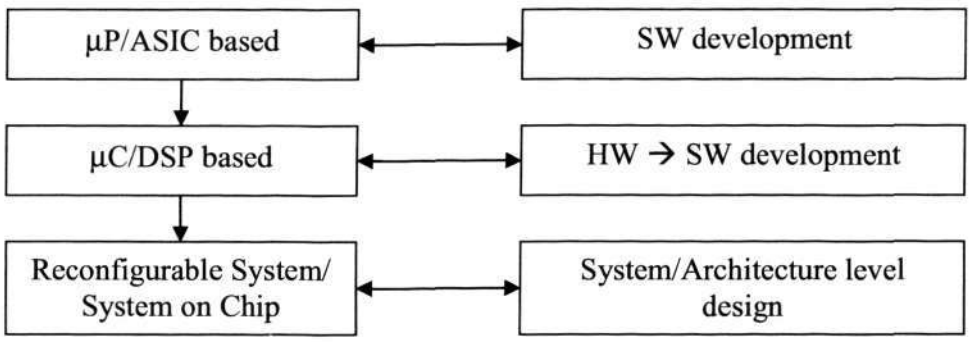


Figure 2.3 Evolution of embedded computing platforms and design methodologies

µP/ASIC-based: This platform comprised of a single microprocessor with ASIC peripherals that communicate with the outside world. System development focuses on software functions that process data and communicate with external devices.

µC-based (DSP-based): The advancement of Digital IC design methodology allows engineers to integrate more components on the same die. Components on these platforms usually consist of a central processor and domain specific components integrated into a single chip. Applications targeted on these platforms are partitioned into software and domain-specific hardware components early in the design cycle. Hardware components consist of either on-chip peripherals or off-chip ASIC. This partitioning is done solely based on the chip-architect's experience [18]. Typically, the hardware components are constructed before the software development process can begin.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

System-on-a-Chip/Reconfigurable System-on-a-Chip: Advancement in fabrication process has enabled the integration of million-gate IP cores onto the same silicon die. This has resulted in larger and more complicated systems that can reside on a single chip, and hence such systems are commonly termed as System-on-a-Chip (SoC). SoCs are usually optimized for an application domain (e.g. automotive, security, telecommunications, etc). However, due to the rapid evolvement of product requirements in emerging embedded applications, SoCs still suffer from high NRE costs and are unable to cope with the tight TTM requirements.

Recent studies [19][20] have reported that combining general-purpose microprocessors and reconfigurable logic on the same computing platform provides the most efficient way of achieving higher performance, flexibility and lower power-consumption at reasonable cost. In line with this, FPGA vendors have recently produced the System-on-Programmable-Chip (SoPC), which integrates high performance FPGA devices with various types of hard-IPs such as PowerPCs, DSP blocks, Ethernet MAC [5], etc on a single die. It is envisaged that the high degree of programmability in SoPC will become the preferred approach for coping with the functional and non-functional requirements of embedded systems [1].

The design methodology for SoPC requires unified hardware and software development, as a module can be implemented on either hardware or software. The embedded industry is driven towards hardware-software co-design that enables the hardware-software implementation trade-offs to be evaluated early in the design cycle. This gives rise to the need for reliable high-level estimation of the application for FPGA implementation. Figure 2.4 shows a general co-design flow with the emphasis on hardware development for reconfigurable system.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

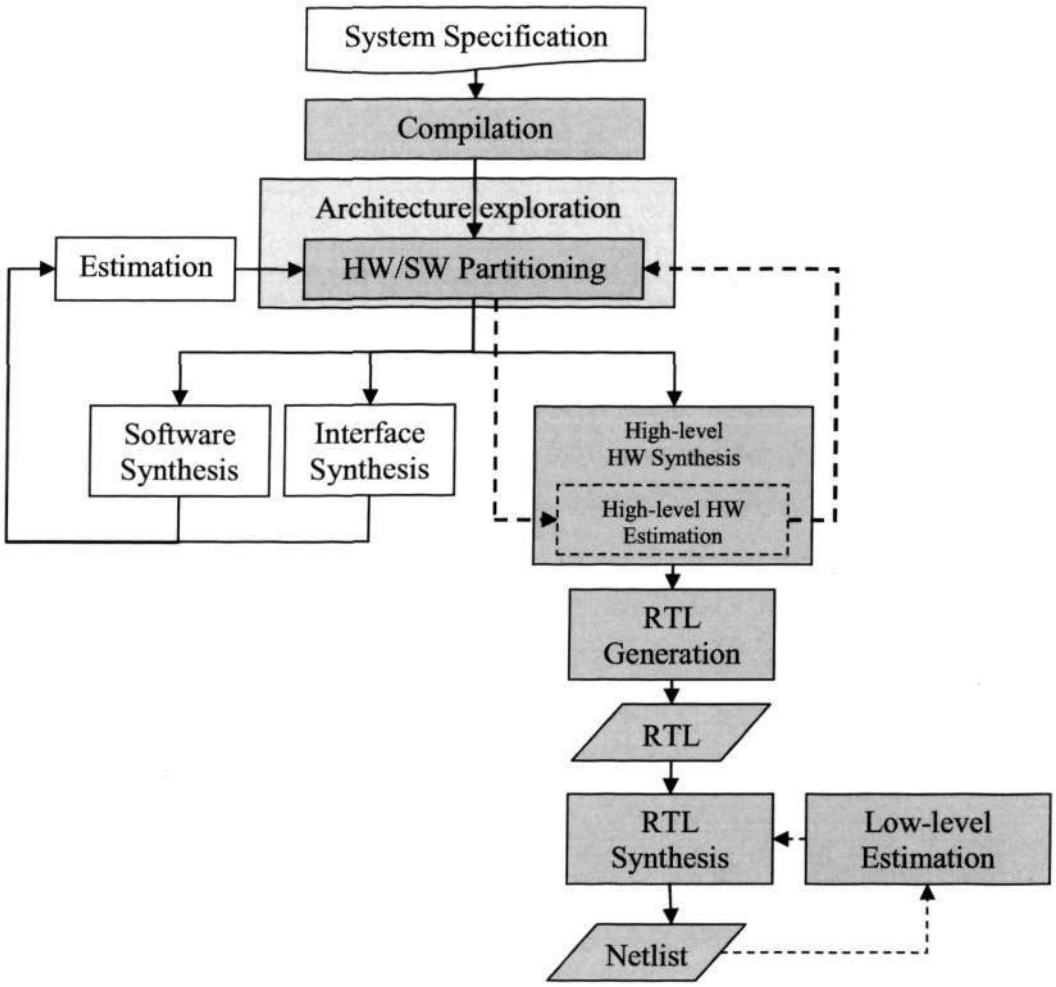


Figure 2.4 General co-design flow with focus on hardware development

The design flow begins with a system specification, which is then compiled and partitioned into an initial set of hardware/software blocks. Subsequent fine-tuning of the hardware/software partitions relies on the high-level estimation results, which computes the projected area-time measures of the hardware implementation without the need to perform time-consuming hardware synthesis. This architecture exploration process, which consists of iterative partitioning and high-level estimation, is used to search for the best architecture configuration that meets system's requirements. When the desired partitioning solution has been achieved, RTL codes are generated either manually or automatically. During RTL synthesis, low-level estimation is commonly employed to guide logic optimization to converge to a good solution.

In the subsequent sections, we will discuss the existing work in the areas of high-level estimation, automated RTL generation, and low-level estimation.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

2.2. High-level Estimation Process

2.2.1. *Background*

High-level estimation aims to predict the area-time measures of mapping an application/function that is specified using high-level language onto FPGA, without performing time-consuming logic synthesis. The estimation is usually performed on the design that has undergone high-level optimizations (during high-level synthesis), which include loop-unrolling, loop-pipelining, function in-lining, etc. High-level estimation results may include high-level area-time measures such as the number of clock-cycles and functional units required, or low-level area-time measures such as the required clock-period and silicon area.

High-level area-time estimation has received considerable interest in the research community for nearly twenty years. Research efforts in this area are motivated by the need to evaluate the hardware performance-cost indices of various design options early in the design phase, in order to reduce time-consuming implementation cycles. Conceptually, the translation of a design from algorithm-level to gate-level encompasses two levels of abstraction. First, un-timed algorithm is translated into a sequence of micro operations. Second, a circuit is realized to perform these sequences of micro-operations. The same flow applies to high-level estimation. The major steps in a typical high-level estimation flow, which are shown in Figure 2.5 include:

a. Transformation of an application/function written in high-level language (C, SystemC, Java, etc.) to an Intermediate Representation (IR) for back-end processing and optimization. This step is usually performed by the compiler in the co-design framework (see Figure 2.4).

b. System-level estimation

c. High-level synthesis

d. Hardware estimation that reports low-level area-time metrics such as clock period, Slice or µm.

The final goal of high-level estimation is to produce a quantitative estimation on the resource requirements and performance for mapping the high-level application to hardware. Estimation can be achieved with or without high-level synthesis using the following flows: 1) a→b→d or 2) a→c→d as depicted in Figure 2.5. Previously reported works either adopt one of these flows or focused on just a single stage.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Currently, there is no tool that incorporates all the steps in the framework shown in Figure 2.5. This thesis takes the a→c→d flow with the emphasis on hardware estimation stage. In subsequent sections, we will discuss existing works in the various stages of Figure 2.5 (i.e. system level estimation, high-level synthesis and hardware estimation).

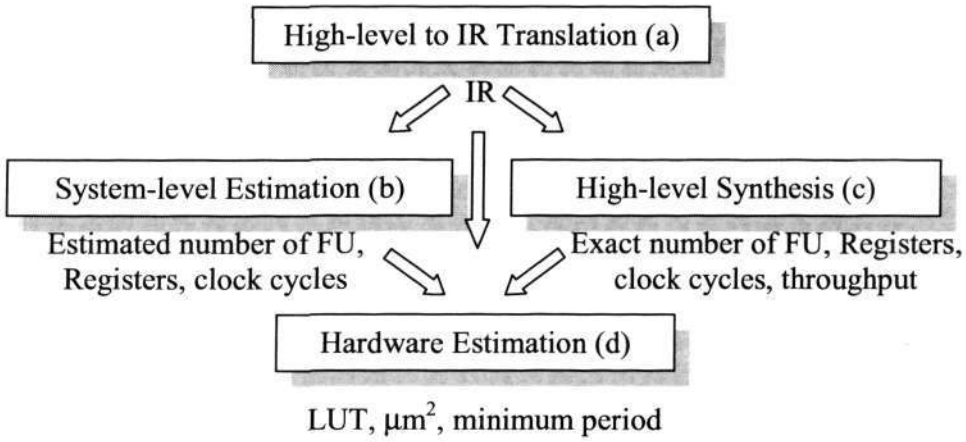


Figure 2.5 Typical high-level estimation flow

2.2.2. *System-level estimation*

System-level estimation attempts to estimate the hardware resources and latency in terms of functional units and clock cycles *without performing scheduling* of the input design. These approaches commonly rely on probabilities or integer linear programming models based on the high-level application characteristics in order to predict the number of functional-units or minimum clock cycles.

The work presented in [21][22][23] perform estimation to obtain a lower-bound execution time of a DFG in the presence of hardware constraints to facilitate efficient design space exploration. Similarly, the method in [24] estimated the minimum number of resources that are required to execute a DFG within a control step constraint. These techniques are useful for pruning the design exploration space. However, they yielded low estimation accuracy, as they did not take into account the physical implementation effects due to routing and the instantiation of components such as multiplexers and buses, which contributes significantly to the final performance and size of circuit.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Nayak et. al. presented a fairly complete estimator in [25]. The authors relied on Matlab codes as inputs to estimate the hardware resources by summing the area of the required operators based on the execution probabilities in the application. Their approach took into consideration data-path, control-path, and bit-width analysis in order to estimate the area-time of operators with different bit-widths. For area estimation, Nayak proposed that the number of Slices could be estimated by C*Max (LUT, FF). The parameter C was derived empirically to take into account the effects of global optimizations on the design and to estimate the number of Slices that were used for routing. However, it remains unclear as to whether the parameter varies with different designs as the authors did not describe the methodology used for recalibrating the parameter. The estimation results of Nayak's technique achieved an average of 90% accuracy with moderate sized benchmarks (around 500 CLBs). The proposed technique in this thesis employs a strategy to predict how the LUTs and FFs are packed onto the Slices. Experimental results show that this method of predicting the distribution of LUTs and FFs can lead to reliable estimation results for a wide range of designs.

The work in [26] presented a high-level area estimation approach for SA-C-based designs. The Single Assignment C (SA-C) (pronounced as "sassy") is a variant of the C programming language that can be used to explicitly specify the hardware design. In [26], the applications that were specified using SA-C were first compiled into DFG. The number of LUTs is then estimated based on analytical models of the arithmetic operators (i.e. nodes in the DFG's). Application scheduling and hardware binding were not considered in this work. The authors showed that their approach could lead to credible area estimation results with an average estimation error of 4%. However, their estimation results did not report on the total Slice utilization, which is a more realistic measure for area estimation. The main contribution of their work was the demonstration that a parameterized mathematical model could be used to estimate the area of RTL operators. However, only a single parameter that defines the operand bit-widths of the RTL operators are used in their mathematical model. This may limit the range of applications for which the model can be applied. For example, the model of a multiplier in [26] cannot be used to effectively estimate the area of a 16x10 bit multiplier. Compared to this work, CRATE has the following advantages: 1) it is capable of reporting the estimation results in terms of Slice count, and 2) the proposed

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

estimation technique in this thesis incorporates more sophisticated mathematical models that can be tuned to estimate the required area of arithmetic blocks in the FPGA with different operand bit-widths.

2.2.3. *High-level Synthesis*

High-level synthesis typically performs scheduling of the IR and resource binding to obtain accurate high-level metrics in terms of the number of clock cycles and hardware resources.

Bilavarn et. al. [27] presented a method that employed high-level synthesis for estimation and design exploration. Their method first transformed the functions (represented in C language) into Hierarchical Control Data Flow Graphs (HCDFGs), which was then used for area-time-power estimation. They considered the estimation of the memory resources, control path, and data-path. However, their approach had several limitations. Firstly, the maximum clock frequency estimation was based solely on the longest latency of the execution unit and ignores post place and route physical effects. Secondly, they employed an oversimplified model for data-path estimation. In addition, the delay from the control logic was not considered, and the effect of logic synthesis optimizations and components with arbitrary bit-width was ignored. These limitations would give rise to inconsistent estimation results. In Chapter 7 of this thesis, we will demonstrate the importance of logic and bit-width optimizations on high-level estimation by showing their impact on the estimation results.

Cardoso proposed a methodology for estimating FPGA implementations of Java byte-codes in [28]. His work was based on a previously reported approach that relied on pre-characterized area-time measure of the components. The paper highlighted the main challenges in interconnect delay estimation is due to the lack of circuit implementation details during high-level synthesis.

Xu and Kurdahi [29] were one of the first who attempted to estimate the interconnect delay during high-level synthesis. In their work, they estimated the number of CLBs and the maximum clock-rate of a net-list, which had been generated by behavioral compiler. In order to compute the total number of CLBs required, the net-list was decomposed into groups of LUTs that could fit into the CLBs. Timing estimation was achieved by examining the CLB and inter-CLB interconnect delays. The Manhattan distance, standard delays of routing segments, and architecture model

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

of the routing resource were used to calculate the interconnect delay between two points. This approach for area-time estimation had several limitations. Firstly, estimation performed at the CLB level is not suitable for high-level estimation as the CLBs do not exhibit the functional characteristics that is necessary for predicting the inter-CLB placement and routing. Secondly, the runtime of the estimation can be undesirable due to the large number of CLBs that need to be considered. Finally, as highlighted in [28], there was no evidence that the method in [29] can be applied to a different FPGA family.

Per Bjureus et. al. in [30] proposed an estimation technique based on simulation to estimate the area and latency of an FPGA implementation from Matlab specification. The authors extracted the execution information from the Matlab simulation model to construct a DFG. The DFG was then scheduled and mapped onto the hardware resources. Next, estimation is performed based on a set of pre-characterized component area-time measures. In general, this work achieved reasonable result with an average estimation error of 10%.

2.2.4. *Hardware estimation*

Hardware estimation techniques commonly employ simple models to accelerate the estimation process. The hardware cost is estimated in terms of LUTs (Look-Up Tables), Flip-flops (FFs) or Slice/Logic Cell, while the performance is often estimated in terms of minimum clock period. It is noteworthy that the estimation of the number of Slices/Logic Cell is more desirable as it gives a more realistic indication of the FPGA area utilization.

In Nayak et. al. [25], the Rent Rule and Feuer's formula and the number of estimated CLBs were employed to estimate the post place and route interconnect delay. This approach led to a large uncertainty of up to 9 ns [25]. This high uncertainty can become unacceptable for designs that need to be clocked at high frequency (e.g. 100 MHz). This uncertainty arose from the fact that the method used for delay estimation relied on the number of CLBs, which is a poor indication of inter-register delays that are pertinent in data-paths comprising of both combinatorial elements and storage elements.

Carlo Brandolese [31] proposed area estimation of applications specified using SystemC. Their method (shown in Figure 2.6) models the low-level metrics (LUTs

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

and FFs) as a function of high-level variables (control steps, codes size, etc.) using High-level Parameters (HLP) and Low-Level Parameters (LLP). These parameters are determined using a large set of test cases. Their method reported a high estimation error of up to 37% for a design with 5000 LUTs.

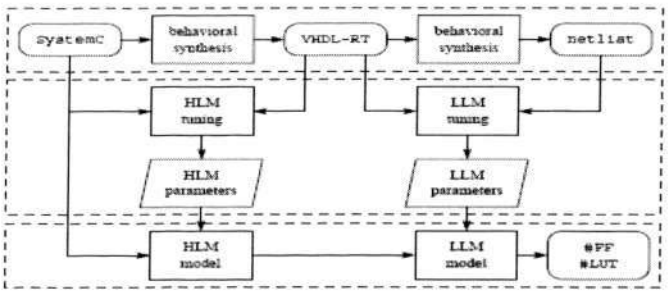


Figure 2.6 Two-level methodology of Carlo

The work in [32] focused on estimation of the control path. Carsten Menn estimated the area of the controller based on the number of control states, number of inputs-output functions and state-transitions. The method worked well with the benchmarks that were considered. Menn highlighted that employing one-hot encoding for the controller can lead to significant reduction in the decoding complexity. However, the work only considered area estimation and extensive modification of the technique is necessary to facilitate delay estimation. This thesis presents a technique to estimate both the area and delay of the control path in Chapter 5.

In order to achieve reliable high-level estimation, there is a need to predict area-time of RTL components based on their required input-output bit-widths. There are two approaches for solving this problem: 1) Constructing area-time models as a function of the inputs-output bit-widths; 2) Maintaining a database that stores area-time information of each component with varying bit-widths. The first approach can lead to high estimation inaccuracies due to the non-linear relationship between area-time and bit-widths, whereas the second approach requires a large database to store all possible configurations of the RTL components. The work in [33] adopts the first approach and focused on developing an area-power model for fixed-point and floating-point RTL components in the FPGA. They have employed a curve-fitting method to derive the analytical models for each component. The reported area estimation error of the technique in [33] for fixed-point and floating point operators is 4%. The proposed framework in this thesis incorporates a hybrid strategy to estimate area-time of commonly used arithmetic operators, i.e. comparators, adder/subtractor, multiplier,

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

barrel shifter, and constant shifter from their input-output bit-widths. The hybrid strategy employs both analytical models and pre-characterized values to achieve accurate estimation for RTL components while keeping the size of the database manageable. The analytical models have been constructed by studying how the synthesis tool maps the components onto the FPGA fabrics. All the parameters used in the proposed method can be directly obtained from the FPGA data-sheet, timing reports or can be derived from other parameters and pre-characterized components in the database.

2.3. Commercial Tools for Automatic RTL Generation

As mentioned above, RTL or hardware generation from high-level codes are becoming increasingly important in ESL design. Due to the need to expedite the development of complex applications in hardware, a number of commercial tools that synthesize high-level languages to FPGA have emerged in recent years. These tools differ in several aspects such as high-level language support, high-level optimization features and the target system. For example, Mitrion-C from Mitronics [34], HandleC from Celoxica [35], and Impulse-C from Impulse Accelerated Technologies [36] support a subset of the ANSI-C language that is extended with constructs for specifying the hardware definitions. These tools cannot be directly employed for most embedded applications that are programmed using ANSI-C. As executable specification is necessary for early simulation and verification, any extension to the original standard programming languages is not preferred. C2H tool from Altera [10], Catapult from Mentor Graphics [7], and Trident [37] support pure ANSI-C applications. Although all the tools discussed above can efficiently convert C-level algorithm into gates, they do not focus on providing accurate high-level estimations for hardware-software partitioning of FPGA-based system. The Celoxica tool is able to provide high-level estimation of the Handle-C specifications. However, designers have to explicitly specify timing information using *par* and *seq* constructs in the Handle-C specification [38]. This restricts the use of Handel-C as high-level system specification. Recently, Synfora released PICO Express FPGA in February 2008 [8], which is claimed to be able to generate hand-crafted-quality RTL codes from un-timed C specification and perform high-level design exploration. This is the only ESL design tool targeting FPGA that supports rapid estimation and design exploration. The newly

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

launched software tool justifies the importance of the research topic in this thesis work.

2.4. Low-level estimation for logic synthesis and place and route

The logic synthesis and place-and-route process often rely on low-level delay estimation to progressively improve the optimization results. This has been the main motivation for interconnect delay estimation in the FPGA research community. Existing low level estimation approaches cannot be directly adopted for high-level estimation as they are usually performed on the input gate-level net-list, which do not incorporate the high-level details such as CLBs, LUTs and FFs. However, these low-level estimation approaches provide valuable insights on how design characteristics affect the final circuit delay.

The work presented in [29] could achieve very accurate estimations of the interconnect delay by analyzing the physical characteristics of the designs. However, due to the complexity of the approach, the estimation results were achieved in the order of seconds and minutes. Karnik and Kang [39] presented an empirical routing delay model for estimating interconnection delays in FPGA. These methods require low-level metrics of circuit such as net fan-out and routing congestion, which is not desirable for efficient high-level estimation. Their method resulted in an estimated delay with 20% error. Hutton highlighted that that delay estimation based on theoretical models, generally produces inferior results when compared to those computed based on empirical data [40]. Manohararaja et. al. reported an interesting finding that the predictability of FPGA implementation is mainly governed by the placement rather than routing process [41].

The physical characteristics that has been commonly used for post place and route delay estimation includes: 1) Design size [39][25], 2) Circuit shape [42], and 3) Fan-in/out [39]. The placement and routing effort is mainly influenced by the interconnectivity in the designs rather than the size or shape of the design due to the fine-grained architecture of FPGA. As the final circuit is a set of connected Slices which contain both combinatorial and storage elements, the size of design have strong correlation to the interconnect delay but not the critical path delay which is measured between flip-flops. Hence, the first two characteristics are not reliable indicators for post place and route estimation. The fan-in/out of a register is defined as the number of input/output connections of that register. Compared to the first two characteristics, the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

fan-in/out provides a better indication on the interconnect complexity of the design. Fan-in/fan-out has two effects on the interconnect delay: 1) effect on the capacitance of the routing fabric and 2) effect on the difficulty in placing the components on the FPGA. Taghavi et. al. [43] has demonstrated that the correlation between net fan-out, driver/loader distance and delay is unpredictable. To the best of our knowledge, the authors have set-up experiments to investigate only the first effect. The second effect is mitigated by register duplication during physical synthesis. Our experiments show that although register-duplication can lead to improved timing in some cases, it could also increase the routing congestion of the circuit. This was inferred in our experiments for some designs, where the final delay after register duplication is higher than the delay obtained from implementations that obviates register duplication. Due to this, there is a need for an accurate model to predict effect of the fan-out on the interconnect delay during high-level estimation. In Chapter 6 of this thesis, a technique to model the effect of fan-out on the interconnect delay during high-level estimation will be discussed.

2.5. Summary

We have discussed various factors that shaped the evolution of embedded system and embedded system design methodology. Economical and technological pressures have been driving embedded system design methodologies towards system level hardware-software co-design. At the same time, reconfigurable systems incorporating microprocessors and FPGAs have demonstrated to be a good solution for meeting the conflicting market demands.

These two factors give rise to the need for high-level design frameworks that target reconfigurable systems such as that shown in Figure 2.4. In the framework, the system specification is compiled and partitioned into an initial set of hardware/software blocks. The high-level estimation tool is responsible for providing reliable area-time measures of the hardware implementation without the need to perform time-consuming hardware synthesis. This facilitates rapid exploration of a large design space to make effective partitioning decisions. In particular, the design exploration process, which consists of iterative partitioning and high-level estimation, is used to search for the best architecture configuration that meets the system's requirements. When the desired partitioning solution has been determined, RTL codes are automatically generated.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

High-level hardware estimation is an essential step in the framework to facilitate rapid architecture exploration for hardware-software partitioning. The main limitations of existing works on high-level estimation are listed below. Table 2.1 provides a summary to compare the existing works with CRATE.

1. Existing estimation models do not take into account the post implementation interconnect delay. Such delay may contribute up to 60% of total delay of critical paths. Only Nayak [25] addressed this issue but their approach led to large uncertainties and was not suitable for designs that need to be clocked at high frequency.
2. Estimation of the resource consumption is commonly reported in terms of LUT although the number of Slices is a more realistic indication of FPGA resource utilization.
3. Existing works often neglect the effects of logic synthesis when translating the high-level input specification to an intermediate form for estimation. This lead to high inaccuracies when the estimation results are compared with results from commercial logic synthesis tools.
4. The area and delay of the control path is often not considered during estimation although this may constitute significantly to the final system's area-time measures.
5. Existing work do not consider the estimation of the utilization of embedded multipliers which are common in current FPGA architectures.
6. Finally, the existing works have not demonstrated the portability of their approach on different FPGA families.

The need for a tool that can aid designers in estimating the hardware area-time early in design cycle and the limitations of existing research works form the motivation for us to develop an efficient area-time estimation framework. In particular, this project aims to devise a high-level estimation framework that targets reconfigurable system to facilitate rapid architecture exploration for hardware-software partitioning. The outcome of the project is a tool that overcomes the limitations of previous works mentioned above. In addition, the tool should possess the following capabilities as discussed in Section 1.1: 1) Reliable, 2) Scalable, 3) Fast

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

and 4) Portable. An area-time analysis tool with such capabilities will be extremely useful in the high-level design framework shown in Figure 2.4.

TABLE 2.1 COMPARISON OF EXISTING WORKS IN HIGH-LEVEL ESTIMATION

				Data Path Estimation					Control Path Estimation	
Authors & Works	**Inputs**	**Outputs (Area/ Time)**	**Target Family**	**Prediction of Logic Optimization Effects**	**Estimation of on-chip Multiplier**	**Area estimation strategy**		**Post Implementation Delay Estimation**	**Encoding Scheme**	**Incorporate Delay of Controller in Total Delay**
						High-level metrics	**Resource Est. Method**			
Nayak et al [25]	Matlab	CLB/ns	Xilinx	Bit-width analysis	No	Estimated	Analytical model	Fueur formular and Rent Rule	Unknown	No
Bilavarn et al [27]	C	SLICE/ns	Xilinx	No	Yes	Exact	Pre-characterized component library	No	Unknown	No
Kulkarni [26]	SA-C	LUT	Xilinx	Bit-width analysis	No	Exact	Analytical model	No	n.a	n.a
Min Xu [29]	LUT netlist	CLB/ns	Xilinx	n.a	n.a	n.a	n.a	CLB-to-CLB analysis + size + shape	n.a	n.a
Carstenn [32]	RTL	LUT	Xilinx	n.a	n.a	n.a	n.a	n.a	Binary	n.a
CRATE (Proposed method in this thesis)	C	SLICE/ns	Xilinx & Altera	Bit-width analysis; constant propagation	Yes	Exact	Pre-characterized component library and analytical model (no calibration required)	1) slack distribution 2) simplified floor-planning	One-hot	Yes

n.a: not within scope of the work.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 3 COMPILER ASSISTED APPROACH FOR EXTRACTING INHERENT PARALLELISM

The proposed high-level estimation framework (CRATE) relies on an existing compiler infrastructure (i.e. Trimaran) to facilitate the translation of high-level ANSI-C functions into suitable IR (Intermediate Representation) for hardware estimation. The focus of this chapter is to discuss the motivation for adopting Trimaran in the proposed framework, and the modifications made to the compiler infrastructure to enable efficient high-level estimation.

This chapter begins by discussing general design considerations for implementing efficient Hardware Accelerators (HWA). We show that parallel processing and Custom Functional Units (CFUs) are the key factors for efficient design of HWA. In order to overcome TTM challenges, we envisioned that future design methodologies would be able to automate the design of HWA by extracting inherent parallelism and identifying suitable CFUs from high-level specifications of the application. Hence, we have adopted Trimaran as the front-end compiler of CRATE as it includes a wide-range of high-level optimization features that leads to efficient hardware realization. In addition, Trimaran enables us to modify the underlying infrastructure to facilitate design exploration of HWA, and to construct an architecture template for high-level estimation. We will provide a detailed description of the proposed architecture template and its corresponding data-path and control-path RTL models. Finally, we present a strategy to map Trimaran's IR onto the proposed architecture template to facilitate high-level area-time estimation.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3.1. Design Approaches and Challenges of HWAs

Incorporating HWAs in embedded systems can lead to performance improvements by many folds [19]. HWAs exist in various forms: special co-processor with domain-specific instruction set (e.g. the Graphic Acceleration Processor) or special IP cores that resides on SoC (e.g. MPEG encoder/decoder in handheld devices). The performance gain in embedded system is achieved by moving critical tasks to hardware so that they can be handled more efficiently when compared to general purpose processors.

```
y[t]=0;
for(i=0;i<N;i++)
{
y[t] +=
w[i]*x[t+i];
}
```

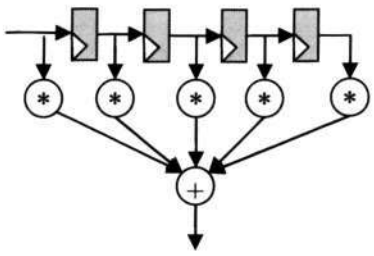


Figure 3.1 FIR filter: software and hardware implementation

Let us take the example of a software-based FIR and hardware-based FIR as shown in Figure 3.1. For an N-tap filter, the software based implementation needs to iterate at least N computation cycles in order to obtain the final result. In constrast, the hardware based implementation can compute the final result in a single cycle. The improvement is achieved by employing CFUs (i.e. 5-input adders) and exploiting the inherent parallelism of the five multiplications (each multiplication can be computed independently). In the software-based implementation, the processor can only perform a 2-input multiplication and 2-input addition in each iteration as only general purpose functional units are available. Hence, it is evident that parallelism and the use of CFUs can lead to high performance gain in HWA.

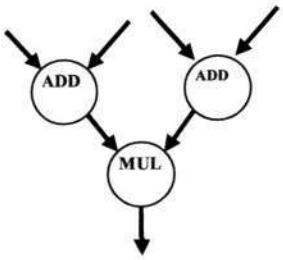


Figure 3.2 Simple DFG

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Parallelism exists among operations that can be computed independently of each other. There are three types of parallelism found in applications: Instruction-level, Loop-level and Task-level. Instruction level parallelism (ILP) exists among data-independent operations. For example in the DFG in Figure 3.2, the two ADD operations are independent and can be executed concurrently. Loop-level parallelism (LLP) exists among data-independent instructions found between loop iterations. Task-level parallelism (TLP) exists between tasks that can be computed independently of each other. Task-level parallelism is commonly exploited in multi-processor platform as in the case of web servers that process many independent clients' requests. ILP and LLP are more commonly exploited in HWAs. Therefore, in order to automate the design of HWA, high-level compilers must incorporate efficient strategies to expose ILP and LLP in applications. ILP is exploited in two ways: sub-instruction pipelining and concurrent execution. The 5-stage pipelined RISC architecture is an example of sub-instruction pipelining. Independent instructions can be executed "concurrently" in the pipe. Concurrent executions are performed using multiple functional units as shown in the FIR filter in Figure 3.1. In Figure 3.1, multiple multiply operations and add operation are executed in one cycle. LLP is exploited through loop unrolling and loop pipelining techniques, which will be discussed in the following sections.

Hardware designers often incorporate CFUs to achieve a higher performance in manual RTL design. Examples of CFUs include bit-reverse operations, rotate operations, multiple-input addition operations, etc. Domain-specific processors also often rely on CFUs to improve performance. For example, CFUs to compute MAC (Multiply-Add) and modulo operations are usually found in DSP processor. The Xtensa configurable processor, which incorporates an extensible instruction-set, is another example of how hardware acceleration is achieved through CFUs [3]. The capability of extending the instruction-set allows Xtensa's customers to customize the processors by augmenting the base processor with CFUs in order to meet their application requirements. This concept is also supported in commercially available reconfigurable platforms. For example, the NIOS II development tools from Altera enable users to define a new instruction and incorporate it into the processor data-path. Xilinx supports custom-instructions using simplex link for Microblaze and APU for

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

PowerPC [5]. It is note-worthy that these tools require the designers to manually identify the custom instructions from the applications.

As mentioned in previous chapter, the design community is moving towards ESL design methodologies which "focus on the higher-abstraction-level-concerns first and foremost" to increase designers' productivity [44]. This trend brings up the need to automate realization of HWA from high-level codes such as C, Matlab, or Java. High level languages such as Handle-C that is based on augmenting the ANSI-C language with timing constructs is not a preferred approach in ESL methodology as it requires the user to make hardware implementation decisions at the high level. In line with this, many commercial efforts have been established that offer tools for un-timed-C-to-hardware compilation such as Synfora FPGA PICO [8], Mentor's Catapult-C [7] and Altera's C2H [10]. The dominant use of C/C-like language as inputs to these tools is mainly due to the legacy of embedded applications that are typically written in C, and the familiarity of the language to embedded designers.

Auto-compilation of C to Hardware is a challenging task as an optimized HWA is very different from original C specifications as shown in the FIR example. The work in [20] has reported that the main challenges for designing hardware from C/C-like applications are due to the concurrency and timing control issues. In addition to this, there is a need to identify suitable CFUs from the application in order to achieve hand-crafted-quality hardware. Hence, a tool that can auto-generate HWA from C must be able to extract parallelism from a given sequential C application and identify CFUs that can lead to performance gain. In this thesis, we have adopted Trimaran as the front-end parallelizing compiler for the proposed high-level estimation framework. In the next section, we will describe the Trimaran compiler infrastructure and illustrate the important optimization features of Trimaran that we have relied on to devise an efficient RTL architecture from pure C applications.

3.2. Trimaran Infrastructure

3.2.1. *Trimaran compiler overview*

Trimaran [11] compiler is an infrastructure for compiler researches which target VLIW architectures. Trimaran has the following advantages over other advanced high-level C compilers such as Spark [45], SUIF [46]: full ANSI-C support, support for user-defined instructions, better documentation, and open source-code.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

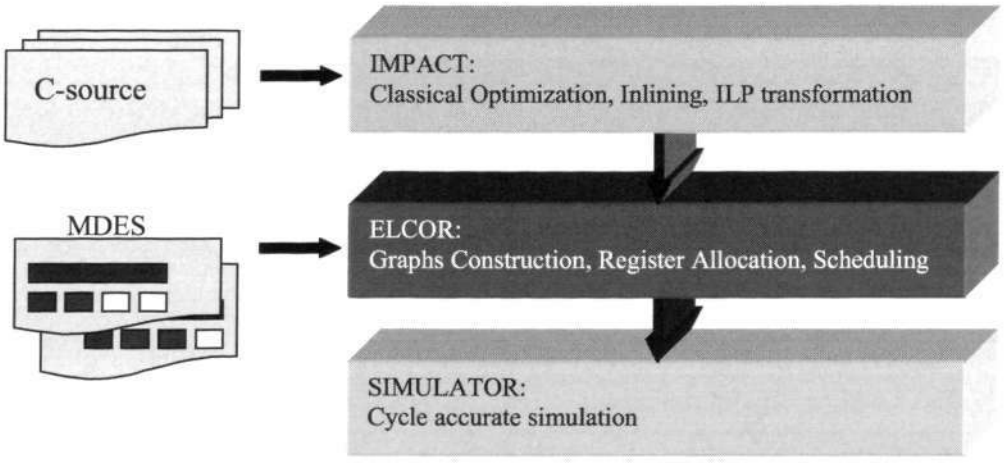


Figure 3.3 Trimaran compiler infrastructure

Figure 3.3 shows the Trimaran system which comprises of four modules: Impact, Elcor, Mdes and Simulator. Impact performs front-end processing to compile functions written in C into sequences of micro operations. Elcor is a retargetable compiler for VLIW machine, which transforms the output of Impact into O_el files. O_el files specify details of operations such as scheduled time, source registers, destination registers using an assembly-like syntax. The term "retargetable" refers to the capability of recompilation of the codes for different platforms or machines. Elcor performs register allocation and scheduling based on resource information described by MDES, which is responsible for maintaining and managing a database of resources. Simulator simulates outputs of ELCOR and gathers all statistic information for system analysis. Machine description system describes attributes of each type of hardware resource such as FU's quantities, latencies etc. The MDES language is capable of describing any architecture belonging to the VLIW family.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3.2.2. *VLIW architecture*

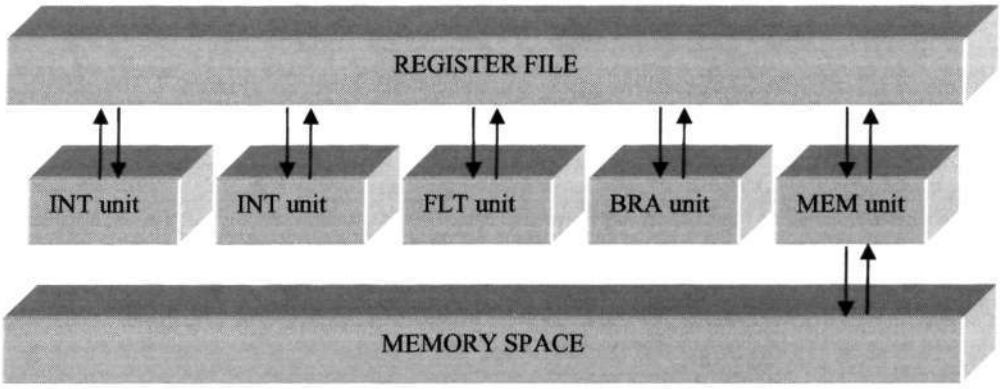


Figure 3.4 VLIW architecture

Figure 3.4 depicts a general view of the VLIW architecture. Integer (INT) unit, floating-point (FLT) unit, memory (MEM) unit and branching (BRA) units are the different types of functional units which perform integer operations, floating-point operations, memory load/store operations and branching operations respectively. Each VLIW machine can have multiple units of each type. Each functional unit can access any register in the register file. The compiler is responsible for exposing the inherent parallelism of the applications and determining which functional units that the operations will be executed on. The compiler then generates a sequence of long instructions, where each instruction encodes multiple operations. Each operation specifies the type of operation, source and destination operands for each functional unit. Trimaran incorporates various sophisticated scheduling techniques for exposing the ILP and LLP, e.g. super-block-scheduling, hyper-block scheduling, meld-scheduling, loop-unrolling, loop-pipelining etc., to fully utilize the available hardware resources.

3.2.3. *Important Trimaran capabilities*

Trimaran supports a large number of optimizations and data analysis capabilities [47]. In this section, we explain the most important features in Trimaran for hardware compilation, which are a) scheduling techniques, b) function in-lining, c) loop unrolling, d) loop pipelining, e) instruction customization, and f) design exploration.

1. **Scheduling techniques:** Trimaran supports a range of scheduling techniques that attempt to expose ILP from sequential representations of applications. These

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

techniques include cycle scheduling (Hyper-block/Super-block scheduling), meld scheduling and modulo scheduling. We will briefly describe these techniques in this section.

A basic block is a segment of codes that has only one entry and one exit (e.g. blocks 1, 2, 3 and 4 in Figure 3.5(a)). ILP in basic-blocks is usually very limited. Super-block is a block that consists of many frequently executed basic blocks, and has one entry and many exits. Super-block scheduling aims to expose more ILP by moving codes across basic blocks within a super-block. Figure 3.5 illustrates the concept of super-block scheduling. Super-block scheduling first performs super-block formation which groups frequently executed basic blocks into one super-block. Figure 3.5(a) shows an original CFG which comprise of basic blocks 1, 2, 3, and 4. Tail duplication technique is performed on block 4 (Figure 3.5(b)) in order to form super-block (1, 3, and 4) in Figure 3.5(c). The purpose of tail-duplication is to remove the entry of block 2 to block 4. Block 3 is chosen for super-block formation instead of block 2 because it is executed more frequently (80% compared to 20% of total time). Super-block formation leads to more operations in the new super block and therefore there is a higher likelihood for more ILPs to be discovered. Scheduling techniques for super-blocks are the same as those used in basic blocks. It has been reported that super-block scheduling can yield 7X speed up for certain applications [49].

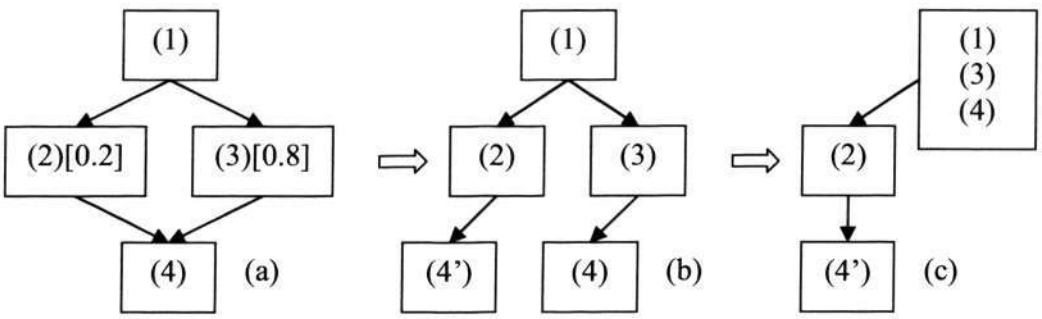


Figure 3.5 Super-block scheduling

Similar to super-block scheduling, hyper-block scheduling also tries to enlarge the block using if-conversion. If-conversion is a technique that utilizes a predicated bit to eliminate the branch. Each operation in VLIW incorporates a predicated bit. Operation is committed only when the predicated bit is true. Let us take the following

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

if-statement as an example: "if(b>c) a=b+c; else a=b-c". This statement is represented by CFG in Figure 3.6(a). In Figure 3.6(b), the branch has been eliminated. In the new hyper-block, at cycle 0, comparisons are performed; at cycle 1, two operations are issued but the value that is written to variable "a" is determined by p1 and p2. This mechanism utilizes multiple functional units and eliminates the overhead of branching. It is noteworthy that this is similar to implementing a time-multiplexed data-path in hardware (c).

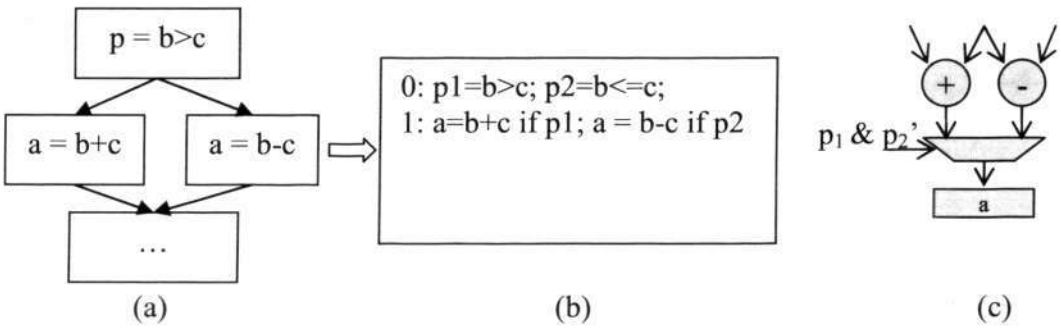


Figure 3.6 Eliminate Branch using If conversion to form hyper-blocks

Meld scheduling allows latency of operations in the preceding block to meld into neighboring blocks to avoid stall cycles. We will use the example in Figure 3.7 to illustrate the meld scheduling technique. The conventional scheduler waits two cycles for the load operation to finish before branching to another block. Meld scheduling melds the latency of Load operation into the destination block without violating the data-dependency. For this example, meld scheduling shortens the execution time by one cycle.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

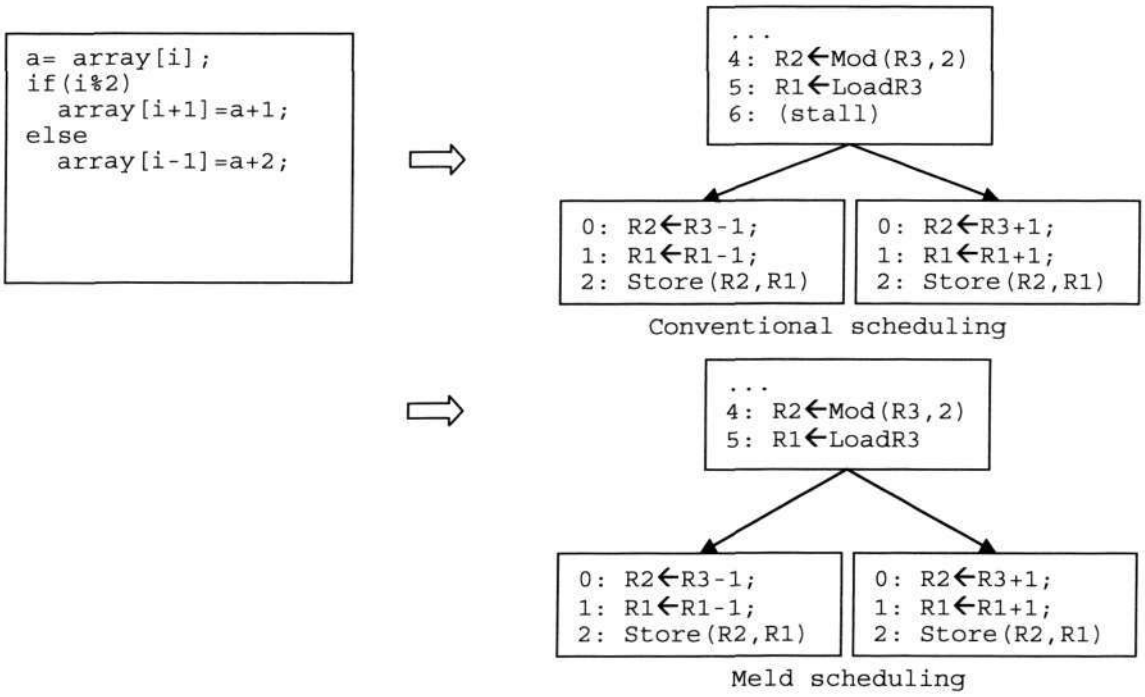


Figure 3.7 Meld scheduling

2. **Function in-lining:** This optimization aims to reduce the function-call overhead by inserting the whole function body at the place where it is called. The drawback is the increase in code size. Impact automatically in-lines functions as long as the code size does not exceed a threshold specified by users. Function in-lining enables area-time analysis of functional blocks at any hierarchical level. For example, in the function call graph shown in Figure 3.8, area-time of function B can be estimated with or without D and E being in-lined.

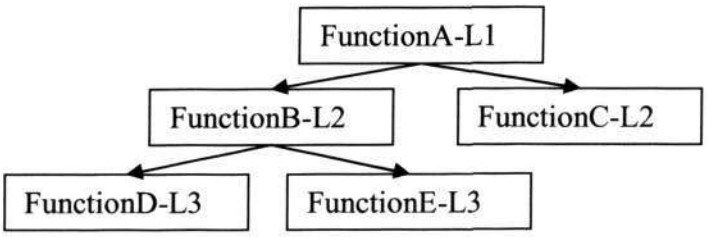


Figure 3.8 Example of Calling functions

3. **Loop Unrolling:** Loop unrolling is used to unroll "for" loops in order to expose loop-level parallelism. Independent operations in different iterations can be executed as long as there are available functional units. Figure 3.9 illustrate the loop-unrolling

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

concept. Assuming that there are two memory ports and two adders available in the VLIW machine, then two loads and two increment operations can be executed at the same time. This results in 2X speed up over the RISC like machine (on VLIW machines, opcodes in the same line are executed in the same cycle).

```
For(i=0;i<10;i++)
{
A[i]++;
}
Assemply code

L1: R1←Load i;
R1←R1+1;
Store i, R1;
i←i+1;
Jmp i<10;

After unroll 2 times:
```

RISC-like machine	VLIW machine with 2 MEMORY PORT, 2 ADDERS
L1: R1←Load A[i];	L1:R1←Load A[i1] , R2←Load A[i2];
R1←R1+1;	R1←R1+1 , R2←R2+1;
Store A[i], R1;	Store A[i1], R1; Store A[i2],i2; i1←i1+2;
i2←i+1;	i2←i2+2;
R2←Load A[i2];	Jmp i1<10, L1;
R2←R2+1;	
Store A[i2], R2;	
i←i+2;	
Jmp i<10, L1;	

Figure 3.9 Example of Loop Unrolling

4. **Loop-pipelining:** This optimization technique is capable of exploiting loop level parallelism that cannot be identified by loop unrolling. It is extremely useful in VLIW architectures as it increases the loop's throughput significantly. In the example below, instruction C has inter-iteration data dependence and therefore, loop unrolling cannot be applied. However, loop-pipelining can still identify ILP to increase the loop's throughput while maintaining code semantics by overlapping iterations i.e. next iteration starts n cycles after the current iteration (n < loop's latency, n=1 if loop is fully pipelined). In the example below, operation A of iteration 1 is executed in 1st cycle; operation B of iteration 1 and operation A of iteration 2 is executed in 2nd cycle, so on and so forth. Therefore, operation C of iteration 2 is executed after operation C of iteration 1, which guarantees correctness of data dependency. After three cycles, the pipeline is full; the machine executes four operations of four different iterations in the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

same cycle and effectively produces one iteration per cycle, which is 4 times faster than traditional scheduling.

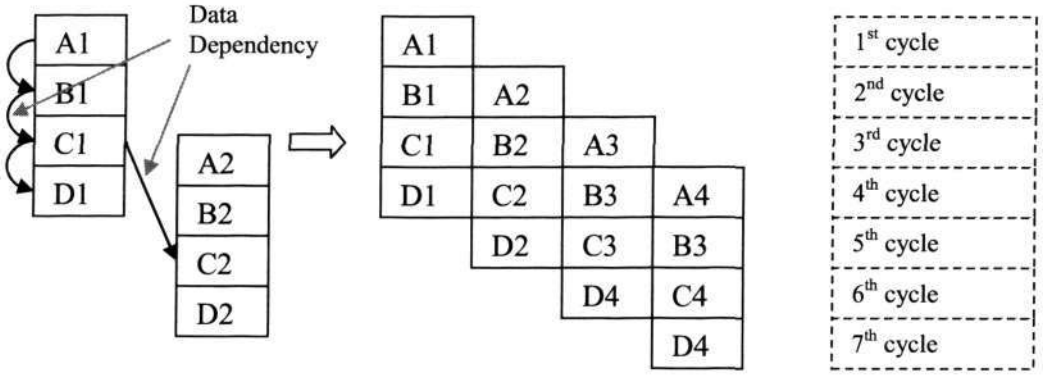


Figure 3.10 Illustration of the concept of loop pipelining

Currently, CRATE does not support software pipelining. Considerations for incorporating loop pipelining will be discussed in Chapter 9.

5. **Instruction Customization:** Trimaran allows user to incorporate user-defined instructions. This is one of the key advantages of Trimaran compared to other available parallelizing compilers. This mechanism allows application-specific instructions to be added, scheduled and simulated. To add new instructions, instructions' formats are defined by modifying MDES. The format of instruction comprises of opcode, number of inputs and outputs, types of inputs and outputs, DFG patterns, etc. Elcor searches the application's DFG and replaces the matched patterns with new opcodes.

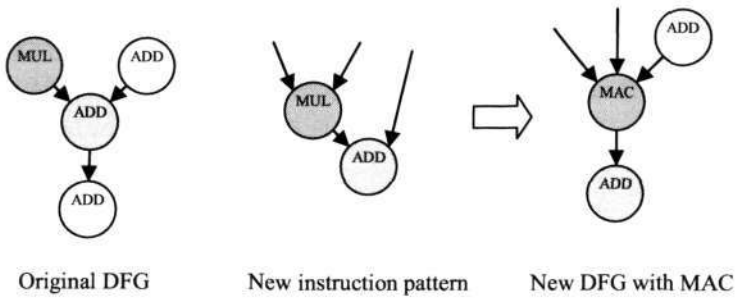


Figure 3.11 Illustration of Custom instruction substitution

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

6. **Design exploration capability:** The machine description system of Trimaran provides a convenient way to define new target machines. A new target machine can be created in seconds by changing number of functional units in the machine description (.hmdes) file, which is provided in human readable format. By changing number of functional units, different design points with different cost-performance characteristic of the same algorithm can be explored. Figure 3.12 shows a snapshot of the compiled schedules for VLIW with 2, 4, 6 functional units. Each schedule takes different cycles to complete. The capability of specifying arbitrary number of functional units makes Trimaran suitable for design exploration.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

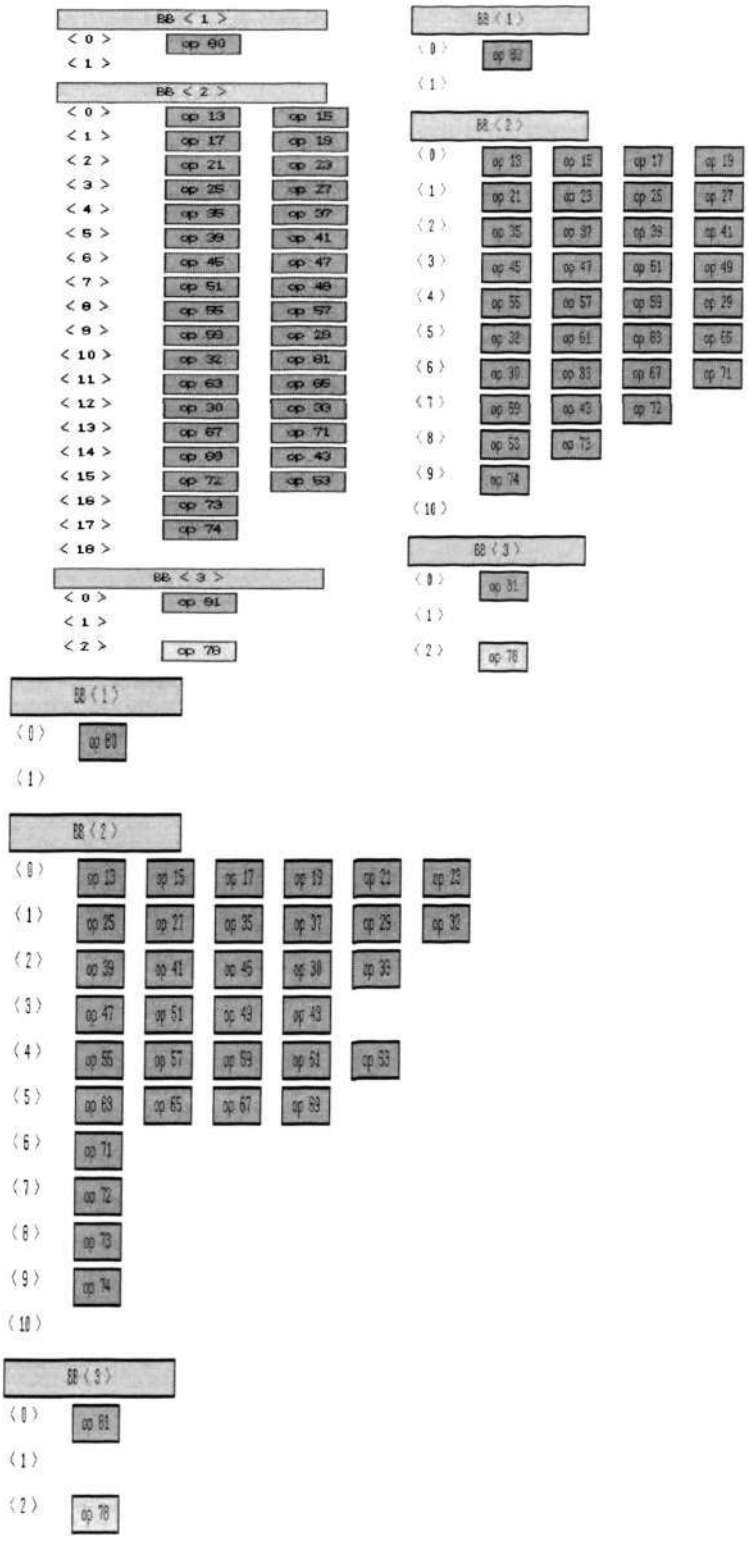


Figure 3.12 Schedules with different number of functional units

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3.2.4. *Inefficiencies of VLIW architecture for hardware implementation*

As mentioned earlier, the Trimaran compiler is an infrastructure for compiler researches which target VLIW architectures. While the VLIW architecture provides some means of parallelism through ILP, it still lacks the degree of customization required for a highly efficient HWA design. The inefficiencies of the VLIW architecture for HWA design is due to the following general-purpose hardware models in the VLIW architecture: 1) Homogeneous functional unit array, 2) Register-file model, 3) Memory model and 4) Branching model. The following explains the reasons for the inefficiencies of the models in realizing efficient HWA.

Homogeneous functional unit array: The functional units found in the VLIW architecture are very coarse-grained and supports multiple operations. Integer Functional unit support ADD/SUB/MUL/AND/OR etc. This implementation is generic but area-time inefficient because the operations are time-multiplexed i.e. at any instant, each unit can only execute a single operation. Therefore, there is a need to modify the target architecture to support CFUs before employing Trimaran for high-level estimation.

Register file model: The VLIW processor has register files with read/write ports. Read/write ports can connect to any functional units and to any register. The FPGA implementation of this model leads to unacceptable size and delay due to the large number of multiplexers required to connect the FUs to the registers [50]. Since a HWA is application-specific, it is possible to achieve significant reduction in area-time by employing application-specific interconnect between registers and functional units. This limits the number of multiplexers to facilitate sharing of FUs and registers only when they are required by the application.

Memory model: The VLIW memory model in Trimaran's machine description is a continuous memory. There can be several ports but the ports are connected to a single memory bank. On the other hand, Block Rams on the FPGA system are discontinuous. Each FPGA Block Ram supports two memory ports. This means that the FPGA system could access n memory banks at a time using 2n memory ports.

Branching model: The branching model employed by Trimaran performs sequential condition checks to determine the branch decisions. In contrast, control units in HWA typically allow 1) concurrent processing of the conditional checks and

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

2) multi-state branching where each state can branch to multiple states, which leads to faster computations.

This project relies on Trimaran for parallelism extraction and code scheduling of the C-based applications to enable high-level estimation. All the mechanics of the underlying target architecture of Trimaran, e.g. branching model, passing of arguments to functions, register files, etc. is adopted in the target HWA. In order to overcome the limitations of the VLIW architecture for efficient HWA design, we have modified the Trimaran's target architecture. In particular, we have devised efficient models to replace the homogeneous functional units array and register file, as they contribute to significant area-time gain for hardware mapping of most applications. Since not all applications can benefit from efficient memory and branching models, efforts to devise efficient memory/branch models will remain as future works.

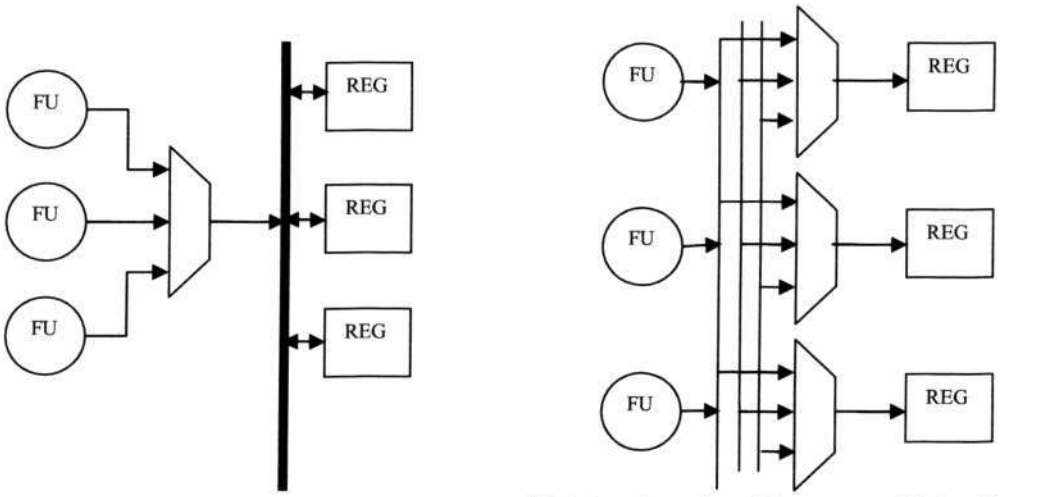
It is noteworthy that the techniques for high-level estimation proposed in this thesis can also be used on the intermediate representations produced by other high-level compilers. This thesis emphasizes on the strategies for area-time estimation of a particular function from the information provided by the compiler. In the next section, we will describe the modifications that have been made to the original hardware model of Trimaran in order to facilitate high-level FPGA estimation.

3.3. Proposed Architecture Template for area-time estimation

3.3.1. *h-VLIW architecture*

In this section, we introduce the architecture template that will be used for hardware estimation. Generally, an algorithm can be mapped onto two types of architectures: mux-based architecture and bus-based architecture. Mux-based architecture employs multiplexers to steer data into functional units whereas in bus-based architecture, data are transferred via buses that are shared between processing elements. The different architecture types require different scheduling algorithms. In this project, mux-based architecture has been adopted due to the following reasons: 1) Trimaran hardware model is mux-based, and 2) bus-based architecture can lead to performance degradation as it limits the number of simultaneous data transfers between functional units and storage units compared to mux-based data-path. Figure 3.13 illustrates the differences between the two types of architectures.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(a) Bus-based architecture, which allows 1 transaction per cycle

(b) Mux-based architecture, which allows 3 simultaneous transactions

Figure 3.13 Bus-based architecture and mux-based architecture

An algorithm can be realized in hardware using two different approaches: pipelined and non-pipelined as shown in Figure 3.14. The two types of realizations can be viewed as special cases of the data-path shown in Figure 3.14(a). The application-specific interconnection defines whether the architecture is pipelined or not. The number of functional units and control signals can be tuned based on the extracted information from the application such as degree of parallelism and data-dependency. Since C-coded applications utilize pointers extensively, the template must support referencing and dereferencing. A memory-shared data transfer mechanism or a stream buffer between host and accelerator can be employed to support memory access by pointers.

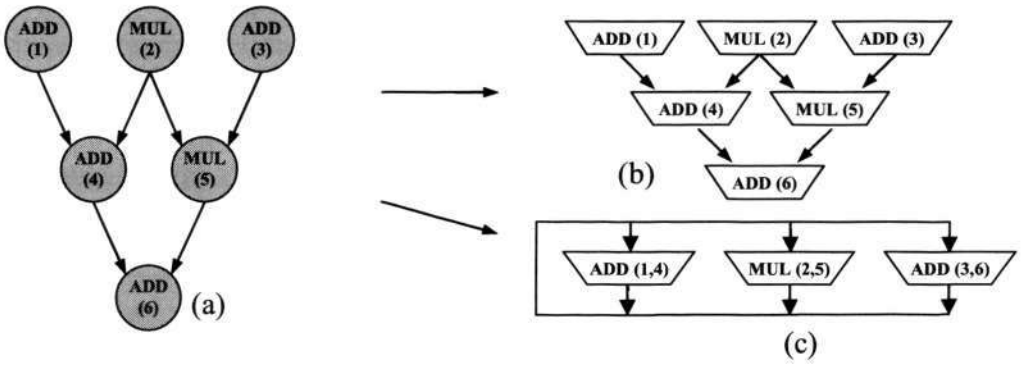


Figure 3.14 Two realizations for the same algorithm (b) pipelined datapath (c) data-path with shared resources

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Based on the considerations discussed above, we employ the architecture template shown in Figure 3.15 for the proposed high-level estimation approach. The architecture template comprises of three main components: Memory, Data-path, and Control-Path. The data-path resembles an application-specific VLIW machine, which is highly optimized for a particular task by employing heterogeneous functional units and application specific interconnection. In this model, each functional unit is a CFU, which performs dedicated operations such as addition, subtraction, shift or multiply-and-accumulate (MAC). As discussed in the previous section, application specific interconnect between functional units and registers can lead to area-time efficient realization of the HWA. This model has been incorporated into the Trimaran machine description system so that Trimaran can perform scheduling to target the h-VLIW architecture.

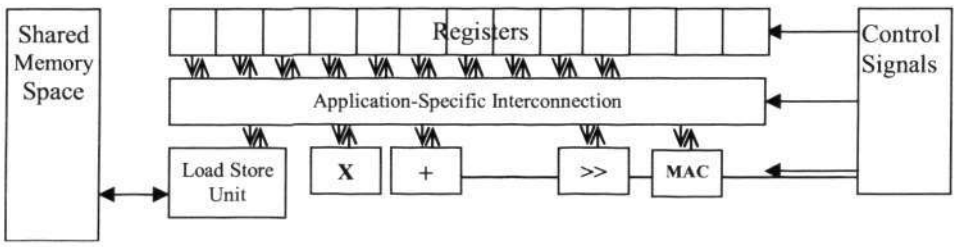


Figure 3.15 Architecture Template with heterogeneous Functional Units (h-VLIW)

3.3.2. *RTL Models of the h-VLIW architecture*

This section describes the RTL models of the h-VLIW architecture. In particular, it defines the control-path, data-path, various delays of components, and the Control Information Table (CIT), which is used to hold control information for each RTL component.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

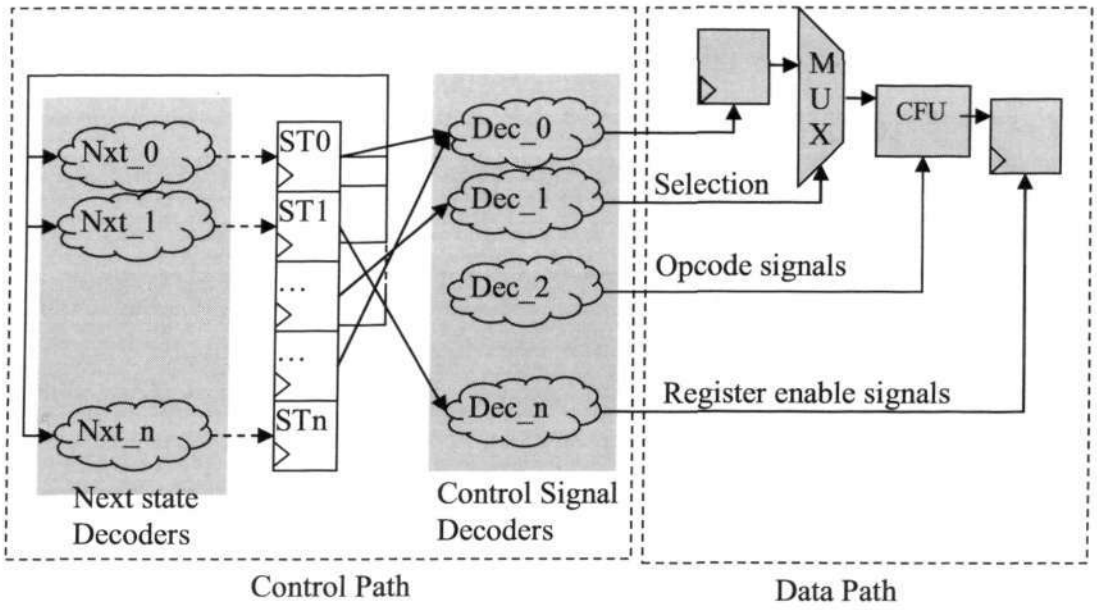


Figure 3.16 General view of accelerator at RTL

Figure 3.16 depicts the control and data-path model of the target architecture. Control path comprises Next State decoding logic (NSDL), control signal decoding logic (CSDL) and state registers (or state flip-flops). Data path comprises Registers to hold data, multiplexers to steer data and CFUs to compute the data.

(a) Custom Functional Units:

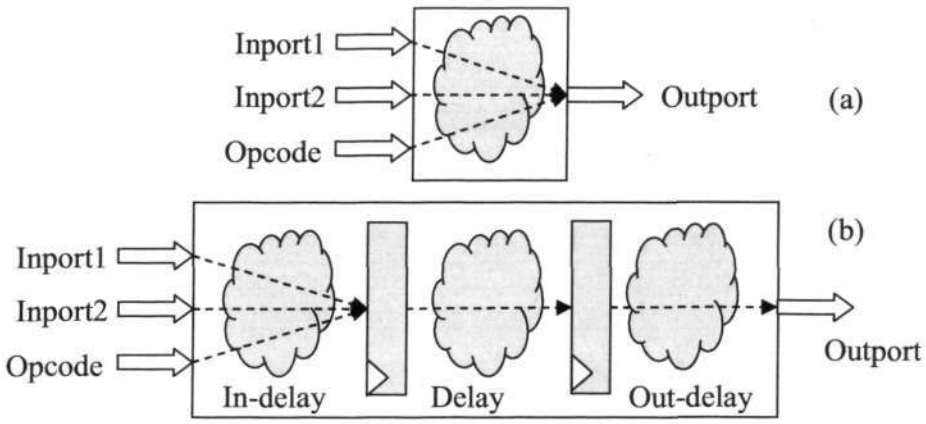


Figure 3.17 RTL Model of Customized Functional Units (CFU)

Figure 3.17 shows the non-pipelined (Figure 3.17(a)) and pipelined (Figure 3.17(b)) functional units which have two input ports, one opcode port and one output port. The clouds and the rectangles with small triangles represent combinatorial logics

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

and flip-flops/registers respectively. Only one delay value is associated with a non-pipelined CFU to represent combinatorial delay from Input1, Input2, and opcode to OutPut. Delays from opcode and from Input to OutPut are assumed to be equal. Similarly, only one area value is use to denote the number of LUT that the non-pipelined CFU consumes. Delays of pipelined CFU are modeled with three values to represent in-delay, internal delay and out-delay. In-delay is the combinatorial delay from inputs to the first register stage. Internal delay represents maximum of internal delays between registers stages. Out-delay is combinatorial delay from the last stage to OutPut. Area is modeled using two values, which represent number of LUTs and FFs. Delay, FFs and LUTs of RTL components will be referred to as the Physical Information.

(b) Multiplexers:

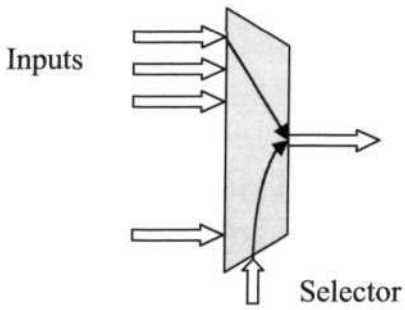


Figure 3.18 RTL model of multiplexer

Multiplexers are used to steer data towards the right path based on values of the selector. Area is measured in terms of number of LUTs. Delay from input and from selector is assumed to be equal. The delay from selector takes into account the effect of state-decode delay, which will be elaborated in a later section.

(c) Registers:

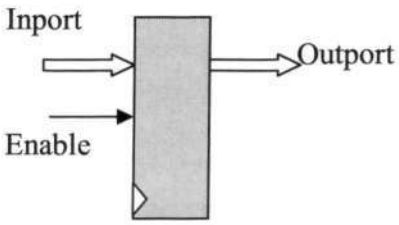


Figure 3.19 RTL model of register

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Registers are used to hold data. Data are latched from in port to out port at rising edge of clock when the enable signal is high. The area of registers is measured in FFs. Timing parameter associated with registers are set-up time and clock-to-output time which are obtained from FPGA datasheet.

(d) Control Information Table (CIT):

TABLE 3-1 CONTROL INFORMATION TABLE (CIT)

States	Selector Values
State_1	001_b
State _2	010_b
State _3	101_b
State _4	001_b

This is CRATE's internal data-structure used to hold control information of RTL components. The opcode, and selector and enable signals (referred to as control signals) are used in control path estimation and RTL generation. To encode the control information of each component, a table (Control Information Table (CIT)) is constructed for each signal (or opcode) to map its values to states. CIT is constructed during the resource binding process that will be discussed later in this chapter. The Table on the right shows an example of mapping values to states of the selector signals of an 8-to-1 multiplexer.

3.4. Mapping Trimaran's IR

In the previous sections, we have discussed the advantages of using Trimaran to expose parallelism in sequential C applications. We have also presented an architecture template that incorporates CFUs for hardware acceleration. In this section, we describe a resource binding strategy for mapping Trimaran IR onto the proposed architecture template.

Figure 3.20 illustrates the concept of resource binding for mapping Trimaran IR onto the architecture template. Figure 3.20(a) shows an example of the IR that has been generated from sequential C code using the Trimaran compiler. The IR contains attributes such as scheduling and the input/output information of each operation. This information is represented in the form of a DFG in Figure 3.20(b). Prior to resource binding, the number and types of available CFUs for the architecture template are

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

specified. Note that the CFUs that are specified in the system must be able to accommodate the minimum number of operations in the DFG. In this example, two adders and two multipliers are specified. Figure 3.20(c) shows a generic architecture template that does not have a fixed interconnectivity between the registers and CFUs. Resource binding is then performed to map the operations in the DFG onto the architecture template in order to define the set of required CFUs and the application-specific interconnectivity as shown in Figure 3.20(d). After resource binding, all the adders that have been specified are fully utilized. As there is only one multiplication in the DFG, the final h-VLIW architecture has only one multiplier.

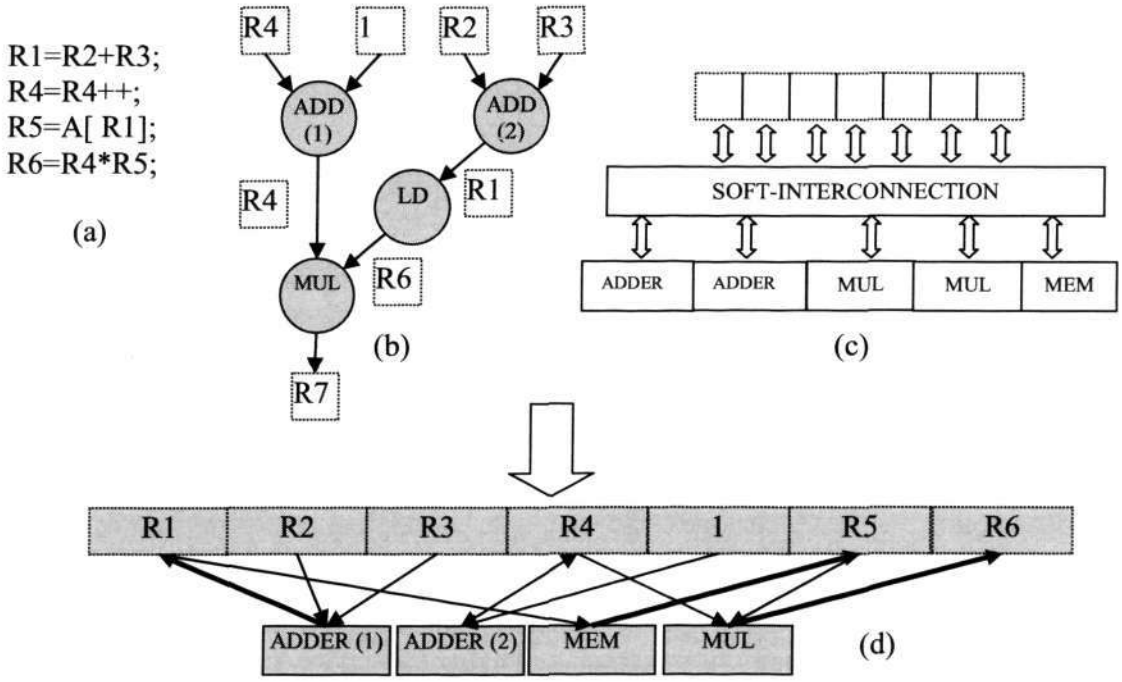


Figure 3.20 Mapping DFG to the architecture template

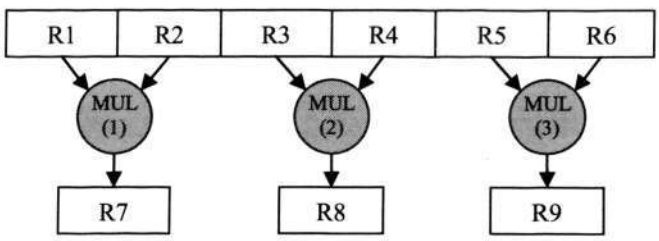
Due to the complexity of the resource binding problem, previously reported works in resource binding methods commonly adopt approaches that result in long iterations in order to converge to a good solution. It has been shown that it is difficult to achieve an optimal resource binding solution due to the inter-iteration dependencies. Resource binding techniques with various optimization goals have been thoroughly researched and are not the focus of this thesis. In this thesis, we employ a relatively simple resource binding algorithm to realize HWAs with reasonable quality. The proposed technique relies on simple heuristics to choose a solution from an exhaustive

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

set of configurations that would minimize the interconnectivity between the CFUs and registers in each iteration.

We will use the example in Figure 3.21 to describe a single iteration in the proposed resource binding method. Let us assume that from the previous iterations, three multipliers (MUL1-MUL3) have been bound to nine registers (R1-R9) in the architecture template (as shown in Figure 3.21). In the current iteration, Op1 and Op2 are two multiplication operations that need to be mapped to the existing multipliers in the architecture template. CRATE attempts to map the operations in each execution cycle one at a time to the architecture template in order to avoid resource conflict.

Op1: R10←R1*R2
Op2: R11←R5*R4



Configuration#1: Op1 → MUL1; Op2 → MUL2; Total Score: 2+1=3
Configuration#2: Op1 → MUL2; Op2 → MUL1; Total Score: 0
Configuration#3: Op1 → MUL2; Op2 → MUL3; Total Score: 1
Configuration#4: Op1 → MUL3; Op2 → MUL2; Total Score: 1
Configuration#5: Op1 → MUL1; Op2 → MUL3; Total Score: 2+1=3
Configuration#6: Op1 → MUL3; Op2 → MUL1; Total Score: 0

Figure 3.21 Example of proposed resource binding strategy

Figure 3.21 shows six possible configurations that are evaluated for binding the two multiplication operations onto the three multipliers. Each cycle with m concurrent operations that need to be assigned to n number of CFUs in the architecture template, will lead to $n*(n-1)*...*(n-m+1)=\frac{n!}{(n-m)!}$ number of configurations to be evaluated. Each configuration is assigned a score that is computed based on the number of sources and destinations that the operations and the CFUs have in common. For example in Figure 3.21, since MUL1 and Op1 have two common sources (i.e. R1 and R2), the score for binding Op1 to MUL1 is 2. Similarly, the score for binding of Op2 to MUL2 is 1 as they have only one common source (i.e. R4). The total score of each configuration is the sum of the scores of individual bindings of the corresponding

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

operations to the CFUs. For example in Figure 3.21, configuration 1 has a total score of 3 as 1) Op1 and Mult1 have two common inputs/outputs resulting in a score of 2, and 2) Op2 and MUL2 has one common input/output resulting in a score of 1. Similarly, the scores for the exhaustive set of configurations are computed.

The configuration with the highest score is then selected as it leads to minimal increase in complexity of the interconnectivity. For example in Figure 3.21, configuration 1 is chosen as it has the highest score. Hence, Op1 will be assigned to MUL1 and Op2 will be assigned to MUL2.

In order to speed-up the binding process, CRATE assumes that low cost components (e.g. adders, comparators, etc.) are not shared. Our investigations reveal that resource sharing of these components does not yield any benefits in FPGA implementation. This is due to the fact that resource sharing of the low cost components generally leads to the introduction of multiplexers which may outweigh the area savings obtained. In order to illustrate this let us use the example in Figure 3.22 where the data-width is assumed to be 16 bits. In case Figure 3.22(a), two adders are implemented independently and this incurs a hardware cost 2x16=32 LUTs and a critical path delay of a 16-bit adder. In case Figure 3.22(b), resource sharing is employed to implement the adder in a time-multiplexed manner. Due to the introduction of the multiplexer in Figure 3.22(b), the total hardware cost that is required is 48 LUTs, i.e. 2x16=32 LUTs to implement two multiplexers + 16 LUTs to implement the adder. In addition, the critical path delay in Figure 3.22(b) is higher, i.e. delay of multiplexer + delay of adder. Therefore the proposed resource binding algorithm does not attempt to share low cost resources.

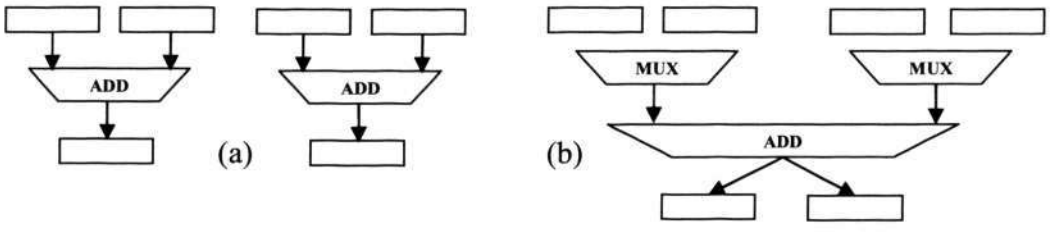


Figure 3.22 Circuits with and without resource sharing

The algorithm for resource binding in the proposed framework is shown in Figure 3.23 Operation binding algorithm. The algorithm begins by binding all the variables in

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

the IR to registers (line 2). The current work in this thesis does not consider register sharing among variables. The concurrent operations in each execution cycle (denoted as Operations_Vector) are then bound to a set of corresponding CFUs (denoted as CFU_Vector) (lines 10). The procedure for binding the Operation_Vector to CFU_vector is described in lines 16-35. This procedure first checks the complexity of the CFU that is selected for binding (line 19). If the selected CFU is non-low-cost type, CRATE generates all possible binding configurations (line 20) and compute their scores (line 21). The best configuration is then chosen (line 22). If a low-cost CFU is selected, CRATE first assigns each operation in the Operation_Vector to an unbound CFU (line 30-35).

```
#1 Procedure Bind Operations
#2 Bind All Variables to Registers
#3 Iterate around region (Basic Block or Hyper Block)
#4 {
#5     For_each CFU_Type
       {
             For_each cycle
                   {
#9                 Create_Operations_Vector
#10                Bind_Operation_Vector_to_CFU_Vector
                   }
       }
       }

#16 Procedure Bind Operation Vector to CFU Vector
       {
#18    If (Operation_Vector > CFU_Vector) return -1; //not enough
resources
#19    If (CFU.Lowcost is false) {
#20       Explore_all_options to bind Operation_Vector to CFU_Vector.
#21       Compute_Binding_Score_For_Each_Option
#22       Choose_the_best_options which has highest score
         }
#23    else // Low-cost components, should relax binding
       {
       For each operation in operation vector
       {
#27     For each CFU in CFU Vector //Evaluate Score for each CFU in
CFU vector
             {
#29          If(CFU.bound is false)
#30               Bind_Operation_To_This_CFU();
             }
       }
       }
   }
```

Figure 3.23 Operation binding algorithm

During resource binding, the CIT for each RTL component is constructed. For example, let us assume that Op1 and Op2 in Figure 3.21 are scheduled to be executed at cycle 4 (i.e. State_4). When operation 2 is assigned to MUL2, R5 is connected to MUL2 as the second input of left-port of MUL2. Hence, one entry is added to CIT of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

left-port of MUL2 which specifies that control value for this port is '1' (inputs are indexed from '0') at State_4. At the same time, a connection between R11 and MUL2 is established. One entry is added to CIT of R11 which specifies the enable signal is "high" at State_4.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3.5. Summary

With the current trend driving towards ESL, it is envisaged that auto-compilation of HWA from high-level applications will become an essential step in future embedded system design methodologies. We have demonstrated that parallel processing and CFU are two key factors for achieving high performance gain in HWA design. The Trimaran compiler infrastructure has been adopted for front-end processing in the proposed high-level estimation framework as it incorporates a wide range of optimizations and features to support the design of HWA. These include various types of scheduling to exploit ILP (super-block scheduling, hyper-block scheduling), loop unrolling and loop pipelining to exploit loop level parallelism, support for instruction customization and capability for rapid design exploration. Trimaran originally targets the VLIW machine. The Trimaran machine description has been modified to target the proposed h-VLIW architecture template for area-time efficient hardware mapping and high-level estimation. The h-VLIW architecture is based on the VLIW architecture that is augmented with CFUs and application-specific interconnectivity.

In order to map the application onto the h-VLIW architecture, CRATE extracts the scheduling information from Trimaran's IR, and performs resource binding to assign the operations onto the CFUs in order to 1) satisfy the data/control dependency and 2) minimize the interconnectivity between functional units and registers. The proposed resource binding technique chooses a binding configuration with the highest score from an exhaustive set of configurations. The score of each configuration corresponds to the interconnectivity complexity between the CFUs and the registers that will be incurred if the configuration is selected. A high score implies that less multiplexors will be required to facilitate resource sharing, hence leading to a more area-time efficient HWA. The outcome of resource binding is an h-VLIW architecture that incorporates the necessary CFUs to implement the application, with low interconnectivity complexity between CFUs and registers. The h-VLIW architectures comprises of data-path with CFUs for data processing and control path to control the data-flow. Based on this architecture, CRATE can perform area-time estimation where the control path and data-path are estimated separately. In the next chapter, we describe the construction and estimation of the control path. Data-path estimation will be addressed in Chapter 5 and Chapter 6.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 4 CONTROL PATH AREA-TIME ESTIMATION

As mentioned in Chapter 2, most of the previous works in high-level estimation do not take into account the control path, even though there are cases where the control unit contributes significantly to the total area-time of hardware accelerators. After the Elcor IR has been mapped onto the h-VLIW architecture, CRATE proceeds to estimate the control and data-path. In this chapter, we focus on area-time estimation of the control path.

The control path in the h-VLIW architecture is an FSM (Finite State Machine) that consists of three fundamental components: 1) State Registers, 2) NSDL (Next State Decoding Logic, and 3) CSDL (Control Signal Decoding Logic). The FSM in the h-VLIW architecture adopts the one-hot encoding scheme as our investigation shows that this scheme generally leads to the best performance for FPGA implementation. We will describe methods for constructing the FSM based on one-hot encoding scheme from the application IR. The NSDL is constructed by realizing each execution cycle (or control step) in the IR as a state in the FSM. Multi-state transitions are constructed based on the branch related operations in the IR. The CSDL is constructed based on the Control Information Table of each component that was generated in the resource binding stage.

The proposed control path estimation strategy performs area-time estimation of the NSDL and CSDL components separately using a set of equations that model the mapping of the FSM components onto the FPGA. Experimental results targeting Xilinx FPGA show that the proposed RTL models and techniques can lead to reliable area-time estimation. In particular when compared to results from a commercial tool, the proposed control path area estimation strategy leads to an average error of only about 10.3%. In addition, the maximum estimation error for the critical path of the FSM is less than 1ns.

It is worth mentioning that the proposed techniques in this thesis can be adapted for different FPGA devices with various LUT structures. In the remainder of this thesis, the examples that are given assumes that the target FPGA consist of 4-input LUTs that is found in popular FPGA devices such as Xilinx Virtex.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

4.1. Overview of FSM and Encoding Schemes

Figure 4.1 shows the general overview of a FSM with three main components: 1) NSDL (Next State Decoding Logic) which computes the next-state of FSM based on the current state and inputs, 2) CSDL (Control Signal Decoding Logic) which decodes control signals (i.e. multiplexer selector signals, register enable signals, CFU opcode signals), and 3) State-Registers, which are FFs that hold the current state of the FSM.

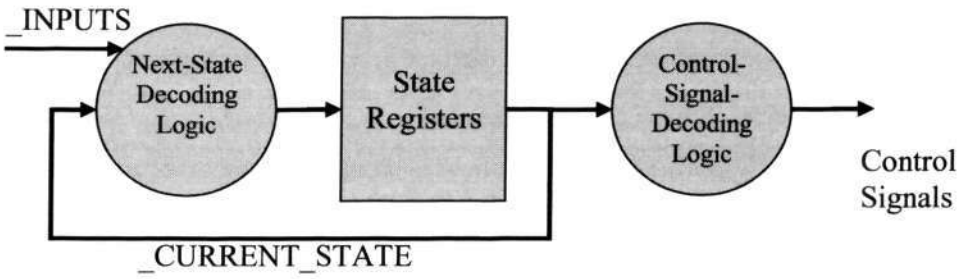


Figure 4.1 General overview of a Finite State Machine

4.1.1. *Common Encoding Schemes*

While the main components of a FSM remain the same across various implementations, the underlying encoding schemes may differ. As the choice of encoding scheme has a notable impact on the resulting area-time efficiency of the HWA, it is important to first discuss some of the most widely used encoding schemes e.g. gray-encoding, binary-encoding and one-hot encoding. In particular, we will show that the one-hot encoding scheme is the best choice for implementing long sequential FSMs on FPGAs.

Table 4-1 compares the various schemes for encoding eight states, where S_i are the bits used for encoding the states. It can be observed that the number of bits used for encoding the states can differ from one scheme to another.

TABLE 4-1 DIFFERENT ENCODING SCHEMES

	Binary	Gray	One-hot
Enumerated States	$S_0S_1S_2$	$S_0S_1S_2$	$S_0S_1S_2S_3S_4S_5S_6S_7S_8$
0	000	000	00000001
1	001	001	00000010
2	010	011	00000100
3	011	010	00001000
4	100	110	00010000
5	101	111	00100000
6	110	101	01000000
7	111	100	10000000

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

The binary encoding scheme uses $\log_2$ (n) bits to represent n-states. For example, in Table 4-1, 3 bits are required to implement an FSM with eight states. This means that $\log_2$ (n) bits must be evaluated in order to determine the current state of the FSM.

Similar to the binary encoding scheme, the gray encoding scheme also uses log2 (n) bits to encode n states. However unlike the binary encoding scheme, the gray encoding scheme ensures that only a single bit is changed during state-transition. This prevents the FSM from transiting to an undesirable state as bit changes do not usually occur simultaneously. For example, if the FSM in Table 4-1 transits from state_1 to state_2 (i.e. 001 to 010 for the binary-encoded scheme), intermediate states such as 011 or 000 may occur. These intermediate states may cause glitches in the control-signals if they are sampled asynchronously. The gray encoding scheme avoids this by ensuring only a single bit changes between consecutive states.

Unlike the binary and gray encoding schemes, the one-hot encoding scheme uses n bits to represent n-state. For example in Table 4-1, 8 bits are required to implement an FSM of eight states. Hence, the one-hot encoding scheme requires more FFs to be used as state registers. While this is clearly unfavorable for ASIC designs where the hardware cost of FFs is significant, the high utilization of FFs for the one-hot encoding scheme is not a major concern for FPGA design due to the abundance of FFs in FPGAs.

The main advantage of the one-hot encoding scheme over the binary and gray counterparts is that only 1 bit needs to be evaluated to determine the state of the FSM, which greatly simplifies the decoding logic. For large FSMs, the complex decoding circuitry of the binary and gray encoding schemes could lead to high critical path delay. In order to illustrate this better, we will use the example in Table 4-2 where each state of a 64-state FSM has an output signal S, which is active at state_1, state_33, and state_49.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 4-2 AN EXAMPLE OF A LARGE FSM

	Binary	Gray	One-hot	Signal (S)
Enumerated States	$S_0S_1S_2S_3S_4S_5$	$S_0S_1S_2S_3S_4S_5$	$S_0S_1S_2\ldots S_{62}S_{63}$	
State_0	000000	000000	$S_0 = 1$	0
State_1	000001	000001	$S_1 = 1$	1
State_2	000010	000011	$S_2 = 1$	0
…	…	…	…	…
State_32	100000	110000	$S_{32} = 1$	0
State_33	100001	110001	$S_{33} = 1$	1
State_34	100100	110001	$S_{34} = 1$	0
…	…	…	…	…
State_48	110000	101001	$S_{48} = 1$	0
State_49	110001	101001	$S_{49} = 1$	1
State_50	110010	101001	$S_{50} = 1$	0
…	…	…	…	…
State_63	111111	100000	$S_{63} = 1$	

The decoding logic for signal S in each of the scheme can be designed as follows:

S = '1' when _state = State_1 or _state = State_33 or _state = State_49

$S_{one\text{-}hot} = S_1+S_{33}+S_{49}$.

$S_{binary} = S'_0S'_1S'_2S'_3S'_4S_5+S_0S'_1S'_2S'_3S'_4S_5 + S_0S_1S'_2S'_3S'_4S_5$

$S_{gray} = S'_0S'_1S'_2S'_3S'_4S_5 + S_0S_1S'_2S'_3S'_4S_5 + S_0S'_1S_2S'_3S'_4S_5$

(S' denotes negation of S. '+' denotes bit-wise "or" operation. S_0S_1 denotes S_0 'and' S_1)

It can be observed that $S_{one\text{-}hot}$ can be constructed using a simple 3-input Or-gate (which can be mapped onto 1 LUT), while implementation of S_{binary} and S_{gray} require Not-gates, And-gates and Or-gates. Figure 4.2 shows the implementation of the decoding logic for S_{binary} and S_{gray} on the FPGA. It is evident that the binary and gray encoding scheme leads to an increase in decoding delay due to the higher number of logic levels. It is note-worthy that the negative effects of the binary and gray encoding schemes only arises when the FSM has more than 16 states (i.e. number of state registers is larger than 4). This is because LUTs on the FPGA will be cascaded in order to implement the large decoder circuit, which incurs additional inter-LUT routing delay. Hence, for large FSM, one-hot encoding is a more preferred approach for implementing large FSMs on FPGA [58]. Since FSMs found in common embedded functions usually have more than 20 states, we have adopted the one-hot

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

encoding scheme in our work. In addition, the FSMs that are derived from the functions used in our experiments consist of at least 27 states.

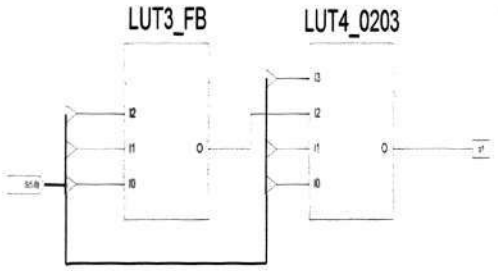


Figure 4.2 Cascaded LUTs for the decoding logic of binary and gray encoding scheme

4.2. Constructing FSM for Area-Time Estimation

Before area-time estimation of the control path can be performed, we need to construct the FSM from the application IR. In this section, we discuss strategies for constructing the NSDL and CSDL components.

4.2.1. *NSDL (Next State Decoder Logic)*

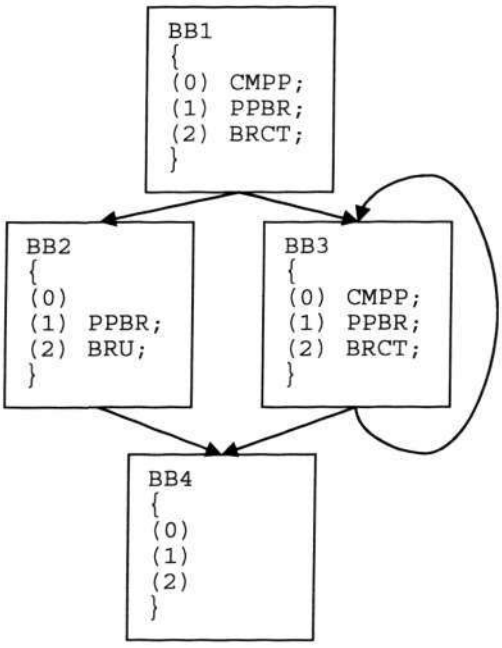


Figure 4.3 An example CFG

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

The NSDL is responsible for computing the next-state of FSM based on the current state and inputs. Each execution cycle or control step in the IR is modeled as a state in the FSM. As most applications are data oriented (as opposed to control oriented), most of the states in the FSM are single-state transition, which means there is only a single transition to the next state. Multi state transitions occur when certain branch-related operations are encountered in the IR. Since identifying single state transitions from the IR is a trivial process, we will only discuss the strategies employed for identifying multi-state transitions.

Trimaran branch-related operations can be categorized into two types: 1) un-conditional and 2) conditional. We use the CFG example in Figure 4.3 that consists of four Basic Blocks (BB), to illustrate these operations. Unconditional branch operations, e.g. BRU (Branch Unconditional) and PPBR (Prepare-To-Branch), are used to branch to a new address without the need to satisfy any conditions. For example, the BRU operation in BB2 results in a branch to BB4. Conditional branch operations, e.g. CMPP (Compare-To-Predicate), BRCT (Branch on Condition True), BRCF (Branch on Condition False) and PPBR are used to jump to an address when some conditions are met. For example, the BRCT operation in BB1 results in a branch to either BB2 or BB3 depending on whether the conditions are met or not. Similarly the BRCT operation in BB3 results in a branch to either BB3 or BB4.

We use the example in Figure 4.3 to illustrate the roles of the conditional branch instructions for branching from BB3 to either BB3 or BB4. Firstly, the CMPP operation (cycle 0 of BB3) evaluates the conditions and sets up the value of the predicate register, which forms one of the inputs of the BRCT instruction (cycle 2 of BB3). The PPBR operation sets up branch target by moving the destination's address into branch target register, which constitutes to the other input of BRCT. Hence, the BRCT operation now consists of two inputs: 1) branch target register which indicates the destination of the branch, and 2) predicate register which indicates whether the condition has been met or not. After executing the BRCT operation, the machine jumps to the address specified in the branch target register if the predicate register is 1 (conversely, if the BRCF operation is executed, the branch to the address in the branch target register takes place only when the predicate register is 0). Otherwise the machine branches to a new address by incrementing the program counter.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

By identifying the unconditional/conditional branch instructions from the IR, multi-state transitions can be constructed for the FSM. CRATE employs an internal data structure to record all the branch information of the IR. Figure 4.4 shows the data structure that records the branching information of the example discussed earlier. Each execution cycle in the IR corresponds to one control step, which in turn is equivalent to one state in the FSM. Each entry in the data structure, which corresponds to a FSM state consist of one ID number, a link to the next control step, and one branch table. The branch table records the branch operations BRCT, BRCF and BRU, and the branch's destination block. For example, control steps 0-2 corresponds to the three operations in BB3 of Figure 4.3 (i.e. control step 0 corresponds to CMPP operation, control step 1 corresponds to PPBR operation, and control step 2 corresponds to BRCT operation). At the entry with ID = 0x0302 (control step 3), the branch table records the information that indicates a possible branch to block BB3 when the predicate register value (predicate_register_0) is true. The target address (i.e. BB3) when this condition is met is obtained from the preceding PPBR operation.

The information in the data-structure can be used to construct the NSDL that will be used for area-time estimation. CRATE also automatically generates the RTL codes for the NSDL from the data structure. To illustrate this, we show the RTL codes for the NSDL in Figure 4.5 that corresponds to the example in Figure 4.3. When a branch operation is identified in the branch table, an if-statement is generated to evaluate the conditions (i.e. whether predicate_register_0 = true/false) in order to determine the next state (i.e. BB3 (ID=0x0300) or next control step).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

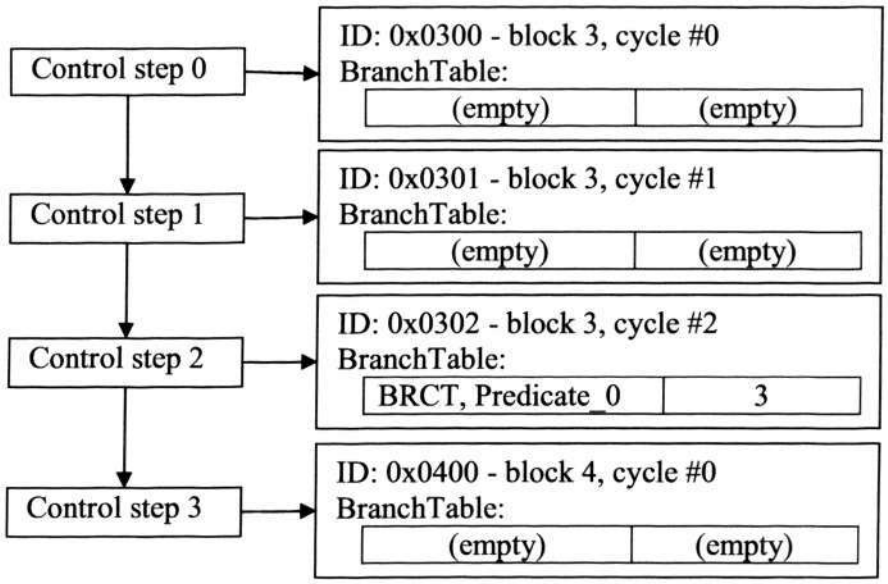


Figure 4.4 Internal Data Structure for FSM construction

```
StateDecode: PROCESS (CUR_STATE, Predicate_Reg0)
 BEGIN
 CASE (CUR_STATE) IS

 WHEN State_0300 =>
      NEXT_STATE<= State_0301;
 WHEN State_0301 =>
      NEXT_STATE<= State_0302;
 WHEN State_0302 =>
      If(Predicate_Reg0 = '1') then NEXT_STATE<= State_0300;
      else
      NEXT_STATE<= State_0400;
      End if;

 END CASE;
 END PROCESS;
```

Figure 4.5 RTL codes of NSDL

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

4.2.2. *CSDL (Control Signal Decoding Logic)*

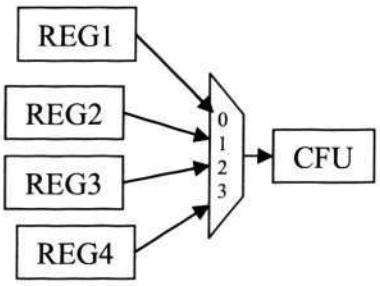


Figure 4.6 Illustration of 4 Registers feeding one CFU via Mux

CSDL is responsible for decoding the control signals (i.e. multiplexer selector signals, register enable signals, CFU opcode signals) from current state. As mentioned in Chapter 3, each component (CFU, Register, and Multiplexer) possesses a CIT (Control Information Table) that is generated in the resource binding stage. The CIT stores the output control signals for each state of the FSM.Table 4-3 shows the CIT of the multiplexer in Figure 4.6, which is used to select the inputs of the CFU from the four registers (REG1-4) in a time multiplexed manner. The CIT of the multiplexer indicates that REG1 is selected as the input to the CFU at State_1_2; REG2 is selected as the input to the CFU at State_1_4, etc. This information is converted into a set of unique values for driving the multiplexer's selector signals as shown in the rightmost columns of Table 4-3. Similarly, the CIT of a register specifies the register that is required to be written to in a particular state, and the CIT of the CFU specifies the CFUs that need to be activated in a particular state. Based on the CIT, the CSDL is constructed for area-time estimation.

CRATE is also capable of automatically generating the RTL codes of each component in the CSDL based on the CIT. The auto-generated RTL code for the example is shown in Figure 4.7. The RTL codes implement the decoder that generates appropriate multiplexer selector signals for each state. The implementation of the decoder is described using a "case" statement in VHDL where at any instance of time, a set of values are assigned to the multiplexer selector signals based on the current state.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 4-3 AN EXAMPLE OF CONTROL INFORMATION TABLE OF A MULTIPLEXER

States	Inputs	Select[1]	Select[0]
State_1_2	0	0	0
State_1_4	1	0	1
State_1_6	3	1	1
State_1_8	2	1	0
State_2_0	2	1	0
State_2_2	1	0	1
State_2_6	0	0	0

```
PortControlInt_Reg5_0: PROCESS (CUR_STATE)
                    BEGIN
               CASE (CUR_STATE) IS
                 WHEN State_1_2=>
          Mux_SelectInt_Reg5_0<="00";
                 WHEN State_1_4=>
          Mux_SelectInt_Reg5_0<="10";
                 WHEN State_1_6=>
          Mux_SelectInt_Reg5_0<="11";
                       ...
                 WHEN State_2_2>
          Mux_SelectInt_Reg5_0<="01";
                 WHEN State_2_6=>
          Mux_SelectInt_Reg5_0<="00";
                 WHEN  OTHERS =>
          Mux_SelectInt_Reg5_0<="00";
```

Figure 4.7 Decoder for multiplexer's select signals in VHDL

4.3. Estimation of FSM

In this section, we present the proposed area-time estimation technique for the FSM that has been constructed using the methods discussed in the previous section. The estimation of the control path relies on a set of equations that model the way the FSM is mapped onto FPGA. Hence, there is a need to first discuss how the RTL models for NSDL and CSDL is generated before we present the proposed estimation technique. The final estimation of the FSM is the sum of all the estimated values of the components in the FSM, namely the state register, NSDL and CSDL.

4.3.1. *Estimation of NSDL*

The example in Figure 4.8 shows an example of a FSM with four states, i.e. S23, S24, S25 and S26, and an input I. It can be observed that state S23 has a multi state transition to either state S24 or state S25 based on the input value of I. The remaining states have single state transitions.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

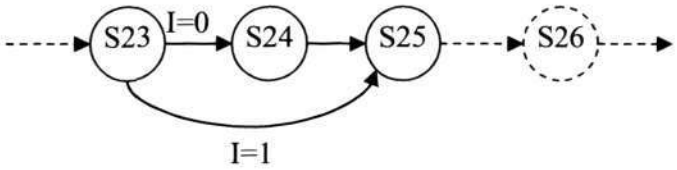


Figure 4.8 An example FSM

Using the one-hot encoding scheme, states S23, S24, S25 and S26 in Figure 4.8 are realized using four FFs (i.e. FF_{S23}, FF_{S24}, FF_{S25} and FF_{S26} as shown in Figure 4.9). It can be observed that inputs to the FFs corresponding to states S24 and S25 are driven by a decoder (i.e. D_{s24} and D_{s25}). These two states are the succeeding states of S23 which has the multi state transition. The decoders are used to compute the values for the FFs that they drive in the next clock cycle. Table 4-4 shows values of decoders at different states for different values of I. The combinational function to derive the value of the decoders is also shown. Since a K-input LUT in the FPGA logic element can implement any combinational function (with up to K inputs) and one output, two LUTs are required to implement the decoders D_{s24} and D_{s25}.

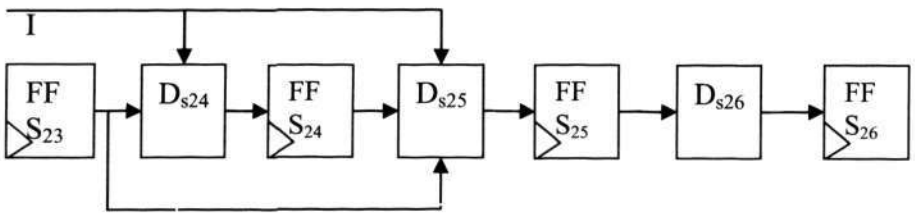


Figure 4.9 Implementation of the example FSM

TABLE 4-4 NEXT-STATE DECODER

Current State	I	Next-state	D_{s24}	D_{s25}	D_{s26}
S23	0	S24	1	0	0
S23	1	S25	0	1	0
S24	X	S25	0	1	0
S25	X	S26	0	0	1
S26	X	S27	0	0	0

D_{S24} = '1' when (FF_{S23}=1 and I = 0) else '0'.

D_{S25} = '1' when ((FF_{S23}=1 and I = 1) or FF_{S24}=1) else '0'.

D_{S26}= '1' when FF_{S25}='1'.

D_{S24} and D_{S25} can be mapped onto 1 LUT of FPGA. D_{S26} requires no LUT.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

As mentioned earlier, Trimaran employs two types of branches: unconditional and conditional. Unconditional branches usually do not require any decoders. An exception to this occurs when multiple unconditional branches have the same target address. This situation occurs when a "break" statement is encountered in C-codes. Since this occurrence is rare, we have assumed that all unconditional branches can be implemented without a decoder. A conditional branch (e.g. S23) leads to multi-state transitions, and this requires a decoder to be instantiated for each destination. In particular, each conditional branch instantiates two decoders as each branch have exactly two destinations. As discussed above, each decoder can be mapped onto a single LUT provided that it has less than 4 inputs (we assume that the target FPGA has only 4 input LUTs). This holds true when the destination state has at most three preceding states, so that the inputs of the decoder are driven by three state registers and one condition input. Our investigations reveal that this is the case with all the applications used in the experiments. Hence, we assume that the each decoder can be mapped onto one LUT and hence, the area of the decoders in terms of LUT (A_{NSDL}) can be calculated by simply computing the number of conditional branches (Cond_Branches) and substituting the value to the following equation:

$$A_{NSDL} = Cond_Branches*2 \qquad (4.1)$$

The estimated number of FFs estimation is equal to number of states as each state requires 1 FF. The critical path of the NSDL is basically the sum of the delay of the state registers and decoders. Since the critical path delay of the NSDL is always shorter than the critical path delay of the data-path, we do not perform delay estimation of the NSDL.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

4.3.2. *Estimation of CSDL*

The CSDL decodes the output control signals for each state using an OR-function. For example, based on the CIT in Table 4-3, the Boolean function of the selector signals for a 4-to-1 multiplexer, can be expressed as follows:

Select [1] = State_1_6 or State_1_8 or State_2_0;

Select [0] = State_1_4 or State_1_6 or State_2_2;

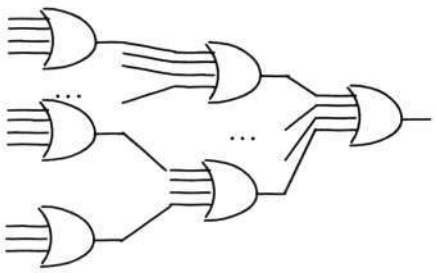


Figure 4.10 Tree of 4-input OR-gates

Since in the one-hot encoding scheme, a single bit is used to represent each state, the Boolean function for select[0] and select[1] can be implemented using a 3-input OR gate. These two signals can be implemented using two LUTs, where each LUT can be configured to implement a 3-input OR gate.

When the control signals are decoded from more than four states, an OR-tree is required. The OR-tree can be implemented by first forming groups of OR-functions with 4 inputs so that each group can be mapped onto a 4-input LUT. The 4-input OR-functions are then cascaded in the form of a tree as shown in Figure 4.10. Xilinx utilizes its fast carry-chain to implement the OR-tree with large depth as shown in Figure 4.11, which results in a faster implementation. In order to estimate the area-time of the OR-tree, CRATE evaluates both implementations (i.e. LUT and carry chain), and selects the estimated values for the faster implementation. It's worth mentioning that implementation is vendor-specific. For example, Altera's FPGA also employ fast-carry chain, but detailed mapping of logic equation on these carry chains are different from Xilinx. In the following sub-sections, we will discuss techniques for estimating the area-time of the OR-tree for each implementation using: 1) LUT and 2) carry-chain.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

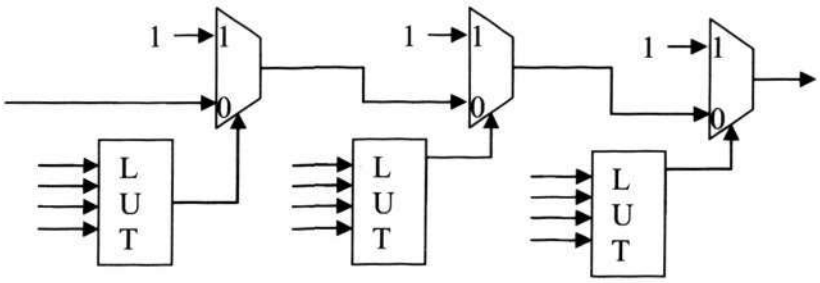


Figure 4.11 implementation of OR-tree using Xilinx carry chain

(a) Estimation of the OR-tree using LUT

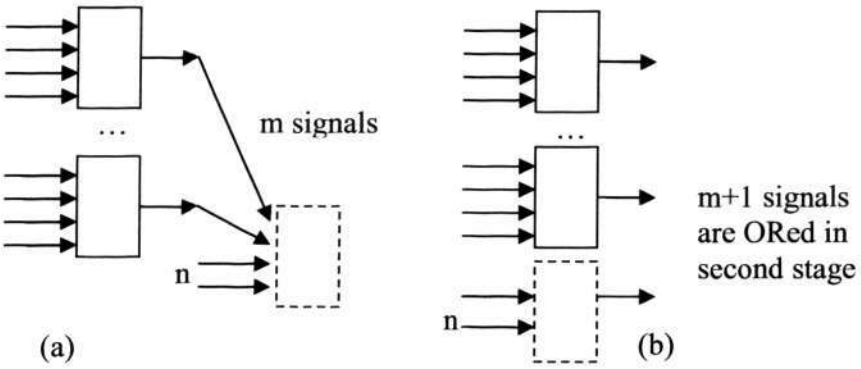


Figure 4.12 Forming OR-tree by cascading LUTs

This model is employed for both Xilinx and Altera FPGA. In order to estimate the area-time of the OR-tree implementation using LUT, we have developed a recursive algorithm as shown in Figure 4.13 for computing the number of LUTs and number of stages (logic depth) required to map the OR-tree onto LUTs. Each iteration of the algorithm computes for a single stage of the OR-tree, and can be briefly described as follows:

- Firstly, the number of inputs of the LUT (N) must be specified (e.g. N = 4) as this is the constraint that is used for forming the OR-function groups.
- If the number of inputs to the OR function used for decoding the control signals is at most N, only one LUT is required to implement the decoder.
- However if the number of inputs is larger than N, m OR-function groups with N-inputs are formed such that each group can be implemented using a single LUT. The number of OR-function groups is computed from x inputs as

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

follows, m=$\left\lfloor \frac{x}{M} \right\rfloor$. If the number of inputs is not fully divisible by N, then there exist a non-zero value of n, where n = x%N.

- If m+n is at most N, then the m outputs of the OR-function groups and the remaining n signals serves as input to an additional LUT (see Figure 4.12(a)). The algorithm terminates as all the stages of the OR-tree has been constructed.
- If m+n is larger than N, an additional LUT is used to implement an n-input OR-function, which in turns produces 1 output (see Figure 4.12(b)). The resulting m+1 signals (m signals from the OR-function groups + 1 signal from the additional LUT) serve as inputs to the next stage of the OR-tree. The algorithm repeats to construct a new stage for the OR-tree with these m+1 signals.

```
//N is number of input of LUT.
void Estimate_LUT_For_OR_Tree(int inputs, int * lut, int* depth)
{
If(inputs<=1) {return;} else
      If(inputs<=N)
      {//1 more stage;
      *lut++;*depth++;
      }
      Else
      {
            m=inputs/N;n=inputs%N;
            if(m>0) {(*depth)++;}
            if((m+n)>N)
                  {
                  Estimate_LUT_For_OR_Tree(m+1,lut,depth);
                  }
            else  {
                  (*lut)++;(*depth)++;
                  }
      }
}
```

Figure 4.13 Algorithm to compute LUT and depth of OR-tree

The function in Figure 4.13 returns 1) the number of LUTs required for implementing the OR-tree, which is denoted as A_{CSDL}, and 2) the number of stages or depth of the OR-tree, which is denoted as *d*. The delay of the OR-tree (D_{CSDL}) can be computed using equation 4.2. The timing parameters T_{LUTI2O} is obtained from the FPGA datasheet and $T_{INTERCONN}$ is obtained from components' database.

$$D_{CSDL} = d * T_{LUTI2O} + (d-1) * T_{INTERCONN} \qquad (4.2)$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

T_{LUT2O} is delay from inputs to outputs of FPGA's LUT obtained from FPGA data-sheet.

$T_{INTERCON}$ is the characterized interconnection delay described in Chapter 5.

(b) Estimation of OR-tree using carry chain

The structure of the OR-tree shown in Figure 4.10 may lead to high critical path delay when the number of stages is large. Xilinx utilizes the fast carry-chain to implement OR-trees with large number of logic depths. As shown in Figure 4.14, the outputs of the OR-function groups at the lower stage of the OR-tree (implemented using LUTs) are connected to the carry-chain which has been configured to implement an OR-function. This implementation is faster than the conventional OR-tree structure as all the OR-function groups at the lower stage of the OR-tree can be computed in parallel, and their results are fed to the fast carry chain to produce the final result.

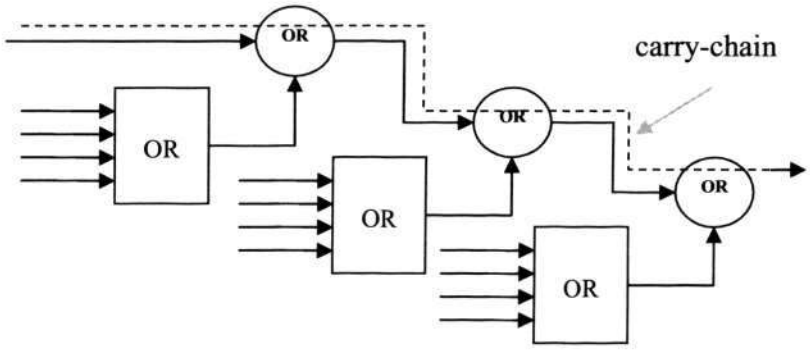


Figure 4.14 Implementation of an OR-tree using carry chain

Figure 4.15 illustrates how the carry chain can be configured to implement the OR- operation. This is achieved by tying one of the inputs of the multiplexers of the carry chain to logic '1'. Since the Boolean function for the multiplexer is:

$$O = S.A + \overline{S}.B$$

In which, S is the selector signal and A, B are inputs, when A is tied to logic '1' the function becomes:

$$O = S + \overline{S}.B \Leftrightarrow O = S + B$$

The above equation implements an OR-gate. It is evident that the number of LUTs required for this implementation is the number of OR-function groups in the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

lower stage of the OR-tree (see Figure 4.14). Hence the area of the OR-tree that is implemented using carry chain is estimated as:

$$A_{CSDL}(x) = \left\lceil \frac{x}{N} \right\rceil$$

$A_{CSDL}(x)$: number of LUTs for x-input or-tree

N : number of input of LUT

The delay of the OR-tree that is implemented using the carry-chain can be estimated as follows:

$$D_{CSDL}(x) = T_{LUT12O} + T_{MUXS2O} + (A_{CSDL}(x) - 1) * T_{CARRY}$$
$$T_{LUT12O} \approx T_{MUXS2O} \qquad (4.3)$$
$$\Rightarrow D_{CSDL}(x) \approx 2 * T_{LUT12O} + (A_{CSDL}(x) - 1) * T_{CARRY}$$

$D_{CSDL}(x)$: delay in ns for x-input or-tree

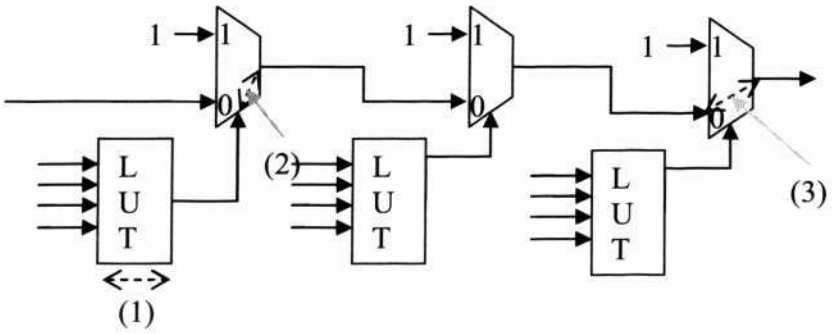


Figure 4.15 Implementation details of OR-tree using carry chain

4.3.3. *Estimation of Total Number of Slices for the Control Unit*

In the previous sections, we have presented the area estimation measures in terms of number of LUTs. However, the number of slices is a more realistic indication of the FPGA area utilization. For the remainder of the discussion, "Slice[1]" refers to a basic configurable logic cell which contains 1 LUT and 1 FF. If all the LUTs and FFs that are used for implementation resides on different slices (i.e. on a slice, either the LUT or FF is used), the total number of slices would be sum of total number of LUTs and total number of FFs. In order to maximize the utilization of the slices, the synthesis

[1] Xilinx's Slice contains 2 LUTs and 2 FFs

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

tool will try to pack LUTs and FFs that are used for implementation onto the same slices. Hence, there is a need to model the way in which LUTs and FFs are packed for estimating the number of slices. Our observation reveals that a LUT-FF pair is usually packed onto the same slice if the LUT drives the input of the FF as shown in Figure 4.16.

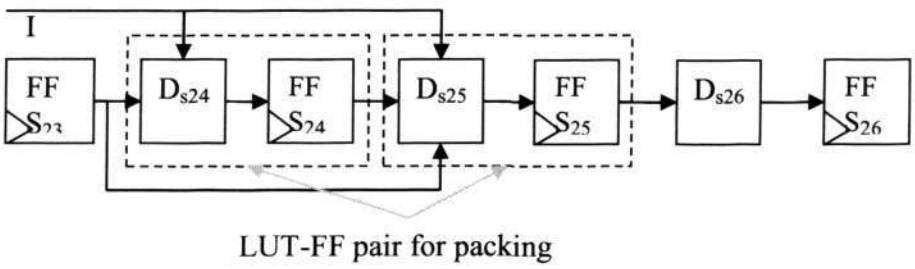


Figure 4.16 LUT-FF packing pattern

Based on this observation, we can now derive a mathematical model to estimate the total number of slices for the control-unit as: $A_{NSDL}^{LUT} + A_{CSDL}^{LUT} + A_{STATE}^{FF} - A_{PACKED}^{LUT}$. A_{STATE}^{FF} equals to total number of states as each FF is used to hold 1 state. LUT_FF_{PACKED} is the number of LUT-FF pairs, which are expected to be packed onto the same slice. As mentioned earlier, each conditional branch produces two LUTs that drive the inputs of two FFs. As the synthesis tool will try to pack each of this LUT-FF pair onto a single slice, the total number of LUT_FF pairs can be estimated as Cond_Branches*2. Therefore, total slices of control unit are:

$$A_{CONTROL} = A_{NSDL}^{LUT} + A_{CSDL}^{LUT} + A_{STATE}^{FF} - Cond_Branches * 2 = A_{CSDL}^{LUT} + A_{STATE}^{FF} \qquad (4.4)$$

4.4. Evaluation of the proposed models

In order to evaluate the accuracy of the proposed area-time estimation for control-path, we have used ten C-functions that are found in common embedded applications in EEMBC benchmarks [48] and Trimaran benchmark [47]. These ten functions were first compiled using Trimaran to produce the IR files. The techniques discussed in this chapter are then used to estimate the area-time of the control path.

In order to compare the estimation results with actual implementation results, we have also generated the RTL codes for the control path, which were implemented

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

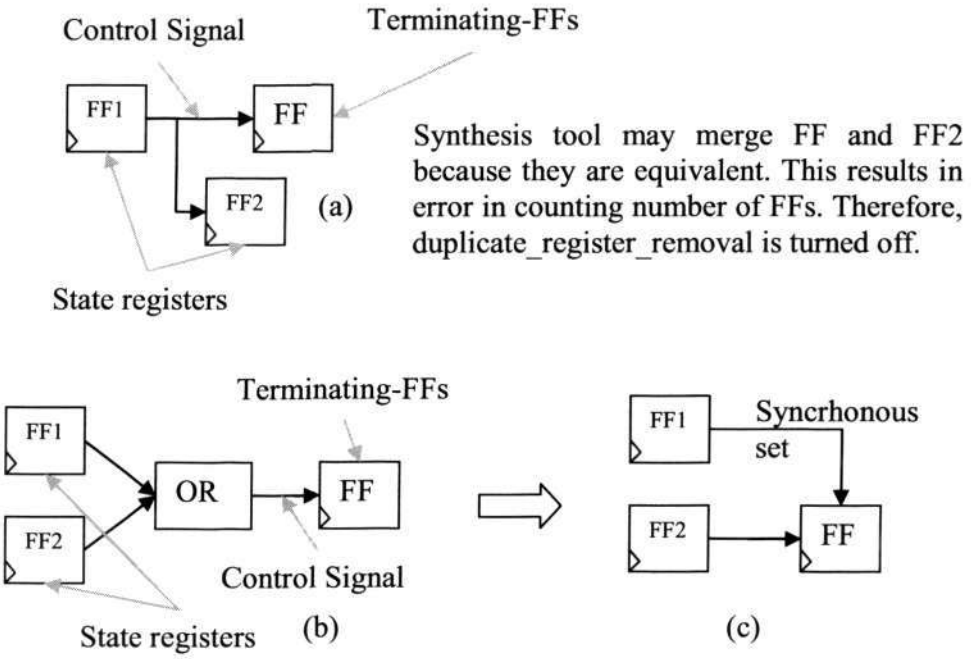
using Xilinx ISE 8.1. It is noteworthy that CRATE can automatically generate the RTL codes for the FSM models discussed in this section. However, there is a need to slightly modify the RTL codes of the control path to facilitate a fair comparison to be made with the estimated results. This is due to the fact that the Xilinx tool may introduce some undesirable changes to the circuit through its optimization features when we segregate the control path from the data-path in order to perform the estimation of the two independently.

For example, in order to measure the delay of the CSDL, we need to register the output of the circuit with a FF as shown in Figure 4.18. We denote this output register as the terminating FF. Otherwise, the synthesis tool will not report any delay for the design as it assumes that the CSDL does not drive any inputs. The delay of CSDL is then calculated by subtracting the delays of components that are not part of the proposed CSDL model from the critical path delay results reported by the tool. The delays that are subtracted include: 1) delay of terminating FFs, 2) interconnect delay from state registers to CSDL, and 3) interconnect delay from CSDL to terminating FFs, all of which are shown as dotted arrows in Figure 4.18. In addition, the following optimization options must be deactivated:

- duplicate_register_removal: Deactivating this option prevents the synthesis tool from merging the terminating FFs with state registers for the case shown in Figure 4.17.
- use_sync_set and use_sync_reset: These options are deactivated to prevent the synthesis tool from replacing the decoding circuit with the synchronous set/reset of terminating FFs.

The insertion of terminating registers is necessary to facilitate the estimation of the data-path and control path independently. Certain optimization features have been de-activated to enable the insertion of "terminating registers" in order to evaluate the proposed data-path estimation method. In the final design that combines the control and data-path, terminating registers are not required and hence, the corresponding optimizations will not be deactivated.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



Two implementations are interchangeable. In (b), LUT is replaced by feeding FF1 directly to synchronous set pin of FF. This may cause incorrect LUT counting. Therefore, use_sync_set/reset is turned off

Figure 4.17 Synthesis option settings

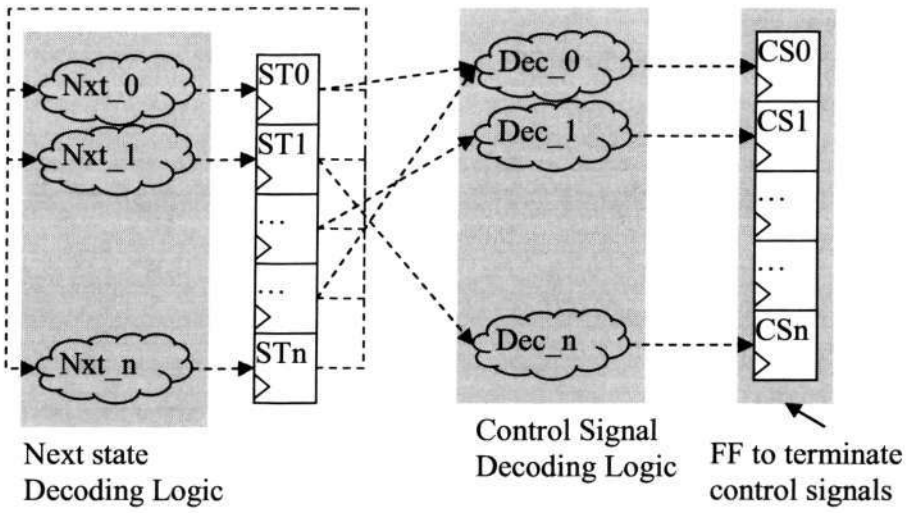


Figure 4.18 Setting up the decoder circuit for verification with the estimation technique

Table 4- 5 shows the comparison between the area-time estimation results and the actual implementation results of the auto-generated RTL codes. The target device is

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

the Xilinx Spartan 3-4. The first column lists the ten C-functions that have been used in the comparison. Columns 2 to 6 list the implementation results that are obtained from the reports generated by the Xilinx tool, while columns 7 to 11 list the corresponding estimation results that are obtained using the proposed methods in this chapter. In particular, columns 2 to 4 and columns 7 to 9 reports on the area utilization in terms of number of LUTs, FFs and slices. Columns 5 and 10 reports on the critical path delay of the control unit that is obtained from the Xilinx implementation and proposed estimation method respectively. Finally, the last two columns reports the absolute error of the proposed area-time estimation (in terms of slices and ns) when compared to the Xilinx implementation.

Xilinx's logic synthesis provides an option for us to specify the encoding scheme to be used for synthesizing the control path. However, we have not imposed on the tool to employ a specific encoding scheme as we want to provide the tool with the flexibility of choosing the best option during its optimization process. Since the number of states in the FSMs for the experiments considered ranged from medium size (i.e. 27 states) to very large size (i.e. 272 states), we expect that the tool would employ the one-hot encoding scheme as this leads to better performance as discussed in 4.1.1. Analysis of the implementation reports shows that our expectation is correct and the one-hot encoding scheme was selected by the tool for all the cases considered. Hence, this justifies the choice of employing the one-hot encoding scheme for the proposed FSM model.

For area estimation, the average percentage error in the number of LUTs and FFs is only about 7.5% and 3.8% respectively. These results demonstrate the accuracy of the analytical models used for estimating the LUTs and FFs of the FSM. The average area estimation error in terms of slices is about 10.3%. In addition, the estimation error (for number of slices) for the largest function used in the experiments (i.e. viterb00-_ViterbiDecoderIS136) is only about 4.3%. The worst case error achieved was 36% with a deviation of 12 slices from actual value. From the results, we believe that the proposed technique yields low average error in general and can cope with the large sized circuit. This is an advantage compared to previous works.

For estimation of the critical path delay, as can be observed in the last column of Table 4- 5, the absolute error of the critical path estimation is less than 1ns for all cases. We do not consider relative errors because the estimated values are too small

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

and in the order of a few LUTs depth. The significance of the proposed technique is that it can predict where the control path may likely affect the overall circuit performance. Although obtaining high relative error in cases of adpcm_decoder estimation and convolutional encoding, the error is within ±1 ns. This would mitigate the overall estimation error when data-path is included.

TABLE 4- 5 AREA-TIME ESTIMATION OF 10 CONTROLLERS FOR XILINX SPARTAN3-4

	Interconn=0.8			T_{luti2o}=0.551							
	Actual				Estimated					Absolute error	
Benchmark	luts	ffs	slices	delay	luts	Ffs	slices	delay	depth	Area (slices)	delay (ns)
autocor00-_fxpAutoCorrelation	21	27	16	0.551	21	25	19	0.551	1	3.00	0.00
adpcm_coder	64	79	51	2.005	64	77	57	1.902	2	6.00	0.10
adpcm_decoder	49	66	41	1.228	47	64	45	1.902	2	4.00	0.67
Comb_sort	21	34	20	0.551	21	32	21	0.551	1	1.00	0.00
sha_transform_sha	27	30	23	2.098	26	28	25	1.902	2	2.00	0.20
cjpegv2-_rgb_ycc_convert_jccolor	46	43	33	2.648	57	41	45	1.902	2	12.00	0.75
conven00-_convolutionalEncode	33	60	39	1.228	40	58	38	1.902	2	1.00	0.67
mpeg2-_dist1_motion	177	170	144	2.174	184	168	139	1.902	2	5.00	0.27
mpeg2-_dist2_motion	71	71	57	2.098	83	69	58	1.902	2	1.00	0.20
viterb00-_ViterbiDecoderIS136	365	272	287	2.996	361	270	300	2.83	29	13.00	0.17
								Average		6.38	0.30

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

4.5. Summary

In this chapter, we have described the proposed technique for area-time estimation of the control path in the h-VLIW architecture. The control path in the h-VLIW architecture is modeled as a FSM that consists of state registers, NSDL (Next State Decoding Logic) and CSDL (Control Signal Decoding Logic). The one-hot encoding scheme is adopted as our investigations show that this will lead to good performances for FSM with large number of states.

We have described techniques for constructing the NSDL and CSDL components from the application IR. The NSDL is constructed by analyzing the IR to extract information that indicates whether single or multi state transitions are required in the FSM. In particular, conditional branch related operations can lead to multi state transitions while other operations will lead to single state transitions. The CSDL is constructed using an OR-tree that decodes the control signal outputs based on the CIT of the data-path components. We have identified two OR-tree structures for implementing the decoder logic in the Xilinx tool, which are based on 1) LUT implementation and 2) Carry-chain implementation. The carry chain implementation is used for realizing OR-tree with large number of stages.

The proposed strategy performs area-time estimation of the NSDL and CSDL components separately by relying on a set of equations that model how the components are mapped onto the FPGA architecture. In particular, the number of FFs in the FSM is equal to the number of states. The number of LUTs required for the NSDL is estimated as twice the number of conditional branches found in the IR. The number of LUTs required for CSDL is estimated based on the OR-tree structure that provides the fastest implementation. Since the number of slices is a better indication of the FPGA area utilization, we have formulated a model that estimates the number of slices by predicting how the LUTs and FFs are packed onto the FPGA architecture.

In order to validate the accuracy of our estimation, we have implemented ten control-paths that are extracted from embedded functions using a commercial FPGA tool (i.e. Xilinx ISE 8.1). Experimental results show that the average area estimation error (for number of slices) is only about 10.3%, while the average delay estimation error is less than 1 ns. These results demonstrate the effectiveness of the proposed technique for area-time estimation of the control path.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 5 DATA-PATH AREA-TIME ESTIMATION

Figure 5.1 shows a typical data-path which includes multiplexers, registers and CFUs. The objective of high-level data-path estimation is to rapidly predict the FPGA area-time measures of the h-VLIW architecture's data-path model that was presented in Chapter 3. The estimated area-time measures consist of area utilization (in terms of FFs, total LUTs and slices), and critical path delay (in terms of ns), which is the longest estimated delay between any two registers in the data-path.

In this thesis, we have developed three approaches for data-path delay estimation:

1. Static delay estimation that relies solely on pre-characterized area-time parameters.
2. Dynamic delay estimation that relies on the pre-characterized area-time parameters and an analytical model that is based on slack distribution to estimate the post-implementation delay.
3. Dynamic delay estimation that relies on the pre-characterized area-time parameters and a simplified floor-planning procedure to estimate the post-implementation delay.

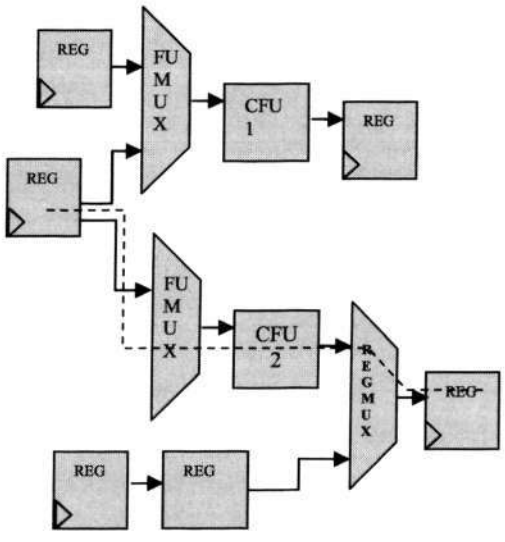


Figure 5.1 Data-path with typical connections

In this chapter, we present the first approach for estimating the data-path which relies on pre-characterized area-time parameters of a set of commonly used

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

components. We will first describe the component characterization process which initializes a database with architecture dependent area-time information of commonly used hardware components and interconnect delay. The component characterization process is fully automated and can be easily retargeted to a new FPGA architecture.

Next, we will present our approach to area estimation of the data-path. Based on the pre-characterized parameters, the proposed approach first computes the number of LUTs and FFs that are utilized by the components in the data-path of the h-VLIW architecture. The interconnection between the LUTs and FFs are then analyzed to determine the LUT-FF pairs that can be packed onto the same slice. Unlike most of the existing work in high-level area estimation (with the exception of Nayak et. al. [25]), the proposed technique is capable of estimating the area utilization in terms of slice count. In the remainder of this thesis, SLICE refers to a basic Configurable Logic Cell which contains one LUT and one FF. We will also present a technique for estimating the critical path delay of the data-path based on the pre-characterized timing parameters. We call this technique *static delay estimation* as it does not consider the placement of the components on FPGA. The critical path delay is computed by first estimating the delay for all possible register-to-register paths, and then identifying the longest delay among the paths.

Finally, we performed experiments using a wide range of functions to evaluate the accuracy of our approaches. Experimental results show that the proposed estimation method can achieve good results for area estimation. In particular, experiments with the functions considered show that the proposed method can achieve an average area estimation error of 5% with a worst error of only 7%. When compared to implementation results from a commercial tool, the static delay estimation method achieves an average error of 4% with a maximum error of less than 15% for the Virtex2Pro device. We believe that the accuracy of the delay estimation can be further improved by considering the placement of the components on the FPGA architecture.

5.1. Characterizing the RTL Components and Interconnect

The characterization of the RTL components is a one-time process that must be performed for each new target FPGA device. This process produces FPGA area and timing parameters for a set of commonly used components that will be stored in the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

pre-characterized component database. This section describes the procedures to characterize the RTL components of the h-VLIW architecture and the interconnect delay. It is noteworthy that these procedures are automated and can be easily re-targeted to a new FPGA device.

5.1.1. *Characterizing the RTL components*

The characterization of RTL components aims to obtain the area-time parameters for implementing a set of commonly used components (e.g. adder, subtractor, shifter, comparator, multiplexers, multipliers, etc.) on a specific FPGA architecture. As such, the characterization process utilizes commercial tools (e.g. Xilinx ISE and Altera Quartus) to synthesize and report the area-time information of the RTL components.

Figure 5.2 shows an example of a UUC (Unit-Under-Characterization) that will be used for characterization. To prepare the UUC for implementation with the commercial tools, registers are first inserted at the inputs and outputs of the UUC to create a synchronous design. The commercial tools will then synthesize the modified UUC and report the area-time results.

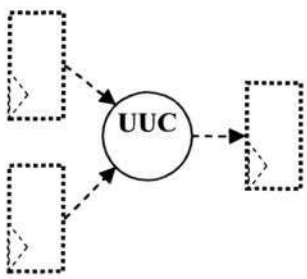


Figure 5.2 Inserting input-output registers to characterize RTL components

Figure 5.3 shows a schematic capture of a UUC (i.e. a 16-bit adder) from the synthesis tool. The critical path delay that is reported by the synthesis tool is shown in Figure 5.4. As can be observed, the reported critical path delay incorporates a series of gate delays that are contributed by the registers (FD), logic (LUT2, MUXCY, XORCY, etc.), and interconnect delay (Net Delay). Since we are only interested in the delay of UUC, we need to subtract the delay of registers and interconnect from the total delay of the critical path. For example the un-shaded region in Figure 5.4 indicates the delay of the adder which must be stored in the pre-characterized

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

component library. The remaining delay in the critical path, consisting of the source register delay (FD:C->Q), interconnect delay from registers to adder, and set-up time of destination register (FD:D), are subtracted from the total delay. In this example, the characterized delay of the 16-bit adder is 2.752 ns (4.493 - 0.370 - 0.568 - 0.803 ns). The characterized area of the adder can also be obtained from the synthesis report and stored in the pre-characterized component library.

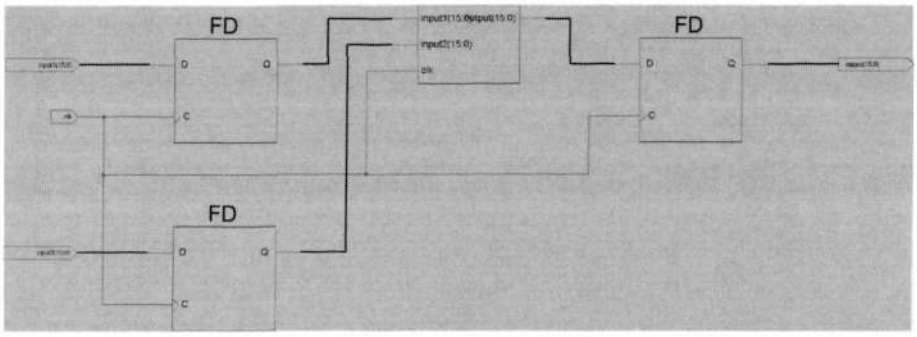


Figure 5.3 Schematic capture of a 16-bit adder that is used for characterization

Data Path: int_input1_0 to output_15

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FD:C->Q	1	0.568	0.803	int_input1_0 (int_input1_0)
LUT2:I0->O	2	0.439	0.000	UT/Madd_output_lut<0> (int_output<0>)
MUXCY:S->O	1	0.298	0.000	UT/Madd_output_cy<0> (UT/Madd_output_cy<0>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<1> (UT/Madd_output_cy<1>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<2> (UT/Madd_output_cy<2>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<3> (UT/Madd_output_cy<3>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<4> (UT/Madd_output_cy<4>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<5> (UT/Madd_output_cy<5>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<6> (UT/Madd_output_cy<6>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<7> (UT/Madd_output_cy<7>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<8> (UT/Madd_output_cy<8>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<9> (UT/Madd_output_cy<9>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<10> (UT/Madd_output_cy<10>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<11> (UT/Madd_output_cy<11>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<12> (UT/Madd_output_cy<12>)
MUXCY:CI->O	1	0.053	0.000	UT/Madd_output_cy<13> (UT/Madd_output_cy<13>)
MUXCY:CI->O	0	0.053	0.000	UT/Madd_output_cy<14> (UT/Madd_output_cy<14>)
XORCY:CI->O	1	1.274	0.000	UT/Madd_output_xor<15> (int_output<15>)
FD:D		0.370		output_15
Total		4.493ns	(3.691ns logic, 0.803ns route)	
			(82.1% logic, 17.9% route)	

Figure 5.4 Timing report of Xilinx Synthesis Tool for the 16-bit adder

The area-time parameters of other RTL components are characterized using the approach described above. Table 5-1 lists the area-time parameters (in terms of number of LUTs and ns) of commonly used RTL components for three Xilinx FPGA families: Spartan3-speedgrade 4 (xc3s5000-4fg1156), Virtex2pro speedgrade-6

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

(xc2vp70-6ff1704) and Virtex 4 speedgrade-12 (xc4vlx15-12ff676). Appendix A.1 contains the procedures and perl script that is used to invoke the synthesis engine of the commercial tools to characterize the set of UUC in Table 5-1 for different FPGA architectures.

TABLE 5-1 AREA-TIME INFORMATION OF RTL COMPONENTS FOR DIFFERENT FPGA FAMILIES

	xc3s5000-4fg1156		xc2vp70-6ff1704		xc4vlx15-12ff676	
	Area (LUT)	Time (ns)	Area (LUT)	Time (ns)	Area (LUT)	Time (ns)
Adder_32bit	32	3.875	32	2.796	32	1.718
Subtractor_32bit	32	3.875	32	2.796	32	1.718
Comparator_NEQ_32bit	16	2.228	16	2.014	16	1.101
Comparator_EQ_32bit	16	2.228	16	2.014	16	1.101
Comparator_LT_32bit	33	5.427	33	3.565	33	2.404
Comparator_GT_32bit	33	5.427	33	3.565	33	2.404
Comparator_LEQ_32bit	32	3.252	32	2.676	32	1.645
Comparator_GEQ_32bit	32	3.252	32	2.676	32	1.645
RA_Shifter_32bit	171	6.548	172	3.74	172	2.686
R_Shifter_32bit	261	8.031	269	5.089	261	3.429
LA_Shifter_32bit	264	7.166	286	4.648	258	3.501
L_Shifter_32bit	264	7.166	286	4.648	258	3.501
mux_2to1_32bit	32	0.551	32	0.313	32	0.147
mux_4to1_32bit	64	0.911	64	0.653	64	0.438
mux_8to1_32bit	128	1.253	128	0.972	128	0.738
mux_16to1_32bit	256	1.595	256	1.291	256	1.038
mux_32to1_32bit	512	5.698	512	3.117	512	2.985
Multiplier_32bit	497	15.26	497	10.8	498	6.05
EmbMultiplier_32bit	30	10.611	30	8.257	498	7.5

5.1.2. *Charactering the Interconnect Delay*

The interconnect delay often dominates the total delay in FPGA designs. Hence, there is a need to take into account the interconnect delay during delay estimation of the data-path. This section describes the procedure for characterizing the interconnect delay. Two types of interconnect delays are characterized: 1) static interconnect delay, and 2) dynamic interconnect delay. The static interconnect delay is used for delay estimation that does not take into account the placement of the components on the FPGA architecture (e.g. the approach described in this chapter). The dynamic interconnect delay is used when the placement of the components on FPGA can be predicted (e.g. in the approach that employs the simplified floor planning process

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

described in the next chapter). The characterized delays are empirically obtained by manually placing eight set of circuits, where each set consist of a FF driving another FF (see Figure 5.5), at a specific distances apart.

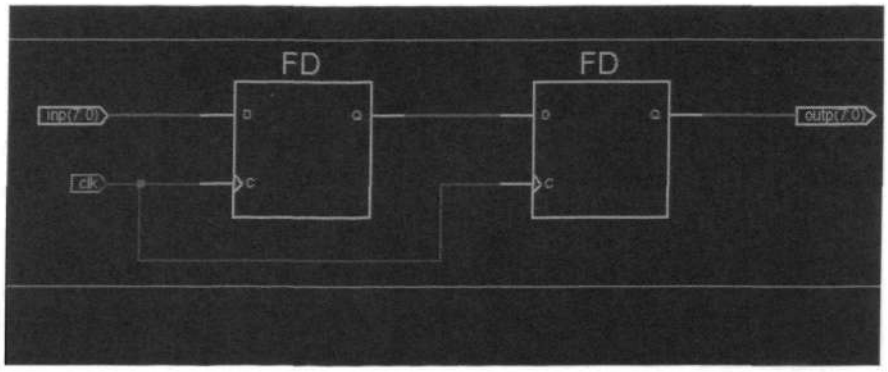


Figure 5.5 Flip-flop circuit for interconnect characterization

(a) Characterizing the Static Interconnect Delay

Static interconnect delay is obtained by measuring the delay between two FFs that are placed one CLB[2] (Configurable Logic Block) apart. In order to characterize the static interconnect delay, each set of FF-pairs are manually placed one CLB apart. The commercial tool is then used to perform interconnect routing between the FF-pairs. The average post-routed interconnect delay between the eight FF-pairs are then obtained as the characterized static interconnect delay. Figure 5.6 shows an image captured from the Xilinx FPGA editor that was used to manually place the FFs into CLB, which are shown as small black rectangles. The post-routed interconnections between the FFs are also shown. Figure 5.6 also shows the report generated by the FPGA editor tool that lists the post routed interconnect delay between a FF pair.

[2] "CLB" is a term used for Xilinx devices which contain programmable Look-up-table (LUT) and FFs. "LAB" is the equivalent term for Altera devices.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

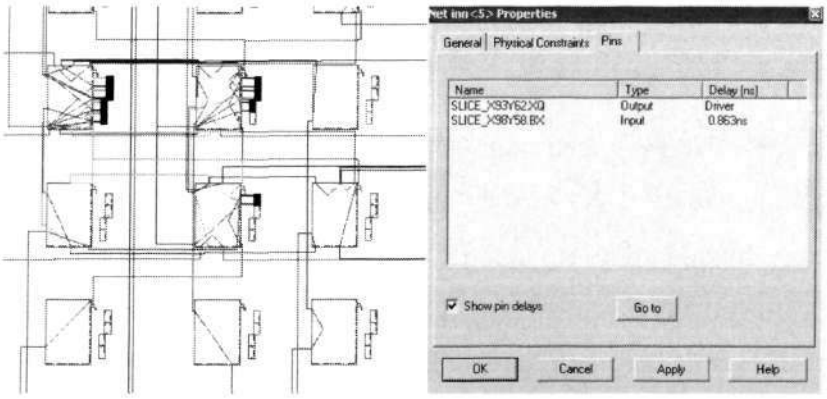


Figure 5.6 Placed and Routed circuit for interconnect characterization and net delay report

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

(b) Characterizing the Dynamic Interconnect Delay

The dynamic interconnect delay can be obtained by measuring the delay between two FFs that are placed multiple CLBs apart. The procedure to calculate the dynamic interconnect delay is similar to the procedure used for obtaining the static interconnect delay. Instead of placing the FF pairs one CLB apart, the characterization of the dynamic interconnect delay requires the FF pairs to be placed within a range of Manhattan distances (in terms of number of CLBs) apart. This process is repeated for distance ranges in steps of five. The average post routed delay that is measured for each range is stored in a form of a lookup table to be used during the estimation process.

Table 5-2 shows characterized delays for the static interconnect delay and the dynamic interconnect delay for three families namely spartan3 speedgrade-4 (SP3-4), virtex2pro speedgrade-6 (V2p-6) and virtex4 speedgrade-12 (V4-12). It can be observed that the dynamic interconnect delay varies with the different range of distances.

TABLE 5-2 CHARACTERIZED INTERCONNECT DELAY FOR THREE XILINX FPGAS

		Delay (ns)		
Distance (CLBs)		SP3-4	V2p-6	V4-12
Dynamic Interconnect	0 to 5	0.8	0.5	0.5
	5 to 10	1.5	0.9	0.7
	10 to 15	2.1	1.3	1
	15 to 20	2.6	1.6	1.1
	20 to 25	2.7	1.7	1.2
	25 to 30	2.8	1.8	1.3
Static Interconnect		0.8	0.5	0.38

Conservative estimation is a preferable approach. Therefore, it is suggested that a small offset (approximately 10%) should be added to the above characterized values to estimate conservatively.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

5.2. Area Estimation

Based on the area parameters in the pre-characterized component library, the number of LUTs and FFs that are required for the RTL components in the h-VLIW architecture can be easily computed. The total number of LUTs is the sum of LUT consumed by the multiplexers and CFUs. The total number of FFs is the sum of FFs consumed by the registers.

CRATE also analyzes the interconnection between the LUTs and FFs in order to predict the packing of LUT-FF pairs onto FPGA slices. If all the FFs (FF in equation 5.1) and LUTs (LUT in (5.1)) are predicted to be placed on separate slices, the total slice count is the sum of FFs and LUTs. However, if some of the LUT-FF pairs are predicted to be packed onto same slices, then the total number of slices ($AREA_{SLICE}$ in (5.1)) is computed as shown in (5.1). Unlike the work in Nayak et. al [25], this strategy for estimating the number of slice utilization avoids the need to recalibrate the estimation model for different devices.

$$AREA_{SLICE} = FF + LUT - LUT_FF_{packed}. \qquad (5.1)$$

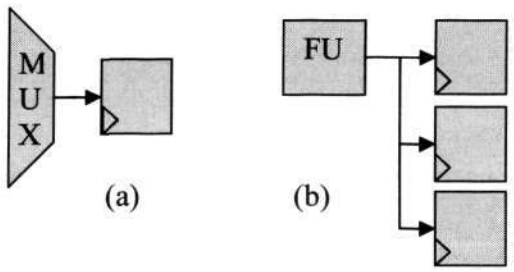


Figure 5.7 Cases for detection of LUTs-FFs packing

Hence, there is a need to predict the number of packed LUT-FF pairs in order to estimate the slice count of the data-path. Figure 5.7 shows two scenarios where the FFs will be packed onto the same slice with the driving components. In case (a), the FF input is driven by a multiplexer that is implemented using a LUT. During place and route, this FF will be packed onto the same slice with the multiplexer. In case (b), if the CFU (implemented using a LUT) is driving the inputs of multiple FFs, one of the FFs will be packed onto the same slice with the CFU. CRATE analyzes the data-path to identify these cases in order to estimate the number of LUT-FF pairs that will be packed onto the same slices.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

It is noteworthy that over-packing LUT-FF pairs onto the same slices may increase the routing demands to these slices. This could incur high interconnect delay in the data-path due to the increase in the routing overhead. FPGA tools can also pack unrelated LUTs and FFs (LUTs and FFs that are not directly connected to one another) onto the same slice if there are available routing resources to facilitate this. As these factors can affect the accuracy of the estimation, they will be considered in future versions of CRATE.

Figure 5.8 shows the pseudo-code for estimating the number of slices of the data-path that takes into account the two cases in Figure 5.7. The variables LUT and FF denote the total number of LUTs and FFs that are consumed by the data-path. Packed_FF refers to the number of LUT-FF pairs that are predicted to be packed onto the same slice. For the first case (Figure 5.7 (a)), the algorithm evaluates each register in the data-path to increment the number of FFs (line #5) by adding FFs consumed by each register (REG.FF), and to determine if the register is driven by a multiplexer (i.e. if number of register inputs is larger than 1). If the register is driven by a multiplexer, Packed_FF is incremented accordingly (line 8). For the second case (Figure 5.7(b)), the algorithm evaluates each CFU to increment the number of LUTs. If the CFU drives a register directly (i.e. without an intermediate multiplexer), it is assumed that a LUT-FF pair is found and the algorithm increments Packed_FF accordingly (line 18). The remaining registers that are driven by the CFU which corresponds to an existing LUT-FF pair will not be evaluated further.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

```
#1  LUT=0;FF=0;Packed_FF=0;
#2  //case (a)
#3  For each Register REG
#4  {
#5        FF+= REG.FF;
#6        If(REG.Inputs.Size>1)
#7        {
#8        Packed_FF+= REG.FF;
#9        }
#10 }
#11 //case (b)
#12 For each functional unit FU
#13 {
#14       LUT+=FU.LUT
#15       For each destination REG
#16             {if (Reg.Inputs.size==1)
#17                   {
#18                   Packed_FF+=REG.FF;
#19                   break;
#20                   }
#21             }
#22 }
#23
#24 SLICE=LUT+FF-Packed_FF
    If (XILINX) SLICEs=SLICEs/2;
```

Figure 5.8 Algorithm for area estimation of data-path

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

5.3. Delay Estimation

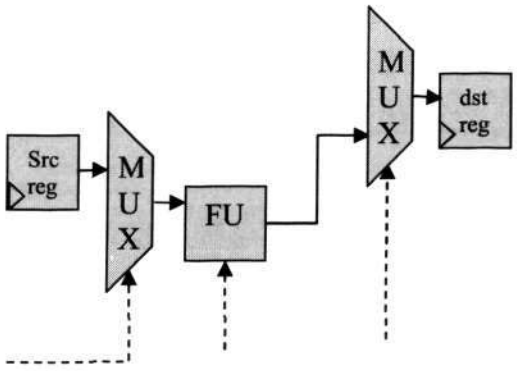


Figure 5.9 Model of a delay path

In order to estimate the critical path delay of the data-path, it is necessary to first estimate the delay of all the register-to-register paths in the data-path. We call each register-to-register path a "delay-path". The critical path delay is the longest delay among the delay-paths.

Figure 5.9 shows an example of a delay path, which comprises of a source-register, CFU-multiplexer, CFU, destination-register-multiplexer and destination-register. There are several delay paths that could end at the same destination register (shown as dotted lines). Some of these paths originate from other source registers in the data-path, while others originate from the control path. For data-path estimation, we are only interested in delay paths that originate from registers in the data-path.

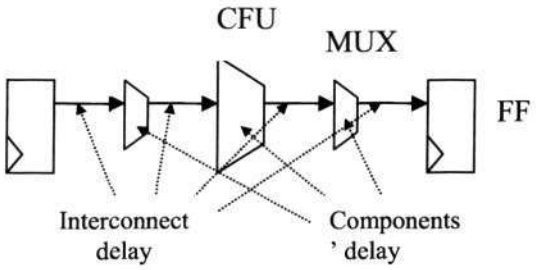
The delay estimation is computed by first summing the pre-characterized delay of the LUT-based components and the static interconnect delay between these components in the individual delay-paths. Note that this approach uses the pre-characterized static interconnect delay (discussed in Section 5.1.2) and ignores the placement of the RTL components on the FPGA fabric. Specifically, this approach assumes that the connected components are placed on adjacent CLBs. The critical path delay of the data-path is then identified as the longest estimated delay among the delay-paths.

Since this approach only uses the static interconnect delay, we denote this approach as the static delay estimation method. In the following subsections, we will present estimation models for computing the static delay estimation of delay-paths that are found in the h-VLIW architecture. In particular, the delay-paths can be categorized

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

into the following: 1) delay-paths with non-pipelined CFUs, and 2) delay-paths with pipelined CFUs.

5.3.1. *Static Estimation Delay for Delay-Paths with Non-Pipelined CFUs*

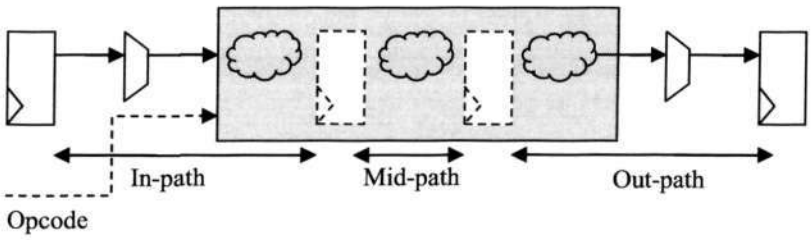


$$D_{path} = FF_{Clk2q} + D_{int} + D_{comp} + FF_{setup} \quad (5.2)$$

Figure 5.10 Estimation model for Delay-path with Non-pipelined CFUs

Figure 5.10 shows a delay-path with non-pipelined CFUs. As can be observed the delay path has only two FFs (or registers), one at the beginning of the path and the other at the end. The total delay of this path can be computed by summing up the pre-characterized component delays (D_{comp} that is contributed by the CFUs, multiplexers and FFs) with the total static interconnect delay (D_{int}) between the components. All the timing parameters mentioned above are obtained from the pre-characterized component library. The equation above (5.2) shows the estimation model for estimating the delay (D_{path}) of a delay-path with non-pipelined CFUs.

5.3.2. *Static Estimation Delay for Delay Paths with Pipelined CFUs*



$D_{in\text{-}path} = FF_{Clk2q} + D_{int} + D_{mux} + D_{fu\text{-}in\text{-}path} + FF_{setup}$

$D_{out\text{-}path} = FF_{Clk2q} + D_{int} + D_{fu\text{-}out\text{-}path} + D_{mux} + FF_{setup}$

$D_{mid\text{-}path}$ = Obtained from characterization

$D_{path} = MAX\ (D_{in\text{-}path}, D_{out\text{-}path}, D_{mid\text{-}path})$

Figure 5.11 Estimation model for Delay path with pipelined CFUs

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

The delay-path with pipelined CFUs has been discussed in Chapter 3. As can be observed from Figure 5.11, the delay path with pipelined CFUs comprises the in-path, mid-path and out-path. The estimated delay of the delay-path is the maximum delay of in-path, out-path and mid-path.

5.3.3. *Estimation delay caused by CFU fan-out*

Additional interconnect delay is incurred for CFUs that drive multiple destination registers. This is due to the fact that the FPGA implementation tool will find difficulty in placing these registers close to the CFU. We model this effect as follows. We assume that the registers are placed horizontally on consecutive columns of islands as shown in Figure 5.12. The worst interconnect delay due to fan-out is caused by the register that is placed farthest from the CFU. We predict the worst distance and obtain the corresponding pre-characterized delay from the pr-characterized component library (see Table 5-2). We opt to use a table to predict the delay instead of relying on the Mahattan-distance, as the Mahattan-distance has been demonstrated to have a low correlation with the FPGA delay [43].

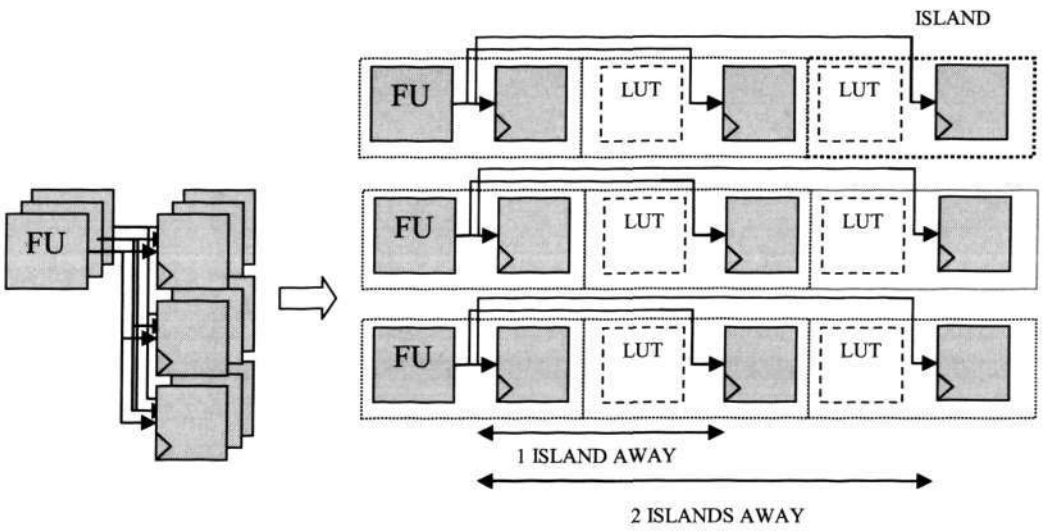


Figure 5.12 Placement of components with fan-out effect

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

5.4. Evaluation of the proposed approaches

5.4.1. *Area Estimation*

In order to evaluate the accuracy of the proposed area estimation, we have created ten dummy C-functions for our experiments. We have employed dummy functions in this experiment, as functions from real applications are often subjected to logic optimizations by the commercial tools. As the proposed method in this chapter have not taken into account the logic optimization effects, there will be a large discrepancy between the estimated results and actual implementation results if functions from real applications are used. It is worth mentioning that the final versions of CRATE take into account the logic optimization effects during estimation, and this will be discussed in Chapter 8. However for this experiment we have generated dummy circuits that have little opportunities for logic optimizations by the commercial tool.

Figure 5.13 shows examples of dummy functions that have been used in the experiments to compute the output 'q' from several inputs (a, b, c ...k, l). These functions were first compiled with Trimaran to produce the Elcor IR files. The data-path is then constructed based on the methods described in Chapter 3. We then employed the proposed technique to estimate the area of the data-path. We have also generated the RTL codes of the data-path for implementation using the commercial tool. The RTL codes were synthesized and mapped onto the Virtex2pro device using Xilinx ISE 8.1. In order to increase the packing effort of the mapping procedure, an area constraint of 1.2 times the estimated area is applied to the tool.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

```
void DummyAlgo0(void)//0multiply,48 addition
{
        int a,b,c,d,e,f,g,h,i,j,k,l;//inputs
        int m,n,p,q;//outputs
        m=a+b+c+d+e+f+g+h+i+j+k+l+12;
        n=m+a-b+c-d+e-f+g-h+i-j+k+42;
        p=m-n+b-a+d-c+f-e+h-j+k-i+1;
        q=m-n+p-a+b-3+d-c+f-e+i-k+j;
}
void DummyAlgo1(void)//1multiply,48 addition
{
        int a,b,c,d,e,f,g,h,i,j,k,l;//inputs
        int m,n,p,q;//outputs
        m=a+b+c+d+e+f+g+h+i+j+k+l+12;
        n=m*a-b+c-d+e-f+g-h+i-j+k+42+l;
        p=m-n+b-a+d-c+f-e+h-j+k-i+1;
        q=m-n+p-a+b-3+d-c+f-e+i-k+j;
}
void DummyAlgo2(void)//2multiply,48 addition
{
        int a,b,c,d,e,f,g,h,i,j,k,l;//inputs
        int m,n,p,q;//outputs
        m=a+b+c+d+e+f+g+h+i+j+k+l+12;
        n=m*a-b+c-d+e-f+g-h+i-j+k+42+l;
        p=m*n+b-a+d-c+f-e+h-j+k-i+1-m;
        q=m-n+p-a+b-3+d-c+f-e+i-k+j;
}
```

Figure 5.13 3 samples of dummy circuits for evaluating the area estimation

Table 5-3 reports the experimental results for the area that is estimated using the proposed method (Estimated) and the post-map area that is reported by Xilinx ISE (Post-Map). The percentage estimation error between Estimated and Post-Map is also shown. The size of the functions ranged from medium size (1037 slices) to large size (3000 slices; total slice of the FPGA is about 5120). As can be observed in Figure 5.14, the proposed technique is able to achieve very consistent results for the functions considered. In particular, the average estimation error for LUT and FFs is only 3% and 3.5% respectively. The average estimation error for number of slices is only about 5%. In addition, the maximum slice estimation error was found to be less than 7%. This results which is based on a wide range of circuit sizes, demonstrates that the proposed technique for data-path area estimation is highly accurate and scalable. In Chapter 9, we will show that the proposed area estimation technique can achieve similar results for functions that are obtained from realistic applications.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 5-3 EXPERIMENTAL RESULTS FOR AREA ESTIMATION

	Post-Map			Estimated			Error (%)		
	LUT	FF	SLICE	LUT	FF	SLICE	LUT	FF	SLICE
1	1445	2065	1037	1501	1952	976	3.88	5.47	5.88
2	5786	2801	3099	6006	2848	3211	3.80	1.68	3.61
3	5531	2538	2955	5750	2592	3083	3.96	2.13	4.33
4	5302	2244	2829	5465	2304	2941	3.07	2.67	3.94
5	4984	1914	2618	5145	1984	2781	3.23	3.66	6.21
6	4856	1750	2912	5017	1824	2701	3.32	4.23	7.26
7	1976	2161	1252	1998	2048	1207	1.11	5.23	3.59
8	2538	2321	1537	2527	2176	1472	0.43	6.25	4.26
9	3070	2354	1688	3024	2272	1720	1.50	3.48	1.90
10	4726	2673	2540	5012	2656	2714	6.05	0.64	6.85
						Average	3.04	3.54	4.78

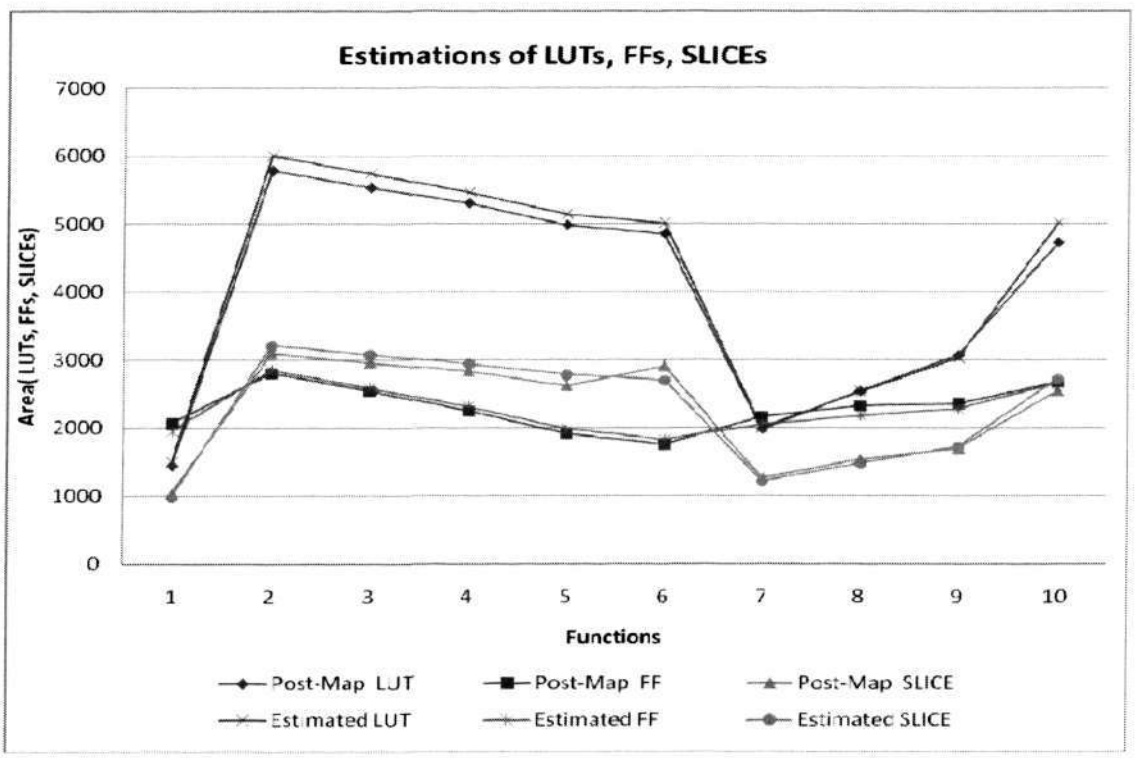


Figure 5.14 Area estimation results

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

5.4.2. *Delay Estimation*

In order to evaluate the proposed static delay estimation, we carried out experiments using twelve functions that are found in widely used embedded applications. Table 5-4 provides a brief description of these functions. Each of this function is compiled using Trimaran and the data-path of the h-VLIW architecture is constructed based on the methods described in Chapter 3. Static delay estimation is then performed on the data-path. In order to compare with the post-PAR results of the commercial tool, we generated RTL codes for the data-paths to be implemented using Xilinx ISE. We have enabled the "keep_hierarchy" option in Xilinx ISE to prevent the synthesis tool from introducing undesirable optimizations to the netlists. Next, we used the Xplorer Script [52] to implement the data-path by gradually tightening the timing constraint.

Table 5-5 shows the critical path delays of the C-functions that are obtained from 1) the proposed static delay estimation (Sim), 2) post-synthesis delay reports of Xilinx ISE (Syn), and 3) post place and route delay reports of Xilinx ISE (PAR). The synthesis results are often used to estimate the final delay of the circuit so that the design cycle can be shortened by preventing unnecessary place and route process. It is noteworthy that the static delay estimation can be performed in a significantly shorter amount of time than the synthesis process. In order to compare the reliability of the estimation results between the static delay estimation and the synthesis engine, the percentage estimation error of Static and Syn are calculated with respect to the PAR delay for two FPGA families, i.e. Spartan3 speedgrade (-4) and Virtex2Pro speedgrade (-6).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 5-4 EMBEDDED FUNCTIONS USED IN EXPERIMENT

1	Random Algorithm	Simple C-codes to compute 1 output from 4 inputs to increase the diversity of benchmarks set
2	Matrix multiplier	Matrix multiplication, inner loop
3	mpeg2-_bdist1_motion	A function in mpeg2 application
4	mpeg2-_bdist2_motion	A function in mpeg2 application
5	mpeg2-_dct_type_estimation	A function in mpeg2 application
6	mpeg2-_dist1_motion	A function in mpeg2 application
7	mpeg2-_dist2_motion	A function in mpeg2 application
8	mpeg2-_idctcol	Inverse cosine transform, column
9	mpeg2-_idctrow	Inverse cosine transform, row
10	sha_transform	Sha transform
11	adpcm_coder	ADPCM coder
12	adpcm_decoder	ADPCM decoder

For the Virtex2Pro device, the results of the static delay estimation are comparable to the post synthesis results of Xilinx ISE in most of the cases. Both approaches (Static and Syn) have an average error of about 5% and maximum error of about 14%. It can also be observed from Figure 5.15 that the error trends of Static and Syn are fairly consistent. For example, both approaches yield very high errors for function 6, and exhibit similar trend for the first three (functions 1-3) and last four functions (functions 9-12). The large errors in function 6 and 12 can be attributed to the fact that both Static and Syn do not take into account the post place and route effects on the delay.

For the Spartan device, both the static delay estimation (Sim) and Xilinx synthesis engine (Syn) resulted in higher errors when compared to the results of the Virtex2Pro device. In particular, the average and maximum percentage error of Static has increased to 6.5% and 24% respectively. The average percentage error of Syn has also increased to 8%, which is slightly higher than the average percentage error of Static. We believe that the reason for the increased in the estimation error for both approaches is due to the fact that the Spartan3's routing architecture is inferior to the routing architecture of the Virtex2Pro family. The highest percentage errors in the experiments for the Spartan3 device are also associated with functions 6 and 12. Note that the static delay estimation can lead to very high inaccuracies for certain functions (e.g. over 24% percentage error for function 6). We believe that improvements can be made by taking into account the post placement effects of the components on the FPGA architecture. We will describe two strategies for this in the next chapter.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 5-5 EXPERIMENTAL RESULTS FOR ESTIMATION OF CRITICAL PATH USING SIMPLE MODEL

	Virtex2Pro-6					Spartan3-4				
	Absolute values			Errors		Absolute Values			Errors	
	Sim (ns)	Syn (ns)	PAR (ns)	Sim (%)	Syn (%)	Sim (ns)	Syn (ns)	PAR (ns)	Sim (%)	Syn (%)
1	11.47	11.22	11.31	1.41	0.80	16.26	18.7	16.85	3.50	10.98
2	9.61	9.62	9.46	1.59	1.69	13.54	14.14	13.28	1.96	6.48
3	10.37	10.41	9.95	4.22	4.62	14.72	15.85	14.36	2.51	10.38
4	10.37	11.54	10.61	2.26	8.77	14.72	17.42	15.49	4.97	12.46
5	9.61	9.74	9.91	3.03	1.72	13.54	14.71	14.41	6.04	2.08
6	6.55	6.65	7.7	14.94	13.64	9.16	10.62	12.06	24.05	11.94
7	10.71	10.97	10.42	2.78	5.28	15.08	16.62	14.98	0.67	10.95
8	10.71	11.22	11.15	3.95	0.63	15.08	17.17	15.78	4.44	8.81
9	10.71	11.22	10.98	2.46	2.19	15.08	17.17	15.79	4.50	8.74
10	5.83	5.98	6.22	6.27	3.86	8.13	9.45	9.04	10.07	4.54
11	6.06	6.41	6.78	10.62	5.46	9.6	11.5	10.2	5.88	12.75
12	5.83	6.01	6.78	14.01	11.36	8.96	10.54	10	10.40	5.40
				5.63	5.00				6.58	8.79

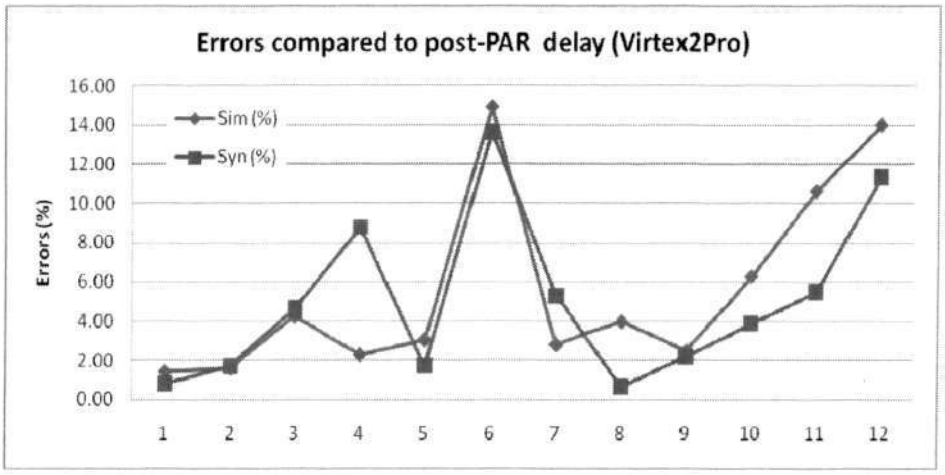


Figure 5.15 Errors compared to Post-PAR delay of Simple model and Synthesis Tool (Virtex2Pro-6)

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

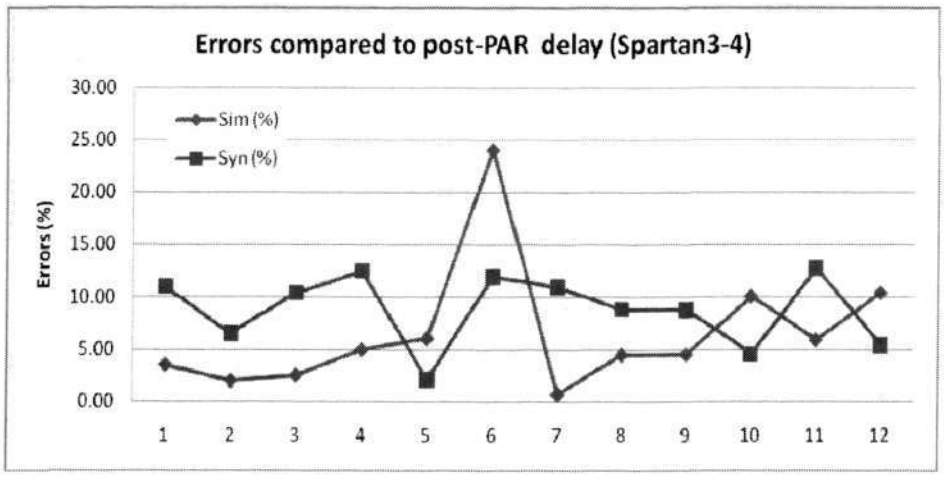


Figure 5.16 Errors compared to Post-PAR delay of Simple model and Synthesis Tool (Spartan3-4)

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

5.5. Summary

In this chapter we have presented our first approach for estimating the data-path of the h-VLIW architecture by relying on a set of pre-characterized area-time parameters. A one-time effort is required to characterize a set of commonly used components and the static interconnect delay for a particular FPGA device.

We have proposed a technique to estimate the number of slices from the total LUTs and FFs consumed by the data-path. The key advantage of the proposed approach for area estimation is the ability to report the estimated area in terms of slice count without the need to recalibrate the estimation model for different functions. This is achieved by analyzing the data-path to identify scenarios which indicate the potential for packing LUT and FF onto the same slice. We have performed area estimation on ten dummy circuits, which ranged from 1500 Slices (~250K gates) to 3500 slices (550K gates). Experimental results for the proposed area estimation technique are very encouraging. In particular, the proposed method for estimating the data-path area has an average percentage error of only 4.7% and a maximum error of less than 7%. In addition, the estimated area and the post-map area reported by the commercial tool exhibit a high degree of correlation.

The proposed static delay estimation relies solely on pre-characterized timing parameters of the RTL components and static interconnect delay. The estimated delay is computed for each delay-path by adding the individual component delays and the static interconnect delay between the components. The critical path delay is identified as the longest delay among the delay-paths. While the proposed static delay estimation leads to fairly good results for most of the functions considered, it exhibits very high inaccuracies for some cases. We believe that these high errors are due to the fact that static delay estimation does not take into account the post place and route effects on the delay. In the next chapter, we will present two approaches to overcome the limitation of the static delay estimation.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 6 DYNAMIC DELAY ESTIMATION PROCESS

In the previous chapter, we presented the static delay estimation technique which relies solely on pre-characterized timing parameters to estimate the critical path delay of the data-path. The static delay estimation technique does not attempt to predict the post implementation interconnect delay. In particular, the static delay estimation assumes that all the related components (components that are directly connected to one another) will be placed on adjacent CLBs. This assumption is unrealistic as the interconnectivity between the components are usually complex (e.g. each component are directly connected to more than one other component), and this leads to difficulty in optimizing the placement of these components on the FPGA architecture in order to minimize the interconnect delay. The commercial tool often performs iterative placement and routing to converge to a good solution. This is the reason for the long runtime of the place and route process in commercial FPGA tools. In complex designs, the placement and routing process often results in a critical path delay that is dominated by the interconnect delay. As the static estimation delay neglects the place and route effects on the interconnect delay, it may lead to high inaccuracies for certain functions as shown in the previous chapter.

In this chapter, we present two novel approaches for data-path delay estimation that overcomes the limitation of static delay estimation. Both approaches rely on the pre-characterized timing parameters to estimate the delay of the RTL components and incorporates strategies to estimate the post implementation interconnect delay. We term these two approaches as dynamic delay estimation methods. The first approach estimates the post-implementation interconnect delay based on the concept of slack distribution and we term this approach as SLK. The second approach attempts to predict the relative positions of components on the FPGA and we term this approach as SFP.

For the SLK approach, we have employed a new metric that models the placement complexity of the circuit. While this method leads to higher accuracies when compared to static delay estimation, it has several limitations. Firstly, it requires a FPGA-device-dependent calibration process which relies on a large dataset of randomly generated functions. Secondly, experimental results show that the SLK

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

cannot be used to reliably estimate the delay of all possible functions. The SFP method overcomes the limitation of SLK by incorporating a simplified floor-planning process to predict the relative distances between RTL components. This enables us to estimate the additional delay that is incurred due to place and route from the pre-characterized dynamic interconnect delay parameters discussed in the previous chapter. Experimental results show that the SFP approach leads to higher accuracy and consistency when compared to static delay estimation. In particular, when compared to the static delay estimation, the average error has decreased from 6% to 3.5% for the benchmarks considered. The maximum error has also decreased from 25% to 10%. The proposed dynamic delay estimation based on the SLP approach outperforms the estimation of the Xilinx synthesis tool by 4.7% for Spartan3 FPGA device.

6.1. Delay Estimation Based on Slack Distribution (SLK)

6.1.1. *Overview of Approach*

It has been previously reported that the placement process plays an important role on the predictability of the final delay (assuming that there is no constraints on the number of FPGA routing resources) [41]. In addition, timing-driven placement relies on the relative delay of the delay-paths. Hence, modeling the relative delay of the delay-paths can lead to reliable predictability of the interconnection delay. The SLK approach first estimate the delay of all the delay-paths using the static delay estimation method that is discussed in the previous chapter. We define the parameter λ to model the relative delay of the delay-paths as follows:

$$\lambda = \frac{D_{mean}}{D_{\max}} \qquad (6.1)$$

In (6.1), D_{mean} denotes the mean delay of all delay paths. D_{max} denotes the longest delay. Lambda models the slack distribution of the nets in the designs. For circuits that has a delay-path that is much longer than the rest (i.e. low-λ), the CAD tool will put in lesser effort to place the shorter delay-paths so that they do not exceed the delay of the longest delay-path. On the other hand for a circuit with high-λ (i.e. all the delay-paths have similar delays), the CAD tool will have less freedom to move the paths around without violating the delay of the longest path. Hence, we expect designs with low-λ design to be more predictable than designs with high-λ.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

The SLK strategy aims to identify through empirical means, the threshold value that will categorize a design as low-λ or high-λ designs. Let's define this threshold value as Λ. The estimated delay of a design with low-λ is the delay of the longest delay-path, while a design with high-λ is computed by multiplying the mean delay of the delay-paths with a constant factor. The constant factor was empirically found to be close to 1/Λ. The following describes the SLK estimation model, where D_{est} is the estimated delay.

$$D_{est} = \begin{cases} D_{\max} & \lambda < \Lambda \\ D_{mean} * \dfrac{1}{\Lambda} & ,otherwise \end{cases} \qquad (6.2)$$

In the next sub-section, we describe the approach used for determining the threshold value of Λ for a specific FPGA architecture.

6.1.2. *Determining the value of Λ*

In order to determine the value of Λ, we have generated 45 random functions and estimated their delays using the static delay estimation method discussed in the previous chapter. The estimated delays are compared with the post place-and-route delay from the commercial tool, and the estimation errors are plotted against λ in Figure 6.1. The plot shows that large errors occurred in regions where λ is large. The shaded region indicates where the predictability begins to fail (i.e. when Λ is 0.78 or 0.8). Based on of $\Lambda = 0.78$, the estimation model in equation 6.2 is then used to estimate the delay of functions from realistic embedded functions. Note that the calibration of Λ must be performed for new target FPGA architecture.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

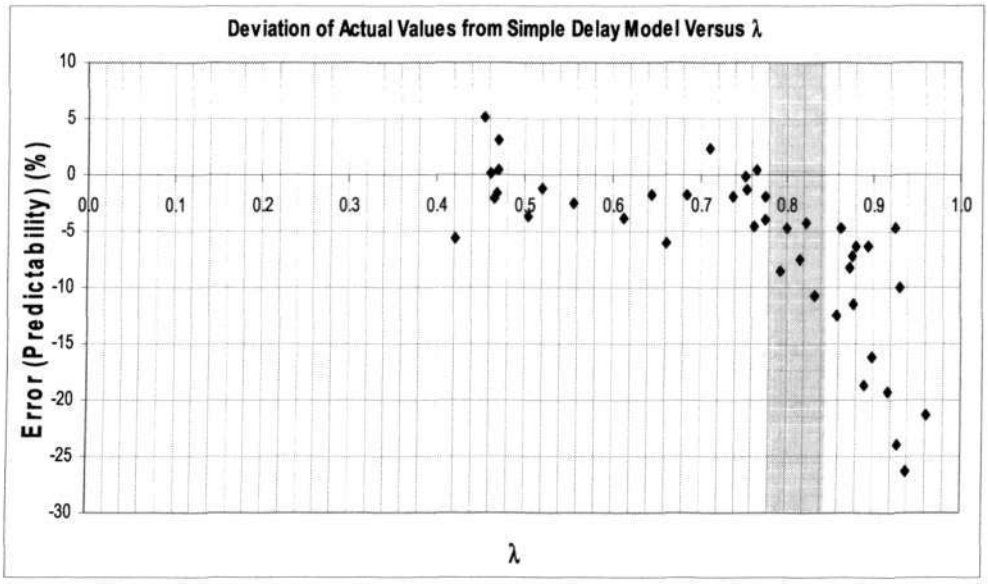


Figure 6.1 Estimation error (Predictability) versus λ

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

6.1.3. *Evaluation of the SLK approach*

(a) Experiment Setup and Results for the Virtex2Pro Family

We evaluate the accuracy of the SLK approach for the same set of functions used in Chapter 5. Table 6-1 shows the results of the static delay estimation method (Static), the post-synthesis results of the ISE tool (Syn), the estimated results using the SLK method (SLK), and the actual post place and route results (PAR) from the commercial tool (i.e. Xilinx ISE). The percentage error of Static, Syn and PAR is calculated with respect to the PAR values. It can be observed that compared to Static and Syn, the SLK approach has the lowest average estimation error.

TABLE 6-1 EXPERIMENTS RESULT WITH VIRTEX2PRO FAMILY

Virtex2Pro	Syn (ns)	Static (ns)	SLK (ns)	PAR (ns)	Syn (%)	Static (%)	SLK (%)
1	11.22	11.47	11.47	11.31	0.80	1.41	1.41
2	9.62	9.61	9.61	9.46	1.69	1.59	1.59
3	10.41	10.37	10.37	9.95	4.62	4.22	4.22
4	11.54	10.37	10.37	10.61	8.77	2.26	2.26
5	9.74	9.61	9.61	9.91	1.72	3.03	3.03
6	6.65	6.55	7.35	7.7	13.64	14.94	4.55
7	10.97	10.71	10.71	10.42	5.28	2.78	2.78
8	11.22	10.71	10.71	11.15	0.63	3.95	3.95
9	11.22	10.71	10.71	10.98	2.19	2.46	2.46
10	5.98	5.83	6.33	6.22	3.86	6.27	1.77
11	6.41	6.06	6.06	6.78	5.46	10.62	10.62
12	6.01	5.83	5.83	6.78	11.36	14.01	14.01
			Average		5.00	5.63	4.39

Figure 6.2 compares estimation errors of Synthesis tool (Syn), Static model (Static) and SLK method (SLK). The designs have been arranged in the order of increasing λ. It can be observed from Figure 6.2 that Static can provide accurate estimation results for designs with low λ only. However, for designs with high-λ (i.e. function 6 and 10), Static and Syn incurs very high estimation error (i.e. up to 14%). These results clearly demonstrates that the SLK approach is capable of providing better accuracy than Static and Syn due to the inclusion of the new metric λ that can be easily obtained for high-level estimation.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

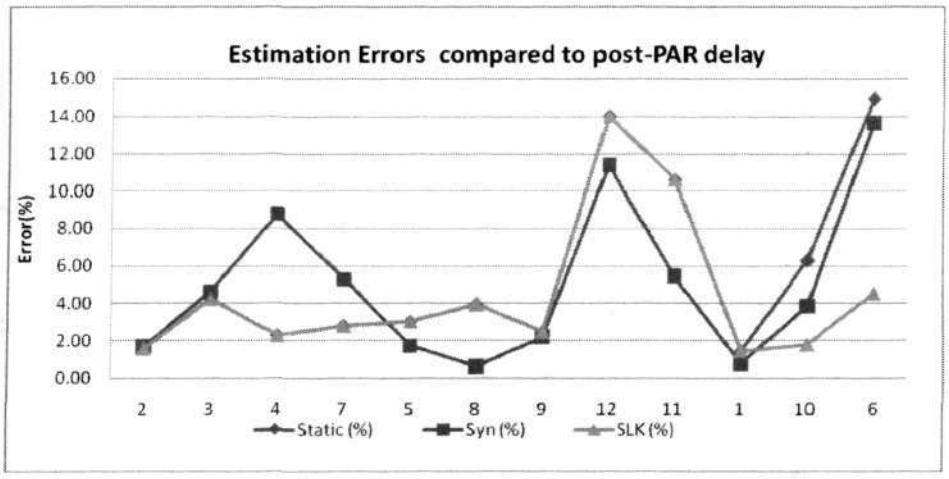


Figure 6.2 Estimation errors for Virtex2Pro Family

(b) Experimental Results for the Spartan-3 Family

In order to evaluate the robustness of the proposed technique, we carried out experiments with the Spartan-3 FPGA. The following processes are repeated with the new target FPGA device: 1) characterization of RTL components and static interconnect delay, 2) identification of the value of Λ and 3) delay estimation. The value of λ that is obtained for the Spartan-3 FPGA is 0.72 (as shown in Figure 6.3).

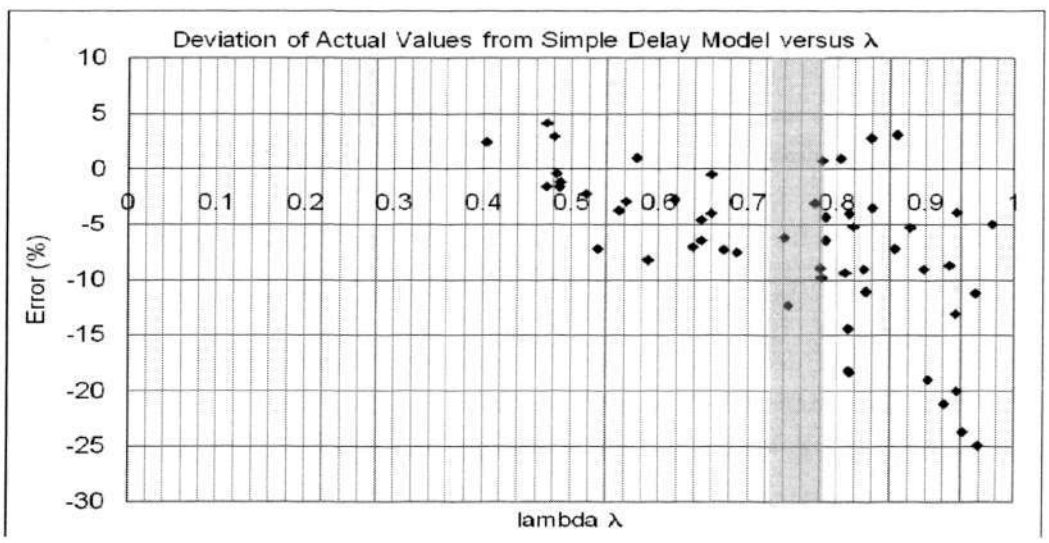


Figure 6.3 Estimation error (Predictability) versus λ for Spartan3-4 FPGA

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Table 6-2 shows the estimation error of the SLK approach that is computed with respect to the PAR results for the Spartan-3 device. The average delay-estimation error of the SLK approach is 4.07%, which is lower than the average estimation error of Static and Syn, which is 6.58% and 8.79% respectively. In addition, the maximum error of SLK is only 9.3% compared to the maximum error of Static and Syn, which is 24.05% and 12.75% respectively. This shows that the SLK method can overcome the limitation of Static, which can lead to very high inaccuracies for certain functions.

TABLE 6-2 ESTIMATION RESULTS FOR SPARTAN3-4 FAMILY

	Absolute Values				Error Compared to PAR		
	Syn (ns)	Sim (ns)	SLK (ns)	PAR (ns)	Syn (%)	Sim (%)	SLK (%)
Random Algorithm	18.7	16.26	17.02	16.85	10.98	3.50	1.01
Matrix multiplier	14.14	13.54	13.54	13.28	6.48	1.96	1.96
mpeg2-_bdist1_motion	15.85	14.72	14.72	14.36	10.38	2.51	2.51
mpeg2-_bdist2_motion	17.42	14.72	15.9	15.49	12.46	4.97	2.65
mpeg2-_dct_type_estimation	14.71	13.54	13.54	14.41	2.08	6.04	6.04
mpeg2-_dist1_motion	10.62	9.16	10.94	12.06	11.94	24.05	9.29
mpeg2-_dist2_motion	16.62	15.08	15.08	14.98	10.95	0.67	0.67
mpeg2-_idctcol	17.17	15.08	15.08	15.78	8.81	4.44	4.44
mpeg2-_idctrow	17.17	15.08	15.08	15.79	8.74	4.50	4.50
sha_transform	9.45	8.13	9.21	9.04	4.54	10.07	1.88
adpcm_coder	11.5	9.6	10.8	10.2	12.75	5.88	5.88
adpcm_decoder	10.54	8.96	9.2	10	5.40	10.40	8.00
Average Error					8.79	6.58	4.07

In Figure 6.4, the functions are rearranged in increasing order of λ. It can be observed that there is a large estimation error obtained with Static and Syn for designs with high value of λ. In contrast, the estimation error incurred with the proposed SLK technique is consistent across the different functions. This implies that the proposed estimation technique leads to a higher degree of predictability when compared to static delay estimation and the synthesis tool.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

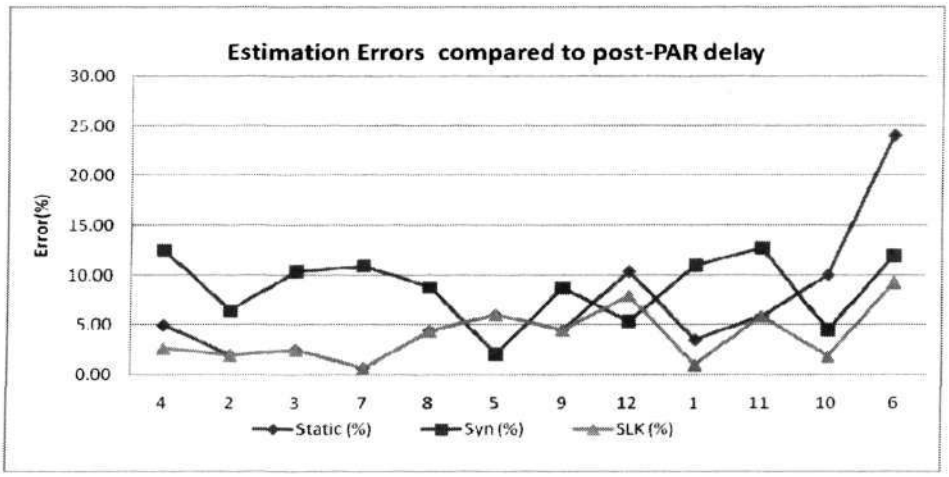


Figure 6.4 Estimation errors for Spartan3-4 family

6.2. Delay Estimation using Simplified Floor-Planning Method (SFP)

In the previous section, we have presented the SLK approach for delay estimation that takes into consideration the post-implementation interconnect delay estimation using the concept of slack distribution. The SLK approach employs the λ parameter, which is the ratio between mean delay and maximum delay of all paths in the circuit, to model the slack distribution of the delay-paths.

Although the experiments showed that SLK outperforms static delay estimation and the synthesis tool with an average accuracy of 95% for the benchmarks considered, it still results in very high inaccuracies for two functions in our experiments. This implies that the SLK approach cannot reliably estimate the post place and route interconnect delay for all applications. Furthermore, calibration of Λ must be performed using a large dataset for each new FPGA family. It is difficult to identify a suitable dataset for the calibration process that guarantees reliable estimation results for all possible applications. Lastly, the optimal value of the threshold is very difficult to determine.

Due to these shortcomings of the SLK method, we have devised another technique to predict the post-implementation interconnect delays, which can improve the consistency and avoid the need for device-dependent calibrations. This technique, which is called SFP, will be discussed in this section.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

6.2.1. *Overview of Proposed Approach*

The SFP method estimates the post implementation interconnect delay by performing a rapid floor-planning pass to predict how components will be placed on the FPGA architecture. This enables the relative distance between the components to be estimated. Based on the predicted relative distances, the interconnect delay can be estimated using the pre-characterized dynamic interconnect delay parameters (discussed in Section 5.1).

It is worth mentioning that the placement procedure in FPGA tools achieves a similar goal to the floor-planning pass adopted in the SFP method. In particular, the placement procedure aims to place the components on the layout of the FPGA architecture with the objective of minimizing the overall interconnect delay. However, these placement procedures are very complex and time consuming. In addition, the placement algorithms in commercial tools often iterate many times in order to converge to the best solution. In contrast, the simplified floor-planning process employed by the proposed SLP method estimates the relative distance of the RTL components in a single pass. The simplified floor-planning process models the physical FPGA architecture in the form of a linear 2D grid, and the geometrical structures of the components are ignored. Specifically, each component is treated as a point that can be placed on arbitrary locations of the 2D grid.

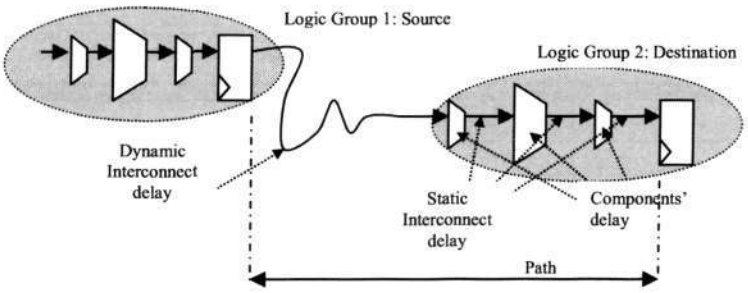


Figure 6.5 Model for delay estimation using simplied floorplanning

Dpath = FFClk2q + Ddynamic_interconn + Ddestination_logic_group

$D_{destination_logic_group} = D_{static_interconn} + D_{comp} + FF_{setup}$

D_{comp} is the characterized delay of the component in Table 5-1 .

D_{int} is the characterized static interconnect-delay within the logic groups.

$D_{dynamic_interconn}$ is the dynamic interconnect delay between the logic groups

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Figure 6.5 depicts our delay model for the SFP approach. The delay-path in Figure 6.5 consists of a connection of components between two registers. The logic components (i.e. input multiplexers, a functional unit, and output registers) and their static interconnection delays within a path are grouped into a logic group. The delay of the path can then be estimated using the model shown in Figure 6.5 which basically comprises destination logic group delay and dynamic interconnect delay. Logic group delay can be obtained by summing the pre-characterized parameters in Table 5-1 (static delay estimation). The estimation of the dynamic interconnect delay can be rapidly obtained using a simple floor-planning procedure, which will be discussed next, that approximates the distances between the logic groups after place and route. As results in previous section suggest that only paths which delay is larger than 70% of the static critical path contribute to the final performance, we only floor-plan the logic groups that have delay larger than 70% of delay of the longest logic group.

6.2.2. *Simplified Floor-planning Algorithm*

There are several floor-planning algorithm proposed in the literature. However, for the purpose of interconnect delay estimation we perform a fast floor-planning process that does not take into account the geometrical shapes and sizes of the components. The proposed floor-planning method employs a greedy algorithm to map a set of nodes (representing the components) to a regular-spaced grid (representing the FPGA space). Unlike the place-and-route algorithms employed in commercial FPGA tools, the proposed floor-planning strategy is not iterative and hence can be accomplished rapidly.

Figure 6.6 and Figure 6.7 are used to illustrate the proposed floor-planning method. Let's assume that we have a circuit of five logic groups that are connected as shown in Figure 6.6. The numbers in brackets are the characterized delays of these groups. The circuit can be represented as a directed graph, with weighted nodes that indicates the characterized delay of the logic groups. The floor-planning algorithm (see Figure 6.8) aims to place these nodes onto a regular grid in order to minimize the dynamic interconnection delay between the logic groups. For the example below, we assume that 1 grid unit is equivalent to 1 ns delay.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

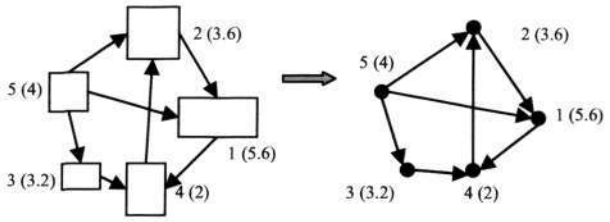
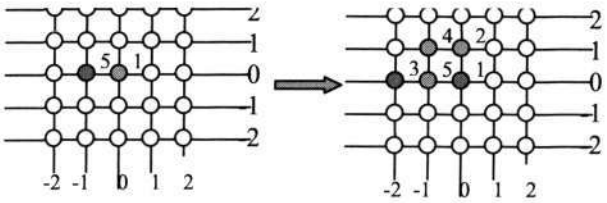


Figure 6.6 Sample circuit and simplified representation



Symbol convention:

Ni	Node i
G	Set of nodes
λ	Location (a place on the grid)
Λ	Set of locations
λ (Ni)	Location that is occupied by node Ni
N (λ)	Node that occupies location λ
D (N)	delay of Node N

Figure 6.7 Regular spaced grid for logic group placement

The algorithm first picks the node with the largest weight (node N1) and places this node onto the grid location (0, 0) (line 2-4). Next, the nodes that are connected to the placed nodes are considered to be placed onto the grid (line 8). In this example, the nodes that are connected to N1 are: N2, N4 and N5. N5 is considered first as it has largest weight. The set of possible locations (Λp) to place N5 includes (0, 1), (1, 0), (0, -1), (-1, 0) (line 12). Placing N5 at any of these locations leads to a distance of 1 unit to N1. In order to determine the location to place N5, we use the following rule: data flow from West to East and from North to South. Therefore, N5 is place at location (-1, 0). The nodes N2 and N4 are considered next. We select N2 as it has the largest weight and the possible locations to place N2 include (-2, 0), (-1, 1), (-1, -1), (0, 1), (0,-1), (1, 0). Note that N2 is connected to both N1 and N5. In order to determine the location to place N2, we have to compute a placement score for each of this location. Let's denote Nx as the node that is being considered for placement (e.g. Nx = N2). The score of location $\lambda \in \Lambda p$ is computed as Max (D(Ni)), where D (Ni) is the delay from λ to an existing node Ni on the grid, where Ni is connected to Nx. D(Ni) is computed by summing the grid distance from λ to λ(Ni) and delay of Ni, if Ni is the destination node. If Ni is the source node, D(Ni) is computed by summing the grid

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

distance from λ to λ(Ni) and the delay of Nx (line 20-21). The location with lowest score is chosen to place N2. For example:

If N2 is placed at (-2, 0) max delay = 7.6 because:

- N2 to N1: 2 grid unit + DN1 = 7.6
- N5 to N2: 1 grid unit + DN2 = 4.6

Similarly, if N2 is placed at (-1, 1) or (-1, -1) max delay = 7.6. Placing N2 at (0, 1) or (0, -1) or (1, 0) gives max delay = 6.6.

It can be observed that there are three locations (0, 1) (0, -1) and (1, 0) with the same score. By adopting the data flow preference, we place N2 at (0, 1). These steps are repeated until all the nodes in the graph have been placed.

```
1    //Initialization:
2    Set up list of nodes G
3    N = Pop_longest(G)
4    Place N into (0,0)
5    Push N into G_just_placed
6    While(G is not empty)
7    {
8          G_to_be_placed = Pop connected of G_just_placed from G
9          While (G_to_be_placed is not empty)
10         {
11               Nx = Pick_longest(G_to_be_placed);
12               Λp = Find_possible_locations()
13               //Compute Score
14               For each λp in Λp
15               {
16                     Λc = Get Connected Locations of Nx
17                           For each λc in Λc
18                           {
19                           D = Distance λp to λc + D(N(λc)) or D(Nx)
20                           Score of λp = max D
21                           }
22               }
23               Place Ni to λp which has lowest score.
24               Push Nx to G_just_placed
25         }
26   }
27
```

Figure 6.8 Simplified Floorplanning Procedure

This simplified floor-planning method enables rapid estimation of the interconnect distance between the logic groups. Based on these distances, we can derive the estimated interconnect delay using the pre-characterized dynamic interconnect delay shown in Table 5-2.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

6.2.3. *Performance Evalutations*

We carried out experiments with the same 12 functions used in previous section for both Virtex2Pro-6 and Spartan3-4 FPGA. The experimental results are presented in Table 6-3 and Table 6-4. Using the post-PAR delays as reference, the results show that the proposed approach (SFP) outperforms the static delay estimation (Static) and synthesis tool (Syn) for estimating the post place and route delay. In particular, the SFP achieves better results in terms of both maximum and average accuracy. The maximum and average estimation error of the proposed approach is only 6.5% and 3.5% respectively. In addition when compared to the other two approaches, the estimation error produced by SFP is more consistent. This implies that the proposed approach leads to a higher degree of predictability across different applications. We have also reported the FPGA area utilization of the functions in Table 6-3 to show the relative complexity of the functions. The average estimation time for each function is 0.3s which is significantly faster than the average place and route time of 40 minutes for each benchmark. It is noteworthy that the SFP method leads to the lowest average and maximum estimation error for both the Spartan3 and Virtex-II devices.

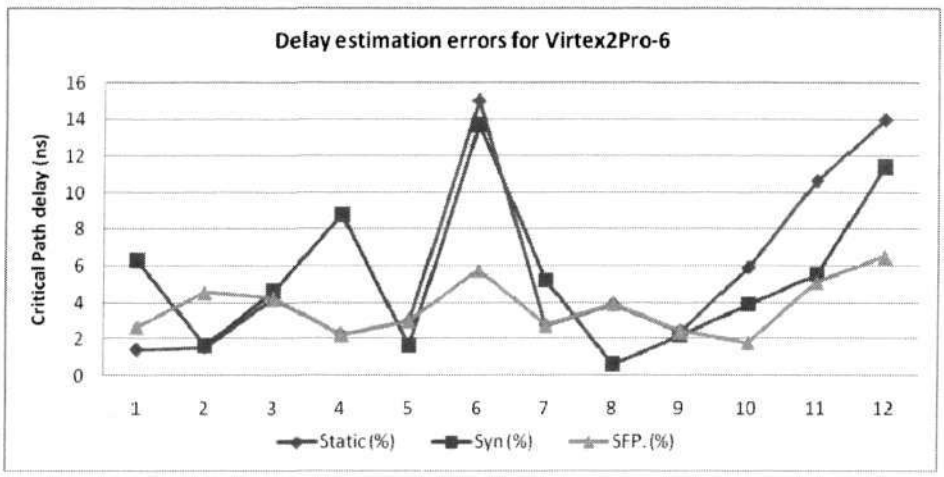


Figure 6.9 Delay estimation errors for experiments with Virtex2pro-6

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

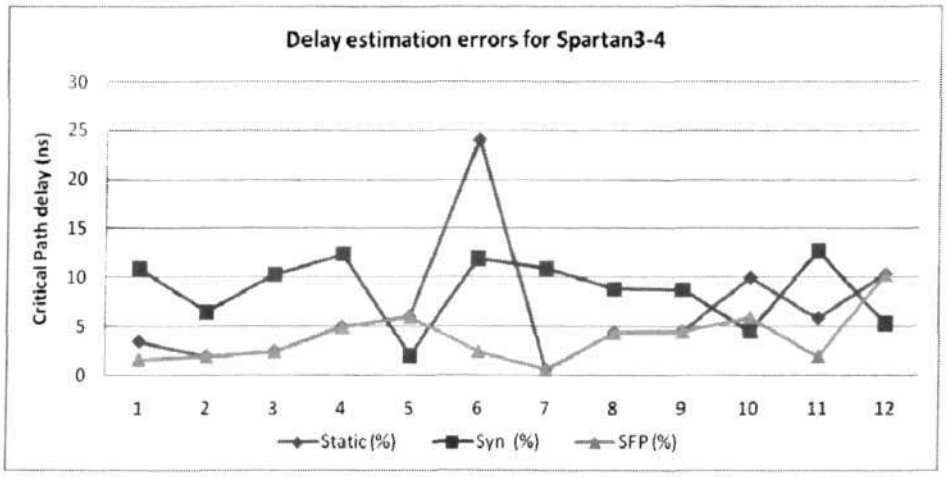


Figure 6.10 Delay estimation errors for experiments with Spartan3-4

TABLE 6-3 DELAY ESTIMATION RESULTS FOR VIRTEX II P-6 FAMILY (XC2VP70-6FF1517)

	Absolute Values				Error Compared to PAR values			
	Sim (ns)	Syn (ns)	SFP (ns)	PAR (ns)	Sim (%)	Syn (%)	SFP (%)	Area in Slices
Random Algorithm	11.47	12.02	11.61	11.31	1.41	6.28	2.65	1322
Matrix multiplier	9.61	9.62	9.61	9.46	1.59	1.69	1.59	530
mpeg2-_bdist1_motion	10.37	10.41	10.37	9.95	4.22	4.62	4.22	1695
mpeg2-_bdist2_motion	10.37	11.54	10.37	10.61	2.26	8.77	2.26	1905
mpeg2-_dct_type_estimation	9.61	9.74	9.61	9.91	3.03	1.72	3.03	786
mpeg2-_dist1_motion	6.55	6.65	8.15	7.70	14.94	13.64	5.84	1691
mpeg2-_dist2_motion	10.71	10.97	10.71	10.42	2.78	5.28	2.78	2102
mpeg2-_idctcol	10.71	11.22	10.71	11.15	3.95	0.63	3.95	4421
mpeg2-_idctrow	10.71	11.22	10.71	10.98	2.46	2.19	2.46	2861
Sha_transform	5.85	5.98	6.33	6.22	5.95	3.86	1.77	2670
adpcm_coder	6.06	6.41	7.13	6.78	10.62	5.46	5.16	1283
adpcm_decoder	5.83	6.01	7.22	6.78	14.01	11.36	6.49	1363
Average Error					5.60	5.46	3.52	

TABLE 6-4 DELAY ESTIMATION RESULTS FOR SPARTAN3-4 FAMILY

	Absolute Values				Error Compared to PAR values		
	Sim (ns)	Syn (ns)	SFP (ns)	PAR (ns)	Sim (%)	Syn (%)	SFP (%)
Random Algorithm	16.26	18.70	16.58	16.85	3.50	10.98	1.60
Matrix multiplier	13.54	14.14	13.54	13.28	1.96	6.48	1.96
mpeg2-_bdist1_motion	14.72	15.85	14.72	14.36	2.51	10.38	2.51
mpeg2-_bdist2_motion	14.72	17.42	14.72	15.49	4.97	12.46	4.97
mpeg2-_dct_type_estimation	13.54	14.71	13.54	14.41	6.04	2.08	6.04
mpeg2-_dist1_motion	9.16	10.62	11.76	12.06	24.05	11.94	2.49
mpeg2-_dist2_motion	15.08	16.62	15.08	14.98	0.67	10.95	0.67
mpeg2-_idctcol	15.08	17.17	15.08	15.78	4.44	8.81	4.44
mpeg2-_idctrow	15.08	17.17	15.08	15.79	4.50	8.74	4.50
sha_transform	8.13	9.45	9.58	9.04	10.07	4.54	5.97
adpcm_coder	9.60	11.50	10.40	10.20	5.88	12.75	1.96
adpcm_decoder	8.96	10.54	11.03	10.00	10.40	5.40	10.30
Average Error					6.58	8.79	3.95

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

6.3. Comparison Between the SLK and SFP Approaches

Figure 6.11 and Figure 6.12 compares the estimation errors (with respect to PAR results) between the two proposed dynamic delay estimation methods (i.e. SLK and SFP) for Xilinx Virtex2Pro and Spartan3 FPGA. The designs are arranged in increasing λ, where λ is the ratio between the mean delay and maximum delay of the delay-paths as shown in formula 6.1. Both figures show that for circuits with small λ, the two techniques converge to the same values which are the estimated delays of the static delay estimation method. The average errors of the SLK and SFP method begin to vary significantly for high-λ circuits. In terms of average performance, SLK and SFP both performs equally well for the set of benchmarks considered. However, the estimation results reported by SFP are more consistent. In addition, as mentioned earlier, unlike the SLK method the SFP approach does not require an FPGA-device dependent calibration process. Hence, we have adopted the SFP approach for data-path delay estimation in CRATE.

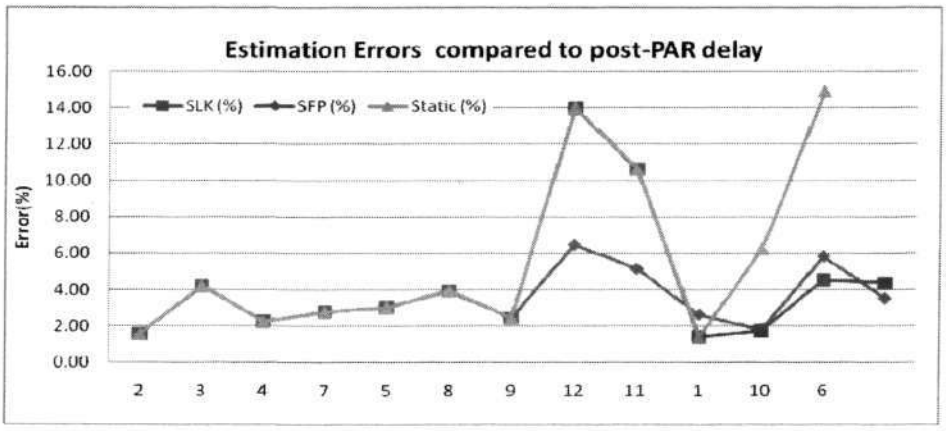


Figure 6.11 Comparison of 3 techniques for Virtex2Pro

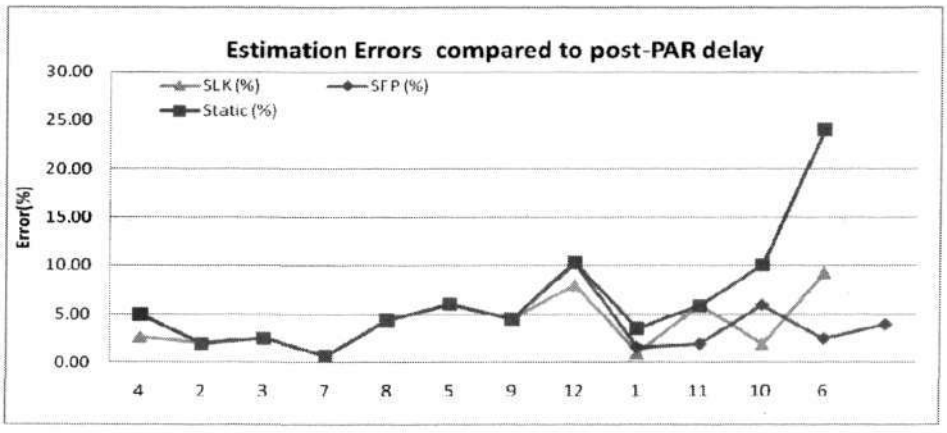


Figure 6.12 Comparison of 3 techniques for Spartan3

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

6.4. Summary

In this chapter, we have presented two methods for delay estimation of the data-path that overcomes the limitation of static delay estimation. Similar to static delay estimation, both approaches rely on the pre-characterized timing parameters to estimate the delay of the RTL components. However, unlike static delay estimation, the proposed approaches in this chapter incorporates strategies to estimate the post implementation interconnect delay.

In the first technique (SLK), we proposed the λ parameter to model the slack-distribution of the circuit. Our investigations reveal that data-paths with high-λ impose greater placement difficulty on the FPGA tool, and this leads to highly unpredictable interconnect delay. The SLK method predicts the post-PAR critical path delay by multiplying the delay obtained from static delay estimation with a constant for data-paths with high-λ. Although the SLK method leads to lower average error when compared to static delay estimation, it has several shortcomings. First, experimental results show that the reliability of the estimation method is not consistent with different FPGA families. Secondly, a large data set is required to find the threshold Λ that determines whether the data-path has high-λ. Finally, the reliability of this approach is dependent heavily on the underlying FPGA routing resource and an extensive calibration process has to be performed for different FPGA architectures.

In order to overcome the shortcomings of SLK, we have devised another method for delay estimation that takes into account the post-PAR interconnection delay. In the second approach called SFP, we perform a simplified floor-planning pass to predict the relative distances between RTL components on the FPGA architecture. This approximated distance can then be used to derive the additional delay caused by the physical placement of the components on the FPGA architecture.

When compared to static delay estimation and synthesis engine of the commercial tool, both the SLK and SFP techniques have better accuracy and consistency. When compared to the SLK approach, the estimation results reported by SFP are more consistent for the applications considered. Based on the quality of the estimation process and the simplicity of the technique (SFP does not require architecture-dependent calibration process), we have adopted the SFP approach for data-path delay estimation in CRATE.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 7 LOGIC OPTIMIZATIONS FOR HIGH-LEVEL ESTIMATION

In the previous three chapters, we have described the proposed methods for area-time estimation of the control and data-path of the h-VLIW architecture. The experimental results have shown that the proposed area-time estimation methods have a high degree of accuracy, and hence are suitable for high-level design exploration.

However, the methods described so far do not take into account the low-level optimizations performed by the synthesis tools when compiling the RTL codes to gate-level netlists. Comparisons with results from the commercial tool have so far been made by restricting logic optimizations of the synthesis engine. For example, we have enabled the "keep_hierarchy" synthesis option which results in larger and slower circuits in order to evaluate the performance of our delay estimation without the effects of logic optimization. A more desirable approach for high level estimation is to incorporate strategies that take into account the typical logic optimizations that are performed by the FPGA synthesis tools. Two common logic optimization techniques that are often employed are bit-width analysis and constant propagation, which can lead to more area-time efficient hardware. This chapter focuses on two tasks: 1) bit-width analysis and constant propagation of the RTL components of the h-VLIW architecture and 2) area-time estimation of RTL components by taking into account the logic optimizations in the first task.

In the first task, bit-width analysis and constant propagation is performed to infer the required bit-width of the RTL components. The second task employs a hybrid approach to estimate the area-time of the RTL components based on the results obtained from bit-width analysis and constant propagation. Hybrid approach refers to the use of the pre-characterized parameters and analytical models for area-time estimation. Analytical models are used to complement the pre-characterized parameters as while it may be possible to characterize all possible types of hardware components (since we can define the required component types in Trimaran's machine description file), it is impractical to characterize all possible variants of each component type (e.g. different bit-widths etc.). We will present analytical models for estimating the common RTL components, namely multipliers, adder/subtractors, comparators and multiplexers with varying bit-widths. Unlike existing work, the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

advantage of the analytical models presented in this chapter is that they require minimal pre-calibration process.

7.1. Bit-width Analysis and Constant Propagation

7.1.1. *Bit-width analysis*

Bit-width analysis is performed to infer the resulting bit-width requirements of the RTL components after they have been subjected to logic synthesis. Without bit-width analysis, the RTL components will have to incorporate the maximum bit-width. This leads to unnecessary area and at times delay, if the operands of the RTL components only occupy a limited segment of the maximum bit-width. For example, the result of an addition of two 8-bit numbers will have a result of at most 9-bits, or the result of a 5-bit right-shift operation of a 32-bit number is a 27-bit wide output. The logic synthesis engine of commercial tools will look for opportunities to trim off unused logic through bit-width analysis.

The RTL components that are selected in the resource binding stage (Chapter 3) can only incorporate bit-widths that correspond to the width of the integer data-types in C language (i.e. 8, 16 or 32 bits wide). Hence, there is a need to perform bit-width analysis to automatically infer the optimized bit-widths of the RTL components based on the variables and operations in the input functions. CRATE incorporates the technique presented in [51] to infer the optimized bit-widths of the RTL components. The bit-widths of the inputs and outputs of the RTL components are computed based on the transfer functions shown in Figure 7.1. For example, the output bit-width of the multiplication operation can be computed as the sum of the input bit-widths. The forward transfer functions propagate the bit-widths from the inputs to outputs, while the backward transfer functions propagate the bit-widths from outputs to inputs of the RTL components. Details of bit-width analysis can be found in [51].

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Opcode	Forward	Backward
add	d = MAX(s1, s2) + 1	s1 = d, s2 = d
subtract	d = MAX(s1, s2) + 1	s1 = d, s2 = d
unary negate	d = s1 + 1	s1 = d
multiply	d = s1 + s2	s1 = d, s2 = d
divide	d = s1 + 1	s1 = max_width, s2 = max_width
left shift by const	d = s1 + C	s1 = d - C, s2 = max_width
right shift by const	d = s1 - C	s1 = d + C, s2 = max_width
compare	d = 1	s1 = max_width, s2 = max_width
bitwise and	d = MIN(s1, s2)	s1 = d, s2 = d
bitwise or	d = MAX(s1, s2)	s1 = d, s2 = d
bitwise xor	d = MAX(s1, s2)	s1 = d, s2 = d
bitwise complement	d = max_width	s1 = d

Form of an operation is dest = src1 opc src1. We use *d*, *s1*, and *s2* to represent the widths of dest, src1, and src2, respectively.

Figure 7.1 Transfer functions for bit-width analysis (source: [51])

7.1.2. *Constant Propagation*

During logic synthesis, the constant 1s or 0s are propagated through the combinatorial logic to prune unnecessary logic. This process is termed as constant propagation.

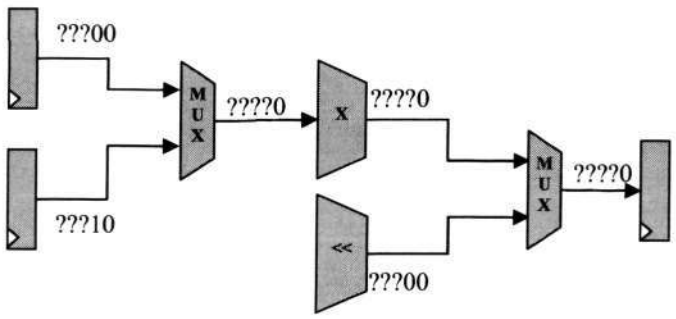


Figure 7.2 Illustration of the concept of constant propagation

We use the example shown in Figure 7.2 to illustrate the concept of constant-propagation and how it can lead to hardware savings. In Figure 7.2, the constant bits are propagated from the source registers, through the CFU-multiplexer, CFUs, CFU output multiplexer and finally to the destination registers. The output values of each component is shown, where a question mark represents a bit value that is unknown before application runtime. The 0's and 1's represent the constant bit values that will remain unchanged throughout the application runtime. Let us assume that the last two bits of the source registers are constants. These two source registers store the operands for the multiplier in the example.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Constant propagation infers the LSB of multiplexer output to be '0' as both inputs of the multiplexer has a LSB of '0'. Since the LSB of the multiplexer output is a constant, the data-width of the multiplexer can be reduced by one. This constant '0' is then propagated to the output of multiplier. At the same time, the last two bits of the 2-bit left-shifter's output are always '0's. The '0's at shifter's output and multiplier's output are propagated to the destination register, where the LSB is inferred to be a constant '0'. This implies that the number of FFs for the destination register can be reduced by 1.

TABLE 7-1 BIT INFORMATION INFERENCE RULE

0: bit is always 0 1: bit is always 1 ?: bit is unknown	Output's bit information = input1's bit-info + input2's bit info The '+' operator performs on bit info using the following rules:
Adder/Subtractor	0+0=0 1+1=0 1+0=1 1+?=? 0+?=?
Logic And	0+0=0 0+1=0 1+1=1 0+?=0 1+?=?
Logic Or	0+0=0 0+1=1 1+1=1 0+?=? 1+?=1
Multiplier	Forwards Least Significant Zeros
Constant n bit left Shifter	n least-significant bits = 0
At registers and Mux	0+0=0 1+1=1 0+1=? 0+?=? 1+?=?

CRATE employs constant propagation to improve the accuracy of area-time estimation. The algorithm presented in Figure 7.3 is used to perform constant propagation for each RTL component based on the inference rules shown in Table 7-1. Procedure Forward_bit_info_at_functional_units first propagates constant bits from the source register to input ports of the CFU based on the inference rules for multiplexers (line 13). It then propagates the constant bits from inputs to output of the CFUs (line 14 and 15) based on the inference rules for propagating the constant bits of different operations. Finally, the procedure forward_bit_info_at_registers propagates the constant bits from the driving CFUs to destination registers (line 26) and checks whether the bits has changed or not (line 21 and 29). This process to propagate constant bits from inputs to the outputs of the CFUs and registers are repeated until the bits are stable (while loop from lines 1-5).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

```
#1   Do
#2   {
#3         Changed =false;
#4         Forward_bit_info_at_Functional_Units;
#5         changed = Forward_bit_info_at_registers;
#6
#7   }while(changed)
#8
#9   Procedure Forward_bit_info_at_Functional_Units
#10  {
#11        For each CFU in the accelerator
#12              {
#13              Compute_Input_BitInfo;
#14              OutPut.BitInfo=Bit_Info_Transfer_Function(Fu_Type,
#15                                                  Inputs.BitInfo);
#16              }
#17  }
#18
#19  Procedure forward_bit_info_at_registers
#20  {
#21        Last_BitInfo = Register.BitInfo[b];
#22        For_each_fu_driving_this_register
#23        {
#24              For each bit b of register
#25              {
#26              Register.BitInfo[b]= Register.BitInfo[b] +
#27  fu.BitInfo[b];
#28              }
#29        }
#30        Changed = (Last_BitInfo != Register.BitInfo[b]);
#31  Return Changed;
#32  }
#33
#34  Procedure Compute_Input_Info
#35  {
#36        For_each_register_driving_this_port reg
#37        {
#38              Port.BitInfo = Port.BitInfo + reg.BitInfo;
#39        }
     }
```

Figure 7.3 Bit-Information Inferrence Procedure

7.2. Bit-width Aware Area-Time Estimation of RTL Components

In this section, we present estimation models that have been developed for area-time estimation of commonly used hardware components. Based on the bit-width information computed in the bit-width analysis and constant propagation process, the proposed models are capable of estimating the area-time of the components with varying bit-widths.

We propose a hybrid approach for the bit-width aware estimation of the hardware components that relies on both the pre-characterized parameters and a set of analytical models. The motivation for using the hybrid approach stems from the impracticability of maintaining a database to record the pre-characterized parameters for all possible

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

bit-width configurations of each component. Existing works that employ only analytical models for bit-width aware estimation requires a process to calibrate the constants in the estimation models for each function. In addition, these techniques often lead to high inaccuracies caused by the non-linear relationship between the bit-width and area-time measures of the hardware components.

In the proposed approach, the pre-characterized component library only stores a single set of area-time parameters for each hardware component. Let's define this set of parameters as the base parameter. In order to perform bit-width aware estimation, the base parameter of the hardware component is first obtained from the pre-characterized component library. Using the analytical models, the base parameter is then altered accordingly to accommodate the bit-width that is inferred from bit-width analysis and constant propagation of the component. For example, the area-time of a 16-bit adder can be inferred from the pre-characterized area-time values of a 32-bit adder easily, as the area and delay of adder /subtractor scales linearly with the bit-width. The proposed analytical models were developed to simulate the way components are implemented on the Xilinx and Altera FPGA devices. Unlike existing works, which develop the estimation models by analyzing the relationship between the bit-widths and area-time measures, the proposed approach avoids the need to calibrate the estimation models for different devices provided that these devices are similar in CLB architecture. The followings define the notations used in the description of the estimation models in this chapter:

A: area measured in LUT

D: delay measured in ns

A_C/D_C: area/delay of characterized component with bit-width C.

A(x)/D(x): area/delay of component with bit-width x

N: maximum number of inputs of LUT

The following section presents the estimation models of commonly used RTL components, i.e. adder/subtractors, multipliers, comparators and multiplexers, for Xilinx and Altera devices. We will also provide the area-time estimation results of various bit-width configurations of these components. The shaded values in the estimation results indicate the base parameters, while the rest are the estimated values which are inferred from the base parameters using the proposed analytical models.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

7.2.1. *Estimation Model for the Adder/Subtractor*

In both Xilinx and Altera FPGA devices, n-bit adder is implemented by cascading n 1-bit full-adder. Figure 7.4 and Figure 7.5 show how full adder is implemented on Xilinx and Altera FPGA devices. Although the full-adders are implemented differently on Xilinx and Altera devices due to the differences in the architectures, they share common area-time characteristics. In particular, the area-time of the adder varies linearly with the bit-width in both architectures.

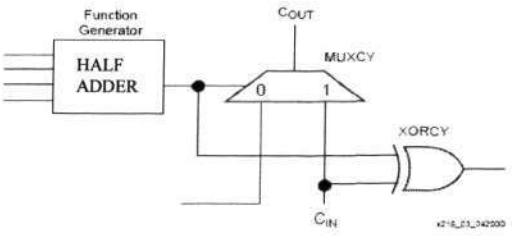


Figure 7.4 Xilinx Full-adder

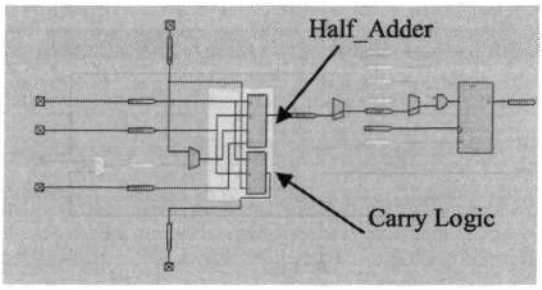


Figure 7.5 Altera Full-adder

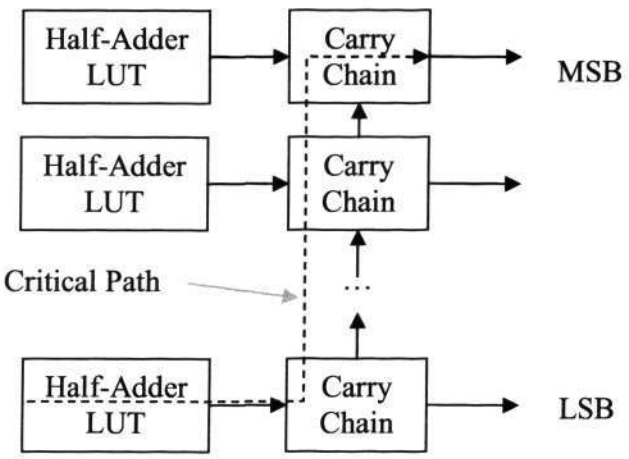


Figure 7.6 Implementation an adder on FPGA

Figure 7.6 shows a generic view of an adder, in which multiple 1-bit full adders are cascaded. Each 1-bit full-adder utilizes one LUT. Therefore, the number of LUTs consumed by an n-bit adder can be estimated as n LUTs. Hence the area (in terms of LUTs) of an x-bit adder is simply:

$$A_{ADDER}(x) = x \quad (7.1)$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Both Xilinx and Altera synthesis tool utilizes fast carry chain to minimize the carry propagation delay in the adder. Since all the half-adders are computed in parallel, the delay of adders with different bit-widths differs in the number of carry-chain stages only. For example, an m-bit adder has (m-n) carry-chain stages more than an n-bit adder (assuming m>n). The critical path delay of the adder begins from the first half-adder (that computes the LSB), continues through the carry chain and ends at the MSB output of the last adder. The delay of an x-bit adder can be estimated by Equation 7.2, where T_{carry} is delay of carry logic that can be obtained from FPGA datasheet.

$$D_{ADDER}(x) = D_C - (C - x) * T_{CARRY} \qquad (7.2)$$

TABLE 7-2 AREA-TIME OF ADDERS FOR XILINX SPARTAN3-4

xc3s5000-4fg1156					
Circuit	LUTs	Syn Delay	Estimated Area computed from formula 7.1	Estimated Delay computed from formula 7.2, D_C=3.875, C=32	T_{carry}
Adder_generic_8bit	8	2.339	8	2.339	0.064
Adder_generic_12bit	12	2.595	12	2.595	
Adder_generic_16bit	16	2.851	16	2.851	
Adder_generic_20bit	20	3.107	20	3.107	
Adder_generic_24bit	24	3.363	24	3.363	
Adder_generic_28bit	28	3.619	28	3.619	
Adder_generic_32bit	32	3.875	32	3.875	

TABLE 7-3 AREA-TIME OF ADDERS FOR ALTERA CYCLONE II C8

EP2C70F896C8					
Circuit	LUTs	Syn Delay	Estimated Area computed from formula 7.1	Estimated Delay computed from formula 7.2, DC=4.9, C=32	T_{carry}
Adder_generic_8bit	8	2.3	8	2.836	0.086
Adder_generic_12bit	12	2.7	12	3.18	
Adder_generic_16bit	16	3.3	16	3.524	
Adder_generic_20bit	20	3.56	20	3.868	
Adder_generic_24bit	24	3.9	24	4.212	
Adder_generic_28bit	28	4.77	28	4.556	
Adder_generic_32bit	32	4.9	32	4.9	

The delay and area of seven adders with varying bit-widths were measured for both the Xilinx and Altera FPGA devices as shown in (7.2)

Table 7-2 and Table 7-3, the estimated results for the Xilinx device were compared with the synthesis results (Syn) from the Xilinx tool. It can be observed that

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

the proposed estimation method yields exactly the same values reported by the synthesis engine of Xilinx. The implementation results of the Altera tool were obtained from the post-fit timing report, which included actual interconnect delay. Therefore, there were small deviations between estimated delays and the actual delays for the Altera device. The average and worst case estimation error is only 0.3 ns and 0.54 ns respectively for the 8-bit adder.

7.2.2. *Estimating area-time of Embedded Multipliers*

Both the Xilinx and Altera devices incorporate 18x18 bit embedded multipliers to perform fast multiplications. When the multiplier's operands are smaller than 18 bits, only one multiplier is instantiated. However, when the operands are larger than 18 bits, multiple multipliers are employed to generate the partial products. The partial products will be summed by an adder tree as shown in Figure 7.7. Hence, the delay of the multiplier can be computed by summing the delay of the embedded multiplier stages and delay of the adder tree.

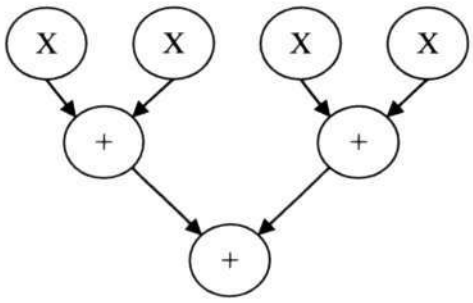


Figure 7.7 Partial products and adder tree

Figure 7.8 shows a two stage adder tree. Each square with '+' sign represents a 1-bit half adder. The dotted line shows a possible critical path of the adder tree. Delay of any critical path that starts from LSB of the first adder and ends at MSB of the last adder can be computed using the 7.3, where n is number of adder stages, T_{half_adder} is delay of the 1-bit half-adder, $T_{interconn}$ is delay of interconnect between adder stages. P is number of bits in the final sum, and T_{carry} is the delay of the carry logic.

$$T_{ADDER_TREE} = (T_{HALF_ADDER} + T_{INTERCON}) * n + P * T_{CARRY} = k * n + P * T_{CARRY} \qquad (7.3)$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

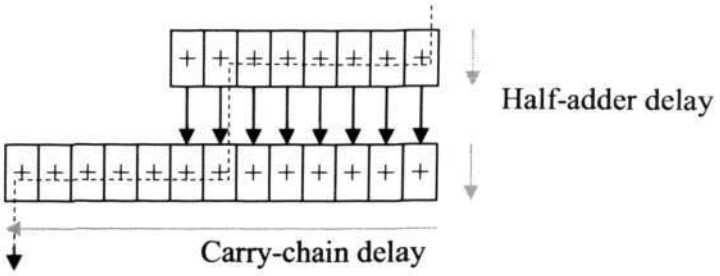


Figure 7.8 2-stage adders

The delay of the embedded multiplier stages is calculated using equation 7.4, where $T_{EMBMULT}$ is delay of the hard multiplier block, m is number of multipliers. Variable m is computed by $\left\lceil \frac{M}{18} \right\rceil * \left\lceil \frac{N}{18} \right\rceil$. Parameter k models the delay of a half-adder and interconnects delay. P is number of bits in the final Product. $T_{EMBMULT}$ and T_{CARRY} are obtained from FPGA datasheet.

$$D_{MULT} = T_{EMBMULT} + k\lceil \log_2(m) \rceil + P * T_{CARRY} \quad (7.4)$$

For each FPGA device, k is derived from the pre-characterized component library. For example, for Spartan3 speedgrade -4 device, $T_{EMBMULT}$ = 4.7 and T_{CARRY} = 0.064, we have characterized a multiplier with operands and output bitwidths of 32 and obtained area time info as m = 4, A = 32 LUTs and D = 10.61 ns. We substitute these values into equation 7.4 to compute k:

$$k = \frac{D - T_{EMBMULT} - P * T_{CARRY}}{\lceil \log_2(m) \rceil} = 1.93 \quad (7.5)$$

Figure 7.10 describes how the area of a large multiplier that is implemented using embedded multipliers and adder tree is estimated. In the example, we multiply A by B, where A is M-bit wide and B is N-bit wide, M and N is larger than 18 and smaller than 36. A[m:n] denotes a bus segment from the m^{th} bit to the n^{th} bit of signal A. Firstly, A and B are split into 18-bit buses: A[M-1:18], A[17:0], B[N-1:18] and B[17:0]. Then the four partial products P1, P2, P3 and P4 are computed as follows:

P1=A[17:0]xB[17:0], which is 36 bits wide.

P2=A[17:0]xB[N-1:18], which is N bits wide.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

P3=A[M-1:18]xB[17:0], which is M bits wide.

P4=A[M-1:18]xB[N-1:18], which is (M+N-36) bits wide.

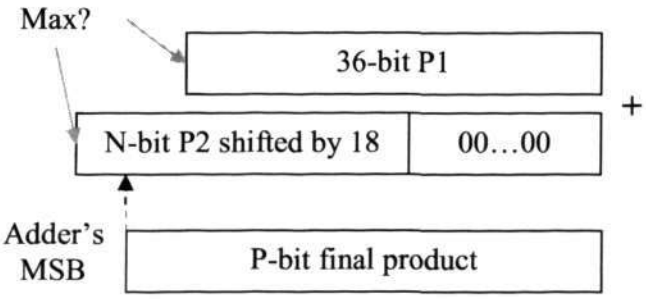


Figure 7.9 Summing Partial Product for multiplication

The final Product P is computed by summing the four shifted partial products: P = (P1+P2<<18)+(P3<<18+P4<<35). In Figure 7.10, we illustrate how to compute the number of LUTs for adding P1 and P2. P3 and P4 are omitted. Number of LUTs for adding P1 and P2 depends on the width P of final product and value of N. N-bit P2 is shifted by 18 bits before being added to P1. Therefore, the index of MSB of P2 is N+17. MSB of the adder is min(max(N+17,35), P-1). Because last 18 bits of P2 are 0s, total number of LUTs for the adder is min(max(N+17,35),P-1)-18+1.

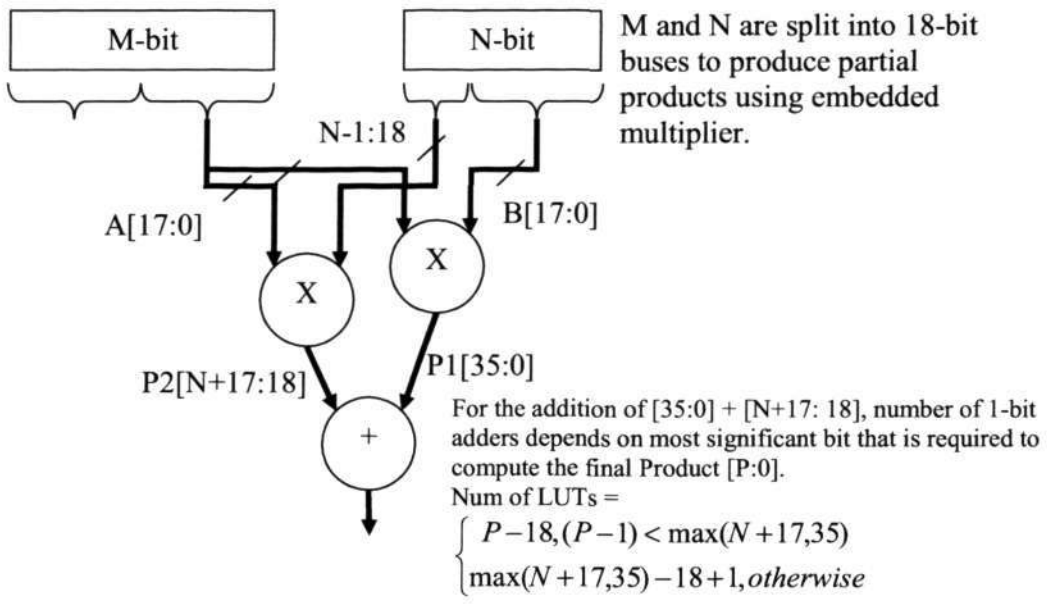


Figure 7.10 Implementation of large multiplier with multiple embedded multipliers

TABLE 7-4 ESTIMATION OF EMBEDDED MULTIPLIERS FOR XILINX SPARTAN3-4

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Embedded Multiplier				$T_{EMBMULT}$=4.7			T_{CARRY}=0.064		
xc3s5000-4ff1156				k = 1.9315 (from values at #2)					
				Actual			Estimated		
	M	N	P	Embedded Mult	LUT	Delay	Embedded Mult	LUT	Delay
1	32	32	64	4	111	13.70	4	92	12.66
2	32	32	32	3	30	10.61	3	30	10.61
3	32	16	32	2	15	7.73	2	15	8.68
4	24	24	36	4	40	10.26	4	39	10.87
5	24	24	24	3	14	10.05	3	14	10.10
6	24	16	24	2	7	7.00	2	7	8.17
7	16	16	32	1	0	4.85	1	0	4.70

TABLE 7-5 ESTIMATION OF EMBEDDED MULTIPLIERS FOR ALTERA CYCLONEII-8

Embedded Multiplier				$T_{EMBMULT}$= 4.25			T_{CARRY}=0.086		
EP2C70F896C8				k=1.449 (from values at #2)					
				Actual			Estimated		
	M	N	P	Embedded Mult	LUT	Delay	Embedded Mult	LUT	Delay
1	32	32	64	4	108	13.64	4	92	12.65
2	32	32	32	3	29	9.90	3	30	9.90
3	32	16	32	2	15	8.19	2	15	8.45
4	24	24	36	3	37	11.00	4	39	10.24
5	24	24	24	3	13	9.09	3	14	9.21
6	24	16	24	2	7	7.02	2	7	7.76
7	16	16	32	1	1	4.43	1	0	4.25

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

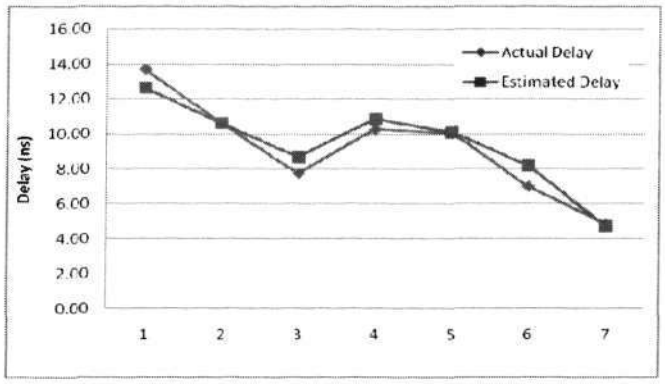


Figure 7.11 Comparison of actual and estimated values for Multipliers on the Xilinx device

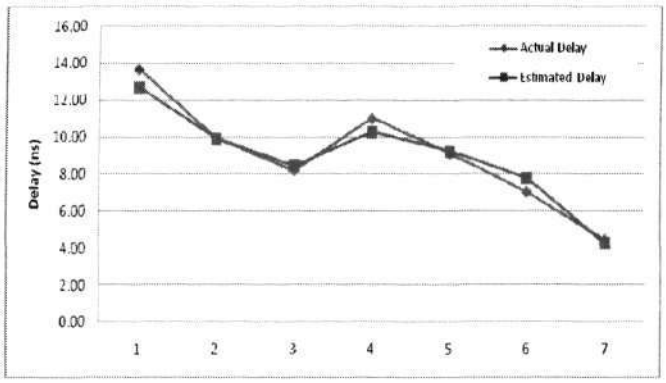


Figure 7.12 Comparison of actual and estimated values for Multiplier on Altera

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

In order to evaluate the accuracy of the models, we synthesized 7 multipliers using ISE 8.1 and Quartus II 7.2 with different inputs (M, N) and outputs (P) as shown in Table 7-4 and Table 7-5. The experiment results show that the proposed technique applies for both Xilinx and Altera devices. In particular, area estimation produces negligible errors for most cases (experiments 2-7) with a maximum error of only 2 LUTs. For large multipliers (32x32=64 bit) the error in LUT estimation is found to be 20 LUTs. We believe that this is negligible compared to the area of designs that employ such large multipliers. For delay estimation, experimental results in Figure 7.11 and Figure 7.12 show that the proposed model yielded consistent estimation with average and maximum errors of 0.5ns and 1.2ns respectively. The advantage of this model is that only the parameter k needs to be recomputed for different devices.

7.2.3. *Estimating Equal Comparator*

(a) Altera Cyclone II

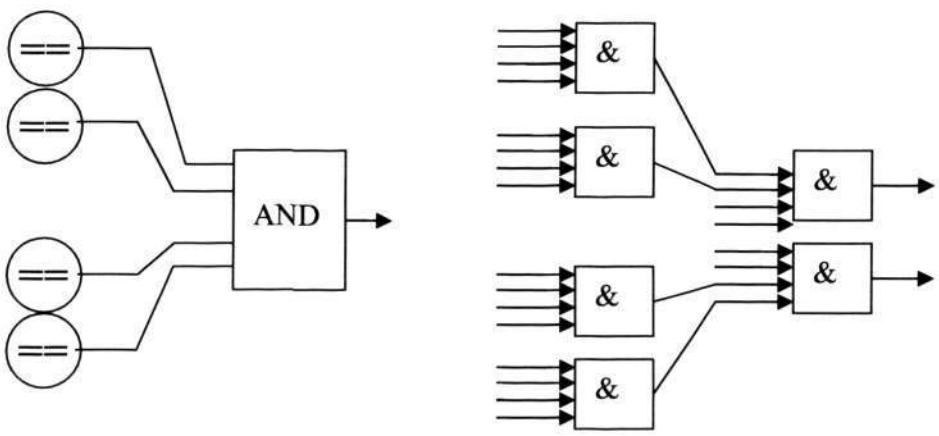


Figure 7.13 Equal comparator circuit

The equal comparator analyzes two input values to determine if they are equal. If the two inputs are equal, the comparator will output a value of '1', otherwise it will output a value of '0'. For the Altera FPGA device, the equal comparator is implemented by an array of 1-bit comparators and a large multiple-input AND-gate as shown in Figure 7.13. To implement x-bit comparators, x 1-bit comparators are required. One N-input LUT can be configured to implement (N/2) 1-bit comparators. Therefore, x/(N/2) LUTs are used to implement x 1-bit comparators. The large AND-gate is implemented by cascading many N-input AND-gates in the form of a tree as shown in Figure 7.13. The number of inputs of the AND-tree is K=x/(N/2). One N-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

input LUT can be configured to implement N-input AND-gate. Hence, the first stage of this tree requires K/N LUTs and produces K/N outputs, the second stage requires (K/N)/N LUTs and so on. Therefore for FPGAs with N-input LUTs, the number of LUTs required to implement the equal comparator can be calculated as follows, assuming x≥8 and N=4:

$$A(x) \approx K + \frac{K}{N} + \frac{K}{N^2} + \frac{K}{N^3} ..., K = \frac{x}{N/2}$$

$$A(x) \approx K \sum_{i=0} (1 + \frac{1}{N^i} c_i), c_i = \begin{cases} 1, \frac{K}{N^i} > 0 \\ 0 \end{cases}$$

$$if \begin{cases} x >= 8 \\ N = 4 \end{cases} \Rightarrow A(x) \approx K * (1 + \frac{1}{N}) \approx \frac{x}{N/2}(1 + \frac{1}{N})$$

$$\Rightarrow A(x) = \frac{x}{C} A_c \qquad (7.6)$$

The delay of the equal comparator varies linearly with number of stages of the comparator, which in turns varies logarithmically with the input bit-width. Total number of stages of comparator can be computed by total number of stages of AND-tree plus one stage of comparators array: $\lceil \log_N(K) \rceil + 1$. Number of interconnects between the stages are computed by: $\lceil \log_N(K) \rceil * T_{INTERCONN}$. The estimation model for delay estimation of an x-bit equal-comparator is shown in (7.3). Our investigation shows that better estimation can be achieved by removing the ceiling operation.

$$K = x/(N/2)$$
$$D(x) = [1 + \lceil \log_N(K) \rceil] * T_{LUT} + \lceil \log_N(K) \rceil * T_{INTERCONN}$$
$$D(x) = \lceil \log_N(K) \rceil * (T_{LUT} + T_{INTERCONN}) + T_{LUT}$$
$$Assume\ T_{LUT} \sim T_{INTERCONN} \Rightarrow T_{LUT} \approx 0.5 * (T_{LUT} + T_{INTERCONN})$$
$$D(x) = (\lceil \log_N(K) \rceil + 0.5) * (T_{LUT} + T_{INTERCONN})$$

$$\Rightarrow D_{Eq_COMPARATOR}(x) \approx \frac{\log_N(K) + 0.5}{\log_N(C/2) + 0.5} * D_C \qquad (7.7)$$

In order to evaluate the accuracy of the estimation model for the comparator, we synthesized and implemented 6 comparators with different widths using Altera Quartus II version 7.2. The number of LUTs (Actual LUTs) and delay (Actual Delay) reported by the tool were obtained after the fitting process. The estimated number of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

LUTs (Estimated LUT) and estimated delay (Estimated Delay) were obtained using the proposed model in (7.7). The area estimation results showed that our model can predict the exact number of LUTs. For delay estimation, results revealed that the worst case error was only 0.2 ns. Hence, we can rely on this model to infer the area-time of comparators with arbitrary widths.

TABLE 7-6 ESTIMATION OF EQUAL COMPARATOR FIR ALTERA CYCLONE II-8

EP2C70F896C8 Bit-width	Actual LUT	Actual Delay	Estimated LUT	Estimated Delay
8	5	2.32	6	2.22
12	8	2.7	8	2.65
16	11	2.83	11	2.96
20	14	3.05	14	3.20
24	16	3.51	16	3.39
32	21	3.7	21	3.70

(b) Xilinx FPGA

The Xilinx devices implement the equal comparator differently from Altera devices, and hence a different estimation model is required. Xilinx uses carry chain, multiplexers and LUTs to implement the equal comparator as shown in Figure 7.14. This implementation is very efficient and simple. The physical multiplexers residing on the FPGA slices are used to implement the AND-gate by tying one of the inputs to logic '0'. The 4-input LUTs are configured to implement a 2-bit comparator. In general, N-input LUTs can be configured as (N/2)-bit equal comparator. Therefore, the number of LUTs to implement x-bit equal comparator for the Xilinx architecture is:

$$A_{EQ_COMPARATOR}(x) = \frac{x}{(N/2)} \tag{7.8}$$

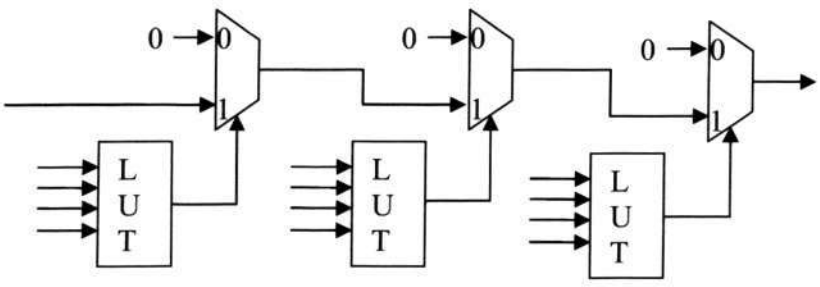


Figure 7.14 Equal comparator circuit

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Similar to the estimation of the adders, the delay of the equal comparators with different bit-widths can be estimated based on the number of carry-chain stages. Therefore, the delay of an x-bit equal-comparator for the Xilinx device can be computed as follows:

$$D_{EQ_COMPARATOR}(x) = D_C - \frac{(C-x)}{N/2} * T_{CARRY} \quad (7.9)$$

TABLE 7-7 ESTIMATION OF EQUAL COMPARATOR FOR XILINX SPARTAN3-4

				Estimated from Dc=2.228, C=32
Family: xc3s5000-4 Bit-width	Actual LUTs	Actual Delay	Estimated LUT	Estimated Delay (Xilinx Model)
8	5	2.098	4	1.46
12	6	1.588	6	1.588
16	8	1.716	8	1.716
20	10	1.844	10	1.844
24	12	1.972	12	1.972
28	14	2.1	14	2.1
32	16	2.228	16	2.228

We have used Xilinx ISE 8.1 to synthesize seven comparators with different bit-widths. The results are reported in Table 7-7 . The number of LUTs (Actual LUTs) and delay (Actual Delay) of comparators were extracted from the synthesis report. The 'Estimated LUT' column shows the number of LUTs that is computed using equation 7.8. The 'Estimated Delay' column shows the delay of the comparator that is estimated using Xilinx Model discussed above. In general, the area estimation model can predict the exact number of LUTs that is reported by the synthesis tool. For delay estimation, our model can predict the exact critical path delay of the components for all cases except for the 8-bit comparator. However, the error difference for this case is only 0.5ns, which is negligible compared to the data-path delay.

7.2.4. *Estimating LT/GT (Less Than/Greater Than) Comparator*

The Less Than/Greater Than comparator determines if the first operand input is less than/greater than the second operand input. Similar to the equal comparator, the LT/GT comparator outputs a value of '1'/'0'. The implementations of the LTGT comparator for the Xilinx and Altera devices are similar and are shown in Figure 7.15. Note that the difference between the two devices is the logic that is implemented using the LUTs. The example shows a LT comparator for inputs A and B, i.e. comparing

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

whether A<B. In order to compare n bits, the comparator first compares the MSB of A and B (i.e. A_n and B_n). If A_n is smaller than B_n or A_n is larger than B_n, then the evaluation is complete. Otherwise if A_n is equal to B_n, then the next lower order bits A_{n-1} and B_{n-1} are evaluated in the same fashion. This is repeated for all the bit pairs in descending bit order until a pair of bits with non-equal values is found. The Boolean equation for the LT comparator is as follows:

$C_n = A'_nB_n+(A'_nB'_n+A_nB_n)C_{n-1}$. (A' denotes negation of A)

$= A'_nB_n + (A'_nB'_n+A_nB_n)C_{n-1} + A_nB_nB'_n$ (last Product is always '0')

$= A'_nB_nB_n + A_nB_nB'_n + + (A'_nB'_n+A_nB_n)C_{n-1}$ ($A'_nB_nB_n$ is equal to A'_nB_n)

$= (A'_nB_n+A_nB'_n)B_n + (A'_nB'_n+A_nB_n)C_{n-1}$

$= (A_n \text{ xor } B_n)\, B_n + (A_n \text{ xnor } B_n)C_{n-1}$.

$= E'.B_n+E.C_{n-1}$ ($E = A_n$ xnor B_n)

E can be implemented using a single 4-input LUT. B_n and C_{n-1} are the inputs of the carry-chain. In both the Xilinx and Altera devices, the area (in terms of LUTs) can be estimated as the bit-width:

$$A_{LT_COMPARATOR}(x) = x \tag{7.10}$$

Similar to the estimation of the adders, the delay of the equal comparators with different bit-widths can be estimated based on the number of carry-chain stages. Therefore, the delay of an x-bit LT/GT comparator for Xilinx and Altera devices can be computed as follows:

$$D_{LT_COMPARATOR}(x) = D_C - (C - x) * T_{CARRY} \tag{7.11}$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

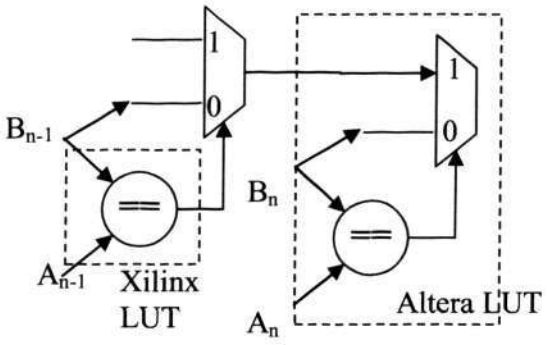


Figure 7.15 Greater than comparator circuit

Experiment Data:

TABLE 7-8 ESTIMATION OF GREATER THAN COMPARATOR FOR ALTERA CYCLONE II-8

EP2C70F896C8 Bit-width	Actual LUT	Actual Delay	Estimated LUT	Estimated Delay	T_{CARRY}
8	8	2.31	8	2.84	0.086
12	12	2.75	12	3.18	
16	16	3.35	16	3.52	
20	20	3.65	20	3.87	
24	24	4	24	4.21	
32	32	4.9	32	4.90	

TABLE 7-9 ESTIMATION OF GREATER THAN COMPARATOR FOR XILINX SPARTAN 3-4

xc3s5000-4ff1156 Bit-width	Actual Lut	Actual Delay	Estimated LUT	Estimated Delay	T_{CARRY}
8	9	3.891	8	3.891	0.064
12	13	4.147	12	4.147	
16	17	4.403	16	4.403	
20	21	4.659	20	4.659	
24	25	4.915	24	4.915	
28	29	5.171	28	5.171	
32	33	5.427	32	5.427	

Table 7-8 and Table 7-9 show the actual area-time and estimated area-time of seven comparators with different bit-widths for Altera Cyclone2 FPGA and Xilinx Spartan3-4 family. Experimental results show that we can use the proposed area-time delay model to reliably estimate the LT/GT comparator for the Xilinx device. For Altera FPGA, the experimental results revealed that relatively higher estimation errors

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

were found for smaller bit-widths. However, the maximum error was found to be only 0.5 ns, which is negligible compared to the delay of typical data-paths.

7.2.5. *Estimating area-time of Multiplexers*

The pre-characterized area-time of fixed bit-width (i.e. 32 bits) multiplexers is stored in the pre-characterized component library as shown in Table 7-10. Note that the pre-characterized area-time values for multiplexers with varying number of inputs (not bit-widths) are stored. These values are used along with the analytical models to estimate the area-time of multiplexers with arbitrary inputs. In this section, we will present the proposed analytical models for Xilinx and Altera devices.

(a) Xilinx FPGA

Our experiments reveal that the number of LUTs for implementing multiplexers on the Xilinx device varies linearly with the number of inputs as shown in Table 7-10. Hence, we can predict the number of LUTs for n-bit multiplexer based on the number of inputs as follows:

$$A(inputs) = n * \lceil inputs / 2 \rceil \quad (7.12)$$

Estimated delay is obtained from the pre-characterized component library after the input is rounded up to the nearest value available in library. For example, a 12-input multiplexer has the same delay of a 16-input multiplexer. The delay estimation model is:

$$D(inputs) = D_C(2^{\lceil \log 2(inputs) \rceil}) \quad (7.13)$$

TABLE 7-10 EXAMPLE MULTIPLEXER AREA-TIME INFORMATION STORED IN DATABASE (XILINX SPARTAN3-4)

Inputs	Data Width	Area	Time
2	32	32	0.9
4	32	64	1.2
8	32	128	1.5
16	32	256	1.8
32	32	512	3
64	32	1024	6

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

(b) **Altera FPGA**

Figure 7.16 shows how a 4-to-1 multiplexer is mapped onto 4-input LUT, in which a, b, c, d are inputs. S0 and S1 are select signals. For multiplexers with larger number of inputs, inputs are partitioned into groups of 4-to-1 multiplexers as shown in Figure 7.15. Each group of 4-input multiplexer is denoted as a stage. The outputs of the previous stage serve as inputs to the next stage.

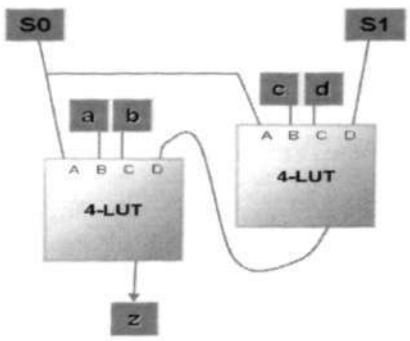


Figure 7.16 Mapping of 4-to-1 multiplexer on Altera FPGA LUT

The algorithm that recursively estimates number of LUTs from inputs is shown in Figure 7.17. If number of inputs is smaller or equal to 4, the corresponding number of required LUTs is derived as shown in lines 6 to 10. Otherwise, in the first stage, inputs are grouped into groups of 4 as shown in Figure 7.17. Hence, total LUTs for these multiplexers in the first stage are computed by: 2*(inputs/4) + Estimate (inputs%4).

The outputs of the first stage are: inputs/4+1, which will be the inputs of the next multiplexer's stage and so on. Therefore, the recursive equation to estimate the total LUTs is:

$$A_{MUX}(\text{inputs}) = 2*(\text{inputs}>>2) + \text{Estimate}(\text{inputs}\%4) + \text{Estimate}(\text{inputs}>>2+1) \quad (7.\ 14)$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

```
1#   Int Estimate (inputs)
2#   {
3#   If(inputs<=4)
4#   {
5#   Switch inputs
       {Case 0: return 0;
       Case 1: return 0;
       Case 2: return 1;
       Case 3:
       Case 4: return 2;
       Default: 0;}
12# } else
     {
     a = 2*(inputs/42)+
       Estimate(inputs%4) +
             Estimate (inputs/4+1);
     }
#18 Return a;}
```

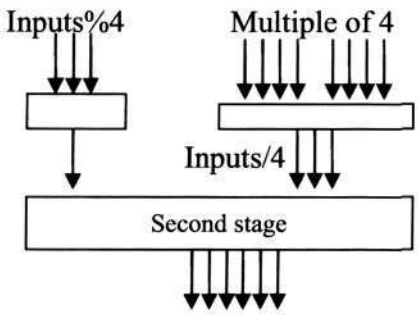


Figure 7.17 Implementation of Multiplexers and area estimation algorithm

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

7.3. Summary

In order to achieve realistic estimation, the effect of logic optimization has to be considered. Logic optimization techniques such as bit-width analysis and constant propagation infer the required bit-width of the hardware components and prunes away unnecessary logic in order to reduce the circuit sizes and improve circuit delay. We have incorporated the following logic optimizations in the proposed estimation process: 1) bit-width analysis to determine the required bit-width of the hardware components based on the type of operations involved and 2) constant propagation to identify segments of the hardware components that do not require actual logic due to the fact that the operands are always constants. We have adopted the approach in [51] for bit-width analysis to infer the final bit-widths of registers and functional units. For constant propagation, we have devised component-specific rules to propagate constant '0's and '1's through the data-paths in order to identify redundant segments of the hardware components. Based on the information from bit-width analysis and constant propagation, we can then perform bit-width specific area-time estimation for each RTL component.

We have presented bit-width aware estimation models for commonly used hardware components such as adders, multipliers, comparators and multiplexers for the Xilinx and Altera devices. The key advantage of our proposed models is that only one parameter needs to be calibrated for different devices. In addition, the unknown parameter can be computed without using complex regression analysis procedures that are employed by previous methods. Experimental results show that the proposed models can effectively estimate the area-time of hardware components with varying bit-widths.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 8 PERFORMANCE EVALUATION OF ENTIRE AREA-TIME ESTIMATION FRAMEWORK

In the previous chapters, we have described in detail the essential steps for high-level estimation, i.e. front-end compilation and high-level optimization, resource binding, control path estimation, data-path estimation and strategies for estimating the effects of logic optimizations. In this chapter, we will put all these steps into perspective by presenting the complete framework for high-level area-time estimation. Figure 8.1 compares our approach to a typical hardware design flow. The key advantage of our approach is the ability to provide area-time estimation of C-based algorithms in a significantly shorter amount of time compared to the typical hardware design flow. The final experimental results that are generated based on the proposed framework will be presented in this chapter. In particular, we will demonstrate that the proposed framework 1) can lead to reliable results (measured in terms of accuracy and fidelity), 2) is portable across different FPGA platforms, and 3) can be executed efficiently to facilitate rapid design exploration. We will also demonstrate the hardware efficiency of the h-VLIW architecture by comparing the implementation results of CRATE with results of the Altera C2H tool.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

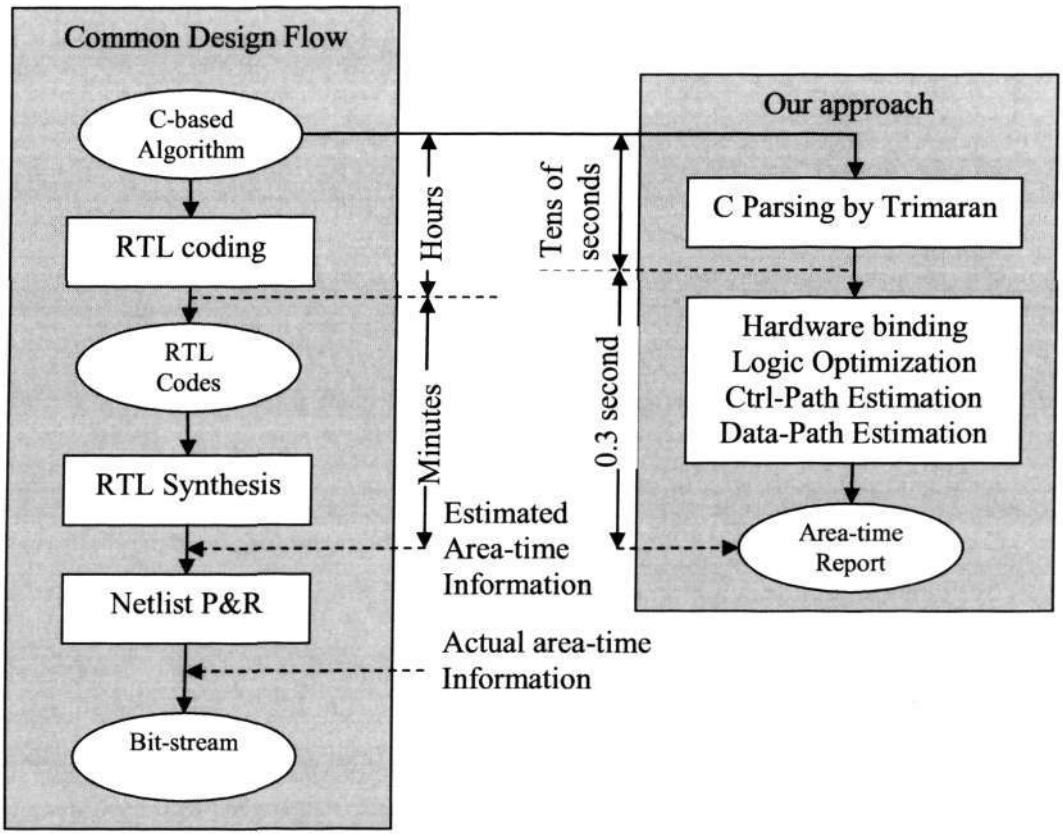


Figure 8.1 Comparing proposed approach to a common design flow

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

8.1. Complete framework

Figure 8.2 shows the proposed framework for high-level estimation. We will summarize the main steps in the proposed framework.

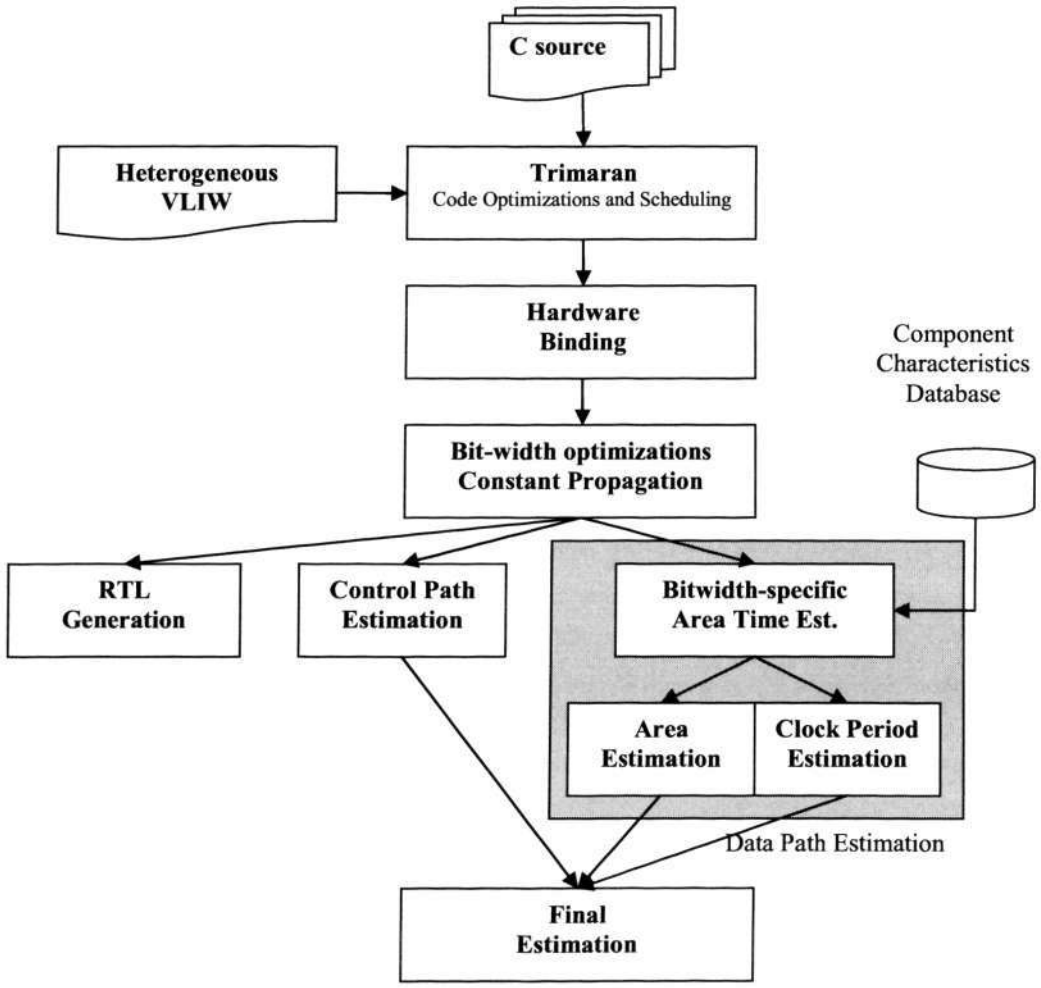


Figure 8.2 Proposed framework implemented in CRATE

Front-end processing with Trimaran:

The proposed framework relies on Trimaran to transform un-timed C codes into a sequence of micro-operations. High-level optimization techniques such as loop unrolling and loop pipelining are employed to expose the ILP of the application. Further details of the high-level optimization techniques have been discussed in Chapter 3. Based on the availability of resources (i.e. CFUs of h-VLIW), Trimaran schedules the execution of each micro-operation. At the end of compilation, Trimaran generates O_el file, which specifies detailed attributes of all operations and their order

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

of execution. Figure 8.3 illustrates a fraction of O_el file which is generated from the fibonacci application.

```
bb 8 (
  weight(9)
  entry_ops(76) exit_ops(39)
  entry_edges(ctrl ^32 ctrl ^33) exit_edges(ctrl ^33 ctrl ^43)
  flags(sched)
  attr(lc ^109)
  Subregions (
    op 76 (C_MERGE [] [] s_time(0) s_opcode(C_MERGE.0) in_edges(op-32(1)
      op-39(8)) flags(real_merge sched))
    op 38 (ADD_W [r<1:i gpr>] [r<1:i gpr> i<1>] p<t> s_time(0) s_opcode(ADD_W.0)
      Flags (sched))
    op 33 (ADD_W [r<5:i gpr>] [r<2:i gpr> r<3:i gpr>] p<t> s_time(0)
      s_opcode (ADD_W.1) flags (sched))
    op 61 (PBRR [r<17:b btr>] [b<8> i<0>] p<t> s_time(0) s_opcode(PBRR.2) attr(lc
      ^113) flags (sched))
    op 35 (MOVE [r<2:i gpr>] [r<3:i gpr>] p<t> s_time(0) s_opcode(MOVE.1)
      flags(sched))
    op 62 (CMPP_W_LEQ_UN_UN [r<18:p pr> u<>] [r<1:i gpr> m<int_ret>] p<t>
      s_time(1) s_opcode(CMPP_W_LEQ_UN_UN.0) flags(sched))
    op 36 (MOVE [r<3:i gpr>] [r<5:i gpr>] p<t> s_time(1) s_opcode(MOVE.0)
      flags(sched))
    op 39 (BRCT [] [r<17:b btr> r<18:p pr>] p<t> s_time(2) s_opcode(BRCT.0)
      attr(lc ^117) out(op-76(8) op-77(1)) flags(sched))
  )
)
```

Assembly codes:
s_time/ Cycle 0: Op 38: R1←R1+1; Op 33: R5←R2+R3; Op 35: R2←R3;
s_time/ Cycle 1: Op 36: R3←R5;
s_time/ Cycle 2: Op 39: Branch to BB8 if R1 Less Than or Equal to Macro Int_Ret

Figure 8.3 Example of IR output

Hardware Binding:

The resource binding step has also been discussed in Chapter 3. Based on the O_el file, CRATE assigns operations and variables to the physical hardware components (i.e. CFUs and registers in the h-VLIW). CRATE binds operations and variables to optimize the interconnectivity between the CFUs and registers. During this process, connections between components are defined and the CIT (Control Information Table), which is used for implementing the decoding logic for each component, is constructed. For example, let us assume that operation 33 in the above code snippet is about to be bound to Adder 1 and its destination (r<5:i gpr>) has been bound to Register 9. The hardware binding process will establish a connection between output of Adder 1 and input of Register 9. At the same time, a new entry is

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

added to CIT of Register 9, which specifies that the enable signal is high at State_BB8_Cycle0.

Bit-width Analysis and Constant Propagation:

CRATE performs bit-width analysis and constant propagation to prune off unnecessary logic which will be eventually trimmed away by the logic synthesis engine. For example, the destination of a shift left operation would have zeros in some of LSBs. If this register is the source of another functional unit such as multiplier or adder, these LSB zeros will lead to logic reduction in the functional units. Therefore, this step is very important as it can significantly improve the area-time of the final design. Estimation without incorporating this step would incur large errors when compared to results of the logic synthesis tools. The algorithms for bit-width analysis and constant propagation have been discussed in Chapter 7.

Control Path Estimation:

Each cycle in the O_el file is realized as a state in the accelerator's FSM. CRATE extracts branch information and branch conditions to construct a state table. Based on the CIT of each component, CRATE performs estimation of the controller based on the one-hot encoding scheme. The estimation reports number of LUTs required for NSDL and CSDL, and number of Flip-flops that are used to hold state variables. This step also predicts the delay of the CSDL. Chapter 4 provides a detail description of the approaches for estimating the control path.

Data Path Estimation:

Data-path estimation is accomplished in two phases. In the first phase, CRATE adopts a hybrid strategy for estimating the data-path by taking into account the bit-widths of the resulting RTL components. In particular, CRATE estimates the area-time for components with arbitrary bit-width using analytical models and pre-characterized component parameters. The pre-characterized component library, which has been discussed in Chapter 5, only stores area-time information of RTL arithmetic components with 16 or 32 bits. Area-time information of the actual bit width (other than 16- or 32-bits) is inferred using analytical models presented in Chapter 7. Using a hybrid approach for estimation has the following advantages over other approaches that only employ analytical models: 1) the avoidance of a complex pre-calibration

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

process and 2) better estimation accuracy. The second phase estimates the collective area and time effects of the components and interconnections.

Area and delay estimation of the data-path are performed separately. Area estimation examines connection between CFUs and registers to predict the total SLICE/Logic Cell count by taking into account the effect of logic packing (i.e. packing of LUT and FF onto the same slice). This was discussed in Chapter 4. Delay estimation predicts critical path delay after PAR. The final estimation combines control path delay and data path to determine the minimum clock period. This step is required as delay from decoder circuits in the control path may exceed delay of the data-path. It is noteworthy that existing works have not addressed this concern.

RTL Generation:

CRATE generates synthesizable HDL codes for the complete HWA (with data-path and control path) based on the h-VLIW architecture model. This is a useful step as it provides a means for us to validate the reliability of our estimation with results from commercial FPGA tools. In addition, this step enables us to compare the quality of the proposed h-VLIW architecture with implementations that are generated by commercial C-to-hardware tools. We will discuss the comparison with the Altera C-to-H tool later in this chapter.

Final Report

Figure 8.4 shows the final area-time estimation report generated by CRATE. The first section reports on the number of LUT/FF/Slice consumed by the control-path. It also reports the LUTs consumed by the individual components of the control path e.g. NSDL and CSDL. The maximum delay of the decoding-logic for each component is also reported. For example, 'FU Port Delay' reports the maximum delay of CSDL for the CFU's multiplexers and 'Reg Ctrl Delay' reports the maximum delay of CSDL for the register enable signals.

The second section of the final report lists the data-path area-time measures. This includes area-time before and after bitwidth optimization. The pre-optimization areas are reported as 'Total FF', 'Total LUT' and 'Total Slice', while the optimization delay is reported as 'Total Delay'. CRATE reports another set of area-time measure after bitwidth optimization. The number of FFs, LUTs, and Slices consumed after bitwidth optimization are shown as 'Optimized FFs', 'Optimized LUTs', and 'Optimized LC'

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

in the report. The number of on-chip multiplier (i.e. '18x18 Multiplier') required is also reported. Delay after bitwidth optmization is reported as 'Optimized Delay'.

The third section of the report shows the statistical values of the delay-paths (e.g. mean, max, min, λ, σ).

Finally, the 'worst cluster delay' reports final delay after the simplified floor-planning process has been performed. This section also indicates whether the worst case delay is contributed by the control signal or data-path. For example, if 'from Cluster 0 control signal' is 'yes', then this indicates that the worst delay originated from the control path.

```
==========================================================
Control Path Info
Assum average interconnect delay
Assume that each branch requires 2 LUT
----------------------------------------------------------------------------
Total Lut              : 147
Total FF               : 154
Total Slice            : 154
  Reg enable lut       : 10
  Reg Mux Lut          : 0
  FU Ctrl lut          : 12
  FU Mux Lut           : 97
  State Decode         : 28
Reg Port delay         : 0
FU Port Delay          : 2.92
Reg Ctrl Delay         : 0.44
FU Ctrl Delay          : 2.92
State Decode Delay     : 0.44
==========================================================
Data-path Info
----------------------------------------------------------------------------
Total Area             : 2147   LUT
Total Delay            : 5.938  ns
Total FF               : 2550
Total Slices           : 2350
Optimized Area         : 1473
Optimized FF           : 2261
Optimized LC           : 3641
18x18 Multiplier       : 0
Optimized Delay        : 5.938
----------------------------------------------------------------------------
All path               : 4385
Mean                   : 5.59759
Sigma                  : 0.0107946
Max                    : 5.938
Min                    : 1.738
Mean/Max               : 0.942672
Note: these values is dependent on whether you have performed logic optimization or not
==========================================================
Worst cluster delay: 8.858 from Cluster 0 control signal: yes
```

Figure 8.4 CRATE's report

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

8.2. Result Analysis for Xilinx and Altera FPGAs

8.2.1. *Experimental Setup*

Twenty benchmarks from 14 embedded functions as shown in Table 8-1 were used to evaluate the effectiveness of the proposed framework for high-level area-time estimation. Each of this benchmark is compiled using Trimaran and the corresponding h-VLIW architecture is constructed based on the methods described in the previous chapters. Control path and data-path estimation is then performed for different FPGA families to obtain the total estimated area-time values of each benchmark. Note that the reported values are the sum of the estimated area-time values of the control and data-path.

TABLE 8-1 EMBEDDED FUNCTIONS USED IN FINAL EXPERIMENTS

	Functions	Description
1	autocor00-_fxpAutoCorrelation	Fix-point Autocorrelation (EEMBC)
2	adpcm_coder	Adaptive Differential Pulse Code Modulation – Coder (MediaBench)
3	adpcm_decoder	Adaptive Differential Pulse Code modulation – Decoder(MediaBench)
4	comb_sort	Comb Sorter Algorithm
5	idctcol_idct_11	Inverse Discreet Cosine Transfrom performing on Column using 11 x 2-clock mult
6	idctcol_idct_2	Same as above but using 2 x 2-clock mult
7	idctcol_idct_4	Same as above but using 4 x 2-clock mult
8	idctcol_idct_8	Same as above but using 8 x 2-clock mult
9	idctrow_idct_11	Inverse Discreet Cosine Transfrom performing on Row using 11 x 2-clock mult
10	idctrow_idct_2	Same as above but using 2 x 2-clock mult
11	idctrow_idct_4	Same as above but using 4 x 2-clock mult
12	idctrow_idct_8	Same as above but using 8 x 2-clock mult
13	sha_transform_sha	SHA transfrom in SHA application
14	cjpegv2-_jpeg_fdct_islow	
15	cjpegv2-_rgb_ycc_convert_jccolor	RGB color space to YCC color space converter (EEMBC)
16	conven00-_convolutionalEncode	Convolutional Encoding (EEMBC)
17	fbital00-_fxpBitAllocation-_div2	Fixed-point Bit Allocation (EEMBC)
18	viterb00-_ACS_viterbi	ACS function in Viterbi Decoder
19	viterb00-_FindMetrics_viterbi	FindMetrics Function in Viterbi Decoder
20	viterb00-_ViterbiDecoderIS136	Viterbi Decoder

The IDCTCOL and IDCTROL benchmarks [47] were estimated for four different configurations, where each configuration employs different number of multipliers in the h-VLIW architecture. The number of multipliers that is used in each configuration is specified in the Trimaran's machine description file. In Table 8-1, 'MxCclock

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

multiplier' is used to denote that M multipliers with latency of C-cycles were allocated for resource binding. Note that each multiplier in the h-VLIW architecture can be implemented with one or more embedded multipliers that are available in the FPGA devices (as discussed in Chapter 7).

The RTL codes of the h-VLIW architecture were also auto-generated for each benchmark. These RTL codes serve as inputs to the commercial tool to obtain the area-time of the actual implementation for comparison with the estimated values. In order to restrict the synthesis engine from introducing additional delay due to allocation of registers to the I/O port as shown in Figure 8.5, the "packing IO registers into IO-PADs" option in the synthesis engine is disabled. Figure 8.5 (a) shows the desirable placement of a LUT that drives the input of a FF, where the FF is placed closed to the driving LUT (they should ideally be packed onto the same slice). When the above-mentioned synthesis option is enabled, Figure 8.5 (b) shows the resulting situation, where the output register is allocated to the IO register that are physically located at the boundary of the FPGA. Hence, enabling the synthesis option could result in unnecessary delay being incurred between the LUT-FF pairs that are placed far apart.

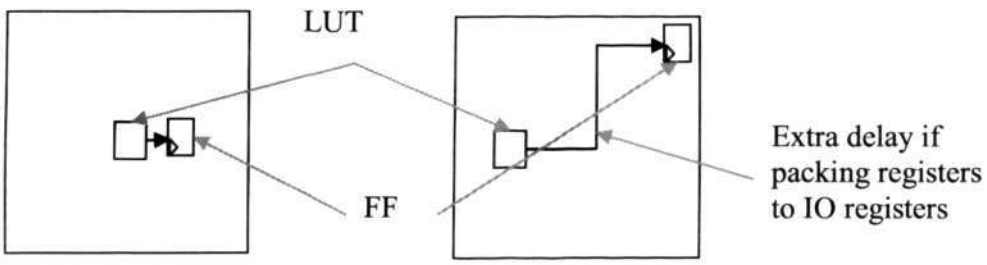


Figure 8.5 Illustration of IO register packing

In addition, the RTL circuits were implemented using the commercial tool with area and time constraints. Area constraints have been imposed to restrict the placement of the components to a limited region in order to force the implementation tool to achieve the best area utilization. The area constraint imposed on each RTL circuit in the following experiments is initially set as 1.15~1.2 times the estimated slices. In the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

event that the implementation tool is unable to meet the specified area constraint, the constraint value is then progressively relaxed. Similarly, timing constraint is applied to force the implementation to achieve the best performance for the design. The timing constraint is set based on the Xplorer script [45] provided by Xilinx, which gradually tightens the constraint to achieve minimum critical path delay.

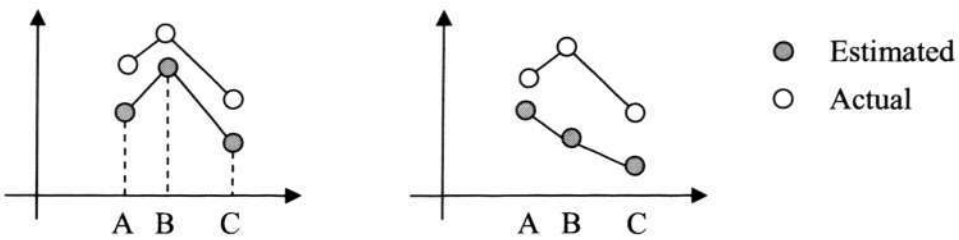
The reliability of estimation framework is measured based on *accuracy* and *fidelity*. Accuracy is a measure of how close the estimated values and actual values are. Accuracy is calculated using (8.1), where E is the value obtained by estimation techniques and M is the value measured from the commercial tool.

$$Accuracy = 1 - \frac{|E - M|}{M} \tag{8.1}$$

"Fidelity of an estimation method is defined as the percentage of correctly predicted comparison between design implementations" [18]. For example, in Figure 8.6(a), the estimator gives correct comparison for three designs which means fidelity = 100%. In Figure 8.6(b), the estimator gives incorrect comparative relationship for design A and design B; comparative relationship between A and C, and between B and C are still correct; Hence fidelity = 2/3 = 66%. In general, fidelity F is computed by:

$$\mu_{ij} = \begin{cases} 1 & ((E_i > E_j) and (M_i > M_j)) or ((E_i = E_j) and (M_i = M_j)) or ((E_i < E_j) and (M_i < M_j)) \\ 0 & otherwise \end{cases}$$

$$F = 100 * \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mu_{ij}$$



(a) Fidelity = 100% (b) Fidelity = 66%

Figure 8.6 Illustration of fidelity

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

8.2.2. *Estimation with Xilinx FPGA(Spartan3-4 and Virtex2Pro-6)*

(a) Area estimation

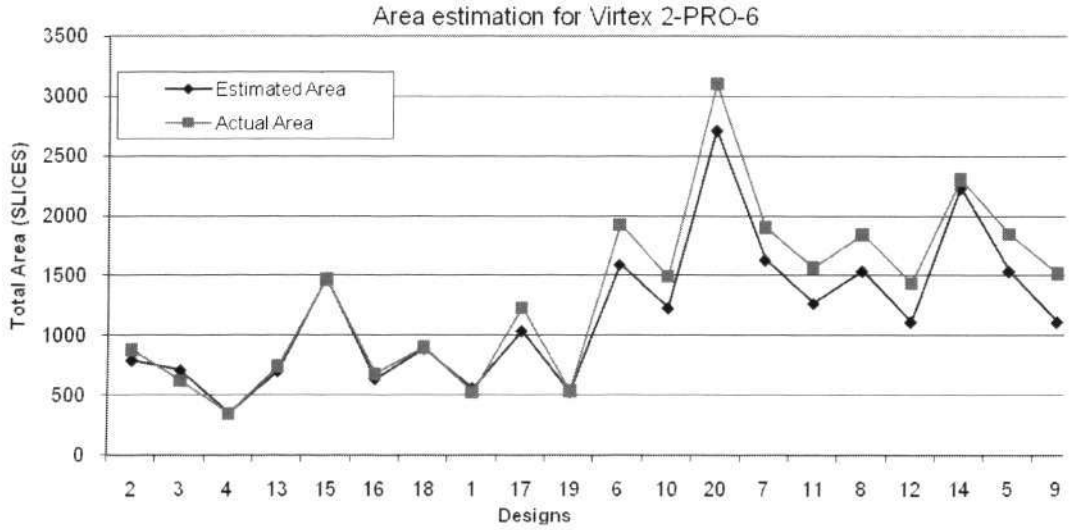


Figure 8.7 Area estimation for Virtex2Pro-6

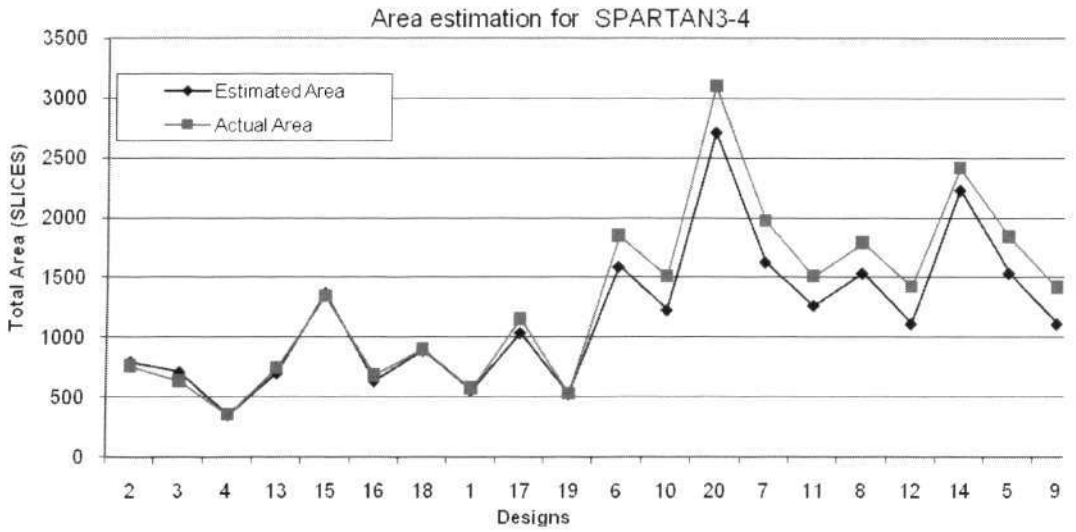


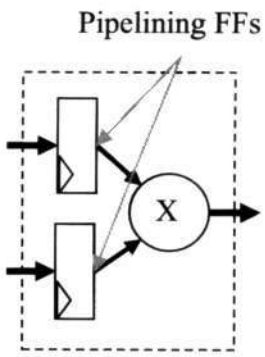
Figure 8.8 Area estimation for Spartan3-4

We have arranged the results in the order of increasing number of embedded multipliers that has been instantianted for the implementations. For both FPGA families, the experimental results are very encouraging in the first 10 benchmarks (i.e. 2, 3, 4 … 1, 17, 19) which use less than four embedded multipliers. In particular for these ten benchmarks, we achieve average accuracy of 94.3% and 95% for Virtex2Pro and Spartan3 family respectively. The worst case errors of the two devices is 14% and 12% respectively. In Nayak [25], the proposed area estimation method achieved an

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

average accuracy of 89.3% and worst case error of 15.8%. Compared to that work, we have an accuracy improvement of 5% for Virtex2Pro family and 6% for Spartan3 Family. It is noteworthy that our estimation method achieve very high fidelities: 92.7% and 94.5% for the Virtex and Spartan families. These encouraging results demonstrate that our approach is capable of estimating the number of slices with an average accuracy of 94% for most medium sized applications.

For the remaining circuits that employ more than four embedded multipliers, our estimation reports an average error of 16.7% for Virtex2Pro and 16% for Spartan3 FPGAs. Maximum errors are 26.7% and 21.6% respectively. We achieve large deviations from the actual values because we have ignored the FFs that are used in the multiplier's pipeline. As mentioned earlier, we employed 2-clk-latency multipliers for our experiments and hence the embedded multipliers' inputs were registered as shown in the figure on the left. For example, each 32x32 embedded multiplier with 2-clock latency will instantiate 64 input FFs. If these FFs are not driven by LUTs, they will consume some additional slices. Hence, the actual area is larger than the estimated area as the current approach do not take into account the registers at the inputs of the embedded multipliers. This will be considered in the future work.



Using the twenty benchmarks, we then performed area estimation for the Virtex4 speedgrade-12 FPGA device. We achieved an average error of 7.6% with a worst case error of 17%. We observe that the estimation accuracy for the circuits that use embedded multipliers are much better than the estimation accuracy of the Virtex2pro and Spartan3 devices. In particular, the average error for this set of functions that use embedded multipliers is only 7% with worst case error of 17% compared to average error of about 16% for the other two families. This is due to the fact that in the Virtex4 device, the multiplier's pipeline registers are mapped onto DSP slice. Hence, this does not incur additional slices for the pipeline registers.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

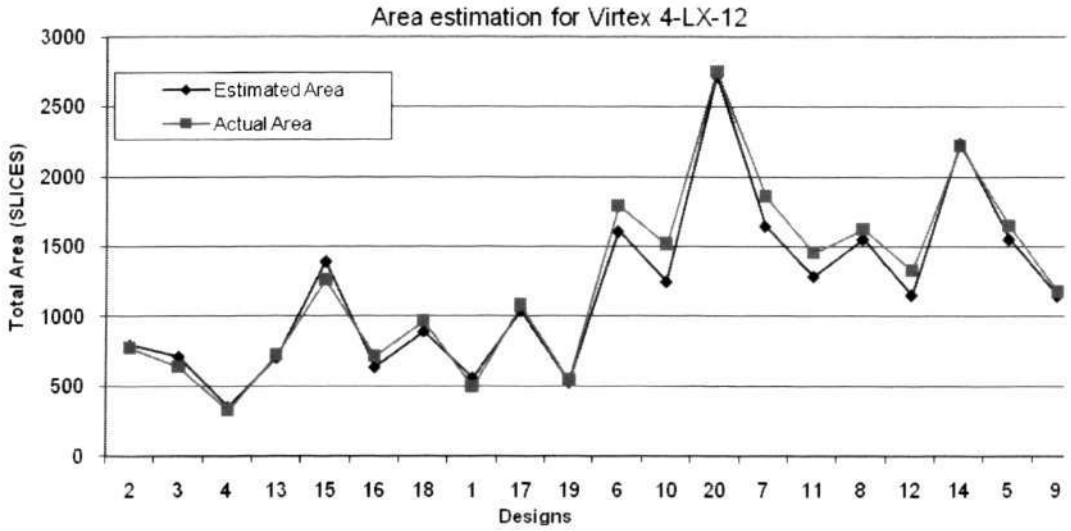


Figure 8.9 Area estimation for Virtex4LX-12

Overall for the twenty benchmarks considered, our area estimation achieved an average accuracy of 88.8%, 89.5%, and 92.4% for Virtex2Pro, Spartan3 and Virtex4 devices respectively. The respective worst case errors for the three devices are 26.7%, 21.7%, and 17.9%.

As mentioned in Chapter 2, most of the existing works in high-level estimation do not take into account the area of the control path. However this could result in high estimation inaccuracies for functions with a complex control unit (e.g. with large number of states). Column 2 and 3 of Table 8-2 shows the estimated area of the control and data-path for the 20 benchmarks. The total estimated area (sum of the estimated area of the control and data-path) and the area reported by the implementation tool is shown in columns 4 and 5. Columns 6 and 7 shows the estimation errors before and after taking into account Control-Path's area. The errors in columns 6 and 7 are calculated with respect to the actual implementation area. The last column shows how much the estimation has improved by taking into account the control path in the final estimation results. It can be observed that by incorporating the estimation of the control-path, an improvement in the estimation accuracy is achieved in almost all the benchmarks. In particular for benchmarks with a large control path (e.g. benchmark 20), an accuracy improvement of close to 10% can be achieved.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 8-2 AREA ESTIMATION WITH AND WITHOUT CONTROL PATH

	Estimated Data Path (Slice)	Estimated Control Path (Slice)	Total Estimated Area (Slice)	Actual Area (Slice)	Estimation Error (Without Control Path) (%)	Estimation Error (%)	Improvement (%)
1	539	19	558	523	3.06	6.69	-3.63
2	735	57	792	880	16.48	10.00	6.48
3	664	45	709	621	6.92	14.17	-7.25
4	327	21	348	348	6.03	0.00	6.03
5	1478	58	1536	1844	19.85	16.70	3.15
6	1528	62	1590	1927	20.71	17.49	3.22
7	1571	58	1629	1901	17.36	14.31	3.05
8	1478	58	1536	1840	19.67	16.52	3.15
9	1079	32	1111	1516	28.83	26.72	2.11
10	1191	36	1227	1492	20.17	17.76	2.41
11	1234	32	1266	1566	21.20	19.16	2.04
12	1080	32	1112	1435	24.74	22.51	2.23
13	676	25	701	742	8.89	5.53	3.37
14	2142	88	2230	2302	6.95	3.13	3.82
15	1436	44	1480	1470	2.31	0.68	1.63
16	596	38	634	679	12.22	6.63	5.60
17	974	63	1037	1230	20.81	15.69	5.12
18	828	62	890	899	7.90	1.00	6.90
19	496	29	525	536	7.46	2.05	5.41
20	2411	301	2712	3100	22.23	12.52	9.71
Ave					14.69	11.46	3.23
Max					28.83	26.72	2.11

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Evaluating the Benefits of Bitwidth Optimization

The figures below show that the area can be significantly reduced after bit-width optimization. In particular, the average reduction for number of SLICEs, number of LUTs and number of FFs is 3.12 times, 2.19 times and 1.67 times respectively. It is noteworthy that a worst case area reduction of up to 4.68 times can be achieved by incorporating bit-width optimization strategies in the proposed high-level estimation.

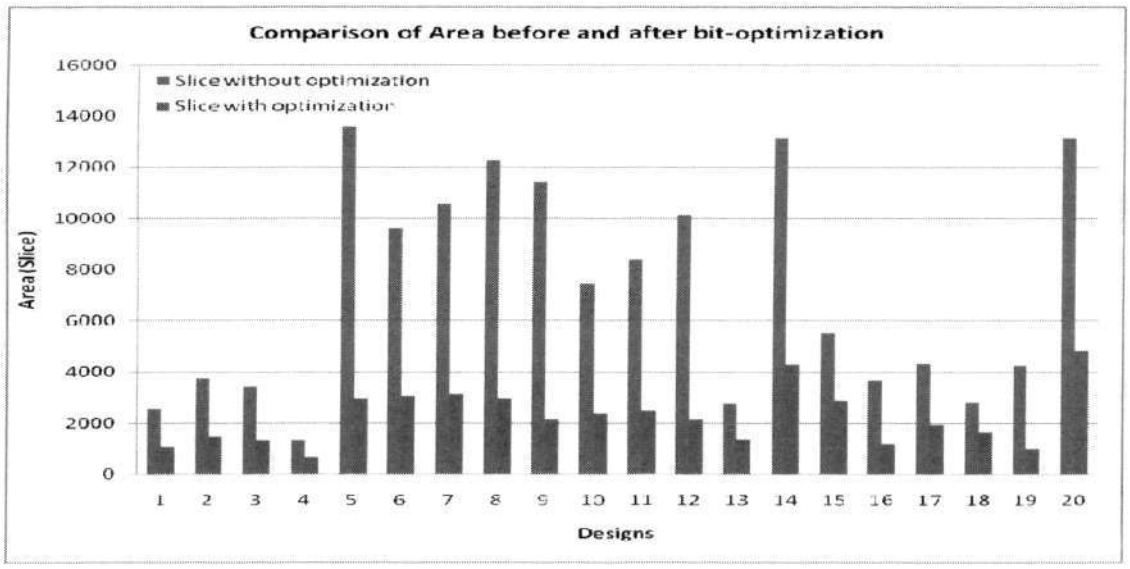


Figure 8.10 Comparison of SLICEs before and after bit-width optimization

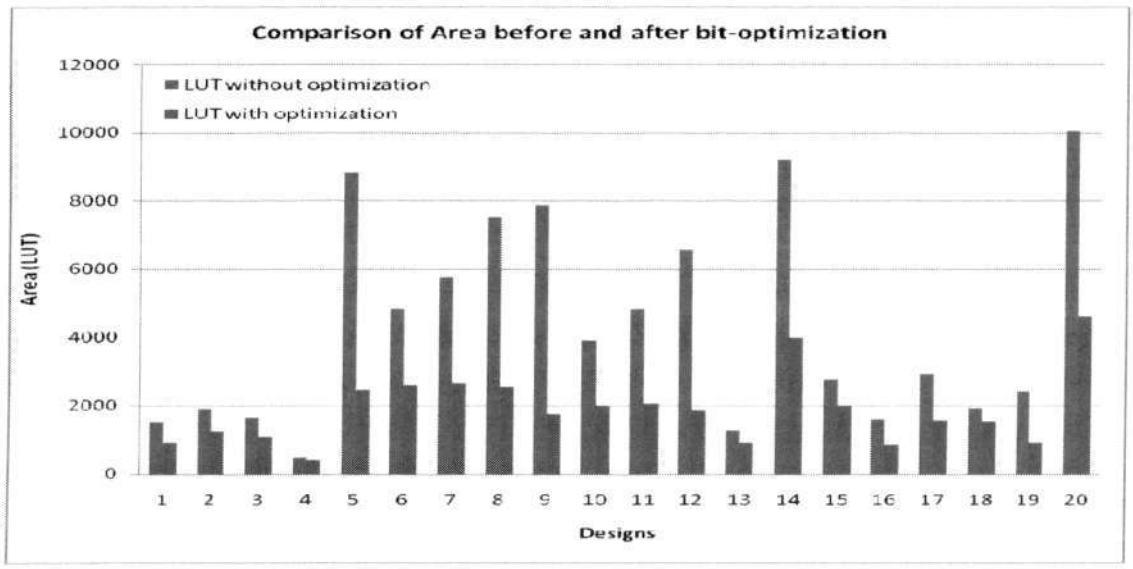


Figure 8.11 Comparison of LUTs before and after bit-width optimization

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

(b) Delay estimation

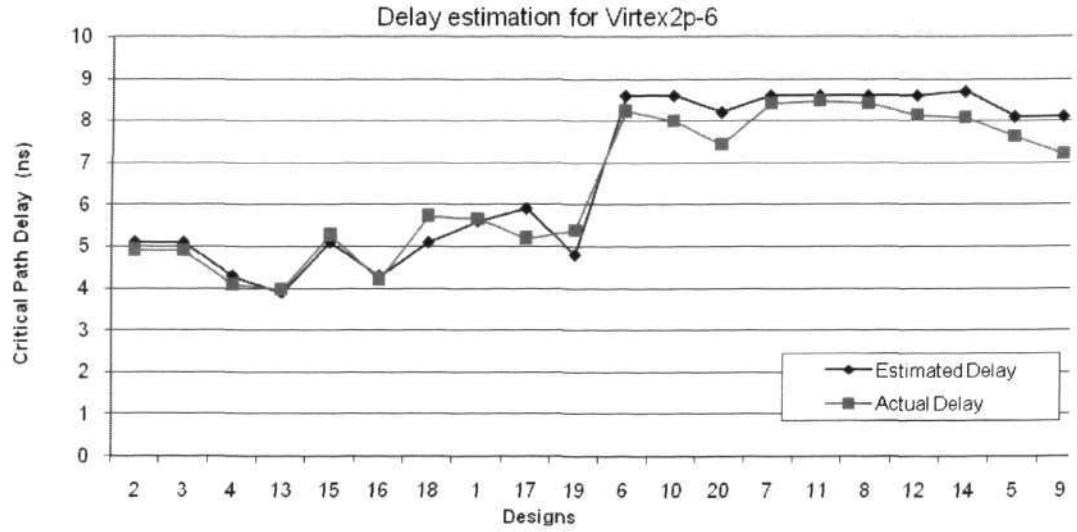


Figure 8.12 Delay estimation for Virtex2Pro-6

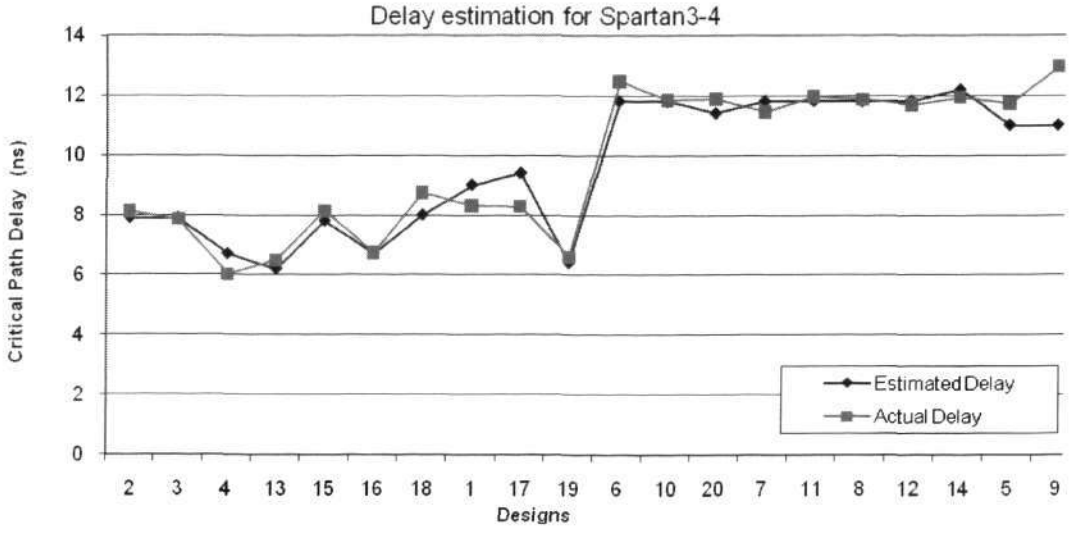


Figure 8.13 Delay estimation for Spartan3-4

Figure 8.12 and Figure 8.13 compare the estimated delay using the proposed approach with the actual delay from the commercial tool for the Virtex2Pro and Spartan3 devices respectively. We achieve very good results for delay estimation with average errors of only 5.92% and 4.83%, with worst case errors of 13.6% and 15% for the Virtex2Pro and Spartan3 devices respectively. In additon, the absolute differences between the estimated delay and the actual delay is very small. In particular, for the Virtex2Pro device, the absolute estimation error is in the range of ±1ns. To the best of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

our knowledge, there is no existing methods that is capable of estimating the delay to this degree of accuracy. This is largely due to the fact that unlike the proposed dealy estimation approach, most of the existing methods do not attempt to predict the post place and route inteconnect delay.

For the Spartan3 family, the absolute error is within ±2ns. Our investigation reveals that large errors occur in benchmarks that employ many embedded multipliers. We believe that this large errors are caused by the distribution of embedded multipliers on FPGA. For example, Figure 8.14 shows the distribution of embedded multipliers (dark blocks) for the Virtex2Pro and Spartan3 devices. The Virtex2Pro has a higher density of embedded-multipliers as compared to the Spartan3, where only two columns are allocated to the embedded multipliers. The shaded regions represents the placement of the accelerators for the Virtex2Pro and Spartan3 devices. The accelerators that make use of the embedded multipliers are often placed along the multiplier columns in order to reduce the interconnect delay between the multipliers and the related hardware components. Hence for accelerators that make use of many embedded multipliers in the Spartan3 device, the hardware components will be placed along the multiplier column as shown in Figure 8.14(b). This placement that takes advantage of the physical location of the embedded multipliers may introduce large interconnect delay between the hardware components, and this in turn causes a larger discrepancy in the estimation results for the Spartan3 device. In constrast, the placement of the components for the Virtex2Pro device is less rigid due to the ease of accesibility to the embedded multipliers, and hence we achieve a better accuracy for the Virtex2Pro device.

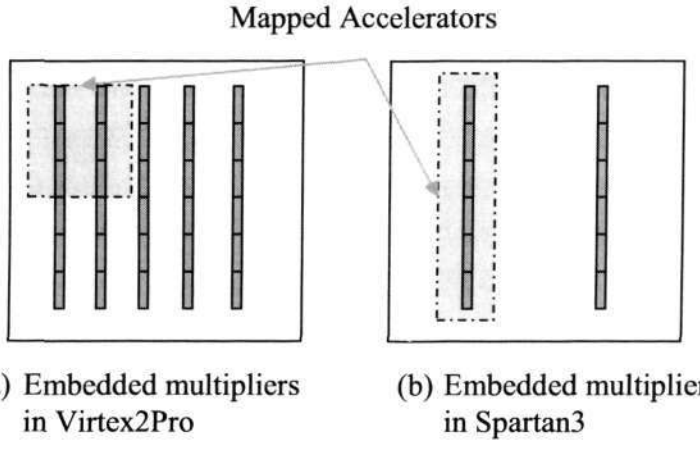


Figure 8.14 Distribution of Embedded Multipliers on Virtex and Spartan

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

We also evaluated the accuracy of our estimation with Virtex4 speedgrade(-12) family as shown in Figure 8.14. However for the Virtex-4 device, we only consider the first seven benchmarks that do not employ embedded multipliers (i.e. 2, 3, …, 16, 18), as the structure of embedded multipliers in Virtex4 is very different from those in Virtex2Pro and Spartan. The current version of CRATE does not incorporate models for estimating the embedded multipliers of Virtex4. The results show that we can achieve an accuracy of 92.3% with a worst case error of 12.2%. It is worth mentioning that the maximum delay of these benchmarks is 4.5 ns, which means that this set of circuits can run at a frequency that is higher than 200MHz.

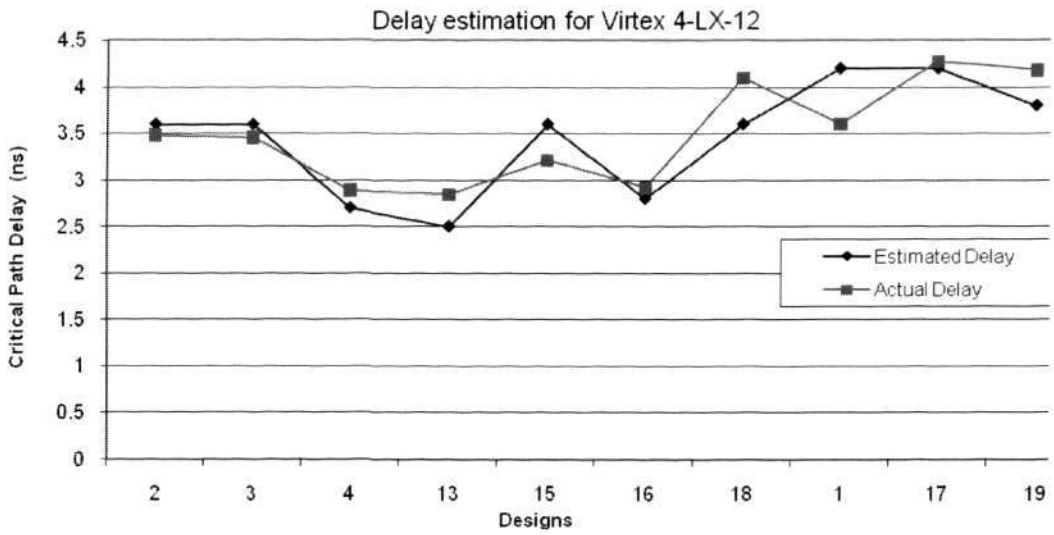


Figure 8.15 Delay estimation for Virtex4LX-12

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Evaluating the Benefits of the Simplified Floor-planning Process

Table 8-3 compares the delay estimation results of the Static Delay Estimation (Static) method and the Simplified Floor-planning (SFP) method, with respect to the actual delay. For our dataset, only three designs benefit from SFP. Two of these functions have very high errors 19.6% and 12.7%. The improvement made by SFP pass is shown in the "Improve" column of Table 8-3. The experimental results illustrate that floorplaning helps to reduce the worst case error from 19.6% downto 15% and improves the overall average accuracy.

TABLE 8-3 EFFECT OF FLOOR-PLANNING PROCESS

	Static (ns)	SFP (ns)	Actual Delay (ns)	Error Static (%)	Error SFP (%)	Improve (%)
1	5.608	5.608	5.662	0.95	0.95	
2	4.604	5.104	4.913	6.29	3.89	2.40
3	5.047	5.104	4.906	2.87	4.04	-1.16
4	4.144	4.368	4.098	1.12	6.59	-5.47
5	7.894	7.894	7.617	3.64	3.64	
6	8.394	8.394	8.217	2.15	2.15	
7	8.394	8.394	8.405	0.13	0.13	
8	8.394	8.394	8.397	0.04	0.04	
9	7.894	7.894	7.2	9.64	9.64	
10	8.394	8.394	7.987	5.10	5.10	
11	8.394	8.394	8.453	0.70	0.70	
12	8.394	8.394	8.125	3.31	3.31	
13	3.947	3.947	3.984	0.93	0.93	
14	8.473	8.473	8.071	4.98	4.98	
15	4.604	5.104	5.275	12.72	3.24	9.48
16	4.368	4.368	4.211	3.73	3.73	
17	5.974	5.974	5.193	15.04	15.04	
18	4.604	5.104	5.723	19.55	10.82	8.74
19	4.808	4.808	5.384	10.70	10.70	
20	7.973	7.973	7.421	7.44	7.44	
			MAX	19.55	15.04	
			AVE	5.55	4.85	

TABLE 8- 4 AREA AND TIME OF BENCHMARKS BEFORE AND AFTER BITWIDTH OPTIMIZATION

	Before bitwidth optimization						After bitwidth optimization						Reduction (times)		
	LUT	FF	LC	DELAY	CTRL	MULT	LUT	FF	LC	DELAY	CTRL	MULT	LUT	FF	SLICE
	1510	1024	2534	8.2	38	0	917	644	1077	5.6	38	1	1.65	1.59	2.35
	1914	1824	3738	5.8	113	0	1246	815	1470	5.1	113	0	1.54	2.24	2.54
	1642	1760	3402	6.2	89	0	1072	870	1328	5.1	89	0	1.53	2.02	2.56
	490	832	1322	5.3	41	0	399	476	654	4.3	41	0	1.23	1.75	2.02
	8805	4768	13573	8.2	115	0	2459	2955	2955	8.1	115	22	3.58	1.61	4.59
	4844	4768	9612	8.2	123	0	2599	2948	3056	8.6	123	4	1.86	1.62	3.15
	5774	4768	10542	8.2	115	0	2652	2980	3141	8.6	115	8	2.18	1.60	3.36
	7506	4768	12274	8.2	115	0	2551	2956	2956	8.6	115	16	2.94	1.61	4.15
	7877	3520	11397	8.2	64	0	1776	2158	2158	8.1	64	22	4.44	1.63	5.28
	3916	3520	7436	8.2	72	0	1989	2182	2382	8.6	72	4	1.97	1.61	3.12
	4846	3520	8366	8.2	64	0	2042	2214	2467	8.6	64	8	2.37	1.59	3.39
	6578	3520	10098	8.2	64	0	1868	2159	2159	8.6	64	16	3.52	1.63	4.68
	1290	1472	2762	5.1	50	0	935	994	1351	3.9	50	0	1.38	1.48	2.04
	9202	3936	13138	8.2	176	0	3994	2624	4283	8.7	176	16	2.30	1.50	3.07
	2762	2752	5514	5.8	88	0	1983	1860	2872	5.1	88	0	1.39	1.48	1.92
	1594	2080	3674	5.8	76	0	864	1012	1192	4.3	76	0	1.84	2.06	3.08
	2925	1376	4301	8.2	125	0	1565	1564	1947	5.9	125	1	1.87	0.88	2.21
	1930	896	2826	5.8	123	0	1527	606	1656	5.1	123	0	1.26	1.48	1.71
	2412	1824	4236	8.2	57	0	896	852	992	4.8	57	2	2.69	2.14	4.27
	10047	3072	13119	9.5	601	0	4597	1719	4822	8.2	601	4	2.19	1.79	2.72
													2.19	1.67	3.11

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 8-5 AREA-TIME ESTIMATION FOR XILINX SPARTAN3-4

Spartan3-4 Family		Estimated					Actual					Errors		
Designs	lut	ff	lc	delay	DSP		lut	ff	lc	delay	DSP	Slice (%)	Delay (%)	Delay (ns)
1	917	644	558	9	1		965	671	575	8.307	1	3.04	8.34	0.69
2	1252	815	795	7.9	0		1,268	889	761	8.115	0	4.40	2.65	0.22
3	1076	870	711	7.9	0		1,010	845	634	7.849	0	12.07	0.65	0.05
4	403	476	350	6.7	0		376	597	355	6.022	0	1.55	11.26	0.68
5	2459	2955	1535	11	22		2,635	3,214	1,838	11.723	22	16.49	6.17	0.72
6	2599	2948	1590	11.8	4		2,933	3,106	1,846	12.475	4	13.89	5.41	0.67
7	2652	2980	1628	11.8	8		2,946	3,199	1,971	11.444	8	17.40	3.11	0.36
8	2551	2956	1536	11.8	16		2,684	3,130	1,789	11.871	16	14.17	0.60	0.07
9	1776	2158	1111	11	22		2,008	2,509	1,417	12.953	22	21.59	15.08	1.95
10	1989	2182	1227	11.8	4		2,332	2,422	1,512	11.846	4	18.85	0.39	0.05
11	2042	2214	1266	11.8	8		2,342	2,512	1,507	11.968	8	16.03	1.40	0.17
12	1868	2159	1112	11.8	16		2,059	2,424	1,419	11.689	16	21.67	0.95	0.11
13	935	994	701	6.2	0		1,206	1,124	745	6.489	0	5.97	4.45	0.29
14	3994	2624	2230	12.2	16		4,024	2,871	2,410	11.932	16	7.49	2.25	1.22
15	2023	1838	1373	7.8	0		1,900	1,818	1,341	8.139	0	2.39	4.17	0.27
16	864	1012	634	6.7	0		1,052	1,114	683	6.736	0	7.17	0.53	0.34
17	1567	1564	1037	9.4	1		1,811	1,796	1,153	8.281	1	10.06	13.51	0.04
18	1527	606	890	8	0		1,710	707	899	8.749	0	1.06	8.56	1.12
19	896	852	525	6.4	2		946	766	532	6.611	2	1.41	3.19	0.87
20	4597	1719	2712	11.4	4		5,329	2,077	3,096	11.876	6	12.42	4.01	0.90
Average												10.46	4.83	-0.13
Worst Case +												21.67	15.08	1.11
Worst Case -														-1.95

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 8-6 AREA-TIME ESTIMATION FOR XILINX VIRTEX2PRO-6

Virtex2p-6 Family		Estimated					Actual					Errors		
Designs	lut	ff	lc	delay	DSP	lut	ff	lc	delay	DSP	Slice (%)	Delay (%)	Delay (ns)
1	917	644	557	5.6	1	929	671	523	5.662	1	6.59	1.10	-0.062
2	1246	815	792	5.1	0	1,266	869	880	4.913	0	10.05	3.81	0.187
3	1072	870	709	5.1	0	1,008	843	621	4.906	0	14.11	3.95	0.194
4	399	476	348	4.3	0	376	597	348	4.098	0	0.11	4.93	0.202
5	2459	2955	1535	8.1	22	2,636	3,214	1,844	7.617	22	16.74	6.34	0.483
6	2599	2948	1590	8.6	4	2,933	3,106	1,927	8.217	4	17.50	4.66	0.383
7	2652	2980	1628	8.6	8	2,947	3,198	1,901	8.405	8	14.36	2.32	0.195
8	2551	2956	1536	8.6	16	2,685	3,130	1,840	8.397	16	16.54	2.42	0.203
9	1776	2158	1111	8.1	22	2,008	2,509	1,516	7.2	22	26.69	12.50	0.9
10	1989	2182	1227	8.6	4	2,332	2,421	1,492	7.987	4	17.74	7.67	0.613
11	2042	2214	1266	8.6	8	2,342	2,511	1,566	8.453	8	19.18	1.74	0.147
12	1868	2159	1112	8.6	16	2,059	2,424	1,435	8.125	16	22.53	5.85	0.475
13	935	994	700	3.9	0	1,206	1,124	742	3.984	0	5.60	2.11	-0.084
14	3994	2624	2230	8.7	16	4,024	2,874	2,302	8.071	16	3.14	7.79	0.629
15	1983	1860	1480	5.1	0	1,845	1,872	1,470	5.275	0	0.67	3.32	-0.175
16	864	1012	634	4.3	0	1,064	1,114	679	4.211	0	6.62	2.11	0.089
17	1565	1564	1036	5.9	1	1,811	1,796	1,230	5.193	1	10.00	13.61	0.707
18	1527	606	889	5.1	0	1,711	710	899	5.723	0	1.09	10.89	-0.623
19	896	852	524	4.8	2	946	766	536	5.384	2	2.20	10.85	-0.584
20	4597	1719	2712	8.2	4	5,289	2,067	3,100	7.421	6	12.52	10.50	0.779
Average											11.20	5.92	0.2329
Worst Case +											26.69	13.61	0.9
Worst Case -													-0.623

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 8-7 AREA-TIME ESTIMATION FOR XILINX VIRTEX4-12

Virtex4-12 Family		Estimated					Actual					Errors		
Designs	lut	ff	lc	delay	DSP		lut	ff	lc	delay	DSP	Slice (%)	Delay (%)	Delay (ns)
1	917	644	558	4.2	1		933	648	494	3.6	1	12.85	16.67	0.6
2	1246	815	792	3.6	0		1,269	870	775	3.482	0	2.13	3.39	0.118
3	1072	870	709	3.6	0		1,011	940	640	3.457	0	10.70	4.14	0.143
4	399	476	348	2.7	0		375	607	329	2.889	0	5.62	6.54	-0.189
5	2497	2988	1552	5.6	22		2,470	2,699	1647	4.834	22	5.80	15.85	0.766
6	2631	2972	1606	6.5	4		2,936	3,024	1795	6.352	4	10.56	2.33	0.148
7	2684	3004	1644	6.5	8		2,895	3,056	1860	6.162	8	11.61	5.49	0.338
8	2589	2989	1552	6.5	16		2,568	2,789	1626	6.188	16	4.55	5.04	0.312
9	1884	2225	1145	5.6	22		1,842	1,991	1177	4.795	22	2.76	16.79	0.805
10	2025	2209	1245	6.5	4		2,336	2,338	1518	6.327	4	17.98	2.73	0.173
11	2078	2241	1284	6.5	8		2,290	2,369	1454	6.187	8	11.73	5.06	0.313
12	1976	2226	1151	6.5	16		1,940	2,081	1327	6.217	16	13.30	4.55	0.283
13	935	994	701	2.5	0		1,206	1,124	724	2.849	0	3.25	12.25	-0.349
14	4012	2624	2238	6.6	16		3,876	2,644	2222	6.212	16	0.72	6.25	0.388
15	2064	1838	1393	3.6	0		1,828	1,986	1264	3.21	0	10.17	12.15	0.39
16	864	1012	634	2.8	0		1,032	1,220	715	2.92	0	11.33	4.11	-0.12
17	1565	1564	1036	4.2	1		1,809	1,761	1081	4.268	1	4.16	1.59	-0.068
18	1527	606	890	3.6	0		1,636	675	969	4.094	0	8.20	12.07	-0.494
19	896	852	525	3.8	2		946	730	541	4.176	2	3.05	9.00	-0.376
20	4612	1719	2719	5.8	4		5,279	2,582	2751	6.158	7	1.16	5.81	-0.358
Average												7.58	7.59	0.14115
Worst Case +												17.98	16.79	0.805
Worst Case -														-0.494

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

8.2.3. *Experiments for Altera FPGA (CYCLONEIIC8)*

In order to test the robustness of our approach, we have performed experiments with the same set of benchmarks for Altera CycloneII FPGA. This is achieved by first computing the pre-characterized component parameters for the CycloneII device using the methods discussed in Chapter 5. The commercial tool that was used in the pre-characterizing process is Quartus II version 7.2. It is worth mentioning that the analytical models used for estimating the area-time of the control and data-paths remain unchanged for FPGAs from different vendors. The RTL codes of the h-VLIW architecture for each benchmark was also auto-generated and implemented using Quartus II version 7.2 to obtain the actual implementation results.

(a) Area estimation

Figure 8.16 shows the estimated area and actual area reported by the Quartus II tool. For area estimation, we achieve an average accuracy of 93.1% with a worst case error of only 15%. For the CycloneII device, we did not encounter the problems caused by the embedded multiplier's pipeline registers such as that discussed in previous section. This is due to the fact that similar to the Virtex-4 device, the pipeline registers are mapped onto DSP slices. Figure 8.16 clearly shows the consistency of our area estimation. The consistency is evaluated by the Fidelity Coefficient that is calculated using (7.2). The proposed area estimation approach achieves a fidelity of 95.78% for the benchmarks considered. This implies that our area estimation can correctly report the comparative relationship between two implementations with 94% confidence. This demonstrates that our area estimation method can also be applied for FPGA devices from different vendors.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

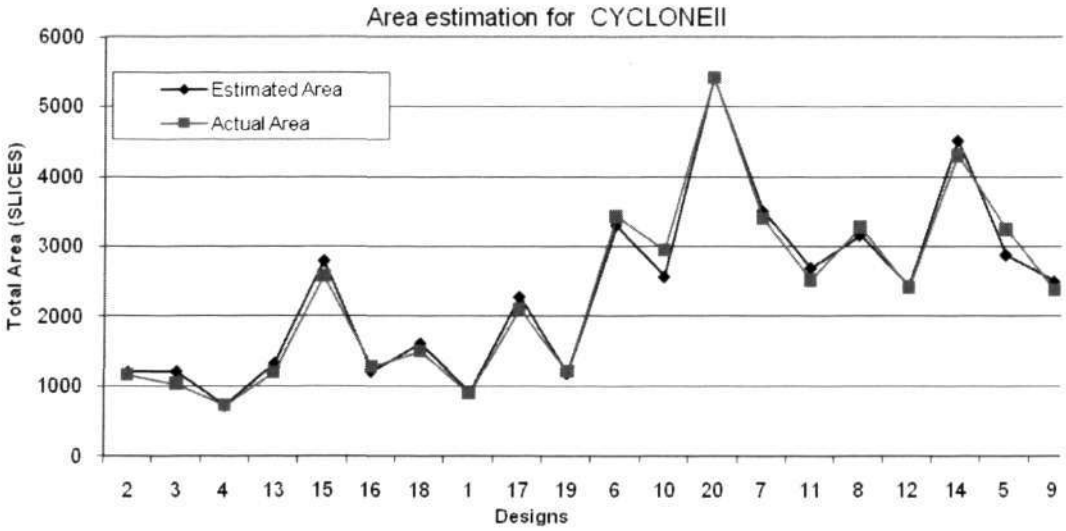


Figure 8.16 Area estimation for Altera CycloneII

(b) Delay estimation

Figure 8.17 shows the estimated delay and actual delay reported by the Quartus II tool. In order to use the SFP method for Altera CycloneII device, we need to specifiy the number of LUTs per LAB[3]. For the Xilinx devices, the number of LUTs per CLB is 8, while for the Altera CycloneII, the number of LUTs per LAB is 16. We achieve an average accuracy of 94.6% with a worst case error of 13.2% for the CycloneII device. It is noteworthy that the speed of the benchmarks considered in our experiments ranges from 12 ns (i.e 83MHz) to 6.5ns (i.e 150MHz), which is the typical execution frequency of embedded IP cores. The absolute delay error is ±1.1ns. To the best of our knowledge no existing work has performed experiments on this device. The experimental results show that our estimation approach could achieve similar accuracies for both the Xilinx and Altera devices. Hence, this demonstrates that the proposed SFP approach can be applied across different FPGA architectures.

[3] Logic Array Block which consists of 16 LUT and FF pairs

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

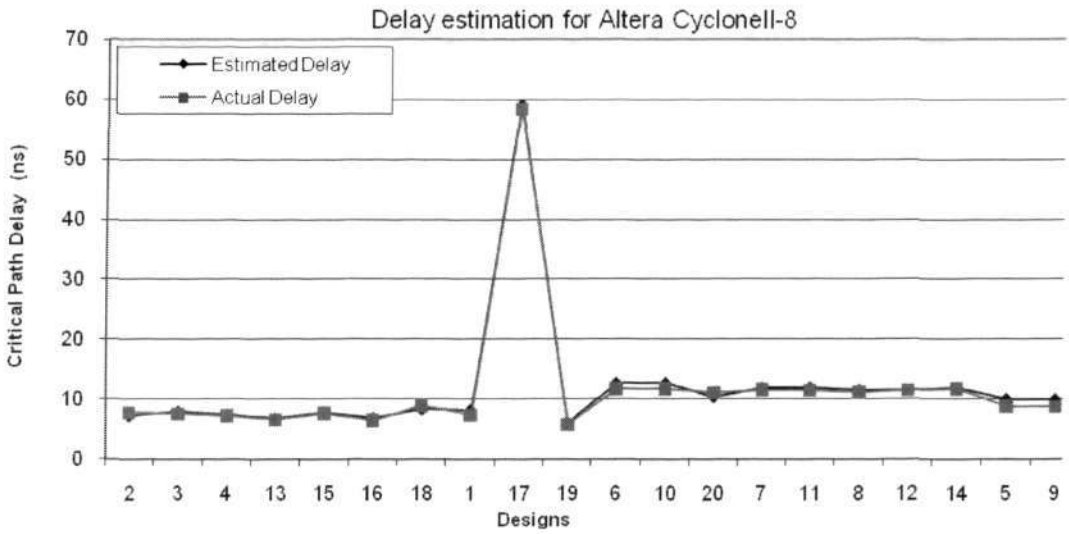


Figure 8.17 Delay estimation for Altera CycloneII

TABLE 8-8 AREA-TIME ESTIMATION FOR ALTERA CYCLONEII-8

CycloneII-8		Estimated			Actual				Errors		
Designs	lut	ff	lc	delay	lut	ff	Lc	delay	Slice (%)	Delay (%)	Delay (ns)
1	742	489	865	8.15	675	671	893	7.33	3.14	11.19	0.82
2	988	660	1258	7.19	919	778	1162	7.7	8.26	6.62	-0.51
3	913	774	1202	7.89	749	751	1037	7.47	15.91	5.62	0.42
4	403	476	690	7.4	335	666	727	7.2	5.09	2.78	0.2
5	2721	3205	2914	9.89	2276	2852	3243	8.76	10.14	12.90	1.13
6	2790	3221	3337	12.66	2309	2898	3428	11.71	2.65	8.11	0.95
7	2893	3221	3438	11.85	2320	2952	3398	11.559	1.18	2.52	0.291
8	2788	3206	3169	11.49	2298	2852	3275	11.139	3.24	3.15	0.351
9	2041	2442	2140	9.89	1736	2137	2377	8.737	9.97	13.20	1.153
10	2116	2458	2575	12.66	1755	2160	2946	11.73	12.59	7.93	0.93
11	2218	2458	2669	11.85	1797	2244	2519	11.38	5.95	4.13	0.47
12	2114	2443	2401	11.49	1791	2134	2414	11.38	0.54	0.97	0.11
13	884	1091	1328	6.65	819	974	1193	6.5	11.32	2.31	0.15
14	3974	2593	4588	11.85	3492	2658	4302	11.75	6.65	0.85	0.1
15	2106	1837	2794	7.78	1805	1804	2574	7.57	8.55	2.77	0.21
16	876	959	1209	6.75	863	1065	1282	6.45	5.69	4.65	0.3
17	2043	863	2320	59.15	1586	1082	2100	58.364	10.48	1.35	0.786
18	1424	575	1617	8.26	1286	708	1503	8.95	7.58	7.71	-0.69
19	1051	852	1177	5.95	1108	739	1216	5.77	3.21	3.12	0.18
20	5101	1719	5781	10.34	5011	2043	5419	11.083	6.68	6.70	-0.743
Average									6.94	5.43	0.3304
Max (upper bound)									15.91	13.20	1.153
Max (lower bound)											-0.743

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

8.2.4. *Comparison of the Execution Time*

Table 8-9 compares the execution time for area-time estimation (CRATE) with the runtime of the logic synthesis engine (Syn) and the runtime of the place and route process (PAR) for the twenty benchmarks. The execution time of the estimation is compared with the runtime of both the Xilinx ISE and Quartus II tools. The results show that CRATE can estimate the area-time with a high degree of accuracy in a significantly shorter amount of time when compared to Syn and PAR of both the commercial tools. In general, CRATE completes the estimation in less than half a second. This is significantly faster that the implementation runtime of Xilinx and Quartus, which typically take hours to report the final area-time information. These results confirm the suitability of the proposed framework to be used for rapid design exploration.

TABLE 8-9 COMPARISON ON ESTIMATION TIME AND EXECUTION TIME

		Xilinx ISE 9.1			Quartus II 7.2			
	CRATE (second)	Syn (sec)	PAR (min)		Syn (sec)	PAR (min)	Speedup (Xilinx)	Speedup (Altera)
1	0.22	102	1.58		14	2.70	896	800
2	0.37	67	34.00		21	2.75	5695	503
3	0.31	60	2.40		18	2.73	657	587
4	0.2	41	48.00		18	2.73	14605	910
5	0.48	136	25.00		80	9.05	3409	1298
6	0.49	111	1.98		80	9.05	470	1271
7	0.52	116	1.75		80	9.05	426	1198
8	0.47	168	1.40		80	9.05	537	1326
9	0.38	147	55.00		36	6.77	9070	1163
10	0.38	97	2.10		36	6.77	588	1163
11	0.35	103	1.85		36	6.77	611	1263
12	0.37	114	1.18		36	6.77	499	1195
13	0.28	53	55.00		17	2.43	11975	582
14	0.48	182	2.32		60	8.75	669	1219
15	0.51	87	3.33		37	6.43	563	829
16	0.29	64	1.52		22	2.90	534	676
17	0.28	89	1.75		30	4.45	694	1061
18	0.38	67	1.40		25	5.75	398	974
19	0.33	71	53.00		21	2.20	9852	464
20	0.9	237	5.88		117	11.23	656	879
	Average						3140	968

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

8.3. Comparison with Altera C2H

Altera C2H software is an EDA tool that is used for automatically translating algorithms written in ANSI-C into RTL codes that can be implemented as a hardware accelerator using Altera Quartus II FPGA tool. The accelerator is interfaced into Altera's Embedded System through Avalon bus. The advantage of this tool is that the hardware accelerator can be generated with minimum modifications in the C codes. This section compares the RTL design generated by CRATE and RTL generated by Altera C2H. The functions in Table 8-1 were compiled using C2H tool to generate the hardware design. Figure 8.18 shows area-time product of CRATE's RTL and C2H's RTL, which is computed by multiplying the circuit's area (in Slice) with the circuit's minimum clock period (in ns). It can be observed that the area-time product of the hardware generated by CRATE is smaller for most cases. CRATE has three notable advantages over C2H that needs to be highlighted. Firstly, the C2H tool cannot facilitate design exploration as it automatically determines a fixed number of functional units. In contrast, CRATE is capable of generating varying hardware configurations for a single function. For example in the experiments, four implementations for the IDCTROW and IDCTCOL functions have been generated with different area-time characteristics. Secondly, CRATE employs Trimaran which has better optimization features compared to the C2H tool. For example in the case of the Viterbi-ACS function, Trimaran was able to convert a division operation into a shift operation, which led to more efficient area-time product compared to the C2H tool. In addition for large functions (e.g. Viterbi_All), C2H produced a cost inefficient hardware which could not be fitted onto the available FPGA devices. In constrast, Trimaran's in-lining optimization is able to flatten the Viterbi_All function in order to perform more optimizations on the function. Finally, the compilation time of CRATE is much faster than C2H (less than a second compared to tens of minutes).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

TABLE 8-10 COMPARISON BETWEEN C2H'S RTL AND CRATE'S RTL

#	Functions	CRATE's RTL		C2H's RTL	
		Area (LC)	Min Clock Period (ns)	Area (LC)	Min Clock Period (ns)
1	AutoCorrelation	865	8.15	1058	8.05
2	ADPCM_CODER	1258	7.19	2338	9.85
3	ADPCM_DECODER	1202	7.89	1900	9.19
4	COMB_SORT	690	7.4	974	8.75
5	IDCTCOL_2MUL	4447	16.05	3700	18.579
6	IDCTROW_2MUL	3359	16.05	2985	16.24
7	SHA	1328	6.65	1466	9.74
8	CJPEG2_FixpointDCT_17mul	4257	9.89	4212	11.33
9	cjpegv2-_rgb_ycc_convert	2794	7.78	2900	11.709
10	Convolutional Encoding	1209	6.75	1323	8.5
11	FixPointBit_Allocation_Div	2320	59.15	2000	58.55
12	Viterbi_ACS	1617	8.26	3500	57.25
13	Viterbi_FindMetics	1177	5.95	1133	7.768
14	Viterbi_All	5781	10.34	14037	not available

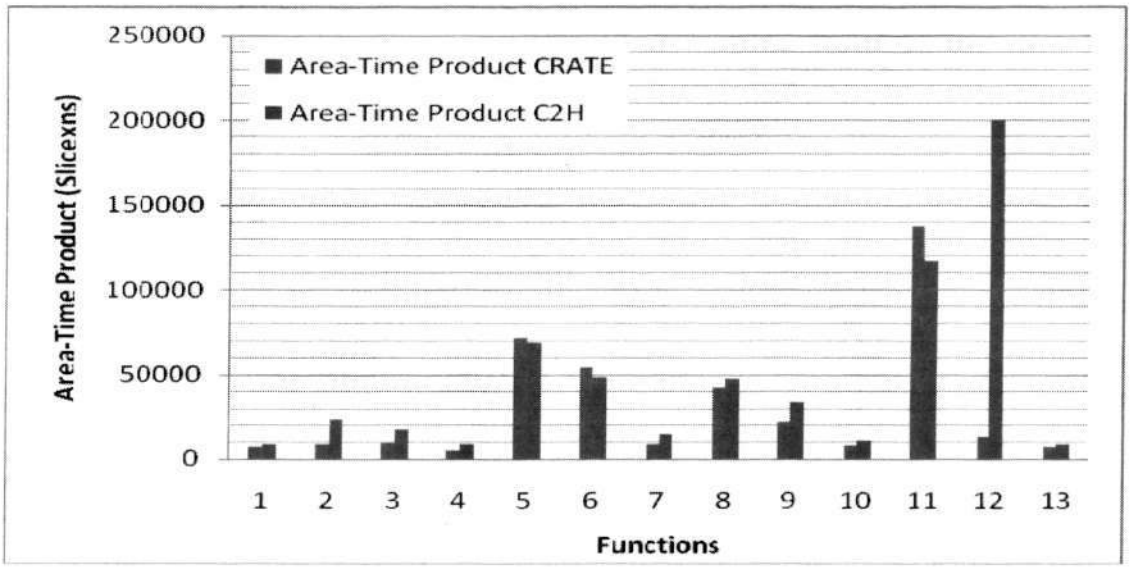


Figure 8.18 Comparison on Area-time Product of CRATE's RTL and C2H's RTL

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

8.4. Summary

In this chapter, we presented the proposed framework (CRATE) for high-level estimation. We also presented experimental results to demonstrate the reliability of the proposed estimation techniques as well as the efficiency of the proposed h-VLIW architecture. Our proposed framework employs Trimaran as the front-end C-compiler to expose the inherent parallelism of the applications and to perform high-level optimizations. Trimaran then generates IR files which contain scheduling information of the C-applications. CRATE parses the IR files and performs resource binding, bitwidth optimization, and area-time control/data path estimation. CRATE also automatically generates synthesizable HDL codes for the HWA. The final report of CRATE provides the following analysis: 1) estimated area measures for the HWA in terms of SLICEs, LUTs, FFs and number of embedded multipliers, 2) maximum clock speed at which HWA can run.

We have performed high-level estimation for a number of commonly-used embedded functions to demonstrate the capability of CRATE to facilitate rapid design exploration for hardware-software partitioning. In particular the proposed tool CRATE is capable of:

1) Reporting reliable area-time information. In particular, our area estimation achieves average accuracy of 93% to 95% for medium sized functions, with estimation fidelity of 94%. In the worst case, the area estimation error was found to be 26%. We have identified the reason for the discrepancy that have led to the worst case estimation error, and will be incorporating strategies to overcome this problem in future work. The proposed area estimation method can achieve an average fidelity of 95%.

2) Performing area-time estimation for different FPGAs architecture. We have demonstrated that our framework can target different Xilinx FPGA families namely the Virtex2Pro, Spartan3, and Virtex4 devices. In addition, we have shown that consistent estimation results can be achieved for FPGA devices from a different vendor, i.e. Altera Cyclone II device. We believe that the framework can be employed for other more advanced FPGA such as the Xilinx Virtex5 or Altera Stratix with minimal modifications.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3) Rapid estimation. The average runtime required to estimate the area-time of the HWA is less than 0.5s which is significantly shorter (~1000 times) than the execution time of the logic synthesis engine and place-and-route process of commercial tools.

4) Estimating area-time of a wide range of applications. The experiments undertaken employ realistic benchmarks that range from 300 slices (60k gates) to 3000 slices (600k gates). This demonstrates the feasibility of the proposed framework for estimating the area-time of applications with varying degree of complexities. In addition, our tool is capable of estimating HWAs that can execute at high frequencies (i.e. upto 200MHz) with an average accuracy of 90%.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

CHAPTER 9 CONCLUSIONS AND FUTURE WORK

9.1. Conclusions

This thesis has presented techniques necessary to implement a framework (named CRATE) for high-speed area-time trade-off analysis to facilitate design space explorations.

The Trimaran compiler infrastructure has been adopted for front-end processing in CRATE as it incorporates a wide range of optimizations and features to support efficient FPGA implementation. The original Trimaran machine description has been modified to target the hybrid VLIW (h-VLIW) architecture template for area-time efficient hardware mapping and high-level estimation. The h-VLIW architecture is based on the VLIW architecture that is augmented with CFUs and application-specific interconnectivity. CRATE extracts the scheduling information from Trimaran's IR, and performs resource binding to assign the operations onto the CFUs to 1) satisfy the data/control dependency and 2) minimize the interconnectivity between functional units and registers. The outcome of resource binding is an h-VLIW-based hardware that incorporates the necessary CFUs to implement the application, with low interconnectivity complexity between CFUs and registers. High-level area-time estimation is performed based on the resulting h-VLIW architecture.

We have proposed two reliable delay estimation techniques by taking into account the post implementation interconnect delay. The first method (SLK) relies on slack distribution of delay paths for estimating the post-PAR interconnect-delay [13]. In this method, the λ parameter was used to model the slack-distribution of the circuit. Our investigations reveal that data-paths with high-λ impose greater placement difficulty on the FPGA tool, and this leads to highly unpredictable interconnect delay. The SLK method predicts the post-PAR critical path delay by multiplying the delay obtained from static delay estimation with a constant for data-paths with high-λ. The second method (SFP) relies on a simplified-floorplaning procedure to estimate the relative distances between RTL components after PAR [14]. This approximated distance can then be used to derive the additional delay caused by the physical placement of the components on the FPGA architecture. We have verified the proposed techniques by comparing the estimated results with actual post place and route results of commonly

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

used embedded functions such as mpeg2, fft, etc. from the EEMBC benchmark set. The proposed techniques achieved an average accuracy of 90% when compared to actual implementation results. The average estimation runtime is only 0.3s, thereby enabling rapid design space explorations.

For area estimation, *Slices* is a more realistic indication of FPGA utilizations compared to LUTs and FFs. By analyzing the interconnect patterns between LUTs and FFs at high-level to predict the packing of LUTs and FFs onto the same slice, we achieved average accuracy of 90%. Unlike previous works, the proposed approach for slice estimation does not require constant calibration for performing estimation on different devices.

We have adopted the one-hot encoding scheme for control path estimation, as our investigations show that this scheme leads to good performances for FSM with large number of states. Results from Xilinx implementation tool confirm that the one-hot encoding scheme is the preferred choice for implementing FSMs in widely known applications. Area-time estimation of the NSDL and CSDL components are performed separately by relying on a set of equations that model how the components are mapped onto the FPGA architecture. The number of LUTs required for the NSDL is estimated as twice the number of conditional branches found in the IR. The number of LUTs required for CSDL is estimated based on the OR-tree structure that provides the fastest implementation. Since the number of slices is a better indication of the FPGA area utilization, we have formulated a model that estimates the number of slices by predicting how the LUTs and FFs are packed onto the FPGA architecture. This approach yields estimation error (for number of slices) only 5.1%, while the average delay-estimation error is less than 1ns. These results demonstrate the effectiveness of the proposed technique for area-time estimation of the control path.

As discussed in chapter 7, ignoring the effects of logic optimizations by synthesis tools that prunes away unused logic can lead to large errors and unrealistic estimation. CRATE have incorporated the following logic optimizations in the proposed estimation process: 1) bit-width analysis to determine the required bit-width of the hardware components based on the type of operations involved and 2) constant propagation to identify segments of the hardware components that do not require actual logic due to the fact that the operands are always constants. Based on the information from bit-width analysis and constant propagation, we can perform bit-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

width specific area-time estimation for each RTL component. Our experiments revealed that by taking into account the bit-width optimization effects, a circuit area reduction of 2 to 3 times can be obtained.

The efficiency of the proposed hardware generator has been compared with the commercial Altera C2H RTL generator. When compared to the Altera C2H tool, the HDL generated by the proposed framework leads to higher performance in almost all cases with comparable area measures. In addition, the applicability of the proposed methods to a variety of devices such as Xilinx Low Cost Spartan3 family, Xilinx Virtex II pro family, Xilinx Virtex 4 family and Altera Cyclone II family has been demonstrated.

These techniques have been successfully incorporated into a high-level estimation (CRATE) framework implemented on the Linux platform. Finally, unlike existing commercial tools, the proposed framework can be relied upon to facilitate area-time trade-off analysis at high-speed.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

9.2. Future Works

In this thesis, the proposed high level estimation techniques have been evaluated on the Xilinx Virtex2, Virtex4 and Spartan3 devices. Experimental results have demonstrated that the proposed estimation techniques can achieve a high degree of accuracy for these devices. In order to further demonstrate the capabilities of the proposed techniques, evaluations must be undertaken for the newer Virtex5 family and Stratix2 architectures. The LUT structures of the Virtex 5 and Stratix architectures are quite different from the previous FPGAs. Since the proposed hardware estimation models takes into account the number of LUT inputs and routing structure of the targer FPGA, there may be a need to slightly modify the proposed estimation models to support the Virtex5 and Stratix2 architectures.

The proposed framework relies on the Trimaran compiler infrastructure to generate IR files, which contain scheduling information of the C-applications, in order to perform high level estimation. As mentioned in Chapter 3, the hardware model and programming model of Trimaran may lead to inefficiencies when compared to hand-coded RTLs. This is due to the fact that the Trimaran compiler infrastructure was built to target VLIW architectures, which consist of a limited set of generic components. In this research, we have modified the Register File and Functional unit components of the Trimaran machine description to generate more realistic RTL codes. However, in order to generate a fully dedicated hardware model that is suitable for FPGA implementation, the following modifications should also be incorporated:

- Multi-bank memory system: This is used to model the block-RAM resources on FPGA. The memory description system of Trimaran must be modified to allow description of multi-bank memory system that supports discontiguous memory accesses. Elcor must also be modified to utilize this type of memory sytem. Finally, pragmas for memory allocations should be supported by the front-end C parser.
- Branching model: Trimaran constructs the control flow of applications based on the implied microprocessor-based branching model which allows either branch or fall-through. For complex "switch" statements, the VLIW machine takes a few cycles before identifying the correct state. In constrast, a dedicated hardware can perform comparisons in the "switch" statement in parallel and identify the correct state in one clock cycle. Thus, a method that can map the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

switch structure into hardware that enables parallel comparison would improve the quality of the circuit.

- Predicated operations support: Currently CRATE does not support predicated operations. Predicated operations would reduce the overhead caused by branching instructions.
- Software pipelining: Software pipelining is crucial for hardware acceleration as it improves the system's throughput significantly. Trimaran supports software pipelining using Rotating Register File. Currently CRATE does not incorporate RTL models for Rotating Register File and therefore cannot support software pipelining. This remains as one of the most important work in the further development of CRATE.
- Automatic operations chaining/grouping: Each operation in the Trimaran architecture model has a latency of one clock cycle. As discussed in Chapter 3, the primitive operations of Trimaran can be chained or grouped into more efficient application specific functions. Although Trimaran provides a means for defining these new functions, it is incapable of automatically identifying them from the application. A procedure that can automatically identify and group the primitive operations to form new application specific functions can lead to a more efficient hardware accelerator.

Finally, a design exploration framework that employs CRATE for rapid high level estimation must be developed. Strategies that can intelligently vary the number of resources allowed in the hardware accelerator must be adopted to enable the framework to efficiently explore the design space. A framework that incorporates rapid and reliable high level estimation, along with intelligent design exploration strategies would be indispensible for overcoming future market and technological challenges in embedded systems.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

BIBLIOGRAPHY

[1] Denice Cabel. Trend shaping FPGA Industry. http://www.edn.com/article/CA6389417.html. EDN Asia. 11/9/2006.

[2] Ahmed A Jerraya. Long Term Trends for Embedded System Design. In Proceedings of EUROMICRO Systems on DSD'04. pp. 20-26. 2004.

[3] Tensilica Corporation. www.tensilica.com. 2008

[4] Bryan Lewis, Ivo Bolsens, Rudy Lauwereins, Chris Sheddon, Bhushan Gupta, and Yankin Tarnuhan. Reconfigurable SoC--what will it look like? In Proceedings of Design, Automation and Test in Europe Conference and Exhibition. pp. 660 - 662. 2002.

[5] Xilinx Corporation. PowerPC Embedded Core. http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex4/capabilities/powerpc.htm. September 2008.

[6] ChipEstimate.com. Incyte Chip Estimator. http://www.chipestimate.com/overview.html. September 2008.

[7] Mentor Graphic Corporation. Catapult C. http://www.mentor.com. 2008.

[8] Synfora Corporation. http://www.synfora.com. 2008.

[9] Xilinx Corporation. www.xilinx.com. 2008.

[10] Altera Corporation. www.altera.com. 2008.

[11] L. N. Chakrapani et al. Trimaran: An infrastructure for research in instruction-level parallelism. In *Lecture Notes in Computer Science (Langurages and Compilers for High Performance Computing)*. pp. 32–41. September 2004.

[12] Robert Schreiber PICO-NPA: High-level synthesis of non-programmable hardware accelerators. In Journal of VLSI Signal Processing Systems. Vol 31, Issue 2, pp. 127-142. June 2002.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[13] Lieu My Chuong, Lam Siew Kei, Thambipillai Srikanthan. "High-Level Delay Estimation Technique for Porting C-based Applications on FPGA". In Proceedings of IEEE ISIE. pp. 1991-1996. 2008.

[14] Lieu My Chuong, Lam Siew Kei, Thambipillai Srikanthan. "Rapid Area-Time Estimation Technique for Porting C-based Applications on FPGA". Journal of Scalable Computing: Practice and Experience. pp. 359-371. 2008.

[15] Embedded System. http://en.wikipedia.org/wiki/Embedded_system. July 2007.

[16] Wikipedia. Field-programmable gate array. http://en.wikipedia.org/wiki/Field-programmable_gate_array. September 2008.

[17] SiliconBlue Corporation. www.siliconblue.com. 2008.

[18] Daneil D. Gajski et al. Specification and Design of Embedded System. Prentice Hall. 1994.

[19] Katherine Compton, Scott Hauck, Reconfigurable Computing: A Survey of Systems and Software, ACM Computing Surveys, Vol. 34, No. 2, pp. 171–210, June 2002.

[20] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk and P.Y.K. Cheung, Reconfigurable computing: architectures and design Methods, IEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, pp.193-207, March 2005.

[21] Min Joong Rim, Rajiv Jain. Lower-bound performance estimation for high-level synthesis scheduling problem. In IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 8, pp. 451-458, April 1994.

[22] Minjoong Rim and Rajiv Jain. 1994. Estimating performance characteristic of Loop transformation. In Proc. IEEE International Symposium on Circuits and Systems. ISCAS '94. pp. 249 – 252. 1994.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[23] Jain, R.; Parker, A.C.; Park, N. Predicting system-level area and delay for pipelined and non-pipelined designs. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. pp. 955-965. 1992.

[24] Chauduri S and Walker. Computing the lower bound of functional unit before scheduling. In Proceedings of the 7th international symposium on High-level synthesis ISSS '94. pp. 273-279. 1994.

[25] A. Nayak, M.Haldar, A.Choudhary, and P.Banerjee. "Accurate Area and Delay Estimators for FPGA". In Proceedings of International Conference DATE, Paris, France. pp. 862-869. Mar 2002.

[26] Dhananjay Kulkarni et al. Compile Time Area estimation for LUT-based FPGAs. In ACM Trans. Design Automation of Electronic Systems, vol. 11, pp. 104-122. Jan 2006.

[27] Sebastien Bi Lavarn et at. Design Space Pruning through estimations of Area/Delay Trade-off for FPGA. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. Volume 25, Issue 10, pp. 1950-1968. October 2006.

[28] João M. P. Cardoso. On Estimations for Compilation of Software for FPGA. Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05). pp. 225-230. 2005.

[29] Min Xu. Fadi J. Kurdahi. 1996. Area and timing estimation for lookup table based FPGA. Proceedings to ED&TC. pp. 151-157. March 1996.

[30] Bjureus, P.; Millberg, M.; Jantsch, A. FPGA resource and timing estimation from Matlab execution traces. Proceedings of the Tenth International Symposium on Hardware/Software Codesign CODES. pp. 31-36. 2002.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[31] Carlo Brandolese et al. An area estimation methodology for FPGA based Designs at system C- level. In Proc. of Design Automation Conference. pp. 129-132. June 2004.

[32] Carsten Menn et al. Controller estimation for FPGA target architectures during high-level synthesis. Proceedings of the 15th international symposium on System Synthesis. pp 56 – 61. 2002.

[33] Tanyi Jiang. Macro-models for high level area and power estimation on FPGAs. Proceedings of the 14th ACM Great Lakes symposium on VLSI. pp 162 – 165. 2004.

[34] Mitrionic Corporation. www.mitrionics.com. 2008.

[35] Celoxica Corporation. www.celoxica.com. 2008.

[36] Impulse Accelerated Technologies. http://www.impulsec.com/. 2008.

[37] Trident Compiler. http://trident.sourceforge.net/. 2008.

[38] Handel C Language Reference Manual. www.celoxica.com. 2008.

[39] Tanay Karnik and Sung-Mo Kang. An Empirical Model For Accurate Estimation of Routing Delay in FPGAs. In Proc. IEEE/ACM Int. Conf. on Computer-aided design ICCAD '95. pp. 328-331. Dec 1995.

[40] Micheal Hutton. Interconnect Prediction for Programmable Logic Devices. In Proceedings of International Proceedings on SLIP. pp. 125-131. 2001.

[41] Valavan Manohararajah et al. Difficulty of Predicting Interconnect Delay in a timing driven FPGA CAD Flow. In Proceedings of International Proceedings on SLIP. pp 3-8. 2006.

[42] Seong Y. Ohm et al. A comprehensive estimation technique for High-level Synthesis. In Proc. of the Eighth International Symposium on System Synthesis. pp. 122-127. Sept 1995.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[43] Taraneh Taghavi, Soheil Ghiasi. Innovate or Perish: FPGA Physical Design. Proceedings of the 2004 international symposium on Physical design. pp 148 – 155. 2004.

[44] Wikipedia. Electronic System Level. http://en.wikipedia.org/wiki/Electronic_system_level. September 2008.

[45] University of California. SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits. http://mesl.ucsd.edu/spark/. 2008.

[46] Standford University. SUIF compiler. http://suif.stanford.edu/. 2008.

[47] Trimaran. Machine-dependent analysis and optimizations: functionality and capabilities. http://www.trimaran.org/documentation.shtml. 2008.

[48] EEMBC. The embedded microprocessor benchmark consortium. http://www.eembc.org/home.php. 2008.

[49] Wen -Mei W. Hwu et al. The superblock: An effective technique for VLIW and superscalar compilation. Springer Netherlands' The Journal of Supercomputing. pp. 229-248. 1993.

[50] Raymond R. Hoare et al. Rapid VLIW Processor Customization for Signal Processing Applications Using Combinational Hardware Functions. EURASIP Journal on Applied Signal Processing. Volume 2006, Article ID 46472, Pp. 1–23. Hindawi Publishing Corporation. 2006.

[51] Mahlke, S.; Ravindran, R.; Schlansker, M.; Schreiber, R.; Sherwood, T. Bitwidth-cognizant architecture synthesis of custom hardware accelerators. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. pp. 1355-1371. November 2001.

[52] Xplorer Technology. http://www.xilinx.com/products /design_tools /logic_design /implementation/ xplorer.htm.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[53] Byongro So et al. Using Estimates from Behavioral Synthesis Tools in Compiler Directed Design Space Exploration. Proc. of the 40th Conf. on Design automation DAC '03. Pp. 514-519. 2003.

[54] Tensilica Inc. *Xtensa Architecture and Performance*, September 2002. http://www.tensilica.com/xtensa arch white paper.pdf.

[55] Henkel, J.; Ernst, R. High-level estimation techniques for usage in Hardware/Software Co-design. Asia and South Pacific – Automation Design Conference 1998. pp 353-360. Feb 1998.

[56] Codesign Research Group at University of Berkeley. Introduction on POLIS system. http://embedded.eecs.berkeley.edu/Research/hsc/abstract.html. July 2008.

[57] Lanping Deng, Kanwaldeep Sobti, Chaitali Chakrabarti. Accurate models for estimating area and power of FPGA implementations. Proceedings of ICASSP 2008. pp 1417 – 1420. April 2008.

[58] State encoding scheme. Xilinx tool box.

[59] Micaela Serra and William B. Gardner. Hardware/Software Codesign - introducing an interdisciplinary course. http://cs.ubc.ca/wccce/program98/micaela/micaela.html. 1998.

[60] Shankar Balachandran and Dinesh Bhatia. A-priori wirelength and interconnect estimation based on circuit characteristics. In Proc. International workshop on SLIP '03. pp. 77-84. April 2003.

[61] M. Feuer. Connectivity of Random Logic. IEEE Trans. Computers, 31(1), (1982), pp.29-33. 1982.

[62] Lou Scheffer, Eric Nequist. Why Interconnect Prediction Doestn't Work. In Proc. to International Conference on SLIP. pp. 139-144. 2000.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[63] Welson Sun et al. FPGA Pipeline Synthesis Design Exploration using Module Selection and resource sharing. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems. pp. 254-265. Feb 2007.

[64] Benoit Miramond. Decision Guide Environment for Design Space Exploration. In Proceedings of International conference of Emerging Technologies and Factory Automation, EFTA. pp. 880-888. 2005.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

APPENDIX A.1 Synthesis Script to characterize components

```
#!/usr/bin/perl -s
@DataWidths = (32);
@PARTS=("xc2vp70-6ff1704","xc3s5000-4fg1156","xc2v4000-6ff1152");
@Designs=("EmbMultiplier_32bit");

foreach $design(@Designs)
{
foreach $family(@PARTS)
{
$syn_op_filename="Synoptions.xst";
$projectfilename = "CharacterizeHW.prj";
$bit="bit";
$OutputFiles="$design";
$foldername="Characterize_$design\_$family";
$command="mkdir $foldername";
system($command);
#create project file
open(PRJFILE,">$projectfilename");
#print PRJFILE "vhdl work \"TestSource/mux_32_8to1.vhd\"\n";
#print PRJFILE "vhdl work \"TestSource/mux_32_16to1.vhd\"\n";
#print PRJFILE "vhdl work \"TestSource/mux_32_4to1.vhd\"\n";
#print PRJFILE "vhdl work \"TestSource/mux_32_2to1.vhd\"\n";
#print PRJFILE "vhdl work \"TestSource/Register16bit.vhd\"\n";
#print PRJFILE "vhdl work \"TestSource/FU.vhd\"\n";
print PRJFILE "vhdl work \"TestSource/$design.vhd\"\n";
print PRJFILE "vhdl work \"TestSource/toplevel_$design.vhd\"\n";
close PRJFILE;

#create synthesis options file
open(SYNOPS,"SynthesisOptionTemplate.xst");
open(OPTIONS,">$syn_op_filename");
@array=<SYNOPS>;
foreach $line(@array)
{
        @words=split(" ",$line);
        foreach $word(@words)
                {
                if($word eq '_PART_NAME_')
                        {print OPTIONS "$family";}
                elsif($word eq '_PROJECT_FILE_'){print OPTIONS
"$projectfilename";}
                elsif($word eq '_OUTPUT_FILE_'){print OPTIONS "$OutputFiles";}
                else {print OPTIONS $word;print OPTIONS " "};
                }
        print OPTIONS "\n";
}
close(OPTIONS);
close(SYNOPS);
```

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

```
$synthesiscommand="xst -intstyle xflow -ifn $syn_op_filename -ofn
$OutputFiles\.syr";
system($synthesiscommand);

$translatecommand="ngdbuild -intstyle xflow -i -p $family $design.ngc $design.ngd";
system($translatecommand);
$mapcommand="map -intstyle xflow -p $family -cm area -pr b -k 4 -c 100 -tx off -o
$design.ncd $design.ngd $design.pcf";
system($mapcommand);

$command = "copy $syn_op_filename.* $foldername";
system($command);
$command = "copy $OutputFiles.* $foldername";
system($command);
system("del $projectfilename");
system("del $OutputFiles.*");
}
}
```

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

APPENDIX A.2 **Algorithm to compute number of embedded multipliers and LUTs for adder tree**

```
Struct EMBEDDED_MUL
{
Int Al; //lower index of first input
Int Au; //upper index of first input
Int Bl; //lower index of second input
Int Bu; //upper index of second input
Int Pl; //lower index of Product
Int Pu; //upper index of Product
Bool used; //true if this multiplier is used
}
//Multiplier: Pbit=MbitxNbit
//Count number of MUL required:
EMBEDDED_MUL EmbMult[10]; //create an array of mults

For(m=0;m<ceil(M/18);m++) //partition M into ceil(M/18) mults
{
      For(n=0;n<ceil(N/18);n++) //same for N
      {
      i = n+ ceil(N/18)*m; //multiplier i in the array

      //Set index for input B
      EmbMult[i].Bl=n*18; //lower index
      if(n*18+17>=N) //max index is larger than input
                  EmbMult[i].Bu=N-1;//restrict to N
      Else
                  EmbMult[i].Bu=n*18+17;

      //Set index for input B
      EmbMult[i].Al=n*18; //lower index
      if(m*18+17>=M) //max index is larger than input
                  EmbMult[i].Au=M-1;//restrict to N
      Else
                  EmbMult[i].Au=m*18+17;

      //Set Index for output
      EmbMult[i].Pl=EmbMult[i].Al+ EmbMult[i].Bl;
      EmbMult[i].Pu=EmbMult[i].Au+ EmbMult[i].Bu+1;

      if(EmbMult[i].Pu<=P)
            EmbMult[i].used = true;
      Else EmbMult[i].used = false;
}
}

//count number of adders required
//iterate a round multiplier, start from second one
Last_MSB=EmbMult[0].Pu;
Adders=0; //number of 1-bit adder or LUT
for(j=1;j<i;j++)
{
if(EmbMult[j].used=false) continue;
if(Last_MSB<EmbMult[j].Pu) Last_MSB=EmbMult[j].Pu;
if(Last_MSB>P)
      {Adders+=P-EmbMult[j].Pl;Last_MSB=P;}
else
      { Adders+=Last_MSB-EmbMult[j].Pl;Last_MSB++;}
}
```