Evaluation Techniques for Mapping IPs on FPGAs

Avinash Lakshminarayana

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Sandeep K. Shukla, Chair
Dong S. Ha
Patrick Schaumont

August 11, 2010
Blacksburg, Virginia

Keywords: Design Space Exploration, High Level Synthesis, Prototyping, Power Estimation, Regression Analysis, Pareto Optimization

Copyright 2010, Avinash Lakshminarayana
All Rights Reserved
Evaluation Techniques for Mapping IPs on FPGAs

Avinash Lakshminarayana

(ABSTRACT)

The phenomenal density growth in semiconductors has resulted in the availability of billions of transistors on a single die. The time-to-design is shrinking continuously due to aggressive competition. Also, the integration of many discrete components on a single chip is growing at a rapid pace. Designing such heterogeneous systems in short duration is becoming difficult with existing technology. Field-Programmable Gate Arrays offer a good alternative in both productivity and heterogeneity issues. However, there are many obstacles that need to be addressed to make them a viable option. One such obstacle is the lack of early design space exploration tools and techniques for FPGA designs. This thesis develops techniques to evaluate systematically, the available design options before the actual system implementation.

The aspect which makes this problem interesting, yet complicated, is that a system-level optimization is not linearly summable. The discrete components of a system, benchmarked as best in all design parameters — speed, area and power, need not add up to the best possible system. This work addresses the problem in two ways. In the first approach, we demonstrate that by working at higher levels of abstraction, one can achieve orders of improvement in productivity. Designing a system directly from its behavioral description is an on-going effort in industry. Instead of focusing on design aspects, we use these methods to develop quick prototypes and estimate the design parameters. Design space exploration needs relative comparison among available choices and not accurate values of design parameters. It is shown that the proposed method can do an acceptable job in this regard. The second approach is about evolving statistical techniques for estimating the design parameters and then algorithmically searching the design space. Specifically, a high level power estimation model is developed for FPGA designs. While existing techniques develop power model for discrete components separately, this work evaluates the option of generic power model for multiple components.
Acknowledgments

I would like to thank my parents for their constant support and encouragement in all my academic activities.

Thanks to my advisor, Dr. Sandeep Shukla for the guidance, support and valuable inputs I received over past two years. Thanks to Dr. Dong Ha and Dr. Patrick Schaumont for serving in my thesis committee.

Special thanks to Dr. Sumit Ahuja, for his mentoring, interesting discussions and useful suggestions during all phases of my thesis. Thanks to my other lab mates — Bijoy Jose, Bin Xue and Mahesh Nanjundappa for their help in thesis work.

Credits are due to Nagendra Krishnamurthy, Athira Chandrasekharan and Nikhil Prakash Rahagude for assistance in preparation and review of thesis draft. Finally, I would like to thank Amol Deshpande and all other Hokie friends, for all the interesting experiences I had.
# Contents

1 Introduction 1

1.1 Trends in semiconductor industry .................................. 1

1.2 Advantages of FPGA-based system design .......................... 2

1.3 Design space exploration in FPGAs ................................. 3

1.4 Related work .......................................................... 5

1.4.1 System level simulation and prototyping ......................... 5

1.4.2 FPGA power estimation using linear regression .................. 6

1.4.3 Design space exploration techniques for reconfigurable systems . . . . 7

1.5 Contributions ......................................................... 8

2 Background 10

2.1 High level synthesis ................................................ 10

2.1.1 Design flow using C2R .......................................... 10

2.1.2 C2R directives .................................................... 11

2.1.3 An illustrative example .......................................... 12

2.2 Multi-variate least square regression model ...................... 13
3 Coprocessor Design Space Exploration Using High Level Synthesis  
3.1 Introduction ......................................................... 16
3.2 Methodology ......................................................... 18
3.3 Case studies and results .............................................. 21
  3.3.1 Fibonacci and Caesar ............................................. 21
  3.3.2 Bitcounter IP ..................................................... 24
  3.3.3 AES .............................................................. 26
3.4 Conclusions .......................................................... 27

4 Regression-based Dynamic Power Estimation for FPGAs  
4.1 Introduction .......................................................... 29
4.2 Background .......................................................... 32
  4.2.1 Multi-variate least square regression model ................. 32
  4.2.2 FPGA power models using linear regression ............... 33
4.3 Methodology .......................................................... 34
  4.3.1 Nature of the IPs used ........................................... 34
  4.3.2 Dynamic power evaluation for reference model ............ 36
  4.3.3 Toggle and resource utilization information ................. 36
  4.3.4 Statistical analysis using JMP ................................. 37
  4.3.5 Model evaluation phase ........................................ 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>Results</td>
<td>37</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Varying data patterns for fixed IP set</td>
<td>38</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Varying IPs for fixed data pattern</td>
<td>40</td>
</tr>
<tr>
<td>4.4.3</td>
<td>IP grouping to increase accuracy</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>Future Work and Conclusions</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Pareto optimization technique for reconfigurable systems</td>
<td>44</td>
</tr>
<tr>
<td>5.1.1</td>
<td>FPGA platforms</td>
<td>44</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Pareto optimization algorithm</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>Other possible extensions to the framework</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>Conclusions</td>
<td>49</td>
</tr>
<tr>
<td>A</td>
<td>Data for Regression Analysis</td>
<td>51</td>
</tr>
<tr>
<td>A.1</td>
<td>Varying toggles with IP set fixed</td>
<td>51</td>
</tr>
<tr>
<td>A.2</td>
<td>Introducing new IPs in evaluation set</td>
<td>53</td>
</tr>
<tr>
<td>A.3</td>
<td>Clustering Strategy</td>
<td>53</td>
</tr>
</tbody>
</table>
## List of Figures

1.1 Multi-Processor SoC system with Heterogeneous Cores ........................................... 2  
2.1 C2R-Based Design Flow .............................................................................................. 11  
3.1 Traditional Hardware-Software system co-design methodology ............................... 19  
3.2 HLS-based high level design flow .............................................................................. 20  
4.1 Regression Methodology for dynamic power estimation ........................................... 35  
4.2 Comparison of average percentage errors with varying data samples ...................... 38  
4.3 Comparison of maximum percentage errors with varying data samples ..................... 39  
4.4 Comparison of average percentage errors with varying IP samples ........................ 41  
4.5 Comparison of different models ............................................................................... 42  
4.6 Error increase with diverse IP sets ............................................................................ 43  
4.7 Comparison of different models ............................................................................... 43  
5.1 Partially Reconfigurable FPGA Platform ................................................................. 45
# List of Tables

3.1 Performance Matrices for Fibonacci and Caesar ........................................... 23
3.2 Performing platform dependent analysis for BitCounter IP ............................ 25
3.3 Hand code versus High level code for DMA implementation ......................... 25
3.4 Different implementations of AES IP ............................................................ 26

A.1 Effect of Increasing Toggle Samples on Hybrid Model ................................. 52
A.2 Effect of Increasing Toggle Samples on Toggle Model .................................. 52
A.3 Effect of Increasing Toggle Samples on Area Model ..................................... 53
A.4 Effect of Increasing Number of IPs on Hybrid Model ................................... 54
A.5 Effect of Increasing Number of IPs on Toggle Model .................................... 55
A.6 Effect of Increasing Number of IPs on Area Model ...................................... 56
A.7 Effect of Introducing Heterogeneity on Hybrid Model ................................. 56
A.8 Effect of Introducing Heterogeneity on Toggle Model ................................... 57
A.9 Effect of Introducing Heterogeneity on Area Model ...................................... 57
Chapter 1

Introduction

1.1 Trends in semiconductor industry

Sophistication of features, superior performance and sleeker form factor have been driving forces of consumer electronics market. At the same time, cost and energy efficiency have turned out to be the essential catalysts for expansion in this market. Phenomenal density growth, governed by the Moore’s law [1], has made it possible to have billions of transistors on a single die. Designing circuits with such a large number of transistors, through existing methodologies, is becoming increasingly difficult. Secondly, the time-to-design is shrinking continuously, due to aggressive competition. The lifetime of a product is turning out to be smaller than a typical cell-based RT - Level design time. There is a need to develop alternate quick turnaround methodologies. In this regard, the role of Field-Programmable Gate Arrays (FPGAs) is increasing.

Another trend in the chip industry involves incorporating many discrete components on a single chip, as shown in Fig. 1.1. A typical system-on-chip (SoC) today encompasses heterogeneous processors and a variety of hardware accelerators. High speed communication units, signal processing components, advanced security and encryption units all form part of
it. FPGAs are heterogeneous in nature having soft and hard processors, a large reconfigurable fabric to implement hardware circuits, certain special purpose units and hard macros. Thus, they offer a good platform for implementing SoCs.

![Figure 1.1: Multi-Processor SoC system with Heterogeneous Cores](image)

### 1.2 Advantages of FPGA-based system design

Although FPGAs have been around for long time, they were mainly used for prototyping. The smaller size and performance issues prevented them from being used in actual products. However, there is enough reconfigurable fabric available in FPGA today to map reasonably large designs. Also, the performance gap is reducing. FPGAs enhance design productivity considerably. This is due to the elimination of design phases at gate-level and beyond. Actual systems can be created and tested, bugs fixed and the new design again implemented
without much cost. This reduces the dependence on simulation and hence the overall verifica-
tion time and effort. Also, high level designing using model-based approaches and high
level languages are showing a good deal of success in both industry and academia [2, 3].
Many communication- and signal processing- related IP cores have been successfully im-
plemented on FPGA using MATLAB [4], LABVIEW [5] and other industry tools. As the
diversity of SoC components grows, more and more engineers from different backgrounds
will be involved. Abstracting out the lower level hardware details and providing a means for
algorithm implementation to application designers would certainly increase the productivity.

1.3 Design space exploration in FPGAs

Design space exploration is an important aspect of SoC design that needs to be looked into.
Broadly, this involves choosing right combination of hardware-software components, selecting
right accelerators, deciding microarchitecture of the IP cores and interfaces. Usually, the goal
is to make delay-energy tradeoffs. This work addresses the design space exploration issues
for FPGAs. So far, most of the exploration and estimation techniques were focused on
cell-based designs and hence there is a need to look into FPGA-specific techniques. Also,
IPs designed on generic FPGA fabric will obviously be behind the cell-based designs, in
terms of performance. So, good microarchitectural decisions can help compensate for this,
up to a certain extent. Finally, there are significant efforts being made in IP reuse and
standardization. As long as the platform is same, IP designed once can be reused in different
systems without many changes. There is also a good deal of effort going on for standardizing
bus protocols, memory models, etc., from different vendors.

The aspect which makes the design space exploration challenge both interesting and com-
plicated is that a system-level optimization is not linearly summable. The constituent IPs
of a system, benchmarked to be best in all 3 parameters of design — speed, area and power
—, need not essentially add up to the best possible system. This thesis, first presents a
quick prototyping framework to perform design space exploration in FPGA. The framework aids in doing hardware-software partitioning, performing platform-specific optimizations and making microarchitectural design decisions. This will help in trying out many permutations of choices quickly. The throughput and other analysis will be more realistic and faster as compared to software-based simulation. Also, it would be time consuming to make simulation models for buses, interfaces, etc. We use High-Level Synthesis (HLS) to aid in such a prototyping. This involves synthesizing hardware from C language-based behavioral descriptions. One can quickly separate components and transform specific parts into hardware [6]. The RTL design phase is completely removed. HLS tools cannot still generate codes comparable to hand-coded RTL in performance. However, design space exploration needs relative comparison among available choices and does not need accurate values of design parameters themselves. The investigation shows that such tools can do a decent job in this regard. Also, the learning curve is small and one can selectively refine the code at any level of granularity.

Although FPGAs are being increasingly integrated in many systems, the power estimation and reduction are not well studied as compared to ASIC and processor-based designs. Therefore, a system level power estimation methodology for FPGAs is provided. This model involves providing generic power model that predicts dynamic power consumption for IPs. FPGAs are known to be power hungry and it is important that power budgeting be done effectively for them. Traditional simulation-based methods are extremely time-consuming and hence the proposed methodology is expected to enhance productivity considerably. A linear regression-based prediction model for dynamic power consumption is proposed. Dependence of power consumption on easily measurable quantities like input-output data pattern and the resource utilization of the design is explored. The results show feasibility of such power models for FPGAs, establish distinction from ASIC counterparts and provide good improvements over existing methodologies.

Finally, an algorithmic approach to the design space exploration problem is provided. With the number of IPs per system continuously increasing, it is believed that ad hoc approaches to the exploration will not be cost-effective. Apart from the software and hardware plat-
forms addressed earlier, FPGAs also come with a partially reconfigurable hardware. Good heuristics are needed to map application IPs across different components of an FPGA. Such an approach involves using the data from the early estimation frameworks, such as the one demonstrated in this work, and searching the design space with multi-objective criteria. A partial pareto algebra-based heuristic is proposed for the same.

1.4 Related work

1.4.1 System level simulation and prototyping

System prototyping has evolved in 3 different phases. The earliest methods involved developing coprocessors using RTL for each possible configuration and then making a suitable choice. Behavioral simulation of coprocessors and the software was done separately. Many industrial tools are available to support such RTL simulations including Modelsim [7], VCS [8], etc. Similarly, for the software, the simulators are available for many general purpose processors. But, evaluation of hardware/software interfaces cannot be done using such methods. Furthermore, the design cycle is too large and making small changes in the configuration will involve high design effort.

Later, a wide range of languages and corresponding cosimulation environments, such as Bluespec [9], HandelC [10], Gezel [11], etc. were developed to consider various aspects of hardware/software co-design. Bluespec comes with an exhaustive tool set encompassing a compiler, simulator and its own IP library. The tool set takes in specification in the form of a specific language called Bluespec System Verilog and generates synthesizable RTL. In another closely related work, Sullivan et al. [12] used Handel-C for algorithm description. This methodology claims to provide considerable advantage for creating communication ports, establish interfaces, etc. with ease. Gezel is a cycle-based hardware description language-based on finite state machine and data path. It provides a cosimulation environment and
also generates synthesizable VHDL. With this, one can perform power-performance analysis and make various tradeoffs [13]. All these methods reduce the verification time and many issues concerning hardware/software interfaces. However, they suffer from a major drawback — the need for using a new language and/or the design environment. This would mean longer design time and investment. It will be demonstrated in our work that learning curve is small for high level synthesis methods since all the exploration activities will be carried out at behavioral level in software environments.

1.4.2 FPGA power estimation using linear regression

Regression-based RTL power models have been studied well over the past decade. One of the early works involved developing power macro models for primitive components at RTL [14]. This work discusses in detail about the power macro-models and the overheads involved, sampling and regression-based power models. [15] includes an exhaustive discussion on the different kinds of RTL power models. In this work, the power estimation is done in multiple phases, involving an offline characterization-based on I/O toggle statistics and online tuning to improve accuracy. These works focus on generic ASIC power models at RT-level taking gate-level simulation as the golden model while this methodology works at a higher level of abstraction taking RTL simulation as golden model. There were also some works which used these techniques for FPGAs [16]. Here, the authors propose an adaptive regression method to model FPGA power consumption. However, all of these works develop power models for each RTL components or IPs separately while we work towards a generic power model involving multiple IPs.

[17] and [18] use an orthogonal approach and characterize power-based on the utilization of various resources of the Xilinx board. [17] estimates effective capacitance of each of the major resources through transistor level modeling and estimates the switching activity using a large set of benchmarks. They combine all the data to evaluate percentage breakdown of dynamic power consumption of different components of Virtex-II FPGA chip. [18] performs
similar activity for Spartan-3 Xilinx board. Firstly, such a characterization for individual chips considering design information as secondary would not be very useful for cost metrics evaluation. Also, compared to these approaches, our methodology has the advantage of giving system level power estimate for IP which will be more meaningful for design space exploration. Instead, [19] developed regression modeling at the operand level (like adders, multipliers etc). Apart from the problems discussed about earlier methods, this kind of an approach can have accuracy problems at a complete design level, as one needs to make a detailed analysis on the effect of combining these operands in different ways, on power consumption.

[20, 21, 22, 23, 24, 25] present several other ways of performing high-level power estimation. [26, 27, 28] discuss many techniques for power-aware high level synthesis. [29] presents a ‘Grey-box approach’ to high level power estimation in ASIC flows. They consider the toggle counts in individual states of a design at FSMD level, instead of counting I/O toggles. This would aid in better accuracy. [30] proposed a single power model for different FPGA power components using non-linear regression and to that extent is similar to our work. However, it has been found in our experiments that having single equation for any design on a particular board, will result in much higher errors. More importantly, the model is static and does not take into account the I/O toggles. This means the model gives the same power estimate for an IP irrespective of the data pattern. One can simply use tools such as Xilinx power estimator [31] to obtain approximate power quickly.

1.4.3 Design space exploration techniques for reconfigurable systems

The traditional exploration techniques were mostly focused on hardware-software partitioning aspects only. Different kinds of implementations within hardware were not studied. Partial runtime reconfiguration is a recent phenomenon. There are few works in academia related to partially reconfigurable systems. [32] provides a detailed list of design space ex-
ploration techniques. [33] gives a survey on the partitioning techniques specific to FPGA. They also provide an integer-linear programming technique for doing hardware-software partitioning for tasks represented as directed acyclic graphs. They also propose Kerninghan-Lin/Fiduccia-Matheysses-based heuristic considering partially reconfigurable fabric’s placement issues. However, this work only considers two implementation platforms — the software and the partially reconfigurable hardware. No special consideration is provided for static hardware. [34] formulates the same problem addressed in this thesis. They consider all three platforms of implementation — software, static and partially reconfigurable hardware. They propose a 0-1 integer linear programming algorithm to solve the problem. In contrast, this thesis proposes a pareto algebra-based heuristic to avoid exhaustive search of the design space. More importantly, most works approach this as a placement problem with the objective of minimizing delays. The work discussed in this thesis considers delay and energy as multi-objective to be reduced and placement requirement as a constraint.

[35, 36, 24, 27, 37, 38] present several approaches to automatically generate power aware RTL from high-level description of the design. Also, [39] approach the optimization and verification issues in Hardware synthesis through concurrent specifications.

[40] has developed a partial pareto algebra for composing pareto optimal configurations dynamically, from pareto optimal configurations of components. In other words, the system level pareto points can be evolved using pareto configurations of individual tasks, without having to exhaustively search the full design space. The paper discusses how this algebra can be used for SoC exploration. The work presented in this thesis uses the pareto algebra to perform design space exploration for partially reconfigurable FPGA systems.

1.5 Contributions

This work addresses design space exploration techniques specific to FPGAs, using FPGAs as targets for SoC design implementation rather than as just prototypes. Some techniques
applicable specifically to FPGAs are developed and certain techniques in the ASIC domain are modified and extended for FPGAs.

1. Chapter [3] discusses a quick prototyping technique for FPGA platforms using high level synthesis. It is demonstrated how various design decisions can be made through a series of benchmarks. The specific contributions of this work include:

   (a) Exploring possibility of using HLS for co-design
   (b) Platform dependent optimization at high level
   (c) Analyzing system level performance parameters in coprocessor selection

2. Chapter [4] presents a linear regression model for FPGA dynamic power estimation. Regression techniques used in ASIC domain is modified for FPGA. Also, an extension of old approaches is made to provide generic power estimation model. To the best of our knowledge, this is the first time such a generic model is being proposed at IP level. The specific contributions of this work include:

   (a) Regression modeling for FPGAs considering I/O toggle and resource utilization
   (b) Unified power model for multiple IPs
   (c) Comprehensive system level power estimation approach for IPs

2.1 High level synthesis

We have used C2R HLS tool for demonstrating our methodology [41]. In this section, we explain the C2R-based high level synthesis flow. We first present the overall view of hardware generation in C2R method and then explain briefly the process of restructuring with C2R directives. This section is intended as a snapshot of C2R synthesis.

2.1.1 Design flow using C2R

Presented in Fig. 2.1 is the high level view of C2R-based hardware synthesis flow. The flow starts with a behavioral C language-based description of the design. The concurrent behaviors are generally expressed through threading in such languages. Of course, there are many limitations on the style as well as flexibility and only a subset of C language can be used in specification for synthesis. Optionally, C code is subjected to a series of transformations to inculcate hardware implementation aspects into the flow. These transformations are together referred to as C2R restructuring. Default implementations will be considered in the absence of restructuring information. This code forms the input to the C2R compiler. The compiler
performs analysis on the restructured code through control flow graphs, after a few tasks such as checking for syntax etc. The hardware will be generated in the form of synthesizable RTL, preserving the functionality of the original code as well as all the hardware semantics explicitly specified in restructuring. This RTL code can be subjected to further reformation to either FPGA or ASIC implementations using standard industry flow.

![Diagram of C2R-Based Design Flow](image)

**Figure 2.1: C2R-Based Design Flow**

### 2.1.2 C2R directives

The restructuring process is carried out using certain keywords at appropriate places in the C code. These keywords are called C2R directives. The C2R framework provides support for compiling of such restructured codes in standard GCC environment using appropriate header files. This compilation time is drastically lesser than the RTL-based compilation time. The C2R directives can be broadly classified as function-, variable- and statement-based
directives. Also, there are many directives which do not belong to any of these categories.

The default size for all the integer data declared is 32 bits. This can be changed in multiple ways. C2R flow provides the means to specify variables at bit level instead of generic int, char etc of C language. There are also special keywords for double, float, narrow and other kinds of generic data types. A process forms the fundamental building block for specifying parallelism in restructured code. Each thread will be called as a process and this corresponds to individual ‘module’ in C2R generated Verilog. Needless to say, each of these blocks operate concurrently and independently of each other. There are special directives to establish communication between different processes and also to establish links between a single serving process and multiple client processes. The latter is done using C2R-shared interfaces. Also, special means is provided to interact with black-boxes and third part IPs through foreign interface functions.

The clocking information can be provided through special directives. By default the tool tries to exploit parallelism as much as possible. The tool can be directed to be conservative or to perform no such parallelizations at all. Explicit clock boundaries can be introduced through wait statements. There are also directives to perform standard parallelizations such as unrolling, pipelining, looping, spawning and forking. Finally, there are keywords to specify storage elements such as flip-flops etc.

2.1.3 An illustrative example

The code snippet shown in Listing 2.1 illustrates some of these directives. The pseudo-code starts with a header file that facilitates GCC compilation. Each of the functions or threads will be identified as individual processes. Also, uint8_t etc specify the data widths. C2R_unroll, as the name specifies, unrolls the loop and establishes parallelism at the cost of increased area. Forking construct demonstrates how intra-process parallelism can be exploited to further enhance performance. The wait directive is optional here as the tool would automatically infer the clock boundaries.
2.2 Multi-variate least square regression model

Linear regression tries to relate a variable of interest, called response variable, to a set of measurable, independent variables, through a linear equation. The actual values of the
response variable is calculated for different sets of independent variables. Curve-fitting is done on these results to arrive at the linear equation. This process is generally called the training phase and the sets involved are called training sets.

Consider a sample of \( m \) observations done on \( n \) variables \((X_1, X_2, \ldots, X_n)\). If these \( n \) variables are assumed to satisfy a linear relation with response variable \( Y \) then it can be represented as:

\[
Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n \tag{2.1}
\]

For the regression model shown in Equation 2.1, there may be various ways to calculate the value of regression coefficients \( \beta \), such that the error between the predicted and measured values of response variables is minimized. Let us denote the \( i^{th} \) observation on these variables as \((X_{i1}, X_{i2}, \ldots, X_{in})\), \( i = 1,2,\ldots,m \). Lets \( Y_i \) be the observed value of the response variable \( Y \) corresponding to the \( i^{th} \) observation. Least squares error method can be used to minimize the difference between predicted and measured values, and the objective function can be represented as:

\[
\left( \sum_{i=1}^{m} (Y - \beta_0 - \beta_1 X_{i1} - \beta_2 X_{i2} - \ldots - \beta_n X_{in})^2 \right) \tag{2.2}
\]

and we need to minimize this objective function by choosing appropriate coefficients.

### 2.3 Partial pareto optimization

Pareto algebra involves finding tradeoff points in multi-objective optimization problems. A set of configurations are pareto optimal if it is impossible to find a single configuration, either within the set or outside, which improves one or more of the objectives without at least one of them. Generally, all feasible configuration points are found out in a design space and the pareto points are evaluated. A recent work on pareto optimization developed an ‘
of Pareto points’, where, given a set of pareto points of components, the pareto points of combination of these components could be found without exploring the design space of all points of the combination [40]. They also discuss how such an algebra can be used in SoC design space exploration incrementally. In this work, an algorithm is proposed specific to reconfigurable FPGA fabrics. The high level estimation information obtained from the techniques demonstrated in this thesis can be used as inputs to the proposed algorithm. Presented below are a set of definitions from [40] to be used in for algorithm discussion.

1. **Pareto optimality**: A solution is optimal if it is impossible to find a solution which improves on one or more of the objectives without worsening at least one of them.

2. **Quantity**: A quantity is a set \( Q \) with a partial order \( \leq_Q \). In our case, \( \leq_Q \) is always a total order as we deal with real values. The two quantities we consider are delay and energy.

3. **Configuration space**: A configuration space \( S \) is the Cartesian product \( Q_1 \times Q_2 \times \ldots \times Q_n \) of a finite number of quantities. A partial order is inherited in the configuration space. The configuration space considered in this work is two-dimensional.

4. **Configuration**: A configuration is an element of a configuration space.

5. **Dominance**: A configuration \( c_1 \) dominates another configuration \( c_2 \) in the given configuration space \( S \) if all the quantities of \( c_1 \) are as good as \( c_2 \).

6. **Pareto Minimal**: A set \( C \) of configurations is said to be Pareto minimal iff for any \( c_1, c_2 \) in \( C \), \( c_1 \) does not strictly dominate \( c_2 \).

7. **Minimization**: Let \( C \) be a set of configurations of a configuration space \( S \), and \( C \) is well ordered, then \( min(C) \) denotes the unique, Pareto equivalent and Pareto minimal set of configurations.

8. **Constraint**: Let \( C \) and \( D \) both be sets of configurations of configuration space \( S \). Then \( C \cap D \) is a set of configurations of \( S \) called \( C \) constrained to \( D \).
Chapter 3

Coprocessor Design Space Exploration Using High Level Synthesis

3.1 Introduction

The traditional computing arena of hardware/software co-design has gained importance in the context of system-on-chip design methodologies. Apart from the classic issues of partitioning, communication and granularity, the need for quick estimation of metrics such as area, power, and latency has turned out to be important. This can be attributed to the large design space under consideration. The number of IPs integrated on a system is large and each IP may have multiple configurations. Many of these IPs are often hardware coprocessors that accelerate compute-intensive tasks. A collection of coprocessors chosen based on Pareto optimal points w.r.t. speed, area and power may not necessarily add up to corresponding system level Pareto optimal points [40]. One way to address this issue could be through developing algorithms or heuristics that lead towards Pareto optimal configuration. Obtaining a generic strategy, keeping in mind global optimization, is hard. Furthermore, evolving mathematical models for such diverse systems is very difficult. Another approach involves using partial or full simulation of systems for different configurations. Various stan-
Standard co-design platforms are available for the same [11, 12]. However, exhaustive design space exploration will be difficult.

Suppose it is required to configure a system involving a 1-) data filter that processes the input data, 2-) a computing core that performs a series of data-intensive transformations, and 3-) an encryptor that encrypts the data before transmission. As already stated, picking the fastest IPs will not necessarily lead to the best solution. One has a plethora of choices when it comes to the filter. Multiple criteria will need to be considered to maximize throughput at the least cost. The computing engine for data transformations has to be as fast as possible. However, one might have multiple choices for interfacing this coprocessor with rest of the system. Also, communication bottlenecks might mask the speed gains obtained from this coprocessor. Such a process will not only cause wastage of resources (in terms of power, area etc.), but also unnecessarily increase the design time. When it comes to the encryptor, apart from multiple encryption algorithms to choose from, one can perform microarchitectural exploration, for example, between a pipelined version and a simple implementation.

In this paper, we show how High Level Synthesis can be used for addressing some of the problems discussed so far. HLS involves synthesizing hardware from high level language descriptions. Firstly, when using off-the-shelf IPs, instead of doing RTL design exploration, one can use behavioral descriptions and perform quick analysis in HLS. Secondly, hardware prototypes can be created quickly for software development. This aids concurrent hardware and software development. Finally, if one goes for C-based synthesis itself, modular designing will be easier at behavioral level than at the RT - level, which involves a lot of microarchitectural detailing. Individual blocks within a design can be subjected to different levels of refinement without affecting other blocks.

We demonstrate a methodology that enables quick design space exploration through prototyping for FPGA-based platforms. We use C2R to synthesize hardware description from C-based designs. Such a methodology provides a fast and easy way to explore the design space. We present a series of case studies of varying complexity in which we start from a
high level description in C language, and then follow a series of C2R specific restructuring, implement the synthesized RTL on the FPGA platform and obtain the performance numbers. A variety of implementations and architectures are tried out to arrive at different tradeoff points. In doing so, bulk of the changes are performed within the C environment and thus design cycle time will be reduced. We also perform power estimation for the generated designs.

The main contributions of this work include

1. Exploring possibility of using HLS for co-design
2. Platform dependent optimization at high level
3. Analyzing system level performance parameters in coprocessor selection

3.2 Methodology

The traditional co-design methodology for FPGA is shown in Fig. 3.1. It starts from system specification which encompasses all the functional requirements and design constraints. The behavioral C model is the most favored choice. This is followed by a profiling of the system and then doing the hardware/software partitioning to meet all the design constraints at minimum cost. The hardware specification is then implemented through RTL design flow using HDLs and software specification is realized in a C/C++ environment. The hardware design undergoes various levels of refinement to meet the essential area and power constraints, and settle all the timing issues while the software flow involves functional verification and throughput considerations. Various cosimulation methodologies are used for functional verification and simulations. And then the implementation is transformed to FPGA and other hardware targets. The system obtained is checked for all system level constraints and specifications. Such a cycle will have to be repeated multiple times before the final design freeze. Amongst the many drawbacks of such a methodology, the cosimulation takes the prominent
role. Compared to hardware emulation-based approach, cosimulation is inherently slow, many details are abstracted and hence not very accurate.

Figure 3.1: Traditional Hardware-Software system co-design methodology

In contrary to the approach discussed above, the HLS-based approach we propose is shown in Fig. 3.2, where integrated development of hardware and software is possible. Our framework takes threaded C code as input and provides a plethora of opportunities to make dynamic hardware-software partitioning, moving components between hardware and software quickly and finally allowing complete functional verification at each stage. A set of compute-intensive threads in the C specification are identified and synthesized into hardware. The system
performance parameters are analyzed and bottlenecks are identified. If the timing constraints are not met, more threads will have to move to hardware. If the area constraints are not met, some of the components may have to be moved back into the software. If both the constraints are not met, multiple threads may have to be moved between hardware and software. Since the synthesis of hardware happens very quickly and the tool automatically generates hardware interfaces, many combinations can be tried out within a short time. Also, mapping algorithms can be applied to shortlist the possible combinations.

The inner loop A in Fig. 3.2, will be frequent and quicker. While the outer loop B will be longer as in the traditional case, however it will be infrequent. The loop A involves obtaining area and speed estimates. The area estimates determine the cost of moving a specific thread to hardware. One can decide what specific components need to be moved to hardware without affecting performance significantly. The speed estimates help identify key communication bottlenecks. However, certain estimates like power analysis would still need
outer loop involving FPGA implementation. Efforts are being made to make the outer loop as infrequent as possible. For example, [29] performs power estimation at high level.

3.3 Case studies and results

We demonstrate the implementations on Xilinx Spartan-3e board [42]. The platform meets all necessary requirements for hardware-software design space exploration through its microblaze softcore for sequential executions and the reconfigurable logic for hardware implementations. Microblaze is a RISC-based softcore provided by Xilinx [43]. C2R tool was used as high level synthesis tool for all the case studies. We dynamically determined the threads that need concurrent execution and turned them to hardware. The remaining threads were run on microblaze.

The area, in terms of slice count and the speedup are the key parameters taken into consideration. We estimated the power for IPs in the context of system design using Xpower tool available with Xilinx ISE environment. Since the tool performs power analysis of post-implementation, place and route designs [44], the accuracy will be higher. We obtained VCDs/SAIFs for different test inputs, individually for the software and hardware modes and obtained the overall system power. Also, along with this data, we have presented the power numbers exclusively for the IP cores generated by the tool. Power estimation was a time consuming process as in other co-design flows. However, there is a proven method to perform power estimation for large IPs at high level with statistical regression-based approaches [29].

3.3.1 Fibonacci and Caesar

We start off with an illustrative Fibonacci IP that gives out the Nth Fibonacci number that is input to the hardware. The high level specification in restructured C version consists of a driver that gives the data input to the Fibonacci thread. This thread computes the Nth
Fibonacci number and returns the output back to driver thread. We retain the driver thread in C to be run on microblaze and realize the Fibonacci function on hardware as an IP core. This IP core is attached to the softcore through PLB bus. There are four ports for the IP - the call input, that activates the hardware, the data input port for the hardware, the data output port for result and finally the acknowledge port to indicate completion of the operation and thus closing the handshake. These 4 ports form a boundary between hardware and software and we create a memory mapped register corresponding to each of them. In software environment, we define these address locations as volatile pointers that can be directly accessed for read and write. This provides ample scope for performing behavioral verification of IP in C environment.

Another IP that we worked on was Caesar cipher that involves streaming data flow framework. In this IP the data to be encrypted is continuously sent to the computation unit. Depending on control signal, the computation unit either encrypts or decrypts data and sends back the data to the processor. There were 3 threads in the design: the cipher thread which did the actual computation, the data input thread for giving data and data output thread for collecting the output data. While the computation thread was realized in hardware, the I/O threads were retained on processor.

The results are tabulated in Table 3.1. The speedup in column 2 of Table 3.1 is ratio of number of execution cycles for coprocessor-based implementation to the execution cycles on software implementation. The area in column 3 is given in terms of slice count. The speedup number obtained for the designs is quite small. Although the raw processing power of hardware is very high, Amdahl’s rule takes over most of the gain in terms of overall design. The speedup obtained in Caesar cipher is slightly higher. This is because we are using a stream of data here rather than a single byte as in previous case. The speed gains obtained from hardware implementation can be justified only when the amount of data processing exceeds the communication overhead of hardware/software co-design. The HLS-based design flow can help in making fine grained decisions in such scenarios efficiently as the design time is less. Various combinations of hardware/software components can be tried.
for a given data pattern. With regard to the area, microblaze itself occupies about half the total number of available slices, which means that the area of coprocessor block is about a tenth of the total system area tabulated here. Nevertheless, overall slice count had to be included as we are considering system parameters.

| Table 3.1: Performance Matrices for Fibonacci and Caesar |
|---------|---------|---------|---------|---------|
| IP      | Spd     | Area    | Sys Pwr SW(mW) | Sys Pwr HW(mW) | IP Core Pwr(mW) |
| Fibonacci | 13      | 2652    | 686.7           | 685.2           | 0.13            |
| Caesar   | 27      | 2675    | 701.3           | 697.4           | 1.51            |

As in the case of performance figures, the IP core power alone cannot be the point of focus. Column 4 in Table 3.1 refers to the total system power in software while column 5 refers to system power for coprocessor-based implementation. Column 6 gives the power of individual IP core. It’s only a fraction of overall system power. We had expected the software power consumption to be lesser than that for hardware implementation. On the contrary, the overall system power remained more or less same. So, although the IP core is turned off, the savings one gets is only limited. In smaller designs, the power numbers for software are in fact slightly higher. But for a large design like AES, it behaves on the expected lines. Though, the power numbers for the software and hardware versions are similar, the hardware implementations turn out to be more efficient. The hardware will be more energy efficient owing to the speed gains of the design. We will have about 13 times energy saving for fibonacci and 27 times for caesar cipher. Thus, depending on the power and performance values, one can make fine grained decisions.
3.3.2 Bitcounter IP

In this case study, we consider an IP that counts number of 1’s in a stream of 16K array. Such an IP finds use in areas like parity checking and cryptography. The data is sent 32 bits at a time to a function which counts the number of 1’s. The result obtained is accumulated for all the 16K array of data. We came up with a threaded version of the C reference implementation. We form 2 threads - a driver thread that initializes a 16K sequence of random numbers and calls the countbits thread which does the computation. At this stage, one can verify the functional correctness of the threaded version in C environment. After this, we do restructuring of C code. The countbits thread will be hardware implementation while the driver will be in software. The first implementation as shown in Table 3.2 is a basic implementation like the previous ones and uses PLB bus and memory mapped registers. Later, we added a FIFO to the Bus-IP interface. This was expected to increase the speed as hardware need not be tied to the slower processor. However, the speed improvement was small as only data output from IP will be enhanced by FIFO. Data coming into the IP will still be delayed by processor. So, we used DMA to eliminate data path from processor. Now, hardware can get data at high speed, meanwhile, the processor can do other jobs. Finally, the bus architecture itself was removed and point-to-point Fast Simplex Link (FSL) was used to assess the speed-area obtained.

The countbits implementation had a higher data size of 16Kbytes. The software version was slow and also the communication overhead had been greatly overcome because of the data size. This explains the big speedup in column 2 of Table 3.2. A FIFO-based implementation, as expected, increased the speed by a certain extent. But this was at the cost of extra slice count as seen in the area numbers. The FSL implementation, gives a relatively smaller speedup. One can consider using such an option for hard-deadline IPs that need dedicated communication channels. FSLs are highly optimized paths for Xilinx environment and hence occupy small area.

The power consumption for overall system for coprocessor implementation is given in column
Table 3.2: Performing platform dependent analysis for BitCounter IP

<table>
<thead>
<tr>
<th>IP</th>
<th>SPEEDUP</th>
<th>AREA</th>
<th>Power(mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Version</td>
<td>243</td>
<td>2870</td>
<td>680.29</td>
</tr>
<tr>
<td>FIFO</td>
<td>276</td>
<td>3212</td>
<td>684.57</td>
</tr>
<tr>
<td>DMA</td>
<td>13402</td>
<td>3072</td>
<td>717.01</td>
</tr>
<tr>
<td>FSL</td>
<td>152</td>
<td>2653</td>
<td>682.39</td>
</tr>
</tbody>
</table>

4 of Table 3.2. The software baseline implementation consumed 694 mW. While the FSL and FIFOs marginally affect the total system power, DMA has a much bigger influence. FSL kind of a dedicated hardware resource gives significant performance gains at a very small power cost and hence can prove effective in many situations.

Presented in Table 3.3 is a comparison between implementation of IP core using Verilog\(^1\) and C2R for DMA implementation. Although the hand code version is faster than C2R code, the overall system speed is same for both the cases. They are limited by the speed at which DMA can transmit the data. This makes a strong case for high level approach. There could be design scenarios like this, wherein we don’t need high speedup. In this case, it’s good enough if the IP can finish its computation before DMA gives new data. Such decisions are difficult to be made at RTL. Despite a slightly higher area of the C2R generated core, we see that the power numbers of the hand coded IP and C2R generated IP are relatively same.

Table 3.3: Hand code versus High level code for DMA implementation

<table>
<thead>
<tr>
<th>IP</th>
<th>SPEEDUP</th>
<th>AREA</th>
<th>Power(mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand Code</td>
<td>13402</td>
<td>74</td>
<td>0.70186</td>
</tr>
<tr>
<td>C2R Code</td>
<td>13402</td>
<td>91</td>
<td>0.71701</td>
</tr>
</tbody>
</table>

\(^1\)The Verilog code was implemented by Mike Henry, ECE, Virginia Tech.
3.3.3 AES

We consider the implementation of advanced encryption standard (AES). AES is a block cipher-based encryption standard [45]. The core consists of mainly 4 functions: adding roundkey, substitute bytes, shift rows and mixcolumns. There are 11 rounds in all. In its simplest form, one could make a single thread that implements all 3 functions of all the 11 rounds into a single big thread. Although very inefficient, such a design can be accomplished in minutes. This can help in performing quick estimations. However, the area of the hardware obtained turned out to be too high for board implementation. One of the major bottlenecks for us was the large memory that was required to store round keys in hardware. Going back to the restructuring phase, we made a few modifications in the C code. This involved adding directives that explicitly indicated the compiler to port the RoundKey data into BRAMs. The next stage for this core was to introduce pipelining. The 10 rounds were divided into individual stages by unrolling the loop in the C code and introducing pipelining signals.

Presented in Table 3.4 are simulation results from ModelSim for standalone IP. By start time, we mean the time first AES encrypted data starts appearing on the output data bus. One could see a slight delay in start time of pipelined version, due to addition of handshake signals in each state and corresponding delays introduced. The BRAM version of the core was generated and because of the area issues, only this was implemented on the board. The core gave a modest speedup of about 3 times. While the overall system power for software version was 1271.24 milliwatts, the coprocessor version had 1345 milliwatts. The AES core contributed about 122 milliwatts to the system power.

<table>
<thead>
<tr>
<th>IP</th>
<th>Start Time(nS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Version</td>
<td>27410</td>
</tr>
<tr>
<td>Pipelined Version</td>
<td>36070</td>
</tr>
</tbody>
</table>
The need for looking at overall system performance metrics instead of individual IP blocks has been demonstrated in [13]. They show how an individual core that gives a speedup of orders of magnitude would only increase the performance 5 times when integrated in a full system. We highlighted similar findings in the high level approach. Incidentally, one of the cores used in demonstration of [13] happens to be AES core, although the two are different implementations and one-to-one comparison cannot be made.

Thus, we were able to obtain relative performance metrics similar to different methodologies, remaining at high level for many of the time consuming activities. Also, many high level synthesis tools have been proven to generate powerful hardwares that are comparable to hand generated cores. But this comes at a cost of increased design time. In fact, [46] demonstrates how one can make suitable changes in C code to obtain substantial speed and area savings using C2R on the AES core.

3.4 Conclusions

In this chapter, we presented a high level synthesis-based approach for solving performance constrained Hardware/software co-design problems. We identified key bottlenecks in a traditional approach and split the design space exploration loop into quick frequent and slow infrequent parts. The abstraction level of quick frequent loop was raised to behavioral level. The exploration at this level was done in C environment. We were able to perform coprocessor design space exploration at multiple levels - in terms of making coprocessor selection, making platform dependent tradeoffs for FSL, DMA etc. considering Xilinx Spartan-3e as target and finally making coprocessor microarchitecture variation in the AES example. Considering the software-only version as the baseline for each of the IP, we provided relative speedups for all of them. The area of synthesized hardware was provided in terms of slice count. The power estimation was carried out for entire system to contrast the hardware and software versions. Results show that HLS-based methods are suitable for design space
exploration at many different levels. Also, there are many potential benefits if high level synthesis is used for system design, completely eliminating RTL designing and debugging.
Chapter 4

Regression-based Dynamic Power Estimation for FPGAs

4.1 Introduction

Design space exploration for SoC is a 2-stage problem. First, it involves finding quick and accurate estimation methodologies to obtain the design space parameters. Next comes developing algorithms to exhaustively and efficiently search the multi-objective space. The increase in number and complexity of the IPs integrated per system has exerted tremendous pressure on both estimation as well as algorithmic aspects. There is a need for good estimation techniques in the context of heterogeneous system design. In particular, doing detailed and accurate power estimation continues to be time consuming. Recently, a few interesting techniques have been developed for doing early estimations with reasonable accuracy. Statistical learning techniques are one of them. But, they are limited both in scope and versatility.

Dynamic power estimation has been studied well in the context of individual IPs in ASIC flow. Relatively, FPGA dynamic power estimation has not received much attention [16,
We focus in this chapter on linear regression-based power estimation technique for FPGAs. Current techniques develop power models for individual IPs with input and output toggles as the parameters for regression. The regression coefficients are arrived at, with extensive training of the model on sample test patterns (refer to section II for further details). The power model is developed for each IP separately in terms of a linear equation relating dynamic power to the I/O toggles. But each IP will have a separate power model. New model will have to be developed even for small changes in design. In this work, we evaluate the possibility of a unified power model for all the IPs in a system instead of developing power models for each IP separately. The random logic in ASIC IPs make the effect of I/O toggle very pronounced. On the contrary, FPGAs are structured. All designs will be mapped to LUT-based fabric or fixed set of FPGA resources like multipliers and Block RAMS. We also show in our results that the effect of I/O toggles for a particular design is small and that there is a significant dependence of dynamic power on resource utilization.

We take into account all these aspects in our analysis to develop FPGA power models. To the best of our knowledge, this is first work which addresses generic power models for IPs, taking I/O pattern variation and resource utilization into consideration.

Currently, RTL tools like XPower from Xilinx are used. XPower is used for post implementation place and route designs and it takes simulation data in either SAIF or VCD formats. Similar tools are available from other vendors. Such tools have a good degree of accuracy as they will be fully aware of internal structure of the resources as well as the design details like toggle rate etc from simulation information. However, the overall flow is very time consuming. VCD file generation forms the key bottleneck. Obtaining simulation data for a set of test vectors can easily take up multiple days even for small-to-mid sized designs. Also, one needs to have synthesized RTL that is ready for bitstream generation in such flows. This may be too late for big designs. These form the primary motivation for developing high level power estimation methodologies for FPGAs.

There are certain interesting commonalities in IP usage across industry for SoC design. IP reuse is extensive, of course with small changes for each system. The generic IPs, either
developed in-house or obtained off-the-shelf, would be generally subjected to extensive profiling and hence a large bank of test vectors and the simulation data in different formats will be available. The changes for each system may involve modification in design spec, IP interfaces differences or variation in the input data pattern. The change in design spec could alter few configuration registers etc or may address larger issues like memory bank size etc. or bring a new IP architecture itself. The changes in IP interfaces could be due to bus protocols or architectural changes in neighboring IPs. The data patterns may vary for each class of applications and corresponding benchmarks. For all such changes in every IP, big or small, one needs to go through the stage of estimation multiple times, followed by exploration algorithms phase to arrive at optimal system configuration.

Finally, the profiling in general, is carried out for individual IPs in isolation. The profiling information can change drastically when the IP is interfaced in a processor-based system [13]. Hence, such methods do not help much in design space exploration. The speed and power specifically can vary significantly when measured at IP level and at system level [47]. In fact, there can be noticeable difference in system level speed and power values even for small changes in design and I/O pattern. We therefore adapt system level profiling in our estimation models. The dynamic power obtained for all our designs is the system power for a processor-IP system with standard bus interface.

The main contributions of this work are:

1. Regression modeling for FPGAs considering I/O toggle and resource utilization
2. Unified power model for multiple IPs
3. Comprehensive system level power estimation approach for IPs
4.2 Background

4.2.1 Multi-variate least square regression model

Linear regression tries to relate a variable of interest, called response variable, to a set of measurable, independent variables, through a linear equation. The actual values of variable of interest is calculated for different sets of independent variables. Curve-fitting is done on these results to arrive at the linear equation. This process is generally called the training phase and the sets involved are called training set.

Consider a sample of m observations done on n variables \((X_1, X_2, \ldots, X_n)\). If these \(n\) variables are assumed to satisfy a linear relation with response variable \(Y\) then it can be represented as:

\[
Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n
\] (4.1)

For regression model shown in Equation 4.1, there may be various ways to calculate the value of regression coefficients \(\beta\), such that the error between the predicted and measured values of response variable is minimized. Let us denote the \(i^{th}\) observation on these variables as \((X_{i1}, X_{i2}, \ldots, X_{im})\), \(i = 1, 2, \ldots, m\). Lets \(Y_i\) be the observed value of the response variable \(Y\) corresponding to the \(i^{th}\) observation. Least squares error method can be used to minimize the difference between predicted and measured values, and objective function can be represented as:

\[
\sum_{i=1}^{m} (Y - \beta_0 - \beta_1 X_{i1} - \beta_2 X_{i2} - \ldots - \beta_n X_{in})^2
\] (4.2)

and we need to minimize this objective function by choosing appropriate coefficients.
4.2.2 FPGA power models using linear regression

The independent variables considered for power models would generally be boundary information like I/O toggle. A single hardware circuit is considered, actual power consumption is measured for a sequence of inputs. The power measured and the toggle count on each port are used to evaluate the coefficients of the equation. Thus an equation is developed to estimate the power consumption of a circuit. Such strategies have been developed for both ASIC and FPGA platforms in the past. However, our experiments revealed that, for smaller circuits, the dynamic power consumption of a circuit varied by small amounts with the toggle count. This is expected behavior for FPGAs which have LUT-based architecture. The logic is mapped to truth-tables as against standard cell mapping in ASICs. So, this gives rise to the possibility of finding generic regression functions for multiple circuits. This means, once the equation is obtained for a set of training data, it may be used to estimate the power consumption of a new circuit. Of course, new variables will be needed to account for varying circuit sizes. In this work, we use FPGA resource utilization as additional variables.

The dynamic power consumption for a circuit is given as

$$P_{dyn} = \frac{1}{2} V^2 F C_{eff} S$$  \hspace{1cm} (4.3)$$

where \(V\) is the circuit voltage, \(F\) the operating frequency, \(C_{eff}\) is the effective capacitance and \(S\) represents input switching activity. There are many variations to this equation. In some cases, the switching activity and capacitance are combined as ‘switching capacitance’, while in some FPGA models, the utilization factor of resources is considered as a separate entity in itself. The voltage and frequency for all circuits will be constant in a given board. Of the 2 remaining variables, switching activity is expected to be accounted for by I/O toggles, while the capacitance by the resource utilization. So, we first develop an equation for power consumption varying just the toggles and keeping other variables fixed (Equation 4.4). Then the area is varied and a new power equation is arrived at (Equation 4.5).
\[ P_1 = K_0 + K_1 S \quad (4.4) \]

\[ P_2 = L_0 + L_1 C_{eff} \quad (4.5) \]

However, the relationship of switching activity and capacitance is multiplicative for power and linear regression cannot adequately model it. So, we take logarithm of the two power estimates and perform linear regression on them.

\[ \log(P_{dyn}) = I_o + I_1 \log(P_1) + I_2 \log(P_2) \quad (4.6) \]

### 4.3 Methodology

In this section, we discuss the overall methodology used for creating the power estimation models. As explained in Fig. 4.1, each IP is run with multiple data patterns. The corresponding VCD files and I/O toggle reports are generated for each pattern. The VCD file is used for estimating dynamic power from XPower. The process is repeated for many IPs and additionally, the resource utilization information of each IP is also obtained. The dynamic power from XPower, toggle count for each data pattern and the resource utilization of each IP, together form the input to the regression tool. We discuss in detail each of these steps, starting with the nature of IPs selected to VCD generation and statistical modeling.

#### 4.3.1 Nature of the IPs used

We considered 13 IPs in all, for various experiments performed. All the IPs were at synthesizable RTL level. While most of the IPs considered were ‘hand-coded’ in Verilog, a few IPs were synthesized from a High level synthesis tool through the behavioral description of IP in C language. HLS is being slowly adapted by industry and certain previous works show
how HLS can be used for design space exploration [47]. Keeping this in mind, we introduced some tool generated IPs.

In order to get a broader picture, we consider system power as our criteria rather than IP power alone. Xilinx Virtex - 2 Pro was the design board used for all our experiments. We interfaced each IP with the microblaze softcore in Virtex board through the standard PLB bus interface and then made measurements and simulations. This gives more accurate timing and power information in the context of SoC designs.
4.3.2 Dynamic power evaluation for reference model

As is the practice in regression-based prediction modeling, we divided each sample set for different experiments into two parts. The first part was used to train the power model and the second part to evaluate the model. For the first part, standard power measurement flow was used. This involved performing timing-based simulation of the design using Modelsim and the simulation data was saved in VCD format. This formed the input to Xilinx XPower tool [48], along with the design files and physical constraints file. The XPower gives a detailed power breakdown based on resource utilization and simulation information for all Xilinx boards. The static power estimate given by XPower, for a given board is constant. Hence, we eliminate that and consider only dynamic power in all our analysis.

4.3.3 Toggle and resource utilization information

Apart from the actual power numbers we get from XPower for the training set, we also need information on toggles and resources used. The toggle count involved counting all the bit flips on a particular port for the duration design is run [14]. There was no particular distinction made between input and output ports for the regression model. The number of ports varied from 2-12 for different designs. In the regression models where different designs were considered, the ones with lesser number of ports were padded with 0. The resources considered for the analysis are multipliers, ram blocks, the FPGA slices, clock multipliers and I/O buffers. This information was obtained from the Xilinx synthesis tools. While performing evaluation, I/O toggle information across ports can be obtained from behavioral models itself. Also, there are techniques to estimate the resource consumption from high level models [30].
4.3.4 Statistical analysis using JMP

We used JMP [49] for creating linear regression models in our analysis. The I/O toggle information and resource utilization information along with power information from XPower tool formed inputs to the tool. Model fitting feature of JMP was used and least squares error was the objective considered.

4.3.5 Model evaluation phase

This step involved finding the accuracy of our models. Various permutations of training and evaluation set were considered to arrive at better estimation models. The I/O toggles and resource utilization information for the models in evaluation phase (not given as input for curve fitting) were plugged into the power model generated by JMP. The average and maximum difference between the actual and estimated values formed the main criteria for model evaluation.

4.4 Results

Multiple experiments were performed to demonstrate the validity of our model, quantitatively demonstrate the improvements in our model over existing approaches and to explain how accuracy can be enhanced. The results are classified into following 3 subsections as under:

1. Obtaining a power model for a given set of IPs and using them to estimate power for varying datapatterns

2. Getting a generic power model to estimate power for new IPs as well as varying datapatterns

3. Grouping strategy to improve accuracy of the estimation model
4.4.1 Varying data patterns for fixed IP set

In this set of experiments, a set of 7 IPs and 10 different data patterns for each IP were considered. For each experiment, the number of data samples in training set was gradually increased. Apart from estimating power from the proposed approach, which we call as ‘Hybrid approach’, we also estimated power by taking only I/O toggle into consideration and separately with only taking resource utilization aspects into consideration. Note that resource utilization-based model would give same power estimate for an IP irrespective of the data pattern variation. Regression-based prediction modeling would generally require around 30 samples to form a useful model. We start with the case of 21 samples (7 IP x 3 data patterns), called s3 to indicate 3 data patterns considered for each IP. We go up to the case of 63 samples in s9.

![Comparison of average percentage errors with varying data samples](image)

Figure 4.2: Comparison of average percentage errors with varying data samples

Presented in Fig. 4.2 is the average percentage errors measured for each sample. Note that, the data set included in evaluation for error calculations are different from the one in training set. Also, shown in Fig. 4.3 is the maximum percentage error measured in each sample size.
It can be seen that the toggle-only model and our hybrid model both give almost similar kind of errors. This is expected behavior since we do not introduce any new IPs in evaluation set. Had the new IPs been introduced, the variation that would occur in resource utilization will not be accounted for by the toggle-only model. The resource utilization-based model suffers as it gives same power estimate for an IP with different data patterns. This clearly shows the need to take I/O toggles and corresponding VCD-based power estimation into consideration, to get higher quality results. Even though average errors remain unchanged with increase in sample size, the maximum error from the model drops down and stabilizes around $s_7$. We consider $s_7$ as the sample size for all other experiments. One of the designs considered (DCT), gave very abnormal results and did not fit well into most of our experiments. We deal with this separately.

This kind of an analysis would come to good use in real-world scenarios where the SoC components are fixed and one needs to make power estimates for different application scenarios. Instead of generating VCD files for new set of benchmarks, one can plug in the I/O
port information and get the power estimates. If the regression model has been developed with sufficient data patterns (like s7 in our case), one can also establish with reasonable confidence, the maximum possible deviation from the actual values. This depends on how far statistically the new data patterns are from the ones in training set. There are many techniques in literature that show how to increase accuracy and the confidence level in such scenarios[16].

4.4.2 Varying IPs for fixed data pattern

In this set of experiments, we increased the number of IPs in both training and evaluation set. The number of data patterns given in each case was held fixed at 7 per IP. This experiment helps evaluate the proposed model more comprehensively, as this is a generic model for IP power estimation. In the first sample space 7 different IPs were considered. Of these, 6 were put in training set in each experiment (and hence the name IP6) and 1 in evaluation set. For each IP in the evaluation set 10 different data patterns were considered and the average error was calculated over all of them. Later, the sample size was increased to IP9 and IP12 and similar experiments were performed.

Fig. 4.4 shows average errors measured over different data patterns for 6 IPs in the sample spaces IP6, IP9 and IP12. As, the number of IPs increase, one can notice that the percentage error goes down. However, some of the the percentage errors, even in IP12 case are above 10%. This is because, the number of IPs in training set is less and the model is not trained enough for resource utilization related parameters. It is expected that the trend shown in these experiments will continue as the number of IPs increase. In real-world scenarios as well, it would be unrealistic to expect such a large number of IPs for a system. We look at ways to address this problem as well.

Shown in Fig. 4.5 is again a comparison of our model with toggle-only and resource utilization-based approaches for all 3 sample spaces. The vertical bars represent average errors for all IPs considered in a particular sample space and not just the 6 designs shown in Fig. 4.4. As
expected, toggle-only approach would show maximum deviation while resource utilization-based approach performs very well. In fact, resource utilization-based approach performs better. This is because, the number of samples with varying toggles was high and the one with resource utilization information was less. As such, lesser weights were assigned to resource utilization related parameters in equation. This trend again shows that increasing number of IPs will aid in increasing accuracy.

Figure 4.4: Comparison of average percentage errors with varying IP samples

4.4.3 IP grouping to increase accuracy

The overall accuracy with regression will not be satisfactory unless the sample space is large enough. We also noticed that in each experiment, there were certain designs which did not fit well in any models (like DCT). Analysis of the input data sequences and average errors revealed a clear pattern. The designs which did not fit were statistically very different from training sequences (toggle pattern, number of ports, resource utilization etc). One possible strategy which addresses both these issues is grouping. In these set of experiments,
we grouped IPs with similar characteristics together and evaluated error varying both data pattern and IP samples.

In the first group of IP samples, 4 IPs were considered. These IPs had same number of control and data ports and also were similar in resource utilization. For each experiment, an IP not included in training set and the data patterns of remaining IPs of the group, not in training set were considered. Error percentage fell sharply for all IPs (Fig. 4.6). The circuit named ‘Bit’ shows relatively more error because of the data pattern. One of the ports was 32 bit size in this case, while all other designs had only 16-bit ports. The group named G4 had most homogeneous composition and hence least average error. The one labeled G8 had the very diverse set of IPs and hence accuracy deteriorates. Fig. 4.7 shows a comparison of the hybrid model with toggle-only and resource utilization-based models.
Figure 4.6: Error increase with diverse IP sets

Figure 4.7: Comparison of different models
Chapter 5

Future Work and Conclusions

5.1 Pareto optimization technique for reconfigurable systems

This chapter looks into algorithmic aspect of design space exploration. It discusses a pareto algebra-based exploration strategy to perform energy-delay tradeoffs. The speed, area and power are estimated for each possible configuration of an IP. The techniques suggested in previous chapters can be used to get these data. The algorithm is developed for heterogeneous FPGAs with 3 different platforms for implementation - the software, hardware and the partially reconfigurable hardware.

5.1.1 FPGA platforms

Traditionally, there have been two different platforms in FPGAs- the processor and the reconfigurable fabric. An application algorithm was partitioned into software and hardware components, and then implemented accordingly on processor and the fabric. The compute intensive components, which caused the bottlenecks were ported to hardware, as the
hardware resources are limited. Also, power considerations changed the mapping patterns. Similar method was used for mapping multiple IPs as well. Many of the designs were larger than the available fabric size. Multiple boards were used for implementing such designs. Another technique called run-time reconfiguration was used. The configuration bitstreams for the hardware components were stored in memory and were loaded when needed. Different components time-shared the available reconfigurable resources. Recent developments in FPGAs involve partially reconfigurable components. Certain parts of the FPGA fabric can be reconfigured while other parts are running. This will be advantageous in two scenarios: when certain components on the hardware have to be up all the time and in situations where the frequency of usage of different components is not same. Thus, there are 3 different platforms on FPGAs to map IPs - the software, static hardware and the partially reconfigurable hardware, as shown in Fig. 5.1.

![Figure 5.1: Partially Reconfigurable FPGA Platform](image)

Each of the platforms offer specific advantages and overheads. The software implementation will be quicker to design. There is no notion of area constraint. However, software implementation turns out to be the slowest, sometimes orders of magnitude slower than other
platforms. The static hardware gives the highest speedup possible. But, the available area is limited and not all applications can be mapped to FPGAs. The partially reconfigurable resource will have theoretically no area bounds. But, partial reconfiguration process introduces timing overheads. Also, the time difference between 2 components sharing the same reconfigurable space has to be greater than the reconfiguration time. The implementation gets more and more complex as one moves from software to static hardware to partially reconfigurable hardware. Finally, many more tradeoffs emerge when power consumption is also considered. Power consumption of software implementation may be lesser than hardware. But since its very slow, overall energy consumption may become very high. A component on hardware will consume certain amount of power always, in addition to the actual computation power consumption. There is also a certain amount of energy overhead involved in doing partial reconfiguration. All these different conflicts will have to be resolved to obtain good system implementation.

5.1.2 Pareto optimization algorithm

Problem formulation

The design space exploration problem is formulated as a multi-objective optimization problem first. The two minimization objectives, also called quantities are energy and delay. The area requirement will be considered as a constraint. The implementation of the task in each of the 3 available platforms will render a new configuration in the configuration space. Additionally, the tasks will have to maintain the execution order strictly. This will be mainly due to data dependency and consistency issues.

Problem statement: Given a set of tasks with an execution order \( \{T_1, T_2, T_3, \ldots, T_n\} \) and 3 implementation platforms - software, static hardware and partially reconfigurable hardware, obtain a mapping for the tasks, with minimum energy consumption and delay, preserving the execution sequence and meeting all the area constraints.
Algorithm

If the number of tasks is $N$, then a 2-dimensional matrix of size $N \times 3$ is formed, denoted as $T[i][j]$, where $i$ varies from 1 to $N$ and $j$ varies from 1 to 3, indicating software, static and partially reconfigurable hardware implementation. Three new functions $E(T[i][j])$, $D(T[i][j])$ and $A(T[i][j])$ that give the energy, delay and area estimate of $T[i][j]$, are evaluated.

1. **Initialization**: First, an initialization set $S = \{C_1, C_2, C_3\}$ is considered, where configuration $C_1$ itself is a set of 3 elements $C_1 = \{C_{1E}, C_{1D}, C_{1A}\}$, representing the values of energy, delay and area functions on $T[1][1]$, that corresponds to software implementation. Similarly, $C_2$ and $C_3$ are for static hardware and partially reconfigurable hardware implementations respectively.

   $$S_I = \{C_1, C_2, C_3\},$$
   where $C_n = \{E(T[1][n]), D(T[1][n]), A(T[1][n])\}$, $n$ varies from 1 to 3

2. **Constraining**: Area requirement forms the main constraint in this optimization. Many more constraints can be added. For example, a feasibility constraint can be added to eliminate implementations that consume a lot of time to design. There are 2 different area constraints possible. $D_H$ represents the set of all the configurations which meet the static hardware area requirement. $D_P$ represents the set of all the configurations which satisfy partially reconfigurable hardware area requirement.

   $$S_C = \{S_I \cap D_H\} \cap D_P$$

3. **Minimization**: This step involves eliminating all the configurations that have been dominated. Two costs are defined at this stage - the time cost and the energy cost.

   Time cost ($C_T$) is a measure which indicates the maximum time for all existing tasks to complete (in all 3 platforms) and a new task can be scheduled for execution only after this time. This helps in scheduling and maintaining dependency.
\( C_T = \max(T_s, T_h, T_p), \)
where \( T_s = \sum T_e S \), sum of execution time of all tasks mapped to software
\( T_h = \max(T_e H) \), the maximum of the execution time of all tasks mapped to static hardware and
\( T_p = \sum T_e P / P_n \), where \( P_n \) is total number of partially reconfigurable regions available.
The sequential execution in software platform causes the summation of execution times.
Due to parallel execution, the total time in static hardware is the maximum execution time of all tasks. The execution time for partially reconfigurable platform is crude.
There will be many execution overheads involved. Refer to [50] for more accurate model.

Calculating energy cost (\( C_E \)) is simpler. It is the sum of energy consumption of all the tasks mapped to different platforms.

In the new set \( S_M = \min \{ S_C \} \), all the configurations which have been dominated (the ones which will be worse in both energy and time cost when compared with another configuration) will be eliminated.

4. **Expansion**: A new set \( A \) is defined,

\[
A = \{ C_s, C_h, C_p \}, \text{ where}\]
\( C_s = T[n+1][1], \) \( C_h = T[n+1][2] \) and \( C_p = T[n+1][3] \)

A new \( S_I \) is formed by taking the cross-product of \( A \) and \( S_M \)

\[
S_I(new) = \{ A \times S_M \}
\]

The process is restarted from constraining phase again and continued till all tasks are mapped. At the end of \( N \) iterations, a pareto minimal set of configurations will be available. Specific weights can be assigned to the energy and delay costs to arrive at a single solution.
5.2 Other possible extensions to the framework

Apart from finding usage in algorithms like the pareto algorithm discusses in previous sections, there are many other interesting extensions possible to the work presented. One useful extension to the quick prototyping technique would be in automation aspects. Once the Verilog file is generated by the C2R tool, certain tweaking is needed to interface the modules on Xilinx boards. In this regard, either wrappers can be generated by the compiler or scripts can be written to generate interface files. Also, compiler can be made aware of partially reconfigurable fabric and compiler optimization techniques could be inculcated to automate the reconfiguration flow.

With regard to regression analysis technique, there are 2 possible extensions. The accuracy of the proposed model can be improved in many ways. Power model using non linear regression could improve accuracy. The independent variable considered and the nature of data considered could be analyzed more statistically. Second extension could be to apply the framework for measuring reconfiguration power. There are 2 components to the delay and energy overheads in reconfigurable systems - the overheads when the task is running on the board and the overheads for doing the initial bitstream configuration for them to run. This will gain importance as the frequency of the partial reconfiguration increases.

5.3 Conclusions

The thesis addressed the issue of evaluation and exploration techniques for FPGAs. A case is made for FPGAs as an alternative for SoC design in consumer electronics industry. The issues involved in design space exploration in FPGA-based design have been analyzed. Two distinct approaches have been proposed for the problem.

The quick prototyping technique identifies and reduces key bottlenecks in the exploration cycle. The slow, repetitive aspects are made faster using High Level Synthesis. This tech-
nique can be used in scenarios where the major parts of the system are being designed from the scratch. This approach also finds usage in early architectural explorations where only behavioral models are available. The approach tolerates high degree of heterogeneity and the bus protocols, interfaces and memory models can be evaluated.

The regression analysis helps in doing quick estimation of design parameters. This can be used in conjunction with algorithms like the pareto technique discussed in this thesis. This approach is more suited for scenarios where reuse is extensive and many of the system details are prefixed. This takes advantage of extensive data profiling that the IPs are subjected to. This technique is more structured and gives more quantitative results. But, its scope is limited. For example, the algorithm presented in this work is applicable to mapping IPs to 3 platforms of FPGA. Thus depending on the design scenario, one of the two approaches can be adopted.
Appendix A

Data for Regression Analysis

This appendix tabulates all the results obtained in regression analysis. In all 13 IPs were considered for experiments. They are AES (AES Encryption unit), BIT (Bit-Counter), CSR (Caesar cipher), COR (Cordic Algorithm), CRC (Cyclic Redundancy Check unit), DES (Encryption Block), DES - A (Area optimized implementation of DES block), FIB (Fibonacci Generator), HAM (Hamming code), SHA (Secured Hash Algorithm), 3-DES (Triple DES unit) and DCT (Discrete Cosine Transform unit).

A.1 Varying toggles with IP set fixed

The percentage errors between the actual and predicted values for Hybrid model (Table A.1), Toggle model (Table A.2) and Area model (Table A.3) are given for 6 IPs considered - FIR, CRC, COR, CSR, BIT, FIB, is tabulated.
Table A.1: Effect of Increasing Toggle Samples on Hybrid Model

<table>
<thead>
<tr>
<th></th>
<th>h3</th>
<th>h4</th>
<th>h5</th>
<th>h6</th>
<th>h7</th>
<th>h8</th>
<th>h9</th>
</tr>
</thead>
<tbody>
<tr>
<td>fir</td>
<td>2.361</td>
<td>2.5714</td>
<td>2.9128</td>
<td>2.983</td>
<td>2.6601</td>
<td>2.9394</td>
<td>0.2277</td>
</tr>
<tr>
<td>crc</td>
<td>2.7278</td>
<td>2.5235</td>
<td>2.9771</td>
<td>2.6539</td>
<td>3.3094</td>
<td>2.613</td>
<td>5.1013</td>
</tr>
<tr>
<td>cor</td>
<td>1.8914</td>
<td>2.0725</td>
<td>1.8115</td>
<td>1.554</td>
<td>1.0536</td>
<td>1.0902</td>
<td>0.4333</td>
</tr>
<tr>
<td>csr</td>
<td>0.2353</td>
<td>0.9021</td>
<td>0.7337</td>
<td>0.8801</td>
<td>0.7405</td>
<td>0.3776</td>
<td>0.8195</td>
</tr>
<tr>
<td>bit</td>
<td>0.4246</td>
<td>0.4933</td>
<td>0.2152</td>
<td>0.2072</td>
<td>0.2709</td>
<td>0.3928</td>
<td>0.2133</td>
</tr>
<tr>
<td>fib</td>
<td>0.4589</td>
<td>0.5277</td>
<td>0.5461</td>
<td>0.5217</td>
<td>0.4948</td>
<td>0.471</td>
<td>0.3399</td>
</tr>
</tbody>
</table>

Table A.2: Effect of Increasing Toggle Samples on Toggle Model

<table>
<thead>
<tr>
<th></th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
<th>t9</th>
</tr>
</thead>
<tbody>
<tr>
<td>fir</td>
<td>2.3663</td>
<td>2.5799</td>
<td>2.9254</td>
<td>3.0002</td>
<td>2.3312</td>
<td>3.0087</td>
<td>0.3062</td>
</tr>
<tr>
<td>crc</td>
<td>2.7387</td>
<td>2.5298</td>
<td>2.979</td>
<td>2.6645</td>
<td>2.7984</td>
<td>2.6307</td>
<td>5.1865</td>
</tr>
<tr>
<td>cor</td>
<td>1.8984</td>
<td>2.0875</td>
<td>1.8201</td>
<td>1.5527</td>
<td>1.0348</td>
<td>1.0998</td>
<td>0.3485</td>
</tr>
<tr>
<td>csr</td>
<td>0.6055</td>
<td>0.8514</td>
<td>0.6823</td>
<td>0.7933</td>
<td>1.1935</td>
<td>0.2397</td>
<td>0.8466</td>
</tr>
<tr>
<td>bit</td>
<td>0.4239</td>
<td>0.4907</td>
<td>0.2145</td>
<td>0.2022</td>
<td>0.3071</td>
<td>0.3962</td>
<td>0.1664</td>
</tr>
<tr>
<td>fib</td>
<td>0.4626</td>
<td>0.535</td>
<td>0.5515</td>
<td>0.5248</td>
<td>0.564</td>
<td>0.4739</td>
<td>0.3691</td>
</tr>
</tbody>
</table>
Table A.3: Effect of Increasing Toggle Samples on Area Model

<table>
<thead>
<tr>
<th></th>
<th>a3</th>
<th>a4</th>
<th>a5</th>
<th>a6</th>
<th>a7</th>
<th>a8</th>
<th>a9</th>
</tr>
</thead>
<tbody>
<tr>
<td>fir</td>
<td>5.991</td>
<td>5.9596</td>
<td>5.9596</td>
<td>5.9778</td>
<td>5.9244</td>
<td>5.9002</td>
<td>6.0419</td>
</tr>
<tr>
<td>bit</td>
<td>0.8577</td>
<td>0.7034</td>
<td>0.7034</td>
<td>0.806</td>
<td>0.8406</td>
<td>0.87</td>
<td>0.8977</td>
</tr>
</tbody>
</table>

A.2 Introducing new IPs in evaluation set

The following 3 results (Table A.4, Table A.5, Table A.6) involve experiments where new IPs were introduced in the evaluation set. IP6 indicates that 6 IPs were considered in the training set.

A.3 Clustering Strategy

The final 3 set of results are tabulated for the grouping experiments for groups of 4, 6 and 8. Again 3 results for all 3 models (Table A.7, Table A.8, Table A.9)
Table A.4: Effect of Increasing Number of IPs on Hybrid Model

<table>
<thead>
<tr>
<th></th>
<th>IP6</th>
<th>IP9</th>
<th>IP12</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>N/A</td>
<td>N/A</td>
<td>4.86</td>
</tr>
<tr>
<td>BIT</td>
<td>11.18</td>
<td>4.41</td>
<td>4.94</td>
</tr>
<tr>
<td>CSR</td>
<td>6.65</td>
<td>6.83</td>
<td>6.72</td>
</tr>
<tr>
<td>COR</td>
<td>11.62</td>
<td>5.35</td>
<td>5.93</td>
</tr>
<tr>
<td>CRC</td>
<td>10.99</td>
<td>6.83</td>
<td>6.62</td>
</tr>
<tr>
<td>DES</td>
<td>N/A</td>
<td>12.37</td>
<td>6.34</td>
</tr>
<tr>
<td>DES-A</td>
<td>N/A</td>
<td>11.4</td>
<td>8.56</td>
</tr>
<tr>
<td>FIB</td>
<td>3.17</td>
<td>3.38</td>
<td>4.83</td>
</tr>
<tr>
<td>FIR</td>
<td>4.67</td>
<td>5.45</td>
<td>5.34</td>
</tr>
<tr>
<td>HAM</td>
<td>N/A</td>
<td>6.92</td>
<td>4.2</td>
</tr>
<tr>
<td>SHA</td>
<td>N/A</td>
<td>N/A</td>
<td>6.07</td>
</tr>
<tr>
<td>3-DES</td>
<td>N/A</td>
<td>N/A</td>
<td>10.47</td>
</tr>
</tbody>
</table>
Table A.5: Effect of Increasing Number of IPs on Toggle Model

<table>
<thead>
<tr>
<th></th>
<th>IP6</th>
<th>IP9</th>
<th>IP12</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>N/A</td>
<td>N/A</td>
<td>8.12</td>
</tr>
<tr>
<td>BIT</td>
<td>11.21</td>
<td>7.7</td>
<td>6.77</td>
</tr>
<tr>
<td>CSR</td>
<td>6.64</td>
<td>8.04</td>
<td>7.9</td>
</tr>
<tr>
<td>COR</td>
<td>11.65</td>
<td>5.88</td>
<td>7.18</td>
</tr>
<tr>
<td>CRC</td>
<td>11.03</td>
<td>8.78</td>
<td>8.48</td>
</tr>
<tr>
<td>DES</td>
<td>N/A</td>
<td>12.7</td>
<td>5.18</td>
</tr>
<tr>
<td>DES-A</td>
<td>N/A</td>
<td>12.58</td>
<td>9.44</td>
</tr>
<tr>
<td>FIB</td>
<td>3.19</td>
<td>6.43</td>
<td>5.18</td>
</tr>
<tr>
<td>FIR</td>
<td>4.67</td>
<td>6.94</td>
<td>6.73</td>
</tr>
<tr>
<td>HAM</td>
<td>N/A</td>
<td>9.81</td>
<td>6.1</td>
</tr>
<tr>
<td>SHA</td>
<td>N/A</td>
<td>N/A</td>
<td>6.95</td>
</tr>
<tr>
<td>3-DES</td>
<td>N/A</td>
<td>N/A</td>
<td>69.07</td>
</tr>
</tbody>
</table>
Table A.6: Effect of Increasing Number of IPs on Area Model

<table>
<thead>
<tr>
<th></th>
<th>IP6</th>
<th>IP9</th>
<th>IP12</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>N/A</td>
<td>N/A</td>
<td>12.94</td>
</tr>
<tr>
<td>BIT</td>
<td>4.9</td>
<td>4.5</td>
<td>4.51</td>
</tr>
<tr>
<td>CSR</td>
<td>9.25</td>
<td>6.8</td>
<td>6.18</td>
</tr>
<tr>
<td>COR</td>
<td>10.87</td>
<td>5.95</td>
<td>5.91</td>
</tr>
<tr>
<td>CRC</td>
<td>8.06</td>
<td>6.18</td>
<td>5.99</td>
</tr>
<tr>
<td>DES</td>
<td>N/A</td>
<td>6.45</td>
<td>7.3</td>
</tr>
<tr>
<td>DES-A</td>
<td>N/A</td>
<td>4.25</td>
<td>4.2</td>
</tr>
<tr>
<td>FIB</td>
<td>6.3</td>
<td>5.16</td>
<td>5.03</td>
</tr>
<tr>
<td>FIR</td>
<td>6.53</td>
<td>5.33</td>
<td>5.25</td>
</tr>
<tr>
<td>HAM</td>
<td>N/A</td>
<td>4.45</td>
<td>4.3</td>
</tr>
<tr>
<td>SHA</td>
<td>N/A</td>
<td>N/A</td>
<td>5.82</td>
</tr>
<tr>
<td>3-DES</td>
<td>N/A</td>
<td>N/A</td>
<td>6.6</td>
</tr>
</tbody>
</table>

Table A.7: Effect of Introducing Heterogeneity on Hybrid Model

<table>
<thead>
<tr>
<th>IP</th>
<th>G4</th>
<th>G6</th>
<th>G8</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>4.422671916</td>
<td>5.576951</td>
<td>11.32849</td>
</tr>
<tr>
<td>CSR</td>
<td>1.109049847</td>
<td>5.978093</td>
<td>12.8033</td>
</tr>
<tr>
<td>HAM</td>
<td>0.328995113</td>
<td>3.093375</td>
<td>4.923756</td>
</tr>
<tr>
<td>FIB</td>
<td>0.766317665</td>
<td>2.897144</td>
<td>2.819894</td>
</tr>
<tr>
<td>COR</td>
<td>N/A</td>
<td>14.27159</td>
<td>1.573383</td>
</tr>
<tr>
<td>SHA</td>
<td>N/A</td>
<td>2.575933</td>
<td>3.270703</td>
</tr>
<tr>
<td>CRC</td>
<td>N/A</td>
<td>N/A</td>
<td>16.07001</td>
</tr>
<tr>
<td>FIR</td>
<td>N/A</td>
<td>N/A</td>
<td>14.60343</td>
</tr>
</tbody>
</table>
Table A.8: Effect of Introducing Heterogeneity on Toggle Model

<table>
<thead>
<tr>
<th>IP</th>
<th>G4</th>
<th>G6</th>
<th>G8</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>7.751121692</td>
<td>8.332570277</td>
<td>23.68830693</td>
</tr>
<tr>
<td>CSR</td>
<td>1.314728182</td>
<td>14.53863526</td>
<td>11.42926278</td>
</tr>
<tr>
<td>HAM</td>
<td>0.454787363</td>
<td>10.68750302</td>
<td>9.352169916</td>
</tr>
<tr>
<td>FIB</td>
<td>0.695032301</td>
<td>0.387614168</td>
<td>3.983069726</td>
</tr>
<tr>
<td>COR</td>
<td>N/A</td>
<td>30.43051397</td>
<td>44.98551863</td>
</tr>
<tr>
<td>CRC</td>
<td>N/A</td>
<td>30.94683254</td>
<td>17.72100568</td>
</tr>
<tr>
<td>FIR</td>
<td>N/A</td>
<td>N/A</td>
<td>36.34785962</td>
</tr>
</tbody>
</table>

Table A.9: Effect of Introducing Heterogeneity on Area Model

<table>
<thead>
<tr>
<th>IP</th>
<th>G4</th>
<th>G6</th>
<th>G8</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>1.373677777</td>
<td>0.838533559</td>
<td>1.924872264</td>
</tr>
<tr>
<td>CSR</td>
<td>2.562245523</td>
<td>10.56121834</td>
<td>16.9627327</td>
</tr>
<tr>
<td>HAM</td>
<td>1.482867579</td>
<td>4.084356282</td>
<td>6.216877449</td>
</tr>
<tr>
<td>FIB</td>
<td>2.593401648</td>
<td>4.377796837</td>
<td>5.420019158</td>
</tr>
<tr>
<td>COR</td>
<td>N/A</td>
<td>16.29497374</td>
<td>20.98821904</td>
</tr>
<tr>
<td>CRC</td>
<td>N/A</td>
<td>14.37980425</td>
<td>11.99628903</td>
</tr>
<tr>
<td>FIR</td>
<td>N/A</td>
<td>N/A</td>
<td>38.61537698</td>
</tr>
</tbody>
</table>
Bibliography


