Mapping large-scale systems on to high density FPGAs

Herath, Kalindu Bandara

2020

Herath, K. B. (2020). Mapping large-scale systems on to high density FPGAs. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/143725>

<https://doi.org/10.32657/10356/143725>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 21:43:21 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**MAPPING LARGE-SCALE SYSTEMS ON TO
HIGH DENSITY FPGAs**

KALINDU BANDARA HERATH
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
2020

**MAPPING LARGE-SCALE SYSTEMS ON TO
HIGH DENSITY FPGAs**

KALINDU BANDARA HERATH

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

2020

"Being the best may not be possible at times. But, being excellent is possible every time"

I would like to dedicate this thesis to my loving sister, Buddhi.

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

26/09/2019

.....

Date

.....

Kalindu Bandara Herath

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

26/09/2019

.....

Date

.....

Prof. Thambipillai Srikanthan

Authorship Attribution Statement

This thesis contains material from 4 papers published in the following peer-reviewed conferences in which I am listed as an author.

Chapter 3 is published as [K. Herath, A. Prakash, J. Guiyuan, and T. Srikanthan. Communication-aware Partitioning for Energy Optimization of Large FPGA Designs. In Proceedings of the on Great Lakes Symposium on VLSI 2017 \(GLSVLSI '17\). ACM, New York, NY, USA, 407-410. DOI: 10.1145/3060403.3060441](#)

The contributions of the co-authors are as follows:

- Prof Srikanthan provided the initial project direction.
- Dr Prakash participated in idea generation and brain storming sessions.
- Dr Guiyuan provided the initial idea for developing a partitioning algorithm.
- I prepared the manuscript drafts. The manuscript was revised by Dr Prakash.
- I designed the studies, performed all the experiments and data analysis.

Chapter 4 is published as [K. Herath, A. Prakash and T. Srikanthan. Performance Estimation of FPGA Modules for Modular Design Methodology Using Artificial Neural Network. Applied Reconfigurable Computing. Architectures, Tools, and Applications. ARC 2018. Lecture Notes in Computer Science, vol 10824. Springer, Cham. DOI: 10.1007/978-3-319-78890-6_9](#)

The contributions of the co-authors are as follows:

- Prof Srikanthan provided the initial project direction.
- Dr Prakash participated in idea generation and brain storming sessions.
- Dr Prakash provided the initial idea for Artificial Neural Network.
- I prepared the manuscript drafts. The manuscript was revised by Dr Prakash.
- I designed the studies, performed all the experiments and data analysis.

Chapter 5 is published as [K. Herath, A. Prakash, U. Kanewala and T. Srikanthan. Communication-Aware Module Placement for Power Reduction in Large FPGA Designs.](#)

2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI).IEEE, 2018. DOI: 10.1109/ISVLSI.2018.00047

The contributions of the co-authors are as follows:

- Prof Srikanthan provided the initial project direction.
- Dr Prakash participated in idea generation and brain storming sessions.
- I prepared the manuscript drafts. The manuscript was revised by Dr Prakash.
- I designed the studies, performed all the experiments and data analysis.
- Udaree Kanewala assisted in experiments

Chapter 5 is published as [K. Herath, A. Prakash and T. Srikanthan. Ant Colony Optimization based Module Footprint Selection and Placement for Lowering Power in Large FPGA Designs. 2018 International Conference on ReConFigurable Computing and FPGAs \(ReConFig\).IEEE, 2018. DOI: 10.1109/RECONFIG.2018.8641731](#)

The contributions of the co-authors are as follows:

- Prof Srikanthan provided the initial project direction.
- Dr Prakash participated in idea generation and brain storming sessions
- Dr Guiyuan provided the initial algorithm model.
- I prepared the manuscript drafts. The manuscript was revised by Dr Prakash.
- I designed the studies, performed all the experiments and data analysis.

26/09/2019

.....

Date

.....

Kalindu Bandara Herath

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor and mentor, Prof. Thambipillai Srikanthan, for his valuable guidance, motivation, and support for a successful and uninterrupted Ph.D. journey. Without his dedication in my research project, this Ph.D. would never be a reality. It was a complete privilege to have not only his expertise in the research domain, but also his valuable pieces of advice given as a beautiful human spirit.

My warmest gratitude also goes to Dr. Alok Prakash. His enormous support and dedication throughout this research is invaluable. Brainstorming and in-depth technical discussions along with his thorough revision of research writings helped me to produce back to back publications. His assistance to me was always one step ahead of my expectations, and this gratitude will never be enough. I would also like to thank Dr. Lam Siew Kei for strengthening the research topic in the early stages. It is also impossible to forget the support given by Dr. Guiyuan in my research. My sincere appreciation also goes to Dr. Ravi Kumar Satzoda for nourishing my interests in FPGAs towards a Ph.D.

I would thankfully like to remind the administrative support of Mr. Jeremiah Chua and Ms. Poh Ling. Mr. Jeremiah's swift responses in technical difficulties I encountered were always helpful. Ms. Poh Ling has always been a strength to me in policies and formalities, especially during candidature extension process when I did not see any hope. I would also like to pay my warmest appreciation for all the beautiful people in NTU canteens who always made me smile with an extra bit of food and care.

I was very fortunate to have my friend, Mr. Aung Lwin Phyto beside me during the time got tough in all aspects. His warmest support assisted me in focusing on my research without any worry. His financial support during the confirmation examination is one of the best things I have ever got from a friend. My colleagues, Deshya, and Thilina, has always given their touching support to me throughout my research. Lunch breaks, coffee times with them, and

of course, their home-cooked treats has helped me to leave heavy minds behind. I always knew that I could count on them. My colleagues Vijeta and Supriya were sisters to me, who always approached me with the most exceptional help. My confirmation examination could not be a success without them. Max Wang's support during the last mile in this Ph.D. has comforted me to stay on track when nothing seemed to be in my favor. Thank you, all.

My family has always been the greatest strength to me. Though my father is not around with me today, he has ensured I grow up this far without any hassle. My mother has given the love of both parents since my father's passing, that I found the worth in this tough journey. My two sisters, my brother, and my brother-in-law, were always my best friends that anyone can envy. Their support in this research was always silent and often disguised as a box of rice, a dessert, a car ride, or even as hidden beautiful gifts. But it will never be forgotten.

Abstract

Modern FPGAs that benefit from advancement in process technology and hard IP cores are increasingly becoming the choice for multi-million logic cell designs due to lower Non-Recurring Engineering (NRE) costs and shorter Time-to-Market (TTM) pressures. While state-of-the-art CAD tools are capable of efficiently mapping small to medium-scale designs, they suffer from prohibitively long compilation time, higher power consumption and sub-optimal compute performance for large applications. In this thesis, novel techniques have been proposed to accelerate the mapping of large applications into high-density heterogeneous FPGAs while lowering the overall power consumption without compromising compute performance.

A communication-aware partitioning technique has been proposed to represent a large design into smaller subsystems by analyzing the signal interaction characteristics at the Register Transfer Level (RTL) in Chapter 3. Basic computation units exhibiting higher intra-unit communication costs are systematically merged to form a subsystem without violating the constraints imposed on the routing resources of target FPGA. Moreover, an area constraint is also imposed during this merging process to ascertain that the basic computation units of a subsystem remain in close proximity to each other. The subsystem generation process has been automated to represent a large design into subsystems, given the area constraint and routing congestion metric and each subsystem was subsequently ported to FPGA using the commercial Intel (Altera) Quartus Prime software. Evaluations based on large applications, derived from the widely used Polybench benchmark suite, have been carried out to show that the proposed technique achieves up to 60% improvement in compute performance (F_{max}) and nearly 20% reduction in the overall energy consumption. Moreover, the proposed method can also be readily applied to other FPGA families and CAD flows.

A scalable technique for characterizing modern FPGA devices has been proposed next to facilitate the target-aware mapping of subsystems of a large design onto high-density heterogeneous FPGAs. This one-time offline characterization process involves the determination of FPGA resources within templates of varying sizes at any given location on the target FPGA. A wide range of computation units, exhibiting the characteristics of Polybench benchmark suite, were then mapped onto these templates for estimating F_{max} at any given location on the target FPGA. A machine learning model was also proposed by leveraging on the large number of data extracted from this characterization process to automate F_{max} estimation with high accuracy. The proposed technique has been extensively evaluated using 26 applications from the Polybench benchmark suite and synthetic benchmarks on 4 FPGA devices from Altera to show that the average error of this automated F_{max} estimation process is less than 4%.

In order to minimize inter-subsystem communication costs of a large design, Ant Colony Optimization (ACO) algorithm is proposed to identify subsystems that must be placed in close proximity to each other. Routing congestion estimation has also been incorporated to limit the size of each group of subsystems. Next, most profitable footprints, defined by position and shape of templates, are selected for each subsystem. A modified ACO algorithm is then employed to determine the most optimal non-overlapping footprint for each subsystem. The proposed ACO algorithm further aims to identify footprints in close proximity to maximize compute performance and power savings. Footprints exhibiting a tight integration are locked in place to facilitate rapid convergence of ACO's iterative process. Experimental results show the subsystem placement technique can outperform Quartus CAD flow, with over 17% of routing power reduction without impeding on performance (F_{max}).

The proposed techniques presented in this thesis have been integrated into an automated subsystem mapping framework that can be leveraged to efficiently map large applications that consume 160K or more logic cells onto high-density heterogeneous FPGAs. To further ensure tight integration of subsystems, a subsystem splitting strategy was also incorporated to overcome inadvertent gaps that are formed during the ACO placement process. Experimental results show that the proposed framework can reduce routing power by over 18% while improving compute performance by 8%. Moreover, the solutions generated by the proposed approach are close to 23% more energy-efficient than those generated using existing commercial CAD tools. Unlike state-of-the-art commercial tools that take prohibitively long time for mapping large applications, the proposed approach is also notably faster and compile deterministically.

The integrated framework has also been deployed for an automated design space exploration in which the parameters used to control the subsystem generation process are varied to influence both the size and number of subsystems for a given large application. Each subsystem so generated are mapped onto target FPGA with the help of latest commercial CAD tools to facilitate the design space exploration. Unlike existing commercial solutions, the techniques proposed have paved the way for notable power savings, particularly for large applications. Moreover, they provide for higher F_{max} performance and lower runtime, thereby overcoming the drawback of state-of-the-art commercial solutions. Therefore, the proposed techniques can be readily deployed to support emerging high-density FPGA architectures while complementing the advances in the emerging commercial tools to map very large applications.

Table of contents

List of figures	xxi
List of tables	xxv
1 Introduction	1
1.1 Motivation	1
1.2 Research Objective	3
1.3 Organization	4
1.4 Publications	5
2 Literature Review	7
2.1 Introduction	7
2.2 Mapping Applications on FPGA	9
2.2.1 Modern Heterogeneous FPGAs	9
2.2.2 Trend in Modern FPGA Applications	11
2.2.3 Compiling Applications into FPGAs	14
2.2.4 Placement Techniques for Large Heterogeneous FPGAs	24
2.3 Techniques for Power Efficient Mapping	27
2.3.1 Power Consumption of FPGAs	27
2.3.2 Static Power Reduction	29
2.3.3 Dynamic Power Reduction	30

2.4	Runtime of FPGA CAD tools	32
2.4.1	Fast Placement with Low Performance	33
2.4.2	Modular Design Methodologies	34
2.5	Summary	36
3	Communication-aware Subsystem Generation	39
3.1	Introduction	39
3.2	Methodology	41
3.2.1	RTL Designs and CAD Compilation	41
3.2.2	Problem Formulation	45
3.2.3	Extracting Communication Frequency from RTL Implementations .	47
3.2.4	Communication-aware Subsystem Generation	51
3.2.5	Automating Subsystem Generation using CAD Tools	56
3.3	Performance Evaluation	58
3.3.1	Comparison Strategy	58
3.3.2	Test Applications	59
3.3.3	Test Vectors	62
3.3.4	Results and Discussion	63
3.4	Summary	66
4	FPGA Characterization for Subsystem Mapping	69
4.1	Introduction	69
4.2	Methodology	72
4.2.1	Footprint Generation for Heterogeneous FPGA Architectures . . .	72
4.2.2	Parameters to Model Target FPGA	74
4.2.3	Problem Statement	76
4.2.4	Generating Training Data	76

4.2.5	Implementation of FPGA Characterization	78
4.3	Performance Evaluation	83
4.3.1	Benchmark Applications and Target FPGAs	83
4.3.2	Evaluation of Training and Testing Phases	83
4.3.3	Discussion	88
4.4	Summary	89
5	Target Architecture-aware Subsystem Mapping	91
5.1	Introduction	91
5.2	Methodology	95
5.2.1	Inter-subsystem Communication-aware Subsystem Grouping	99
5.2.2	Mapping Subsystems	107
5.2.3	CAD Tools Assisted Mapping of Subsystems onto Target FPGA	115
5.2.4	Performance Evaluation	118
5.3	Subsystem Locking for Rapid Convergence	123
5.3.1	Identify Subsystem for Locking	126
5.3.2	Footprint List of a Locked Subsystem	127
5.3.3	Performance Evaluation	128
5.4	Summary	130
6	Framework for Design Space Exploration	131
6.1	Introduction	131
6.2	Methodology	133
6.2.1	Implementation of the Framework for Subsystem Mapping	135
6.2.2	Performance Evaluation	139
6.3	Subsystem Splitting for Optimal Placement	143
6.3.1	Splitting Subsystem by Subsystem Re-generation	144

6.3.2	Performance Evaluation	145
6.4	Design Space Exploration	151
6.5	Summary	156
7	Conclusions and Future Work	157
7.1	Conclusions	157
7.2	Future Research Directions	162
	References	165

List of figures

2.1	Growth in Logic Cells in Xilinx Virtex Devices	8
2.2	Floorplan of Intel(Altera) FPGAs	11
3.1	Compiling Large Application Using State-of-the-art CAD	40
3.2	Typical Design Hierarchy of an RTL Implementation	42
3.3	Different Communication Activities in Applications	44
3.4	Converting an Input Design into a Graph	46
3.5	Connections of an Application in Different Levels	48
3.6	Technique to Extract Communication in RTL Implementation	50
3.7	Execution of the Subsystem Generation Algorithm	55
3.8	Communication-aware Subsystem Generation Methodology	57
3.9	Comparison Strategy for Power and Energy	60
3.10	Structure of Large Applications Consists of Benchmark Applications	61
3.11	Power Reduction with Subsystems	64
3.12	Average Power Reduction with Subsystems	65
3.13	Energy Reduction with Subsystems	65
4.1	Performance (F_{max}) of a Subsystem in Three Shapes	71
4.2	Footprint variation for a given subsystem	73
4.3	Training Data	76

4.4	Generating Footprints for Training	77
4.5	ANN Training Methodology	79
4.6	Use the Characterized Model for F_{max} Estimation	81
4.7	MatLab View of the Neural Network	82
4.8	Training Error Trend	85
4.9	Test Error Percentage Histograms	87
5.1	Communication-aware Subsystem Generation	92
5.2	Mapping Subsystems on FPGA Without Placement Information	93
5.3	Different Shapes and Locations for Mapping Subsystems	95
5.4	Two Step Approach to Subsystem Mapping	98
5.5	Modeling Subsystem Mapping as a TSP	101
5.6	Shape of Subsystems	103
5.7	Congestion Model	105
5.8	Example of a Solution from Subsystem Grouping Technique	106
5.9	Building a Complete Graph for Groups	111
5.10	Implementation of Subsystem Mapping Methodology	117
5.11	Routing Power Reduction Over the Methodology in Chapter 3	120
5.12	Effect of Grouping Technique on Routing Power	122
5.13	Routing Power Reduction due to Subsystem Mapping over Quartus	122
5.14	Effect of Grouping Technique on Performance (F_{max})	123
5.15	Suboptimal Mapping Solutions	124
5.16	Subsystem Locking	125
5.17	Footprint List for Merged Footprints	128
5.18	Effect of Subsystem Locking Technique on Routing Power	129
5.19	Routing Power Reduction due to Subsystem Locking over Quartus	129
6.1	Variation of Compilation Time w.r.t. to Application Size	132

6.2	Framework for Subsystem Mapping	134
6.3	Increase in Compilation Time with Application Size	141
6.4	Comparison of Routing Power Dissipation	141
6.5	Routing Power Reduction over Quartus	142
6.6	Comparison of Performance	142
6.7	Unused FPGA Resource During Subsystem Mapping	143
6.8	The Integrated framework with Subsystem Splitting	146
6.9	Increase in Compilation Time with Application Size	147
6.10	Comparison of Routing Power Dissipation	148
6.11	Routing Power Reduction over Quartus	148
6.12	Comparison of Performance	149
6.13	Integrated Framework Assisted Design Space Exploration	150
6.14	Design Space Exploration for Synth1 Application	152
6.15	Design Space Exploration for Synth2 Application	153
6.16	Design Space Exploration for Synth3 Application	154
6.17	Design Space Exploration for Atax' Application	155
6.18	Design Space Exploration for Bicg' Application	155

List of tables

3.1	Performance (F_{max}) Improvement	64
4.1	Benchmark Applications Used for Characterization and Testing	84
4.2	Target FPGA Architecture Characteristics	84
4.3	Training Error	85
4.4	Time Taken for Data Collection and Training	85
4.5	Time Taken for Testing	86
4.6	Test Error Percentages	88
4.7	Percentage Variation of F_{max} on Footprints	88
5.1	Routing Power Reduction over Quartus CAD Flow	120
5.2	Runtime Analysis of ACO Algorithm	121
5.3	Routing Power Reduction with Subsystem Locking	128
6.1	Test Application Details	139
6.2	Comparison of Compilation Time	140
6.3	Compilation Time for Extreme Cases	140
6.4	Comparison of Compilation Time	147
6.5	Design Points for Synth1	152
6.6	Design Points for Synth2	152
6.7	Design Points for Synth3	153

6.8	Design Points for Atax'	154
6.9	Design Points for Bicg'	154

Chapter 1

Introduction

1.1 Motivation

The continuous technology scaling of the transistors has enabled Field Programmable Gate Array(FPGA) vendors to integrate of logic cells and diverse set of hard Intellectual Property (IP) cores on latest high-density heterogeneous FPGAs. For example, leading FPGA vendors offer devices such as Xilinx Virtex UltraScale+ and IntelFPGA Stratix 10 constituting as many as 3.78 million [1] and 5.5 million [2] logic cells together with hard IP cores such as Digital Signal Processors (DSPs) and Block Memories(BRAMs) in various sizes. On the other hand, the advancements in the process technology has resulted in a significant increase in the non-recurring engineering (NRE) costs. At the same time, time-to-market (TTM) pressures continue to be increasingly stringent. Hence, designing embedded systems with conventional Application-specific Integrated Circuits (ASICs) is becoming a riskier proposition.

As compared to ASICs, FPGA-based designs incur lower NRE and TTM costs, which is favorable specifically in markets with high uncertainty. In addition, the abundance as well as the diversity of resources available on latest FPGA chips has encouraged hardware designers to port increasingly large applications on FPGAs [3]. As a result, FPGAs have become a

popular choice for not only traditional embedded systems applications but also for relatively new applications such as deep neural networks, artificial intelligence, cloud services and acceleration-as-a-service where, in the past, only Graphical Processing Units (GPUs) were typically used. Further, recent research has shown that FPGAs can even outperform GPUs for certain tasks such as the ones where workloads involve irregular parallelism and custom data types [4].

Even though FPGAs offer a promising alternative for a growing spectrum of applications, the design process is cumbersome and time consuming [5]. The accompanying CAD tools provided by the FPGA vendors have improved tremendously over the years, reaching to the point where they produce very high Quality-of-Result (QoR) for small and medium scale applications [6]. Even so, these tools have not kept pace with the tremendous growth in the FPGA hardware capabilities (millions of logic cells and diverse set of IP cores), resulting in mapping solutions with low QoR in terms of both performance (measured by maximum operating frequency - F_{max}) as well as power for large applications. Compilation of large applications require considerable amount of manual intervention and design expertise in addition to the efforts needed for Hardware Description Language(HDL) based design and verification. In addition, compiling large applications using existing state-of-the-art CAD tools incurs a prohibitively long compilation time, effectively negating the ability to provide lower TTM using FPGAs. Several studies have shown that compiling large applications on such tools could take hours or days and even fail to compile for very large applications [7][8][9].

Therefore, in this research, methodologies were explored to complement the current FPGA CAD tools by proposing techniques for compiling large applications in a deterministic time while improving both performance and power efficiency.

1.2 Research Objective

To develop a methodology for improving the performance of current FPGA Computer-Aided Design Tools for mapping large applications by proposing techniques for:

Subsystem Generation: A technique to partition a large FPGA application into smaller subsystems. This technique is intended to minimize power dissipation on FPGA interconnect fabric that occurs due to data-flow between different parts of applications.

Subsystem Grouping: A technique to identify the association of generated subsystems. This process aims to identify the connection between subsystems and grouping them accordingly in order to minimize the power dissipation due to data-flow between subsystems, while the interconnect delays are intact.

FPGA Characterization: A technique to characterize FPGAs. This offline technique should develop a device model to estimate the performance of subsystem mapping on the FPGA.

Subsystem Mapping: A scalable and power-aware technique to map subsystems on target FPGA while utilizing the heterogeneity of the FPGA and managing possible routing congestion.

Framework for Design Space Exploration: A Framework which integrates the techniques for generating power and performance efficient mapping solutions for large application in low runtime. It should facilitates the user to observe different subsystem mapping solutions without requiring design expertise.

1.3 Organization

The rest of the thesis is organized as follows.

- **Chapter 2** presents a detailed discussion of the existing literature. First, it discusses the heterogeneous FPGAs available in the market and the trend in FPGA based applications. Different steps involved in FPGA CAD flow, which maps such applications into FPGAs are discussed and compared, and the scalability of the placement CAD step is further studied. Next, the techniques involved in power dissipation on FPGAs is discussed. Comparison on ways the FPGA power dissipation occurs, the focus is moved for dynamic power reduction using CAD techniques. The study also extends to discuss techniques used for minimizing CAD runtime.
- **Chapter 3** proposes a technique for partitioning large applications into smaller subsystems so that the mapping process can exploit existing CAD flows, despite their inefficiencies in mapping large applications. The method considers data-flow between different parts of the application. It minimizes the distance of high data-flow, and lower power dissipation on interconnect fabric.
- **Chapter 4** discusses a novel technique to characterize an FPGA in order to estimate the performance of a subsystem during mapping. The technique requires many training data for the characterization. Therefore, an automated training process is also proposed. This technique models the effect of FPGA architecture heterogeneity for the performance (F_{max}) of a given subsystem, and it accurately estimates the performance of the subsystem mapped on different locations and shapes on FPGA.
- **Chapter 5** proposes a scalable methodology in mapping subsystems into target FPGA. The technique finds the best location and floorplan shape on the FPGA for each subsystem mapping, to utilize the heterogeneous resources and resource arrangement

available on modern FPGAs. In order to achieve scalability, the technique comprises of a subsystem grouping technique that identifies the subsystem association that ideally should be mapped at close proximity on target FPGA, that drives the floorplan locations and shapes decision making accordingly. A subsystem floorplan merging technique and a selective subsystem regeneration technique are also incorporated into the methodology to ensure better convergence.

- **Chapter 6** presents a framework that integrates the techniques proposed in this thesis. The framework produces a subsystem mapping solution for any given application on the target FPGA. It also allows users to explore the different mapping solutions by modifying the criteria for generating subsystems.
- **Chapter 7** concludes this thesis by summarizing the contributions made in this research followed by directions for future research from this work.

1.4 Publications

The work presented in this thesis has been communicated in international conferences as follows:

- [C-1] **Kalindu Herath**, Alok Prakash, Jiang Guiyuan, and Thambipillai Srikanthan. "Communication-aware partitioning for energy optimization of large FPGA designs.", In Proceedings of the on Great Lakes Symposium on VLSI, 2017.
- [C-2] **Kalindu Herath**, Alok Prakash, Udaree Kanewala, and Thambipillai Srikanthan. "Communication-aware Module Placement for Power Reduction in Large FPGA Designs." In 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2018.

- [C-3] **Kalindu Herath**, Alok Prakash, and Thambipillai Srikanthan. "Performance Estimation of FPGA Modules for Modular Design Methodology Using Artificial Neural Network." In International Symposium on Applied Reconfigurable Computing, 2018.
- [C-4] **Kalindu Herath**, Alok Prakash, Jiang Guiyuan, and Thambipillai Srikanthan. "Ant Colony Optimization based Module Footprint Selection and Placement for Lowering Power in Large FPGA Designs." In 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2018.

Chapter 2

Literature Review

2.1 Introduction

Embedded systems market continues to grow at a rapid pace. It has been predicted that this market will reach US\$116.2 billion globally by the year 2025 [10]. However, realizing embedded systems on traditional Application-specific Integrated Circuits (ASICs) has been significantly affected by high Non-recurring Engineering (NRE) cost and increasingly strict Time-to Market (TTM) pressures[11]. On the other hand, modern Field Programmable Gate Arrays (FPGAs) incorporate millions of logic cells and along with Intellectual Property cores (IP cores) such as Block Memories (BRAMs) and Digital Signal Processors (DSPs) [12][13]. This abundance and diversity of resources coupled with lower NRE and TTM has made FPGAs a viable alternative for the ever-changing as well as demanding embedded applications.

The scale of the applications to be mapped on FPGAs has also grown with the capacity of FPGAs. FPGAs are used not only for conventional embedded systems applications but also large-scale applications such as neural networks and data centers[14][15]. However, the rapid scaling of FPGAs and applications have overtaken the advancements in the accompanying FPGA Computer-aided Design (CAD) tools. As a result, CAD tools tend to produce

Fig. 2.1 Growth in Logic Cells in Xilinx Virtex Devices [16]

sub-optimal solutions during mapping process, i.e. degraded performance, higher power consumption, especially for larger embedded applications [6]. In addition, CAD tools also take a very long runtime to compile such applications, which eventually negates the advantage of low TTM in FPGA based designing [7].

This chapter reviews the landscape of modern FPGA architecture, and applications and domains targeted on FPGAs. Existing state-of-the-art academic and commercial FPGA CAD tools have also been explored in detail to identify their strengths as well as their limitations. The rest of this chapter is organized as follows. Section 2.2 explores the mapping process of modern applications on FPGA using existing CAD tools. Section 2.3 discusses existing efforts towards improving power consumption on FPGAs. Next, section 2.4 discusses the runtime of CAD tools for compiling applications for FPGAs while section 2.5 concludes this chapter by summarizing the limitations of the existing work.

2.2 Mapping Applications on FPGA

2.2.1 Modern Heterogeneous FPGAs

FPGA architecture comprise of four main components [17]. Programmable Logic Blocks or Configurable Logic Blocks (CLBs) are the primitive computation elements in FPGAs which holds the basic logic and storage functionality. Modern FPGAs also include other components called Hard Intellectual Property cores (IP cores) such as embedded memories (Block RAMs), Digital Signal Processors (DSPs), external memory controllers, etc. FPGA chips are also equipped with Input/Output (I/O) pins which facilitates external interaction. In addition to these components, Programmable Interconnect Fabric gives the interconnecting medium for FPGAs. A user defined design could be implemented by desirably interconnecting CLBs, Hard IPs and I/O pins.

Over the past years, the immense scaling in the silicon has led the commercial FPGA vendors to produce FPGA chips that consists of billions of transistors[12][13]. For instance, the figure 2.1 compares the number of logic cells available on Xilinx Virtex family FPGA devices in 35mm x 35mm Ball Grid Array (BGA). The figure lists several FPGA devices from each Virtex family. Virtex 4 device family [18] was introduced in 2004 on 90nm process technology and Xilinx released its 28nm Virtex 7 family [19] in 2010. During this period, the logic cells available on each FPGA devices has rapidly increased over 4 times. However, the largest available Xilinx Virtex 4 device consists of 200k logic cells whereas the Virtex 7 counterpart equipped with 1955k logic cells, which is about 10 times increase. Xilinx Virtex Ultrascale [13] on 20nm is known as the company's high-capacity and high performance FPGA family to-date, which features FPGA devices with logic cells upto 3780k. Similarly, Intel FPGAs offer FPGAs with tremendous capacities. For instance, Stratix V [20] and Stratix 10 [12], the high capacity FPGA families offers FPGAs upto 952k and 2073k logic elements respectively. Besides logic cells capacity, modern FPGAs also equipped with hard

IP cores such as BRAMs and DSPs in large numbers [12][13]. One of the latest Xilinx FPGA series, Virtex Ultrascale [13], has included up to 132 Mega bits of BRAM memory capacity whereas Virtex Ultrascale+ consists over 12k DSP slices. These devices demonstrate the advancement in FPGA Hard IP core capacity, especially when it is compared with 10 kilobits of BRAM and 192 DSP slices. Likewise, Intel's high capacity FPGA family, Stratix 10, includes 229 Mega bits of memory capacity and 11.5k DSP slices.

Whilst the resource capacity of modern FPGAs has grown rapidly, FPGA vendors have made the FPGA resource arrangement on the chip floorplans more diverse in order to cater various resource requirements from modern applications [21]. Such floorplans are often called *Heterogeneous FPGA Floorplans*. Heterogeneity is achieved by having different variations of on-chip FPGA resources and by arranging them in irregular column positions. In relation to different resources, FPGAs in Xilinx Ultrascale+ devices consist of two different types of BRAMs; normal Block RAMs and Ultra RAMs [22]. The former type is the common 36 kilobits BRAM type among all Xilinx devices, whereas, the latter is 288 kilobits larger BRAM type which allows users to implement larger RAM arrays. Similar concept was used in Intel Stratix IV devices [23], which includes two types of BRAMs; M9k BRAMs with 9 kilobits capacity and M144k with 144 kilobits capacity. DSP slices on FPGAs, however, are identical in the fundamental structure across each device. But these DSPs provide different modes of operations and therefore are considered as heterogeneous. For instance, DSP slices in Intel Stratix 10 devices (called Variable Precision DSP Blocks) can be configured as two 18x19 bit multipliers or single 27x27 bit multiplier.

The arrangement of resource columns in modern FPGA floorplans is more irregular in fashion, as compared to regular resource column arrangements in legacy FPGA devices [24][20]. For instance, figure 2.2 shows examples for floorplans of legacy and modern FPGA devices. Resource columns in the modern device serves different permutations of column arrangements in different column gaps. In the Cyclone II device in the figure 2.2, resource

Fig. 2.2 Floorplan of Intel(Altera) FPGAs (a) Cyclone II Device - Legacy [24] (b) Stratix V Device - Modern [20]

columns are fabricated in equidistant style with minimal irregularity. Stratix V device, on the other hand, exhibit the heterogeneity as described above which ultimately provide column arrangement for robust and efficient application mapping on FPGA with numerous demands.

2.2.2 Trend in Modern FPGA Applications

Over the years, FPGAs have acquired a great interest in computer and embedded system applications, predominantly because of their reconfigurability [25], low NRE cost and TTM. However, the rapid growth in resource availability in FPGA chips has attracted non-conventional markets in recent years [26]. In this section, the demand for FPGAs in various domains is explored.

Currently, FPGAs are widely used in various industries such as aerospace [27], automotive [28] [29], broadcast[30], medical[31], wired and wireless networking [32] and ASIC prototyping [33]. Since the FPGAs provide large amount of BRAMs along with thousands of DSPs, FPGAs have been extensively used in image, video and digital signal processing and compression domains [34]. With the availability of modern high capacity FPGAs, hardware

designers have developed high-end signal processing and compression applications to deliver higher throughput and compression [35]. In addition, Apple has included a FPGA powered dedicated video editing card in their latest Mac Pro, called Afterburner [36]. This accelerator is capable of processing up to 6 billion pixels per second, hence allowing user to process high resolution encoding and decoding.

FPGA devices with multi-million logic cells and various IP cores as discussed in section 2.2.1 have motivated hardware engineers to port increasingly large applications in both conventional and modern application domains as follows. FPGAs have been used in implementing many artificial neural network applications in various domains. TuRF [14], a framework which finds efficient convolutional neural network (CNN) model of a given application, has been used for porting CNNs with DSP slice requirements of 1k to 2k. A similar work in [37] also has used CNNs with very large DSP requirement as much as 2.8k. Authors have shown that the speed-up achieved by the FPGA implementation is over 2x as compared to the speed-up achieved by GPUs. However, the FPGA implementation is possible only if the FPGA capacity is over 300k LUTs, 1k BRAMs and 2.5k DSPs. High performance and energy efficiency with respect to GPUs in similar FPGA based large scale neural network applications [38] [39] [40] and deep learning applications [41] indicates that FPGA based NN will be widespread in future. As machine learning applications are widespread in recent computing, FPGA and GPU vendors have been competing to provide fast and efficient devices [42].

Intel's largest ever acquisition of one of the leading FPGA vendor[43], Altera, indicates that the FPGAs will be dominating in acceleration services in the future [44]. Intel have released processors with integrated Altera Arria 10 FPGAs [45] and Intel FPGA Programmable Acceleration Cards (PAC) [46] with Altera's high end Stratix FPGAs in order to accelerate machine learning, video transcoding, big data analytics, financial computation, database acceleration, real-time analytics, artificial intelligence workloads [46]. In addition, FPGAs have

gained significant popularity in data center applications. Microsoft Project Brainwave [15] is a hardware platform which is designed to accelerate deep neural network (DNN) inferencing uses large arrays of Intel FPGAs (Altera). It has enabled Microsoft to deploy a cloud based machine learning accelerator model, Microsoft Azure [47], allows real-time AI calculations in various domains. Similarly, Amazon and Baidu have also demonstrated that FPGAs will take over data centers, with Amazon Web Services Elastic Compute Cloud (AWS EC2) [48] and Baidu FPGA cloud server [49]. Google has also unveiled its application-specific integrated circuit, Cloud Tensor Processing Units (TPUs) [50], to accelerate machine learning workloads. However, FPGA based acceleration-as-a-services, for instance, Microsoft, turning to be more attractive than ASICs based TPUs because of FPGA's ability to quickly adapt silicon implementations as algorithms evolve [51]. On the other hand, irregular parallelism in AI applications makes FPGAs better than GPUs which currently held the lead in accelerating parallel work loads [4].

Advanced Driver Assistance System (ADAS) market shows a 12.24% Compound Annual Growth Rate (CAGR) and is expected to reach notable market value during 2019 and 2023 due to growing demand for safety features in automobile [52]. It is predicted that, there will be 45 million vehicles with at least one installed ADAS camera by 2020, and 25% of vehicles will be connected to the Internet by 2025. In such growing domain, FPGAs are becoming popular because of few reasons. ADAS applications and algorithms changes continuously, hence, the reconfigurability provides greater support to facilitate design changes [53]. On the other hand, ADAS also involves processing real-time processing of numerous sensor data such as radar, image, laser and ultrasound [54]. Capability of performing high performance and low latency computations, as well as performing simultaneous workloads, make FPGAs a feasible solution for ADAS application needs. On the other hand, massively connected ADAS network emerges the necessity of security of the system against cyber attacks. Commercial vendors, therefore, provide secure programmable FPGAs as well [55].

2.2.3 Compiling Applications into FPGAs

Typical FPGA CAD tools take an application coded in a Hardware Description Language (HDL) like VHDL or Verilog as the primary input, along with design constraints and specification of target FPGA device, and it compiles the application into a programmable bitcode [56][57]. CAD flow includes few processes as described follows. *RTL elaboration* identifies data-path operations, such as additions, multiplications, memory blocks, and control logic, and those operations are elaborated into generic Boolean networks. Most of the data-path elements, such as adders with dedicated fast-carry chains and embedded multipliers, have special architectural support in modern FPGAs. Both data-path and control logic are optimized during *architecture-independent optimization*. These two steps generally called the synthesis process, which converts the HDL code to a architecture-independent netlist logic gates and interconnects. During the *technology mapping*, the optimized gate-level netlist is mapped to a functionally equivalent netlist, which is expressed in terms of Basic Logic Elements (BLEs) provided by the target FPGA architecture.

As a further optimization step, the *clustering process* groups the BLEs into netlist of clusters which can be fitted into LUTs on the FPGA[57]. The physical location of each netlist cluster in the FPGA layout is determined during the *placement*, and then the routing resources that are needed to connect the placed logic blocks are determined by the *routing* step. Finally, a stream of configuration bits is produced and is used to program SRAM cells in the FPGA device.

2.2.3.1 Technology Mapping

Technology mapping step of the CAD tools is responsible of converting a netlist of registers and logic gates in to a netlist of k-input LUTs and flip flops [57]. Typical technology mapping algorithms divide the gate and register netlist into groups and then determine the best representation of LUTs. A logic gate and register netlist can be represented as a directed

acyclic graph (DAG), where a node represents a logic gate and an edge represent a connection from an output of a logic gate to a input of another. The DAG is then divided into cuts, where each cut has number of inputs less than K and a single output. Delay of a FPGA circuit depends on two factors; response time of an LUT and the delay in interconnects. In the technology mapping stage, the interconnect details are not available. On the other hand, the delay within the LUT is generally a constant and the delay of a cut is relative to the depth of the graph. Technology mapping algorithms whose objective is to optimize delay [58], therefore minimize the depth. Similarly, by reducing the number of cuts in the main graph and by reducing the number of edges [59] in the solution cuts, area optimized and routability optimized solutions are made respectively.

2.2.3.2 Clustering

In the clustering stage, a netlist of LUTs are grouped together to make a netlist of CLBs [57]. Typical clustering algorithm aims either to minimize the area by reducing the number of CLBs in the final solution, or to minimize the delay by packing the critical path to shared CLBs.

VPack [60] is an early yet a major FPGA clustering algorithm, whose objective is to minimize the number of clusters in the final solution. The algorithm makes one cluster at a time, and on each iteration it chooses a logic cell which is not yet clustered and has the most number of used inputs. The chosen logic cell is then clustered and its suitability to the cluster is determined by sharing of inputs and outputs between added logic cell and the cluster. A logic cell with maximum number of shared input and output with the cluster is finally added to the cluster, provided the addition of new logic cell to the cluster does not violate any other constraints. This addition of logic cell to the current cluster is continued until there are no logic cell to be added without violating constraints. This process is repeated for all the clusters sequentially. T-VPack [61] is slightly enhanced clustering algorithm which optimized

the timing than area. T-VPack follows the same steps as VPack except the suitability of addition of a logic cell in to a cluster is evaluated by timing criticality of the connections. The modified algorithm defines three types of delays; logic cell delay, intra-cluster delay and inter-cluster delay. Since inter-cluster delays are much bigger than the intra-cluster delays, T-VPack tries to reduce inter-cluster connections on the critical path. To calculate the timing criticality, T-VPack uses a delay model, but it is not accurate as the routing information is not available at the clustering stage. Combined clustering-placement algorithms are developed to minimize the effect of incorrect delay models. T-VPack, however, can be fine tuned to have customized degree of area optimization and timing optimization.

2.2.3.3 Placement

After obtaining a netlist of CLBs, placing the CLBs on the physical target FPGA architecture is the next step of the CAD flow [57]. The common optimization goals in a placement algorithm are performance, routability, wirelength or power [62]. There can be some other minor optimization goals such as thermal distribution [63]. In CAD tools, placement, is often combined iteratively with routing step, plays a major role in delivering intended quality to the final outcome as it decides the usage of programmable switches and global interconnects, hence the circuit delay and wirelength. There are three major types of state-of-the-art FPGA placement algorithms; simulated annealing based, partitioned based and analytical.

Simulated Annealing Placement Simulated annealing optimization algorithm searches for a global optima in a large solution space such that it mimics the physical annealing process [64]. During annealing, materials are heated into high temperature and then allowed to be cooled. When the material is at a high temperature, the material molecules absorb large amount of energy and move freely in the material. Once the material is subjected to a slow cooling process, the energy in the molecules also is decreased and molecules moves slower, but they form a crystalline structure which is more stable than the original

configuration of the material. The cost function of the simulated annealing algorithm is similar to the energy in annealing. Cost of a best known solution gradually decreases with the iteration of the algorithm, and the algorithm finds the solution with the minimum cost in each iteration. The current solution is subjected to have small changes, called perturbations, and it resembles the movements in the particles. Simulated annealing algorithm perturbs several times per iteration, in order to explore more solutions in the solution space. The current solution is updated when there is a better one. But at times, a worse solution can be accepted with a higher probability if the temperature is higher, or with a less probability otherwise. Temperature in the simulated annealing is the control parameter, and is gradually reduced in each iteration. The algorithm stops when the temperature reaches to a predefined amount. This algorithm is popular due to its attractive properties. Simulated annealing has non zero probability for accepting a worse solution, which allows the algorithm to get rid of local minima unlike the greedy algorithms. The runtime of the algorithm is also adjustable by deciding the temperature schedule. The user can always get a solution at each iteration, but for a better quality of the result, the runtime should be longer.

Simulated Annealing optimization algorithm is used in a well-known academic placement tool, VPR [62]. VPR has evolved to accommodate different optimization goals such as circuit delay and power. But the original cost function aims to optimize the wirelength. In [65], the VPR cost function is modified to consider the timing criticality of nets. The aim of the new cost function is to reduce the delay of connections on the critical path, and to allow the delay of non-critical connections to be increased. First, the delay of each nets is modeled during the placement and a new parameter, criticality, is defined to indicate the delay. Here, F_{max} is the modeled critical path delay and the slack which represents the amount of delay that can be added to the connection without exceeding the critical path delay. The timing cost of a connection is then defined as a multiplication of criticality and the delay. However, the final cost function considers both wire length and timing criticality by adjusting a weight

parameter to have a good trade-off between critical path delay and wire length. With all these improvements to the basic approach, VPR shows superior critical path and wirelength quality over many other placement paradigms. However, with the increased design complexity of placement problem, VPR takes exponentially high runtime to produce a good quality result.

Quartus [66] is the FPGA CAD tool developed by Intel (Altera) for their designs. The placement tools of Quartus tool, Q2P [67] is a simulated annealing approach and is made on top of VPR. But Altera claims that they have improved the VPR algorithm by changing the characteristics [8]. In conventional simulated annealing, blocks are moved randomly and the cost of the movement is the evaluated. This process is improved in Q2P, such that the selection of the blocks and the destination location is done partially guided manner. This concept is called directed movement of blocks and it has improved the efficiency in exploring the search space. The cost function of VPR is also improved in Q2P. It uses eight linearly combined costs to optimize wirelength, timing, power and localized congestion. Another technique used in Q2P in improving the placement solution is called cluster decomposition. Both VPR and Q2P uses a clustered netlist as the input, and the clusters are placed appropriately. But as [68] suggest, the partition algorithm can follow a multilevel clustering as well. Q2P majorly moves entire clusters during the placement process, but it also can decompose clusters and move component LUTs during the later stages in the placement process. This allows Q2P to make finer-tuned placement, but it comes with a significant runtime cost. However, with all the finer improvements done to VPR base algorithm, Q2P claims that it gives better results in wirelength and critical path as compared to VPR. Though simulated annealing is often believed to scale more poorly than other approaches, [8] also shows that Q2P scales well to larger problems. While Q2P shows superior runtime with better solution quality than VPR in larger problems, [8] claims that VPR could not achieve a better solution than Q2P even if it was allowed to have a 10x more runtime than Q2P. However, there have been many attempts

to improve the scalability and runtime of simulated annealing algorithms which are discussed in next sub-sections.

Unlike ASIC, FPGA consists of prefabricated routing wires [17]. Since these routing resource density is limited on FPGAs, it is necessary to do a careful matching of routing resource availability and design complexity during the placement. A proper congestion analysis can contribute to achieve spatial uniformity in routing resource utilization, and consequently can lead to overall device de-congestion after routing, and savings in area, power, and routing time [69]. Rent's rule [70], is commonly used for congestion analysis. It states that the relationship between the number of pins or external signal connections of a logic block, and the number of logic blocks in a circuit forms a straight line in a log-log plot. The exponent used in the Rent's rule formula is called Rent's exponent, and could be used to characterize the routing resource utilization. The approach in [71], uses Rent's exponent to characterize the routing resources in a given FPGA architecture (p_a) and routing resource utilization of a design (p_d). For every placement, the algorithm evaluates the difference between Rent's exponents $|p_d - p_a|$ and is used to scale the bounding box lengths in the cost function. The basic idea of this scaling is to accommodate more area for a block so that there will be more routing resources to serve the congestion. [69] and [71] achieve a reduction in total routing area and wirelength, and thus achieve better power consumption due to lowered active and inactive interconnects.

In addition, the proposed algorithm in [72] introduces a new bounding box cost function for a routability aware placer. The final bounding box cost is achieved by multiplying the general simulated annealing wiring cost and a new heuristic called congestion coefficient. Authors argue that each net in a placement will require routing tracks around the CLBs it covers. If a CLB region covered by bounding boxes of many nets, the routing tracks placed in that region could be a bottleneck. Therefore the congestion coefficient is proportional to the number of bounding boxes overlapped on each CLB on the chip, and penalizes the

moves resulting higher congestion. However, this approach lacks a method to determine localized congestion. A diffusion based routability driven placement approach is introduced in [73] with similar cost functions. Local congestion cost in this approach, is determined by the ratio of occupancy of two CLB to the distance between the two. Here occupancy refers to the number of connections a CLB has with adjacent routing channels. The algorithm first does the placement and routing process with standard wirelength driven placement cost function and then occupancy of all used CLBs is evaluated. After that, another place and route process is done with an additional cost term for congestion, while penalizing swaps which result increment in congestion cost. However, this technique requires multiple runs of the algorithm, and also a search through all the CLBs used. Therefore, for a large design the placement process will require a longer runtime.

Partition based placement The partition based placement problem is to map a given circuit which is represented as a hypergraph to some specific location in the physical FPGA region [74]. The iterative placement process starts by partitioning the given hypergraph into two sub-graphs (bi-partitioning) P1 and P2. The FPGA placement region is also bi-partitioned by a cutline into two subregions F1 and F2, and the subregions F1 and F2 should contain more FPGA resources than the partition P1 and P2 required. The partitions P1 and P2 are then assign to the physical subregions F1 and F2, respectively. The orientation of the cutline is decided by a cost function and the hypergraph is partitioned using an algorithm like Fiduccia-Mattheyses (FM) algorithm [75]. Further restricted circuit placement is achieved by recursively bi-partitioning each subgraphs and subdividing the physical regions. A physical subregion is assigned to respective subgraph in each recursive iteration. This process continues until each sub-circuit contains a few modules that are assigned to a small subregion.

Basically, a partitioning algorithm like FM algorithm attempts to make a cut on a graph such that the cost of the cutting is minimum. In each iteration, the algorithm greedily moves vertices from one partition to another and evaluate gain for the best cut line. The

FM algorithm works fine for the hypergraph which contains only about few hundreds. For the hypergraphs with beyond hundreds of vertices, a multilevel partition scheme hMetis [76] is suitable in terms of quality of the result and the runtime of the algorithm. The multilevel scheme consists of three stages; coarsening stage, initial partitioning stage, and un-coarsening stage. Starting from the hypergraph, heavily connected vertices are merged to make a coarser hypergraph in the coarsening stage. After several levels of coarsening, the original hypergraph is reduced to few vertices with smaller hyperedges. Then at the initial partitioning stage, the coarsest hypergraph is bipartitioned, randomly (as in hMetis) or following any bipartitioning algorithm. Finally, during the un-coarsening stage, a partitioning in the coarser hypergraph is used to obtain the next finer level hypergraph. FM algorithm is successively used in this stage to refine the partition cut.

Partitioning based placement for FPGA (PPFF) [74] is a key placement method following this technique. Main aim of PPFF is to achieve a faster and more scalable placement method as compared to simulated annealing. While PPFF follows a general portioning techniques as in hMetis [76], it optimizes the placement for delay and congestion. Delay minimization is done by having timing criticality weight on edges. Since in FPGAs, the number of wire segments travelled by a routed net plays a more significant role in the delay of the net than the traditional geometric distance, PPFF minimize the number of segments by a terminal alignment method. Congestion minimization is achieved by minimizing the cut-size of the netlist at different partition level, but the terminal alignment method also result a reduction in congestion. The idea is to align the terminals of the most critical nets during the recursive partitioning. Consider nodes X and Y of a two-terminal net. Assume that node Y is fixed as it is placed in previous iteration of the algorithm. Then it is more likely to use fewer routing segments to connect the two terminals X and Y, if the terminal X is placed vertically or horizontally in line with terminal Y. Experimental results demonstrate that the routing algorithm can effectively harness delay minimization opportunities created by terminal

alignment during placement. In addition PPFF determines the cut direction of a partition based on the timing criticality of the nets which cross the horizontal or vertical borders of the placement region. For example, if there are more critical nets cross from the vertical boundaries, make a cut on horizontal direction would be beneficial, as then the critical nets can be aligned. Finally, PPFF algorithm assign a delay weight value for each net during every partition step. The delay value of a net is modelled based on minimum span of the net in terms of number of partition to be crossed.

Generally partition based placements show degradation of solution quality, as partitioning avoids searching whole solution space, but achieve lower runtimes as compared to simulated annealing algorithms [56].

Analytical Placement Although the analytic placement methods have been used in ASIC designs with a great success [77], it is new for FPGAs. The analytical approach for placement mainly involves a step to formulate the cost functions as mathematical functions of coordinate values of modules. Then the placement problem can be converted to a mathematical problem, which can be solved using general mathematical or numerical techniques. Typical analytical approach consists of three stages; global placement, legalization, and detail placement [77].

In the global placement stage, the main target to place the nets (or macros), regardless of overlaps while minimizing a wirelength cost function. Half perimeter wire length (HPWL) is a primitive metric to evaluate cost. HPWL is calculated by the half of the perimeter of the smallest rectangle which can include all the connected nodes of a net. But it is found that the overlapping evaluation is difficult with HPWL model so that another metric called quadratic wirelength is commonly used. Quadratic wirelength metrics are basically derived by taking the square Euclidean distance between nets. Then the legalization removes the overlap by assigning coordinate position to each of the overlapping blocks with a possibility of increasing the total Wirelength. Finally, during the detailed placement a legal placement

solution is fine-tuned by reassigning some of the blocks onto new position for optimal performance.

2.2.3.4 Routing

Routing is a tedious yet important step in FPGA CAD, which aims to connect all the placed CLBs via the programmable routing fabric [57]. Routing on FPGAs is more restricted than routing in ASIC as the FPGA routing needs to be done on limited number of prefabricated routing resources such as wires, switches and multiplexers [56]. Routing process typically consists of three major steps; routing resource graph generation, global routing (optional), and detail routing.

A model of available prefabricated routing resources is necessary [78]. In general, a given FPGA routing architecture is modelled using a graph technique, where the wires segments and the input and output terminals of logic blocks are represented as the vertices, and the switches are represented as edges. Routing resource graph is simplified in global routing, such that it represent the routing wire regions (routing channels) as vertices and switch boxes as edges, instead of representing detailed wire and switch information. Global routing determines the routing of each net on the simplified routing resource graph, such that the routing channel capacity constraints are not violated, and the timing constraints of all nets are satisfies. Once the global routing solution is obtained for all the nets, the detail routing determines the route in the detailed routing resource graph for each route in the simplified routing resource graph, so that the conflicts for routing resources are eliminated. Routing problem is a NP hard problem and it is usually solved in two stages as global and detail routing in order to handle the complexity of the problem.

Most successful routing algorithm reported in FPGA CAD is based on negotiation-based iterative global router in ASIC designing [79]. It is the base for the well-known FPGA routers called PathFinder [79] and VPR [62]. In this iterative approach, all the nets are routed in

each iteration by evaluating the minimum cost for each net. The cost evaluation is based on the current cost associated with vertices in the routing resource graph, and the routing decision for each net may cause over congestion in some routing channel. After performing routing for all the nets in one iteration, the vertices cost are readjusted based on the current congestion. In the subsequent iterations, all the nets are rerouted based on the new cost value, and as a result, the congested routing channels are eliminated or reduced. This rerouting process is continued for predefined number of iterations or until all the congestion costs are met. In addition to the congestion minimization, VPR cost function [62] contains additional term to penalize the bends in the route of a net, in order to reduce number of routing switches being used. Timing driven routing algorithms on the other hand considers the delay and the criticality of routs in their cost function. For example, VPR uses Elmore delay model to evaluate delay in their timing-driver cost function. The delay term and the congestion term are balanced using a weight term, timing-criticality.

2.2.4 Placement Techniques for Large Heterogeneous FPGAs

Simulated annealing placement algorithms are popular in FPGA CAD because of their superior quality of result [56]. However, when the modern FPGA applications become large, the runtime of simulated annealing placement degrades the design productivity of FPGA design flow [80]. Therefore alternative placement techniques and modification to the existing simulated annealing algorithms have been tried to build scalable placement algorithms.

There have been many attempts to parallelize the existing VPR placement algorithm to make use of multi cores in modern computing systems. For instance, [81] describes a parallel VPR implementation. Even though the placement algorithm can achieve large speedups over VPR, there is a degradation in performance. On the other hand, the authors in [82] shows parallelizing simulated annealing placement using Transactional Memory can achieve large speedups but with unacceptable quality of result degradation. Also they show

parallelization techniques with thread-level speculation do not scale well, but can preserve quality. Finally they explain a new parallel simulated annealing approach which balances speedup and quality of results, but it requires additional programming effort to implement mechanism for inter-thread communication. While there are research going on to parallelize placement algorithms used in academic tools like VPR, efforts have been made to parallelize commercial placement algorithms, for instance Q2P (Altera) [8], as well. Q2P placer involves many improvements in the serial version over VPR. Therefore parallelizing Q2P algorithm is said more challenging. One difference between [8] and other parallel approaches is that [8] has attempt to make the final placement solution equivalent to the serial solution. However, parallel Q2P placement algorithm does not show huge speedup without losing quality of result. A new placement strategy is introduced in the latest Quartus Prime Pro CAD flow, which combines analytical and simulated annealing placement techniques [83] [84]. An initial analytical solution is used for the simulated annealing flow, in order to have a balance between high quality placement and CAD runtime.

Many of the placement techniques which targets multi-million FPGA devices considers alternative approaches to pure simulated annealing techniques. [85] for instance, treats modules to be placed as rectangular areas, annotated with required number of CLBs, memory blocks and multipliers. The physical FPGA area is divided into smaller rectangular regions by recursively cutting a rectangle into smaller rectangles. For a given floorplan, which is represented by a set of smaller rectangles, the proposed algorithm evaluates the suitability in placing all the modules using a different technique than other bi-partition algorithms. A suitable floorplan is then treated as initial placement in simulated annealing, and saves the runtime by running simulated annealing algorithm lesser number of iterations. Approach described in [86] is also a partition based placement algorithm which targets multi-million FPGAs. [86] claims their approach is faster and improves the wirelength as well, but the evaluation has not been done on large size benchmark designs. On the other hand, [87] shows

the quality loss in such multi-million FPGA based hierarchical placement algorithm over non-hierarchical approach. The authors has discussed the variation quality of result based on number of partitions, and the partitioning methodology been used. In [6], authors explain an adaptation of ASIC placement algorithm, SimPL, in analytical placer for FPGAs. Their approach shows 4x average speedup over Altera's non-timing driven placement at a cost of 4% additional wirelength and 9% lesser maximum operating frequency.

Simulated annealing based placers have been avoided by the ASIC community due to its runtime. The work in [6] discusses three alternative paradigms used in ASIC and compares with VPR. The paradigms are, multi-level partition based placement, analytical placement, and scalable quadratic placement. Though the comparison with VPR is difficult, the authors has done a rough comparison. The partitioning-based and analytical placers has shown a comparable quality of result with VPR but with modest speedup around 1.6x to 6x. But for placement problems with larger cluster sizes, VPR outperform many high-quality ASIC placers. On the other hand, scalable quadratic placers achieve huge speedups over VPR but unacceptable degradation in quality of result. Authors also predict that the simulated annealing based placers will be in dominant use for few more FPGA device generations. In addition to these findings, [6] also demonstrate an improved quadratic placer which can drastically reduce runtime while reducing the quality of result gap. The authors has shown the results of the improved placer for large sized benchmark (200K clusters), which many proposed placers has not been evaluated with. However, their placer optimize the half perimeter wire length, and other important optimization metrics for FPGA should be considered.

2.3 Techniques for Power Efficient Mapping

2.3.1 Power Consumption of FPGAs

There are two ways of power dissipation in FPGAs; static and dynamic [88]. In SRAM based FPGAs, the static power dissipation occurs due to the leakage current. There are two types of leakage current: reverse biased PN-junction current and the current due to sub-threshold channel conduction. Power dissipation due to leakage current had been negligible in the past [88]. However, transistor scaling in modern FPGAs has necessitated the need to lower the threshold voltage. As a result, threshold voltage exponentially increases the sub-threshold channel current, leading to rapid increase in static power dissipation. It has been found that the static power dissipation in modern FPGAs is as high as about 30% of the total power dissipation[89].

Dynamic power dissipation happens due to the signal transitions in the FPGA [88]. Transitions causes the capacitive loads to be charged or discharged. On the other hand, high operating frequencies can make these transition more frequent and hence cause more dynamic power dissipation. Dynamic power is commonly modeled as in the following equation.

$$P = \sum_{i \in nets} \frac{1}{2} C_i \cdot \alpha_i \cdot V_{dd}^2 \cdot f \quad (2.1)$$

Here, capacitance and switching activity (toggle rate) of resource i , are noted as C_i , and α_i respectively whereas V_{dd} and f refers to supply voltage and operating frequency respectively.

Sources of power dissipation in an FPGA can be identified as logic, interconnect, clock network, configuration memory and input-output blocks. Several studies have attempted to estimate the power dissipation of each source. In [90], the authors show the estimated power result for Xilinx Spartan-3 FPGA device and the dynamic power is dominated by the routing. Routing resources consumes considerable amount from static power as well. Similar dynamic power breakdown is also reported in [88], and shows 60% of dynamic

power is contributed by the routing. As the Equation 2.1 depicts, the supply voltage has a quadratic relation to the dynamic power dissipation. The operating frequency of typical FPGA designs (F_{max}) is decided by the critical path. As a result, components in the entire design, including the paths with a positive slack, are forced to run on this frequency. However, as the delay of a gate is roughly inversely proportional to supply voltage V_{dd} [91], it is possible to reduce power dissipation by having lower V_{dd} for non-critical paths, without affecting overall performance of the system. This technique has been explored in fixed and configurable dual- V_{dd} architectures in [92][93]. SmartVID [94] is Intel FPGA's approach to scale V_{dd} based on the operating conditions. During the manufacturing stage, the lowest possible V_{dd} for each device is tested and stored in non-volatile registers. These values are used by the voltage regulators. Dynamic Voltage Scaling (DVS) is a voltage scaling technique that has been extensively used in commercial processors. However, DVS is challenging to implement on FPGAs due to its reconfigurability [95], and therefore has not been commonly used on commercial FPGAs. There are academic DVS schemes for FPGAs [96] [97], but these techniques are mostly application specific and device specific. Clock gating [98][99] is another dynamic power reduction technique. Here, the unused clock network is dynamically turned off, resulting reduction in effective capacitance in the clock network.

Accurate power modeling is necessary to evaluate new low power designs, to check whether designs meet power constraints. Modern CAD tools equipped with a power estimator tool, such as Altera PowerPlay [100]. During the early stages in the FPGA design flow, designers can have a power estimation using model based estimators like Altera Early Power Estimator [100] and Xilinx XPower [101]. These estimators take resource usage such as number of adders, multipliers, block memories, MUXes and programmable logic as input and evaluate power consumption using a database of different elements and respective power consumption. The advantage of this method is that designers are not required to place and route the design to do the power estimation, but however estimation is less accurate as the

switching activity and routing resource information is not considered. To obtain more accurate power estimation results, designers can place and route their designs. Switching activity information is then could be estimated vectorless switching activity estimation. However the most accurate power estimation is obtained by performing a gate-level simulation and observing all the nets in the design. Except the difficulty in performing a place and route, these accurate power estimators do not affect the design productivity.

2.3.2 Static Power Reduction

Static power reduction is usually achieved by FPGA device level changes [102]. FPGA vendors offer low power devices for users to balance their performance requirement and power budget. For example, Xilinx UltraScale offers devices operates at lower operating voltage [103]. Such devices are 35% low in static power dissipation, as compared to the devices operated in normal voltage. Similarly, Intel(Altera) offers Cyclone 10LP series [104] with low static power dissipation.

In typical applications, only a fraction of routing paths are timing critical. Therefore, the running entire FPGA chip leads high power dissipation. To mitigate this, Stratix device series introduces Programmable Power technology [105]. Here, the technology allocates high-speed mode and lower-power modes to the paths, by adjusting the back biased voltage of the transistors. Quartus CAD flow [66] identifies the slack in each path of the design and set the appropriate operating mode. In the low power mode, the back biased transistors is made more negative, making the transistor harder to turn on, and the opposite in the high-speed mode. It leads to reduce sub-threshold leakage current, hence reducing over 40% static power dissipation. Intel (Altera) uses various other architecture modifications, such as adding Embedded HardCopy blocks to double the FPGA capacity without doubling static power, and improvements to the process technology [89].

2.3.3 Dynamic Power Reduction

The dynamic power consumption of an FPGA could be lowered in two ways during the technology mapping [106]; minimizing the switching activity of the connection between LUTs or minimizing the total number of connection between LUTs so that the total capacitance is minimized. To minimize the switching activity in connections, approaches like [107] [108] identifies the high switching activity in the gate netlist and then make cuts such that those connections are kept within the cut. The cuts are finally represents LUTs so that this process could be identified as “hiding” the connections in LUTs. Generally, at the technology mapping stage, the exact runtime behavior of a circuit is not known, and therefore the signal activity discussed here is modeled. In [107] and [108] signal activity of a node output is modeled by a parameter called transition density and is derived by considering the rate of changing a signal of each logic function. On the other hand, approach in [109] uses the previous empirical studies which shows the signal activity decreases with the circuit depth. Depth optimization is common in technology mapping, which introduces another aspect called node duplication. While minimizing the depth, one node could be mapped in to two different cuts in the mapping solution. By duplicating the node into two cuts, the output edges of the considering node could be covered inside each cut. Therefore, the output connection from this node does not have a route in interconnect and does not contribute to the total interconnect power. However, the node duplication sometime can result an increase in total input connections to the cuts, and therefore increase more connections. In [12], the authors attempt to reduce number of connections in the mapping solution hence the total capacitance of the wires by limiting node duplication appropriately.

Power-aware clustering algorithms follow a similar approach as power-aware technology mapping [110]. The LUT to LUT connections with higher signal activities are mapped into same CLB. Unlike in the technology mapping, these connections cannot be hidden, but the power could be saved by utilizing the low power local interconnects within the CLB. The

clustering algorithm discussed in [110] is similar to T-VPack but its way of assessing the suitability of a logic cell to a particular cluster is modified. The modified assessment method prioritizes logic cells which are possible to be fully added into the current cluster, and also the logic cells with higher signal activities. It is possible to tune the two parameters to get the power optimal clustering. Clustering algorithm in [111] also uses the same two parameters. Depopulation is another aspect which has been used in clustering algorithm. The idea of depopulation is that to remove LUTs from fully occupied CLBs and cluster them in other CLBs. This process reduces the area utilization, but the works in [112] and [69] shows that it reduces the routing congestion at the routing stage. In [69], the clustering algorithm considers a parameter called separation while adding a logic cell to a cluster. Basically separation of a logic cell indicates the degree of separation from its neighbors, therefore adding a logic cell with low separation value into a cluster will be beneficial in terms of routing congestion. Both [112] and [69] discuss this kind of routability aware clustering is appropriate for complex applications, and the methodology in [69] claims it can achieve better power consumption due to the reduction in interconnect wirelength.

Traditional placement approaches aims to reduce the delay by placing critical CLBs close together, reducing the amount of interconnect a critical net pass through [62]. On the other hand, power aware placers take the activity of the connections in to account and place CLBs connected with high signal activity connections close together [113]. In [110] authors modify the VPR's cost function to have a switching activity component. The cost of swapping, therefore depend on wirelength, delay and the signal activity. In addition, in [114] a power term is added to the cost function, which requires both capacitance and switching activity data. Here, the capacity is modeled using an empirical capacitance estimation technique. The switching activity can be estimated using vectorless estimation methods, but, for more accurate results, the authors in [114] suggest to do a simulation and obtain signal activity data of the nets. However, to obtain switching activity data for all the nets in the design,

the designer might need to perform a non-power based placement and routing, which may degrade the design productivity. These power aware cost functions need to be carefully adjusted because the low power oriented cost function may increase the delay and wire length. In addition, the literature has not discussed the effect of power aware approach on larger and complex benchmarks

In modern FPGAs, there are different types of routing wires with different span [115]. For example, in Altera devices there are two different row wires R4 and R16, which are spanning 4 LUTs and 16 LUTs respectively, and in Xilinx Virtex-5 devices, there are PENT and DOUBLE routing wires which are spanning 5 tiles and 2 tiles respectively [115]. These different types of wires have different delays and capacitance, therefore there is a necessity to identify correct type of wires in power aware routing. To achieve this, the negotiation based cost function in [114] includes a wire capacitance parameter along with other congestion and delay parameters. To avoid the timing critical path to be unaffected by the power aware cost values, [114] consider the capacitance values only for non-critical nets. On the other hand, cost functions in both [114] and [110] consider the signal activities of the nets during the routing, such that the high active nets are given the priority to capture the low-capacitance wires.

2.4 Runtime of FPGA CAD tools

Placement and routing phase of FPGA CAD tool is the most time consuming process during the compilation. [82] shows that the runtime for the placement step can be as high as 49% of total runtime of typical FPGA CAD flow. Time to complete a typical placement and routing can be as much as days[116]. In addition to the standard CAD compilation flow, FPGA design cycle also includes steps for functional verification and timing/power analysis [117]. These design verification steps require re-running the compilation flow, in order to meet timing/power requirements and to resolve functional errors. Therefore, prolonged runtime

of placement step (CAD flow) drastically increases the time for design-debug cycle, hence increases the time-to-market. Therefore, there have been attempts to lower the runtime of placement step.

2.4.1 Fast Placement with Low Performance

Ultra-fast placement [116] describes a placement methodology to speed up the placement process by trading the quality. These approaches would be good for FPGA design and verification engineers who are willing to see a fast compilation, but would not be suitable for a final compilation. Ultra-fast placement algorithm is a hybrid of partition based and simulated annealing approaches. First, the input netlist of logic blocks is partitioned in a bottom-up and multi-level manner, based on the connections between each logic blocks. During each clustering stage s_n , clusters of size s_n are made and the clustering process continues until a predefined number of clusters are being made. After the clustering process, the placement is also done in multi-level fashion. The goal is to make quick placement by making good and fast global decisions at the higher levels of the hierarchy, and following this with iterative local improvement at the different levels of granularity. Each level in the hierarchy is placed in two stages; the first stage is a rough placement depending on the external connections of the cluster, and the second stage is an improved placement using simulated annealing. The parameters of anneal process are tuned to acquire a good quality-time trade-off. The choices of algorithms in these approaches are decided by the following goal which is focused to reduce the complexity of large placement problems by dividing them into manageable portions, and then employ known heuristics that are fast, and effective on each portion. Ultra-fast placement is capable of delivering high quality results if there are no runtime restrictions, but rapid placement results suffers area overhead. This work, however, has not compared the runtime with complexity of the application.

2.4.2 Modular Design Methodologies

The MDM was developed to increase design productivity by allowing designers to compile designs partially and preserve satisfactory partial compilations. Typical MDM flow constitutes two phases; *module creation* and *module assembly*, and are discussed below.

2.4.2.1 Module Creation

In the module creation phase, identified modules of a design are compiled for every possible location on FPGA and stored in a module library. Module creation in Frontier [118] and HMFlow [119] are done at a finer level where RTL components such as muxes, adders, comparators etc. are considered as a module. BPR [7], on the other hand, creates modules with much bigger components such as FFT and FIR Filters. Module creation in QFlow [120] is based on the rate of modifications required in each module creating invariant and evolving sets rather than ones based on granularity. However, all these techniques focus on faster CAD flow runtime, but do not consider performance or power.

2.4.2.2 Module Assembly

In the module assembly phase, individual modules are placed and routed to form the complete design and hence finalize the compilation process [7]. Here, the pre-compiled or pre-synthesized modules are selected from large module database for each module, and selection is made to reduce a placement cost function. In Frontier [118], the placement cost is a combination of timing and wiring cost, whereas BPR [7] also considers expected routing congestion. In the module assembly phase of HMFlow [119], its heuristic placer introduces more parameters such as size of modules, amount of connectivity, and also the presence of BRAMs and DSPs in addition to the conventional wire-length and timing based cost functions. Standard cost function is altered in [121] by adding priority levels to modules during module placement, and in [122] by including area requirement and aspect ratio of modules to the

cost function. The work [123] altered the module placement cost function to perform power-aware module placement by incorporating data-flow between modules as a placement cost. However, most of the MDM work-flows including [123], consider only one shape per each pre-compiled module during module creation. Shape of modules is the deciding factor in the utilization of the heterogeneous column arrangement of FPGAs and therefore, state-of-the-art module placement might cause suboptimal placement and resource wastage.

The authors in [124] extensively analyzed module shape and have shown that multi-shaped pre-compiled modules lead to better resource utilization. One of the reasons that multi-shape modules have not been widely used is the increase in module library size. To mitigate this the authors in [125] proposed a methodology to derive different module shape versions by partitioning a given version of a module. As a result, they could eliminate the need for pre-compiling all the module shape versions on every possible location on FPGA. However, none of these attempts have eliminated the need for pre-compiled module library. In [126], proposed an Artificial Neural Network (ANN) based methodology to estimate performance for a given module on FPGA. The ANN model takes module application characteristics, intended module shape and placement information as inputs and estimates performance of the module. Using this technique, one can rapidly estimate the performance of a module in any shape and placement on FPGA. However, the ANN model is specific to the target FPGA which requires a one-time training process, but the model can be used in estimating performance of any module in any shape and placement eliminating the need for a pre-compiled module library. However, this methodology, has not discussed a module placement mechanism for design with multiple modules.

Commercial FPGA vendors provide tools to perform MDM flow in their CAD software. For instance, Intel's Quartus Prime includes Incremental Compilation flow [127] and Xilinx's Vivado include PlanAhead [128] tools for floorplanning. However, these CAD tools do not perform automatic and informed module creation and placement and require developer

expertise. Academic and commercial state-of-the-art show the need for a power-aware MDM which is able to utilize heterogeneous column arrangements in modern FPGAs.

2.5 Summary

FPGA devices have become an appealing platform for porting applications due to the low NRE cost and TTM [25]. Modern FPGAs which are fabricated with multi-million logic cells and thousands of IP cores such as BRAMs and DSPs, have amplified the demand for FPGAs, becoming a feasible hardware to implement increasingly large applications [12][13]. However, FPGA CAD tools have not been developed at the same pace to cater the efficient and effective compilation of such applications, often causing very long compilation time [7] (several hours or days) and degraded performance and power. In addition, manual intervention and higher design expertise have been required in order to perform a better compilation, which then cause prolonged design cycle, hence the inversely affecting low TTM property of FPGAs. Therefore, there is a need for an efficient application mapping methodology for large FPGA applications.

Typical FPGA CAD flow comprises of several processes, which referred to as Synthesis, Technology Mapping and Clustering, Placement and Routing. Out of these steps, placement consumes the longest runtime during compilation [7]. In addition, it has a great impact on final performance of the FPGA implementation, as it decides the interconnect wirelength required to implement a design [57]. Three types of placement algorithms exist in the state-of-the-art. Simulated-annealing based placement tools produce high quality results but the runtime increases exponentially with the application size. On the other hand, partitioned based approaches are faster but produce deteriorated QoR. With the growth of the FPGA capacity and the application sizes, modern commercial FPGA vendors have adapted to analytical placement techniques or hybrid simulated annealing - analytical technique to have better balance between runtime and QoR. However, irrespective of these efforts, the FPGA still

requires automated methodologies to handle the design complexity without impeding on performance.

Power dissipation on FPGAs occurs due to leakage current in the transistors (static power) and signal toggles on capacitive loads (dynamic power). The dynamic power dissipation has become prominent in typical FPGA applications, and it is dominated by the interconnect power dissipation [90][88]. Therefore the placement step is critical in order to avoid signal toggles on longer routing wires with higher capacitive loads. However, state-of-the-art CAD does not consider signal toggles during placement. Power-aware techniques require an initial compilation of the application and extract toggle rates using gate-level simulation [129]. Such methodologies still requires manual intervention and often deliver solution with sub-optimal performance and power for larger applications.

Literature also show that multi-level placement techniques have been developed to speed up the compilation. However, the QoR degradation is not negligible. Modular Design Methodology (MDM) is introduced to improve the design productivity by modularizing large designs, compile and re-use previously compiled solutions. Creating modules is done based on the area granularity and the other means such as power and performance is not carried out. MDM has also shown runtime reduction, but due to losing global information of the application, the compilation still suffer from degraded performance. Furthermore, MDM requires maintaining a pre-compiled module database at all possible placement locations on target FPGAs, which is not feasible when the compiling application modules change frequently. In addition, floor-planing during CAD flow is performed in order to reduce the solution space of placement. However, such mechanisms lacks the power-aware techniques as well as the techniques to utilize modern heterogeneous FPGA architectures.

Proposed CAD tools in the literature mostly consider on optimizing performance of large application during the compilation, without addressing power degradation and prolong runtime. On the other hand, it is also important to utilize different types of resources arranged

in different column arrangements, in order to deliver solutions with high QoR. Therefore, there is a need for automated techniques that efficiently map large applications into modern heterogeneous FPGAs.

Chapter 3

Communication-aware Subsystem Generation

3.1 Introduction

Field Programmable Gate Array (FPGA) manufacturers such as Xilinx [1] and Intel FPGA (Altera) [2] integrate millions of logic resources into modern FPGAs, providing opportunities to develop increasingly large FPGA designs. The current Computer-Aided Design (CAD) tools, however, are not sophisticated enough to cater for such customizations and perform poorly for large applications. This typically results in very long design compilation times, and poor placement and routing decisions that inevitably lead to degradation in circuit performance, area, and energy consumption [130].

In order to observe the extent of degradation in power and performance of a large application on an FPGA, a case study is performed using the commercial CAD flow of Intel FPGAs (Altera), Quartus [66]. An application is considered to be ‘large’ if it consumes 160k or more logic cells and 80% or more other resources. Otherwise, they are considered to be ‘small-to-medium’ applications. An FPGA that is able to accommodate a large application can be considered as high-density FPGA. Figure 3.1a shows the floorplan of a Quartus

Fig. 3.1 Compiling Large Application Using State-of-the-art CAD

mapping for a large synthetic application. For the application in figure 3.1a, it can be seen that the routing resources have been fully utilized. It is observed that two modules have been connected via an unusually long interconnect path ‘a’, instead of a much shorter path as that indicated in ‘b’. Further, the tool does not consider the communication frequency between the modules during compilation. Therefore, the problem is exacerbated if these two modules communicate frequently, thereby leading to high power consumption. Additionally, the size of the synthetic application is gradually increased and the maximum allowable frequency (F_{max}) for the mapped solution is noted. As shown in figure 3.1b, the F_{max} of the mapped solution significantly decreases when the design size exceeds 160k logic cells.

As discussed in chapter 2, state-of-the-art FPGA CAD maps small-to-medium applications into FPGAs with superior quality of results. Therefore, in this chapter, a communication-aware partitioning technique that targets large FPGA designs is proposed. *The proposed technique partitions such FPGA designs into smaller sub-designs (subsystems) such that the intra-subsystem communication frequency is maximized, while minimizing the inter-subsystem communication frequency.* This strategy reduces the energy consumption by eliminating high-communication frequency on longer routing wires. It relies on existing

CAD tools for the compilation process. The partitioning information is provided to the CAD tool during compilation. The main contributions of this chapter are;

- An algorithm to partition a given RTL design into subsystems
- Analysis of the effect of subsystem size on quality-of-results
- A methodology to extract FPGA resource consumption at computation and memory nodes, and communication frequencies between nodes at RTL level
- An automated methodology to map subsystems to FPGAs using a commercial CAD software

The rest of the chapter is organized as follows. Section 3.2 discusses the methodology. The performance evaluation is presented in section 3.3 followed by the conclusion in section 3.4. This work has been partially published in [131].

3.2 Methodology

In this section, the proposed communication-aware design partitioning methodology is discussed in detail. The Quartus CAD flow is used to map design partitioning solution obtained from the methodology into Intel(Altera) FPGA devices. However, this technique can be used in other vendor specific CAD flows and the respective FPGA devices.

3.2.1 RTL Designs and CAD Compilation

Register Transfer Level (RTL) descriptions of circuit designs using Hardware Description Languages (HDL) such as VHDL and Verilog is a common approach in FPGA based design. Thus, FPGA applications are implemented using RTL code or RTL code generated using High Level Synthesis (HLS) tools. Typically, RTL code of a large application constitutes smaller

Fig. 3.2 Typical Design Hierarchy of an RTL Implementation

code blocks where each block represents a specific functionality. i.e. matrix multiplication, vector addition etc.. Such code blocks are defined as an *entity* in VHDL (or *module* in Verilog HDL). Each entity consists of a number of input and output ports for clock signals, control signals (reset, start, done) and data input/output signals to interact with the rest of the design. Entities are arranged as a hierarchy as shown in the figure 3.2. The entity at the top of the hierarchy (top module) encapsulates all the other entities and connects the input/output ports of the design with to the input/output pins of the FPGA. The design hierarchy is usually multi-leveled as shown in figure 3.2, where each entity comprises of many other entities. Upon synthesis of RTL code using a CAD tool, each entity will have a resource requirement from FPGA such as CLBs, BRAMs or DSPs.

During CAD flow, an RTL code is translated into atomic FPGA resource components (ex. CLBs, BRAMs) and then the individual components are placed and routed on the target FPGA device. Further, there are different compiler optimization strategies. For example,

Intel Quartus compiler optimizes in a balanced mode by default, targeting to meet timing constraints of the application. Alternate optimization goals, such as performance or area cause the compiler to prioritize a particular optimization metric. However, high effort in optimization increases compiler runtime considerably.

As mentioned in chapter 2, state-of-the-art CAD tools produce mapping solutions with superior QoR for small-to-medium applications. In this work, the existing CAD flows will be used in mapping large applications, irrespective of their limitations. The proposed methodology breaks a large application in to a collection of smaller sub-designs, hence the existing CAD flows can be exploited in mapping each sub-design. Additionally, the abstraction level is elevated from individual atomic components (i.e. CLBs, BRAMs) to RTL entities as the smallest component in a large design. The different types of entities constituting an application, which are referred to as *Basic Computation Units*, are defined as follows.

Computation units RTL Entities with operations like arithmetic, logical, comparisons and assignments are mostly mapped to CLBs (LUTs and Registers) and DSPs. In this work such entities are referred to as *computation units*.

Memory units RTL entities that are mainly inferred into BRAMs in the FPGA during the CAD compilation are referred to as *memory units* in the proposed methodology.

Communication In a typical HDL code, connections between the circuit elements are defined in order to obtain the desired circuit. Such logical connections between various RTL elements are mapped on to interconnects on FPGAs upon compilation (Placement and Routing stage). These connections experience signal toggles between HIGH(1) and LOW(0), based on the behavior of the operation of the circuit which eventually is seen as the data-flow. However, in the proposed methodology, the data-flow is considered at RTL

Fig. 3.3 Different Communication Activities in Applications

entity level. i.e. Computation units and Memory units. There can be many RTL input and output connections of different bitwidths between two RTL entities, which are eventually considered as individual signals. The total number of toggles that occur on all the input and output signals between two basic computation units is defined as the *communication* between respective entities.

Applications are categorized into three types based on the communication activity. As shown in figure 3.3, the first type of applications has isolated computation and memory units with high communication between them. In the second type, moderate and higher communication can happen between most of the units in the application, whereas, the communication throughout the application is low in the third type. The proposed methodology is analyzed with applications from each type.

3.2.2 Problem Formulation

The state-of-the-art CAD flows are capable of efficiently compiling small-to-medium applications. Therefore, the proposed methodology partitions a large application into smaller sub-designs, hence, exploiting the state-of-the-art CAD flows in compilation. A partitioning technique, which uses communication as the primary criterion, is used to divide a large application. The problem of dividing a large application is formulated as follows.

Input to the proposed methodology is an RTL code of the design. The RTL code is transformed into a graph $G(V, E)$ where a node $v \in V$ represents a computation or a memory unit (a basic computation unit), and an edge $e_{i,j} \in E$ represents a connection between basic computation units v_i and v_j , as shown in figure 3.4. Additionally, the following information is extracted using the RTL code.

Each basic computation unit (node in G) has a minimum FPGA resource requirement for successful mapping. The resource requirement of the node v_i is annotated as $a_i(l_i, m_i, d_i)$ where l_i , m_i and d_i represent the number of CLBs, number of Block RAMs (BRAMs) and number of DSPs, respectively. The communication between two nodes is analogous to the toggle rate α in the equation 2.1 and is represented as the cost value of edge $e_{i,j}$. Maximum allowed interconnect wires per unit FPGA area (also referred to as connection density in this chapter) is represented by C and is assumed to be a constant for the entire FPGA area.

Given the graph representation of an RTL code $G(V, E)$, area requirement of each computation and memory units $a(l, m, d)$, communication frequency between each units α , and the connection density of the FPGA C , partitioning large application into sub-designs can be formulated as partitioning the graph G into $n (\leq N)$ number of subgraphs, where N is the total number of basic computation units in the design.

Fig. 3.4 Converting an Input Design into a Graph

3.2.2.1 Subsystem

As shown in figure 3.4, a *subsystem* can be defined as a collection of basic computation units. Subsystems are formed by partitioning the graph $G(V,E)$ into subgraphs based on communication. The basic computation units of a particular subsystem, for instance v_1 , v_2 and v_6 in figure 3.4, are placed in close proximity to each other.

3.2.2.2 Intra-subsystem Communication

When two or more computation or memory units are partitioned into a single subsystem, the communication between them become internal to that subsystem. For example, communication on edges e_3 , e_4 and e_5 are now internal to the subsystem. The communication that happens between basic computation units of the same subsystem is referred as *intra-subsystem communication*.

3.2.2.3 Inter-subsystem Communication

After forming subsystems, computation and memory units belonging to different subsystems may still communicate with one or more units from different subsystems. Aggregated communication between such subsystem is referred to as *inter-subsystem communication*.

The objective of the proposed partitioning technique is to maximize the intra-subsystem communication for each subsystem while minimizing the total inter-subsystem communication. The partitioning process is constrained as follows:

1. Area Constraint: Total number of CLBs, BRAMs and DSPs required by each subsystem must be less than predefined area constraint tuple $A(L, M, D)$
2. Congestion Constraint: The ratio between Area and the associated number of connections of each subsystem must be less than C

3.2.3 Extracting Communication Frequency from RTL Implementations

In this section, the concept of communication, that is defined in 3.2.1, is further discussed. During the Technology Mapping step of the typical CAD flow transform a given RTL code into basic logic elements, and hard IP cores such as BRAMs. During the Placement and Routing step, the elements are mapped into the target FPGA and the connection between elements are routed. For instance, figure 3.5a shows a diagram of connections between RTL entities of an application source code, which is taken from RTL viewer of Quartus CAD tool. As discussed in section 3.2.1, RTL entities are defined as basic computation units (computation and memory units) in this methodology. For instance, figure 3.5a shows 2 memory units and a computation unit. A portion of post-synthesis netlist of the same RTL implementation is shown in figure 3.5b. Here, the memory units are converted to BRAMs and computation units are shown as logic gates and DSPs. These logic gates are

(a) Connections between RTL Entities in the Source Code

(b) Connections in Post Synthesis Netlist

(c) Connections in Post Placement & Route Netlist

Fig. 3.5 Connections of an Application in Different Levels

clustered and mapped into architecture specific CLBs (Configurable Logic Blocks) during the target architecture-aware technology mapping step in the CAD flow. Figure 3.5c shows the post-placement and route netlist, with FPGA elements such as CLBs, BRAMs and DSPs. During the placement step in the CAD flow, these elements are mapped to a unique FPGA location, and the connections in the post placement and route netlist are implemented on the FPGA interconnect fabric during the routing step. In a typical FPGA-based design cycle, the signal activities of a design is analyzed in many levels for verification. Functional verification of the application is done by performing an RTL simulation (functional simulation) which

operates on the source code level. Designers also analyze signals in a gate-level simulation (timing simulation), which operates on post-placement and route netlist with signal delays, in order to verify the implementation of the design on target FPGA.

A simulation based technique is used in order to extract signal toggle details (communication). As discussed in section 3.2.1, RTL entities (basic computation units) are considered as the smallest component in defining large FPGA applications. Hence, the RTL connections within an entity and their signal activity can be ignored. Gate-level simulation reports the most accurate and close to actual signal activity. However, it is a tedious process due to the great number of signals, especially when the design is large. On the other hand, RTL simulation analyses RTL connections defined in the source code of the application. i.e. In Verilog: wires and regs, in VHDL: signals and variables. According to figure 3.5b and 3.5c, the post-synthesis and post-placement and route connections are arranged in a similar fashion. Connections between RTL entities shown in figure 3.5a are also similar to such connections in post-synthesis netlist. Therefore, signal activity analysis in RTL simulation approximates the actual signal activity.

3.2.3.1 RTL Simulation based Technique to Extract Communication

RTL connections in a typical application can be categorized into (i) Clock signals (ii) Control signals (iii) Address signals and (iv) Data signals. The clock signals are connected to special clock pins on target FPGA during the mapping, and are routed through clock network routing fabric. Hence, the clock signals do not contribute to the communication between RTL entities. On the other hand, control signals (i.e. reset, start, done) are single-bit signals which are typically controlled by the top-level entity. Therefore, the effect of control signals on communication is insignificant, hence the control signals are excluded from the communication analysis. Majority of signal activity is occurred in address and data signals.

Fig. 3.6 Technique to Extract Communication in RTL Implementation

Figure 3.6 illustrates the technique used for extract communication information from an RTL Implementation.

- (A) The technique take the RTL code as the input. In a typical RTL code, the signals are connected to different RTL entities using a port map. The port map is analyzed by a text processing script to obtain the names of the address and data signals that connects two RTL entities. Extracted signal names are then output to a *Simulation Helper Script* (.do file) which can be used in automating simulation process.
- (B) The *Helper Script* is then provided to the RTL Simulation Software (i.e. ModelSim [132]) along with the RTL source code and its test-bench. The script file adds all the signals obtained in step (a) to the simulation waveform, whereas the test-bench injects necessary input signals to the RTL design. Then the waveform analysis on RTL simulator is run for a specific time period, and the output waveform with signal activity is obtained.

(C) The waveform with populated signal activity is then exported to a tabular list. For each address and data signal in the waveform, there is a entry in the list. It represents the values of signals in binary form with time intervals. The tabular list is process to identify the changes in signal changes (i.e. by XOR operation), and the total number of signal changes for each address and data signal is calculated. The communication value α_i of an edge e_i that connects nodes v_a and v_b in the graph G is obtained as the total number of signal changes of all the signals between the RTL entities (basic computation units) of the respective nodes.

3.2.4 Communication-aware Subsystem Generation

Algorithm 1 presents a bottom-up greedy clustering strategy for partitioning a given large FPGA design. Initially, each of the basic computation units in the graph G , is treated as a subsystem with only one element. In line 1 of Algorithm 1) a list *edge_list* of all the edges and respective communication frequency of the connection is maintained. In lines 3 through 6, each iteration of the *while* loop, an edge is first selected from this list using the *get_valid_edge(edge_list)* procedure. Then, the two subsystems of the selected edge are merged to form one (larger) subsystem using the *merge(edge)* procedure. This selection of edges and their merging process is repeated for all edges.

3.2.4.1 Eligible Edge Identification

An edge can be eligible for the *merging operation* if the associated nodes of the edge satisfy the area and congestion constraints, upon merging. For example, consider the operation of merging a node to the subsystem shown in figure 3.4. Let the area constraint for a subsystem in this case be 16,4,4. Merging the nodes v_0 or v_3 to the existing subsystem will violate this area constraint, hence the edges e_2 and e_9 are not eligible for merging. However, merging the nodes v_5 or v_7 with the subsystem still satisfies the area constraint, hence the edges e_0, e_6

and e_7 are considered as *eligible edges*. If there exists any valid edge, the most profitable edge among them is chosen for the *merging operation* in lines 8 through 18 of Algorithm 1.

3.2.4.2 Merge Operation

Lines 20 through 31 of Algorithm 1, present the *merge(edge)* procedure. The associated nodes a and b , to the edge e are merged to form a new subsystem a' , and the edge is removed from *edge_list*. There can be other adjacent nodes common to the subsystem, which can be denoted as $c \in V$ and $c \sim a, c \sim b$. In *merge(edge)*, the edges associated with such common node c , i.e. e_{c1}, e_{c2} , are merged into one edge x which holds the total communication frequency $\alpha_x = \alpha_{e_{c1}} + \alpha_{e_{c2}}$. Other edges associated with nodes a and b are added to the subsystem without modifying respective communication frequency. However, since the nodes a and b are merged into a' , other edges in the *edge_list* which are associated with a and b needs to be update to a' . Additionally, *edge_list* should be updated if there is a merged edge x as described above. These updates are handled by *update_edge_list()* procedure in line 30.

3.2.4.3 Calculating the profit

Two factors are considered to evaluate the profit of merging two nodes (subsystems) a and b into a' which is connected by edge e . One factor is the communication between all RTL modules in a and b . The higher the communication between modules, the better the quality of the resultant subsystem a' . This communication is referred as *intra-subsystem communication*. The other factor is the communication between the modules in the resultant subsystem a' and the nodes outside which is referred as *inter-subsystem communication*. The clustering process strives to increase the *intra-subsystem communication* and decrease the *inter-subsystem communication*. Therefore, the profit of merging subsystems (a, b) which

Algorithm 1 Communication-aware Subsystem Generation Algorithm

```

1:  $edge\_list \leftarrow$  communication activity  $\alpha$  for all  $e \in E$ 
2:
3: while  $has\_valid\_edge$  do
4:    $edge \leftarrow get\_valid\_edge(edge\_list)$ 
5:    $merge(edge)$ 
6:
7: procedure GET_VALID_EDGE( $edge\_list$ )
8:    $e_{max\_profit} \leftarrow first\_edge(edge\_list)$ 
9:   for each  $e \in edge\_list$  where  $e \notin marked\_edges$  do
10:    if  $meet\_constraints(e)$  then
11:      if  $profit(e) > profit(e_{max\_profit})$  then
12:         $e_{max\_profit} \leftarrow e$ 
13:   return  $e$ 
14:
15: procedure MERGE( $edge$ )
16:    $\{a, b\} \leftarrow get\_connecting\_nodes(edge)$ 
17:    $a' \leftarrow$  merge of  $\{a, b\}$ 
18:    $remove(edge)$ 
19:   for each  $c \in edge\_list$  where  $(c \text{ adjacent } a)$  and  $(c \text{ adjacent } b)$  do
20:      $e_{c1} \leftarrow get\_associated\_edge(c, a)$ 
21:      $e_{c2} \leftarrow get\_associated\_edge(c, b)$ 
22:      $x \leftarrow$  merge of  $e_{c1}, e_{c2}$ 
23:      $\alpha_x \leftarrow \alpha_{e_{c1}} + \alpha_{e_{c2}}$ 
24:    $update\_edge\_list()$ 

```

is connected by edge e , is formulated in equation 3.1. Here $\alpha(u, v)$ is the communication between nodes u and v .

$$profit_{e(a,b)} = \frac{\text{intra-subsystem communication}}{\text{inter-subsystem communication}} \quad (3.1)$$

3.2.4.4 Constraints

The partitioning problem discussed above can lead to a trivial solution, where all the nodes are clustered into a single subsystem so that the inter-subsystem communication is zero and intra-subsystem communication is maximum. Furthermore, it is not always beneficial to have

lesser number of partitions that only satisfy the intra- and inter-subsystem communication goals and maximize the profit term expressed in Equation 3.1. Having a lesser number of subsystems might not lead to efficient FPGA mapping due to the area as well as congestion constraints described in subsection 3.2.2. To avoid such trivial solutions and to satisfy area constraints, the maximum area of a subsystem is constrained during merging RTL components. Let L and M and D be the maximum allowable LUTs, BRAMs and DSPs per subsystem; then

$$\sum_{\forall i \in V} a(l_i, m_i, d_i) \leq A(L, M, D) \quad (3.2)$$

It is noteworthy that while placing high communication links in a relatively small area improves the intra-subsystem communication, merging more nodes into subsystems also involves having more internal edges in the subsystem. These edges will be a physical routing wire when they are mapped into FPGA, thus, having many connections within a subsystems may also cause high routing congestion regions while subsystems are mapped to FPGA. In order, to avoid such regions, the number of connections per unit area is constrained by introducing a term called *connection density*. Connection density of a subsystem v is defined as follows in equation 3.3.

$$connection_density_v = \frac{number\ of\ internal\ links_v}{area_v} \quad (3.3)$$

Let the maximum allowable connection density of a target FPGA be, C , then for all the subsystems in the graph,

$$\sum_{\forall v \in V} connection_density_v \leq C \quad (3.4)$$

Next, algorithm and the concepts presented above are discussed with the help of a example application as shown in Figure 3.7. An iteration of the algorithm is shown in Figure 3.7(a)

(a) Initial Condition and the First Iteration

(b) Execution of the Second Iteration

Fig. 3.7 Execution of the Subsystem Generation Algorithm

and Figure 3.7(b). The figure illustrates the graph representation of the application, $G(V, E)$. Each node in the graph indicates the nodes associated within the subsystem, area requirement of the subsystem (LUTs, BRAMs, DSPs) and intra-subsystem communication in the notation $[node_list] \{l, m, d\} \{\alpha_{intra}\}$ respectively. An edge in the figure is annotated with the edge ID and communication value between associated nodes with the notation $[edge_no] \{\alpha_{inter}\}$. According to figure 3.7 (a), e_{11} is selected for merging in the first iteration as it shows the highest profit. The profit is even higher than merging the nodes associated with e_4 , although $\alpha_{e_{11}} < \alpha_{e_4}$. Merging respective connecting subsystems of e_{11} , makes a new subsystem with $[v_4, v_3]$ of area $(l, m, d) = (16, 0, 0)$, as shown in figure 3.7 b). Since the edge e_{11} is now internal to the new subsystem, the α_{intra} of the subsystem is now 200. In the second iteration, e_{10} is the highest profitable for merging operation, as it makes $\alpha_{intra} = 350$ and $\alpha_{inter} = 250$, resulting in a profit of 1.4 according to the equation 3.1.

3.2.4.5 Complexity of the Algorithm

Let the number of nodes and the edges in the graph G be n and e , respectively. Assume the maximum degree of connectivity of the graph G is d . In the algorithm 1, main loop (lines 3-5) iterates for n times (worst-case). In each iteration, *get_valid_edge* procedure selects the best edge from *edge_list*, hence $O(e)$ complexity. When two nodes are merged in procedure *merge*, all the edges connected to the merged nodes are updated, hence the complexity $O(n.d)$ (worst-case). Therefore, the complexity of the algorithm is $O(n.(e + n.d))$.

3.2.5 Automating Subsystem Generation using CAD Tools

In this section, the steps involved in automating the proposed subsystem generation methodology is discussed. As shown in Figure 3.8, the input to the proposed methodology is an RTL design. The following parameters are extracted from a given RTL design code, in order to execute the algorithm:

Fig. 3.8 Communication-aware Subsystem Generation Methodology

- i) Area of a subsystem: CAD tools calculate area requirement of each design instance after RTL synthesis. Since the runtime of RTL synthesis step for an application is minimal as compared to the runtime of its placement and routing steps, the FPGA area $a(l, m, d)$ of each node in $G(V, E)$ is obtained by performing a quick RTL synthesis.
- ii) Communication Frequency of Edges: After the synthesis step, an RTL simulation is performed. Communication frequency of edge e , α_e , is obtained by logging the simulation details of signal e to a file and by counting ($High \rightarrow Low$) and ($Low \rightarrow High$) logic transitions as discussed in section 3.2.3.1.
- iii) Graph Connectivity: A typical RTL design comprises of different design instances i.e. VHDL components or Verilog modules. These instances are connected using wires or signals to achieve the desired behavior of the overall design. The connectivity information is required to build the Graph $G(V, E)$. Typically, after the RTL synthesis step, a netlist with the required connectivity information, is generated. However,

since commercial CAD tools like Quartus do not allow users to read the generated netlist, a text processing technique on the RTL code is used for extracting the relevant information.

Next, the extracted parameters are provided to the proposed communication-aware subsystem generation algorithm. The output of the algorithm is the generated subsystems and the composition of each subsystem, i.e. basic computation units in each subsystem. The subsystem composition is then provided to LogicLock feature in Quartus tool to implement the subsystems on the target FPGA. Each LogicLock corresponds to a set of RTL modules, belonging to a subsystem, in order to facilitate their placement within a rectangular region of choice on FPGA. However, width and height, and the proximity of the LogicLock on the target FPGA are not specified during the compilation, and the Quartus compiler is allowed to decide them. An application with identified subsystems is then compiled using Quartus.

3.3 Performance Evaluation

In this section, the effectiveness of the proposed communication-aware subsystem generation algorithm is evaluated by executing various applications and observing their performance in terms of F_{max} and energy consumption. Intel's (Altera) Quartus toolchain[66] is used for the experiments. For both the traditional and proposed approaches, the Quartus compiler optimizes for performance rather than for area during the mapping process. The technicalities on comparison strategy and used benchmark applications and test vectors are discussed in the subsequent sections.

3.3.1 Comparison Strategy

The comparison is performed using the following cases.

- i Default Quartus Compilation : RTL code of the test application is compiled using Quartus CAD flow as illustrated in figure 3.9a.
- ii Compilation with the proposed methodology : Same RTL code is input to the proposed methodology in order to identify subsystems. Then, the identified subsystem details are provided to Quartus compilation along with the test RTL code to compile with subsystems in the design. This flow is illustrated in figure 3.9b.

After the Quartus compilation finishes for both cases, the performance (maximum clock frequency - F_{max}) of each case is obtained from the compilation report. Accurate power dissipation of both cases is obtained using a gate-level simulation followed by running Quartus PowerPlay power estimator[133]. Total energy consumption is then derived using equation 3.5.

$$E = P \times t \quad (3.5)$$

where, E - total energy consumption, P - total power dissipation and t - runtime of the application.

It is noteworthy that the maximum clock frequency (F_{max}) can be different across the 2 cases above. Hence, running an application on FPGA with F_{max} clock frequency in each case will differ the runtime of the application and consequently the energy consumption (As per equation 3.5).

3.3.2 Test Applications

In order to show the effectiveness of the proposed approach, it is necessary to use test applications that demonstrate the qualities of a ‘large application’. Therefore, large synthetic applications are created such that they mimic the applications from Polybench [134] benchmark suite. For example the Atax and Bicg application kernels from Polybench, which are relatively small-scale designs, are duplicated multiple times with minor modifications to the

Fig. 3.9 Comparison Strategy for Power and Energy

kernel itself, to exhaust the logic and memory resources of a Altera EP2C70 series FPGA device, as shown in figure 3.10(a). Those applications are referred to as *Atax'* and *Bicg'* respectively. In order to increase the complexity of the applications, an additional demand for routing resources is created by including links between kernels (shown in red in figure 3.10(a)).

Three more synthetic applications with the following properties are created to observe the effectiveness of the proposed approach on smaller FPGAs (experimented on Altera EP2C35 device) : Let us assume that $C = \{c_0, c_1, c_2, \dots, c_N\}$ and $M = \{m_0, m_1, m_2, \dots, m_N\}$ are a set of N computation units and N memories respectively. In order to make characteristics of synthetic test applications similar to actual applications, the computation units are based on different applications from Polybench benchmark suite. Each computation unit c_i is connected to n number of memories m_i to $m_{mod(i+n-1, N+1)}$ consecutively, as shown in figure 3.10(b). In other words, to perform the computation c_i , it requires data from n memories. Parameters N and n are gradually adjusted in order to exhaust the resources on Altera EP2C35

(a) Replicate Real Benchmark Kernels

(b) Synthetic Applications Inspired by Real Benchmarks

Fig. 3.10 Structure of Large Applications Consists of Benchmark Applications

family FPGA. This can be identified as a large application in context of a smaller FPGA device, which demands both high logic and routing utilization in this device.

3.3.3 Test Vectors

Typically application execution follows Pareto Principle or 80/20 rule [135], hence, only 20% of the code runs 80% of the time. The test applications and test-benches are designed such that the input from test-benches lead applications to demonstrate the 80-20 rule.

Each kernel in applications shown in 3.10(a) is a real benchmark application which naturally demonstrate the 80/20 rule. Kernels operate with the memories within each kernel. During the simulation process, these memories are populated with an initialization file, such that data of each memory entry is randomly generated. In addition, an *enable* signal, which is controlled by the test-bench, is introduced for each kernel. The kernels are also activated demonstrating 80/20 rule by a randomly generated enable signal sequence.

To demonstrate the varying execution frequency in the synthetic test applications, memory units in these synthetic applications are controlled by the test-bench. Memories in the benchmark applications can be activated with the *enable* signal. For instance, a benchmark application with 32 memories can be handled with a 32 bit wide input signal. A series of random input signals are generated such that 80% of the input sequence activates 20% of memories and connected computation units. Once the memories are enabled, data-flow between respective memories and connected computation units takes place. It is important to note that, the input sequence is referred as *runtime characteristics* of the application. Communication in the application depends on this input sequence. The same sequence is used in the RTL simulation for our framework as shown in figure 3.8.

Two parameters are measured in order to analyze the effectiveness of the partitioning methodology, (i) performance of the design and (ii) power dissipation of the design. As mentioned earlier, CAD tools are often equipped with a timing analyzer to estimate gate

delays and routing delays with very high precision. As a result, the timing analysis reports the critical path delay and hence the maximum clock frequency (F_{max}). For a given design and CAD compilation settings, typical CAD flow produces the exact same mapping solution. In addition, CAD compilation does not depend on the input. Therefore, different test vectors are not necessary to obtain F_{max} results.

Power dissipation is estimated using simulation data. The compiled application is simulated using a gate-level simulator, which accounts for low-level signal delays and switching. Such signal switching activity of the entire application is recored in a file. (i.e. VCD file). The VCD file is then analyzed using Quartus PowerPlay to obtain the most accurate estimation for power dissipation. The gate-level simulation takes the input into account, so that the signal activity and VCD files depend on the input vectors.

3.3.4 Results and Discussion

The test applications have been mapped to FPGA devices both with and without the subsystems generated in the techniques proposed in this chapter. The relatively large *AtaxI* and *BicgI* are mapped to the larger Altera EP2C70 [24] FPGA device while the three hand-coded benchmarks were mapped to the Altera EP2C35 [24] FPGA device. This is due to the size of the *AtaxI* and *BicgI* applications, which cannot be accommodated in the smaller Altera EP2C35 FPGA device. Table 3.1 shows performance of these five applications in terms of maximum achievable frequency, F_{max} . As evident from this table, partitioning the large applications into subsystems, has significantly increased the performance of the mapped solution with an average improvement of over 61% in their F_{max} .

Quartus PowerPlay reports a breakdown of total power dissipation as static power and dynamic power. Since the proposed methodology does not alter the static power dissipation, only the dynamic power dissipation is considered for the evaluation. Even though the estimate report further breaks down the dynamic component, only dynamic routing power dissipation,

Table 3.1 Performance (F_{max}) Improvement

Application	Without partitioning F_{max} (MHz)	With partitioning F_{max} (MHz)	Percentage F_{max} Improvement
Synth1	50.61	89.17	76.19
Synth2	50.61	69.26	36.85
Synth3	52.43	93.9	79.10
Atax'	51.91	90.23	73.82
Bicg'	57.8	81.01	40.16

Fig. 3.11 Power Reduction with Subsystems

Fig. 3.12 Average Power Reduction with Subsystems

Fig. 3.13 Energy Reduction with Subsystems

which is the power dissipation due to switching activity on interconnect fabric, shows a direct impact from the proposed methodology.

Figure 3.11 shows the percentage reduction in total power and routing power (due to interconnect resources) in the benchmark applications for different seed instances. Here, each seed instance corresponds to a different initial solution seed in the CAD tool. In figure 3.12, the average percentage reductions in total power consumption and routing power consumption are depicted for the benchmark applications. It can be seen that the proposed

approach can lead to a significant reduction in the interconnect power consumption. Even the total power consumption is reduced while achieving higher performance, when compared to the traditional compilation method. From Figure 3.12, it can be seen that applications such as *Synth3* and *Bicg7* do not show significant reduction in power, while still showing a significant improvement in performance gain as seen in Table 3.1. After careful investigation of these applications, it is observed that the CAD tools kept the communicating modules in close proximity for these applications even without partitioning the application into subsystems. Hence, these applications do not show a significant reduction in power after the proposed communication-aware partitioning. However, this behavior of the CAD tools is not guaranteed for every application as evident by the notable improvement in power consumption for other applications. In addition, as shown in Figure 3.13, there is a significant reduction in the overall energy consumption when using the proposed partitioning strategy, especially for applications where both the performance as well as power consumption improve significantly when compared to the existing tools.

3.4 Summary

In this chapter, a communication-aware subsystem generation technique for large FPGA applications is presented. The proposed methodology takes an RTL code of a design as the input, and performs a quick RTL synthesis process using an existing CAD flow to extract area estimation of RTL entities (referred to as computation and memory units) in the application. It also considers the signal toggle rates between computation and memory units into account (referred to as communication), through RTL simulation. The methodology then partitions the input application into subsystems while minimizing inter-subsystem communication and maximizing intra-subsystem communication. The generated subsystems are mapped into FPGAs using existing CAD tools. The methodology ensures that computation and memory units within each subsystem (typically with high communication) are placed at

close proximity, thereby reducing the power dissipation. This technique is generic as it does not require modifications to existing CAD algorithms and can be used in conjunction with different vendor specific CAD flows.

An extensive evaluation has been performed using test applications that are developed based on benchmark applications to mimic characteristics large applications. Tests using Intel Cyclone 2 device family has shown that the proposed approach improves the routing power consumption by over 8% as compared to Quartus CAD flow. It has also improved the maximum achievable clock frequency (F_{max}) by a significant margin. This leads to an overall energy efficient design. Experimental results on 2 FPGA devices, show an average of over 61% improvement in F_{max} and more than 20% reduction in energy consumption when compared to the state-of-the-art design flow.

Even though the proposed methodology is efficient in identifying subsystems with higher intra-subsystem communication and contributes to reduce inter-subsystem communication, some applications may still have high inter-subsystem communication. Since the placement of the identified subsystems is done by the CAD flow, which is not sensitive to communication, the FPGA mapping of such cases can incur higher power dissipation. Thus, the proposed technique is extended in chapter 5 to incorporate techniques for mapping subsystems based on inter-subsystem communication, hence to identify preferable placement on FPGA.

Chapter 4

FPGA Characterization for Subsystem Mapping

4.1 Introduction

In the previous chapter 3, a communication-aware methodology was proposed to partition a large FPGA design into smaller *subsystems*, in order to lower the power dissipation from FPGA interconnect fabric. The proposed partitioning algorithm groups *computation and memory units* of a given application RTL code so as to maximize the intra-subsystem communication while reducing inter-subsystem communication.

While the proposed partitioning technique led to significant power savings, it did not consider the detailed architecture of the target FPGA device during the clustering process. As modern FPGAs continue to become more heterogeneous, incorporating different types of resources at varied locations on the fabric, it is imperative to consider the location of the various resources on the FPGA fabric to find the optimum placement of the subsystems. The optimum placement ensures that the most widely used resources in a subsystem are available at a physically closer distance, thereby reducing the length of interconnecting wires, ultimately leading to better performance and lower power consumption.

To address the varied resource requirement of a subsystem, subsystems with different resources are divided into sub-modules, where all components within a sub-module requires the same resource type for mapping. Sub-modules with only CLBs are placed in a *square* region, whereas the sub-modules only with DSPs or BRAMs are placed as a *single* column. However, in order to ensure the availability of *all* the necessary resources in close proximity for optimum subsystem placement, it is important to identify the *shape* of a subsystem along with its placement location. The shape of a subsystem refers to the rectangular space in the FPGA within which the subsystem can be placed as well as routed.

Existing work in this area typically uses a Modular Design Methodology (MDM) to identify the location of a subsystem for its optimum placement. Subsystem creation and placement, as discussed in this thesis, can be related to MDM, where a subsystem can essentially be considered as a module in MDM. Several MDM techniques, as discussed in section 2.4.2 of chapter 2, use a large database of pre-compiled subsystems. Subsystems are placed and routed at large number of possible locations on FPGA during offline *module creation phase*. During the online *design assembly phase*, best set of pre-compiled subsystems are searched and mapped on the target FPGA to achieve the final design.

Large heterogeneous FPGAs offer a large number of possible locations for a subsystem to be placed on FPGA space, making large solution space for the subsystem assembly stage. However, due to availability of different types of resources on these FPGAs, selecting a place and a shape for a given subsystem from the library becomes more critical. Wrong selection either could lead to resource wastage [124] or could affect the performance of the subsystem [136]. In addition, pre-compiling large number of variations per module is becoming increasingly time consuming. As will be shown later in section 4.2.1, compiling all possible placement variations for a single subsystem requires over 3 hours. Therefore, module creation phase is becoming prohibitively time consuming, especially if the subsystem library changes frequently [125].

Fig. 4.1 Performance (F_{max}) of a Subsystem in Three Shapes

The following example elaborates the effect of subsystem shapes for performance. Consider mapping a subsystem S_i that require 421 LUTs, 20 BRAMs and 0 DSPs. Its resource requirement tuple can be indicated as $\{421, 20, 0\}$. Figure 4.1 shows three different rectangular regions(shapes), whose left bottom placement coordinates (placement) are the same. The dimensions of each region are shown in figure 4.1 and each region satisfies the resource requirement of S_i . The subsystem S_i is then isolated from the rest of the design and mapped into 3 regions using Quartus CAD flow. Maximum allowable clock frequency (F_{max}) for each case is obtained after the mapping process and shown in figure 4.1. It has been observed that the shape given in Figure 4.1 (b) produces 11% greater F_{max} as compared to (a). While similar number of resources are consumed in both cases, the main difference is the arrangement of resource columns within each region. For instance, 27 BRAMs are located in 3 BRAM columns within the region shown in figure 4.1 (a), whereas 31 BRAMs within the region shown in figure 4.1 (b) are located in a single BRAM column. Depending on how the CLBs BRAMs and DSPs are connected within a subsystem, an arrangement of columns in a rectangular region can be more suited for the subsystem. Existing MDM techniques have also not explored the relationship between subsystem performance and its shape as well as the location in the FPGA fabric.

Hence, in this chapter, an artificial neural network (ANN) based methodology is proposed to predict the F_{max} of a given subsystem on a FPGA device, given the shape and other application parameters as explained in the subsequent sections. The main contributions of this chapter are:

- a methodology to train an artificial neural network to characterize and model any FPGA to estimate the F_{max} performance of a given subsystem.
- an ANN based approach to estimate the shape and placement of a given module in order to achieve the best performance.
- swift placement decisions during module based development that eliminates the need for maintaining a database of frequently changing pre-compiled modules.

The rest of the chapter is organized as follows. Section 4.2 discusses the methodology, and the performance evaluation is presented in section 4.3 followed by conclusion in section 4.4.

This work has been partially published in [126].

4.2 Methodology

4.2.1 Footprint Generation for Heterogeneous FPGA Architectures

A subsystem S_i is characterized by a resource requirement, which can be expressed as a tuple $A_i = \{l, m, d\}$, where l, m, d are the minimum requirement of CLBs, BRAMs and DSPs on the FPGA space in order to successfully map the subsystem into FPGA. A *footprint* (fp_i) of subsystem S_i can be defined as a rectangular region on target FPGA that satisfies the resource requirement A_i . The shape and location properties of footprint fp_i can be expressed as a tuple $LL_i = \{x, y, w, h\}$, where x and y are the bottom left corner of the rectangle in FPGA

Fig. 4.2 Footprint variation for a given subsystem

space, and w, h is width and height of the rectangle. In a typical island style FPGA, the bottom left corner block is marked as the origin of the coordinate plane which is $\{1, 1\}$.

For example, consider two different footprints $fp_{1,1}, fp_{1,2}$ for subsystem S_1 with $A_1 = \{421, 20, 0\}$ on an Altera EP2C35 FPGA [24] device as shown in figure 4.2. The shape parameters of the two footprints are $LL_{1,1} = \{1, 1, 56, 8\}$ and $LL_{1,2} = \{1, 1, 52, 9\}$. Here, the second variable (k) in the notations, $fp_{i,k}$ & $LL_{i,k}$, denotes the different footprints and shape parameters respectively of a single subsystem (S_i). Both footprints consist of 3 BRAM columns and 1 DSP column but contain 52 CLB columns in the former case and 48 CLB columns in the latter. $fp_{1,1}$ offers FPGA resources $\{432, 24, 8\}$ which satisfies the requirement A_1 . If the width of $fp_{1,1}$ is reduced to 52 without adjusting the height, the resultant resource coverage on FPGA will be only $\{384, 24, 8\}$ which does not satisfy the CLB requirement hence it is an invalid footprint. However, by increasing the height to 9 while reducing width to 52, $fp_{1,2}$, which offers FPGA resources $\{432, 27, 9\}$, satisfies the requirement A_1 . Note that both $fp_{1,1}$ and $fp_{1,2}$ cause resource wastage. However, the CLB requirements cannot be satisfied by a footprint with a height of 7, even if its width is set to the entire device width (64 for this device) of this FPGA.

An artificial neural-network (ANN) based approach has been proposed to estimate the performance of a given subsystem on its footprints. The proposed technique eliminates the need for actual compilation of subsystem $S_{i,k}$ on each footprint $fp_{i,k}$. The main idea of our approach is to use a supervised learning phase to train an ANN model that can effectively represent the target FPGA. The input training data set consists of application level characteristics of different subsystems and parameters of various footprint versions for each subsystem. During the training stage, all the subsystems are compiled on respective footprints and obtain the performance (F_{max}) values. These performance values are also used in the supervised learning process as output in the training dataset. Afterwards, the trained target FPGA model is used for F_{max} estimation by providing details of any given subsystem and respective footprint details as an input to the ANN model. The subsequent sections describe the proposed ANN technique in detail.

4.2.2 Parameters to Model Target FPGA

In this section, the parameters that are used for modeling the target FPGA device are discussed. The modeling parameters must capture the underlying details of the target FPGA architecture within each footprint as well as the subsystem (application) characteristics. The former is referred to as the architecture parameters while the latter as application parameters.

4.2.2.1 Architecture Parameters

If a subsystem is mapped onto two different footprints, the proximity of FPGA resources in the two footprints is different. FPGA resources can be different in two footprints in two ways: (i) Different *number* of CLBs, BRAMs and DSPs available in the footprint. i.e. Footprints $fp_{1,1}$ and $fp_{1,2}$ in figure 4.2 consist different number of resources (ii) Different *column arrangement* of CLBs, BRAMs and DSPs. i.e Footprints in figure 4.1 consist of same number

of BRAMs. However, BRAMs are allocated in three columns in the footprint in figure 4.1 (a), whereas a single BRAM column in the footprint in figure 4.1 (b).

These differences in the underlying FPGA architecture of footprints forces the CAD tools to map the subsystem differently on the two footprints. As a result, the performance (F_{max}) of the subsystem on the two footprints will be different.

The differences in the underlying FPGA architectures is captured using 3 parameters; the starting horizontal coordinate (x) of the footprint placement on the FPGA, width (w) and height (h) of the rectangular region on FPGA space. It should be noted that the starting vertical coordinate (y) is not considered as a parameter, due to uniformity of the FPGA architecture along the vertical direction. In other words, moving any rectangular region along vertical axis on FPGA space does not change the number or the arrangement of resources within that region. The FPGA architectural details in a region, such as the number and position of resource columns of each resource type, are effectively captured by the (x), (w) and (h) parameters.

4.2.2.2 Application Parameters

1. Different subsystems require varying FPGA resource requirements in terms of CLBs, BRAMs, and DSPs. The resource requirement of subsystems has significant effect on footprint sizes and shapes. Resource requirement also signifies the minimum resources needed to implement a subsystem successfully on FPGA. The resource requirement of a subsystem S_i is parameterized as a tuple $A_i\{l_i, m_i, d_i\}$, where the tuple represent the requirement of CLBs, BRAMs and DSPs respectively. For a successful implementation, the respective footprints must satisfy all three tuple values.
2. Application internal links (c_i): Performance (F_{max}) of a design partition depends on number of available routing resources in the respective region. High routing resource

Training application₁ data points

x	w	h	$A(l, m, d)$	c	cp	F_{max}
1	64	10	(100,8,2)	1342	144.4	132.7
1	62	12	(100,8,2)	1342	144.4	152.4
1	60	13	(100,8,2)	1342	144.4	148.9
...	...	...	...	...	...	...
58	6	30	(100,8,2)	1342	144.4	128.4

Training application₂ data points

x	w	h	$A(l, m, d)$	c	cp	F_{max}
1	64	13	(80,2,0)	944	122.8	124.7
1	62	14	(80,2,0)	944	122.8	140.0
1	60	15	(80,2,0)	944	122.8	162.3
...	...	...	...	...	...	...
58	6	33	(80,2,0)	944	122.8	116.1

Fig. 4.3 Training Data

utilization within the region typically results in longer routing paths for some routes, which could lower the performance.

3. Application default critical path (cp_i): Critical path of a design is the longest path between two registers. Some applications may contain critical paths along the combinational logic while others may have paths along memory and DSP I/O. In order to capture this differentiation in the training dataset, an estimated critical path parameter for each application has also been included. However, it should be noted that this estimated critical path is same for all the footprints of a single application, and should not be confused with the output of the model.

4.2.3 Problem Statement

Given a subsystem S_i with an area requirement A_i , with c_i number of internal links and a default critical path cp_i , estimate the F_{max} if it is mapped to a footprint $fp_{i,k}$ with architecture parameters (horizontal location, width and height of the footprint) $\phi_{i,k} = \{x, w, h\}$.

4.2.4 Generating Training Data

In order to achieve high estimation accuracy from an ANN model, a diverse training set is critical. Therefore, a large number of subsystems and respective footprints is chosen

Fig. 4.4 Generating Footprints for Training

in the training step. The RTL designs of benchmark and synthetic applications are used as subsystems by treating an entire application as a subsystem. For example, benchmark application #1 is defined as subsystem S_1 . In figure 4.3, training data points from two applications are shown. Each training data point contains three types of parameters: (i) architectural parameters - highlighted in orange, (ii) application parameters - highlighted in green and (iii) F_{max} of the application mapped into the respective footprint - in a bold black font. Here, the first two types of parameters are used as the training input in a supervised learning step whereas the last parameter is used as the training output.

Application Parameters : Application parameters (A_i , c_i and cp_i) are obtained by mapping each application into target FPGA, without defining a footprint. As shown in figure 4.3, application parameters of a particular application do not change with different footprint variations it is mapped into.

Architectural Parameters : For each application (subsystem S_i), many footprints $fp_{i,k}$ are generated exhaustively, where each $fp_{i,k}$ has different location and shape parameter values as explained below.

For each application i , a series of footprints $fp_{i,k}$ are generated by setting different horizontal placements and shapes $\phi_{i,k}$ as shown in the figure 4.4. For $fp_{i,1}$, parameters x and

w are initialized to 1 and to the device maximum width (i.e. 64 for our case), respectively. The height h of the footprint is calculated using A_i . For instance, if $A_i = \{421, 20, 0\}$ then for a rectangular region on an EP2C35 FPGA device, with starting coordinate $x = 1$ and $w = 64$, the height should be at least 8 to satisfy the area constraint A_i . Therefore, $\phi_{i,1}$ is set to $\{1, 64, 8\}$. For the next footprint $fp_{i,2}$, w is set to 63, reduced by one compared to $fp_{i,1}$, while keeping $x = 1$. Similar to previous case, h is calculated according to A_i , resulting $\phi_{i,2} = \{1, 63, 8\}$. Likewise, set of footprints are generated by reducing w and calculating respective h while $x = 1$, until w cannot be reduced further to generate a valid footprint. This footprint generation process can be repeated by shifting the starting position $x = 2, 3, \dots$ until there are no more valid footprints. The training data is obtained by changing x and w with an interval of 2.

F_{max} of the application : Each application i can be mapped and compiled into each footprint in its footprint series $fp_{i,k}$ obtained above, using a CAD flow. Successful CAD flow compilation produces the timing report, from where the respective F_{max} value is obtained.

4.2.5 Implementation of FPGA Characterization

In this section, the implementation of the methodology for FPGA FPGA characterization is discussed. Intel(Altera) Quartus CAD tool is used to demonstrate the implementation. However, it should be noted that the proposed methodology does not depend on vendor specific CAD tools. For example, Xilinx offers equivalent functionality in the PlanAhead tool flow [128]. First, some relevant information is presented, before discussing the proposed approach.

Quartus Tools for Incremental Compilation : In the default Quartus compilation flow, also referred to as ‘flat compilation’, all the RTL code files are processed simultaneously in the Analysis and Synthesis step and the entire post-synthesis netlist is placed and routed

Fig. 4.5 ANN Training Methodology

subsequently by the fitter. If any changes need to be made after a flat compilation, one needs to perform a fresh re-compilation. Intel(Altera) includes tools for modular design methodology (MDM) in Quartus, called Incremental Compilation Design Flow. It allows hardware designers to follow a divide and conquer strategy, by partitioning a large application into relatively smaller subsystems (modules) and develop each subsystem separately. Such subsystems can be defined using *Design Partition* tool in Quartus so that subsystems can be compiled separately.

Performing an incremental compilation after defining *Design Partitions* for subsystems creates separate netlist for each subsystem after each CAD flow step. For instance, after the 'Analysis and Synthesis' step a *post-synthesis netlist* or after 'Fitter' step *post-fit netlist* is generated. Quartus allows to preserve the generated netlist of each subsystem at any stage by setting a parameter called *preservation level*.

One Time Compilation for Application Parameters : As discussed earlier, an entire benchmark or synthetic application is used as a subsystem. Typical RTL applications contains input/output connections with FPGA I/O ports. Since the subsystem performance estimation is done without considering inter-subsystem connections and I/O connections, the following steps were used to isolate the training applications for training data generation.

1. An RTL application source file with empty top entity is created in Quartus.
2. Entire training application is wrapped in another RTL entity, which is then included into the top entity
3. RTL entity with training application is defined as a *Design Partition* in Quartus tool. The netlist of this design partition should be created from the source code, hence the *preservation level* is set to *source code*.
4. Netlist of the top entity is ignored in order to isolate the training application. Therefore, netlist preservation level of the top entity is set to *empty*.

During one-time compilation process, the training application is not restricted to any part of FPGA (without a footprint). The application parameters 1, 2 and 3 defined above in Section 4.2.2 are extracted by performing a one-time incremental compilation for the defined designed partition. Resource requirement (A_i) of the training application is obtained using Quartus Fitter Reports. The report also provides the total routing resource utilization, hence the parameter Application internal links (c_i) is also derived. Default critical path (cp_i) is obtained using the timing report of the Quartus compilation.

Footprint Generation in Quartus : In this step, series of footprints for each training application (subsystem) is defined by using the application resource requirement (A_i) and hence changing architecture parameters $\phi_{i,k}$ as discussed in section 4.2.4. Quartus LogicLock feature allows designer to restrict the placement to a rectangular region in the FPGA space.

Fig. 4.6 Use the Characterized Model for F_{max} Estimation

To define a particular footprint $fp_{i,k}$ of training application i on Quartus, a LogicLock is set to the *design partition*. Architecture parameters $\phi_{i,k}$ are provided to the LogicLock to set its rectangular shape (width and height), placement (bottom left coordinate).

Obtain Performance of Footprints : Similar to the one-time compilation, the netlist preservation level of the application is set to *Source Code Level* while the top level entity is kept as *Empty* preservation level in order to treat the application as an isolated module. Once a footprint is defined using *design partition* and LogicLock, it is compiled in Quartus projects using Quartus Incremental Compilation. After the compilation, the performance (F_{max}) of the footprint is obtained using Quartus Timing Analysis Report. During the training data generation, it is required to compile number of footprints per each application. In order to speed up this process, multiple Quartus project instances with different footprint configurations are compiled in parallel.

Training ANN : Once the training dataset is created using all the benchmark and synthetic applications, a Matlab deep learning ANN model [137] is trained. The ANN model is trained with 8 hidden layer neurons using Resilient Back-propagation training algorithm. A compact version of ANN used is illustrated in figure 4.7. The appropriate training function and the number of hidden layer neurons are found empirically.

Fig. 4.7 MatLab View of the Neural Network

Use ANN for Performance Estimation : A technique similar to the training phase is used for estimating performance of a given application on a particular footprint. As shown in the testing phase of figure 4.5, a test data point is provided to the ANN model with following parameters of the given application and its footprint - $(x, w, h, A(l, m, d), c, cp)$ where first three parameters represents the architecture parameters of the footprint and the rest represents the application parameters of the subsystem. Similar to training data generation as discussed in section 4.3, application parameters can be extracted by performing one-time compilation. After obtaining the resource requirement A_i of the subsystem, architecture parameters of a footprint can be derived by deciding shape and location on target FPGA such that the A_i is satisfied. Characterizedwe model estimates the F_{max} of the subsystem if it is mapped to the intended footprint. In a practical environment, it is useful to find a set of footprints of a particular subsystem that gives the best performance. This can be achieved by providing a vector of input data points as shown in figure 4.6. An input data point vector can be obtained using the same principal used in training data generation. The device model outputs the estimated performance for respective footprints, and hence footprints with best performance is found.

4.3 Performance Evaluation

4.3.1 Benchmark Applications and Target FPGAs

A total of 26 applications, 16 from Polybench benchmark suite [134] and 10 handwritten synthetic ones, are used for the evaluation of the proposed techniques in this chapter. Out of these 26 applications, 23 were used during the training phase, while the remaining 3 were used for the cross validation in the testing phase. Application parameters of these applications are shown in Table 4.1. As discussed in section 4.2, Table 4.1 presents the modeling parameters for each application, i.e. Application area ($A_i(l, m, d)$), Application internal connections (c_i) and Application default critical path (cp_i).

To conduct an extensive evaluation, four FPGA devices are incorporated in the experiment setup; (a) Cyclone II, EP2C35 - a smaller device with homogeneous floorplan (b) Cyclone V, 5CGXBC4 - one of the latest device with a heterogeneous floorplan which includes number of BRAM and DSP columns (c) Cyclone V, 5CGTFD9 - largest FPGA of Altera Cyclone series (d) Cyclone 10LP, 10C025 - very recently released FPGA. Characteristics of these devices are listed in Table 4.2.

4.3.2 Evaluation of Training and Testing Phases

4.3.2.1 Training the ANN Based Model

The training data set contains data points from 23 applications as stated above. These applications are compiled at various footprints on each device as described in Section 4.2 in order to generate the large training dataset. The training error is calculated as the percentage difference between the actual performance (F_{max}) and the predicted performance from the model, for each application on every footprint. The average training error across all applications at every footprint for each device is shown in Table 4.3.

Table 4.1 Benchmark Applications Used for Characterization and Testing

	Application	CLB(l)	BRAM(m)	DSP(d)	Connections(c_i)	Default Critical Path(cp_i)
1	2mm	36	40	3	128	98.01
2	3mm	51	56	9	384	234.96
3	bicg	25	12	3	256	235.07
4	conv2d	19	16	2	64	108.31
5	conv3d	25	64	2	64	101.55
6	doitgen	22	72	3	160	167.95
7	fdtd2d	56	25	2	384	91.93
8	gemm	26	24	3	128	197.75
9	gemver	72	16	9	544	163.23
10	gesummv	34	19	6	224	189.54
11	mvt	29	12	6	256	209.47
12	symm	26	24	3	128	212.76
13	syrk	33	16	6	160	194.59
14	syr2k	37	24	6	192	211.73
15	atax	25	11	3	224	239.29
16	trmm	21	16	3	96	234.58
17	subsys1	421	20	0	3200	163.91
18	subsys2	360	18	0	2688	167.5
19	subsys3	349	20	0	2688	165.7
20	subsys4	357	22	0	3008	168.92
21	subsys5	325	21	0	2496	170.39
22	subsys6	121	16	0	960	183.82
23	subsys7	61	14	0	480	191.75
24	subsys8	91	21	0	800	168.63
25	subsys9	523	30	0	4512	157.8
26	subsys10	523	36	0	4704	164.8

Table 4.2 Target FPGA Architecture Characteristics

Family	Device	Logic elements	BRAM columns	DSP columns
Cyclone II	EP2C35	35k	3	2
Cyclone V	5CGXBC4	50k	8	5
Cyclone V	5CGTFD9	301k	11	6
Cyclone 10LP	10CL025	25k	2	2

Table 4.3 Training Error

Device	No.of data points	Average training error(%)
EP2C35	6818	3.5
5CGXBC4	7059	3.27
5CGTFD9	19738	2.65
10CL025	3471	2.58

Fig. 4.8 Training Error Trend

Table 4.4 shows the time taken for data collection and training the ANN for each device. It is noteworthy that, data collection and training are one-time steps. As the table depicts, data collection may need several hours to days depending on the size of the device. However, it is possible to accelerate the data collection process by leveraging on parallel compilation. On the other hand, training the ANN is extremely quick, which takes 1.9 seconds to 4.7 seconds for the devices considered in the evaluation.

Table 4.4 Time Taken for Data Collection and Training

Device	No. of Training Data Points	Data Collection Time (/hours)	Training Time (/sec)
EP2C35	6818	21.5	3.4
5CGXBC4	7059	22.2	3.5
5CGTFD9	19738	62.1	4.7
10CL025	3471	10.9	1.9

Table 4.5 Time Taken for Testing

Device	No. of Data Points Tested	Total Time Taken (/sec)
EP2C35	858	0.1
5CGXBC4	862	0.12
5CGTFD9	3595	0.12
10CL025	519	0.11

4.3.2.2 Testing the model

The accuracy of the performance estimation of the model is evaluated using 3 applications. The expected output (ground truth) for these applications is obtained by compiling the applications at every footprint using Quartus. *atax* and *2mm* applications from Polybench [134] and a handwritten synthetic application, *subsys10*, are used for testing. Table 4.5 shows the total number of data points from 3 applications and the total time taken for testing. The results show that predicting F_{max} using the ANN model takes only a fraction of a second.

The test error is calculated similar to the training error above. Table 4.6 shows the individual test error for the 3 test applications, average across all footprint, for each device. Overall, the maximum average error across all applications and devices is less than **4%**.

To evaluate the error values in a greater detail, the percentage errors are plotted as histograms. Figure 4.9 shows error histogram plots for the 3 test applications on these 4 test devices. It can be clearly observed that for the majority of footprints, the error stays at the lower end of the histogram, confirming the overall effectiveness in predicting the performance (F_{max}) using the proposed ANN based technique. Additionally, table 4.7 shows the percentage difference between the maximum and the minimum ground truth F_{max} when the test applications are mapped to all possible footprints on all 4 test devices. According to the table, the percentage F_{max} difference is between 20% and 30% for most of the cases. Therefore, estimation error of less than 4% can be considered as low. The estimation error is also lower than the F_{max} variations that occur due to seed sweeping (5%) [138].

Fig. 4.9 Test Error Percentage Histograms

Table 4.6 Test Error Percentages

Device	atax	2mm	subsys10
	Average Error(%)	Average Error(%)	Average Error(%)
EP2C35	2.56	1.36	2.30
5CGXBC4	3.76	2.43	2.99
5CGTFD9	3.79	1.76	3.63
10CL025	3.35	1.68	3.08

Table 4.7 Percentage Variation of F_{max} on Footprints

Device	atax (%)	2mm (%)	subsys10 (%)
EP2C35	21	19	31
5CGXBC4	24	15	19
5CGTFD9	21	20	27
10CL025	21	20	28

4.3.3 Discussion

While the above results confirm the accurate prediction of F_{max} using the proposed ANN based technique for a given design partition in a footprint, a more interesting application of our approach is to identify the best footprint that provides the highest F_{max} for a given design partition. It is indeed possible to obtain this using the proposed methodology by rapidly predicting the performance at all footprints using our model and then identifying the footprint with the highest F_{max} . This is immensely useful in a modular design methodology, where, instead of storing a pre-placed and routed database for all the partitions for all possible locations, a smaller database could be maintained during incremental compilation of the full design by setting the preservation level of the best footprint set of *post-fit*. It is also noteworthy that even though the proposed work has been described and evaluated using Intel Altera's Quartus-specific terminology, similar toolsets are available in Xilinx environments as well [128]. Hence the proposed methodology is also applicable for Xilinx FPGA based designs.

It should also be noted that, the model is architecture specific and needs to be trained for each device. However, this requirement is common for all the module based FPGA design approaches since the post-fit netlists are indeed architecture specific.

4.4 Summary

Modern FPGAs consist of millions of logic resources allowing hardware designers to map increasingly large designs. However, the design productivity of mapping large designs is greatly affected by the long runtime of FPGA CAD flow. To mitigate it, modular design methodology has been introduced in the past that allows designers to partition large designs into smaller subsystems and compile and test the subsystems individually before assembling them together to complete the compilation process. Automated decision making on placing these subsystems on FPGA, however, is a slow and tedious process that requires a large database of pre-compiled subsystems, which are compiled on a large number of placement positions. In order to accelerate this placement process during modular design, in this chapter an ANN based performance estimation technique that can rapidly suggest the best shape and location for a given subsystem is proposed.

The proposed techniques can estimate F_{max} of a subsystem at any footprint on a FPGA device. Altera Quartus incremental compilation tools have been used for implementing the subsystems while MatLab was used to train the ANN model. Experimental results on legacy as well as state-of-the-art FPGA devices show that the proposed technique can accurately estimate the F_{max} of subsystems with an average error of less than 4%. The techniques can be of immense benefit in modular design methodology in order to reduce the pre-compiled subsystem database sizes by keeping only the best footprint set for a given subsystem while discarding the other footprints.

Chapter 5

Target Architecture-aware Subsystem Mapping

5.1 Introduction

In chapter 3, it was shown that partitioning a large application into several smaller subsystems, while maximizing intra-subsystem communication and minimizing inter-subsystem communication, led to substantial energy savings by leveraging the state-of-the-art CAD tools to compile the relatively smaller subsystems. However, the subsystem generation step, as discussed in chapter 3, is performed under an area constraint. Hence, all high communication activities in a large design may not get encapsulated within subsystems, resulting in a limited number of inter-subsystem links with high communication. A sub-optimal placement of such highly communicating subsystems on FPGA can result in allocation of longer interconnect wires to the links with high inter-subsystem communication, thereby increasing dynamic power dissipation. It is noteworthy that even if the area constraint is relaxed during the subsystem generation step, the resulting larger subsystems reduce the chances of placing the basic computation units (within the subsystem) with high communication in close proximity. This in turn will negate the usefulness of the subsystem generation step.

(a) Graph of Basic Computation Units

(b) Graph of Subsystems

Fig. 5.1 Communication-aware Subsystem Generation

In order to explain the subsystem placement problem, consider a design with 10 basic computation units as shown in figure 5.1a. Here, each node represents a basic computation unit and shows its area requirement on FPGA space and intra-subsystem communication as $[l, m, d][\alpha]$ where l, m, d and α represent CLBs, BRAMs, DSPs and communication within the node respectively. An edge in the graph is a connection between two design units where data flow occurs. The communication between the two basic computation units (no of times the nodes communicate with each other) is indicated on each edge. Applying the subsystem generation technique in chapter 3 on this design results in a network of subsystems as given

Fig. 5.2 Mapping Subsystems on FPGA Without Placement Information

in figure 5.1b. Thereafter, the CAD flow is guided during compilation to group the relevant design units as subsystems. However, the location of these subsystems on FPGA space is decided by the CAD flow that is not sensitive to inter-subsystem communication information. As a result, subsystems with higher communication between them can be mapped far apart. For example, the subsystem network in figure 5.1 could be mapped on target FPGA as shown in figure 5.2(a) and (b). In both cases, subsystems are mapped in a similar manner. However, the distance between subsystems S_3 and S_4 in mapping in figure 5.2(b) is higher than that of in figure 5.2(a). As the inter-subsystem communication between S_3 and S_4 is considerably higher than the other links (as shown in figure 5.1b), power consumption for the mapping in figure 5.2(b) will be higher than the mapping in figure 5.2(a).

At the same time, diversity in resource types and their locations on modern heterogeneous FPGA fabrics make the placement of large applications a complex task [124], where sub-optimal placement decisions cause poor resource allocation as well as longer routing paths resulting in overall performance degradation. To address this challenge, chapter 4 presented a technique for characterizing FPGA devices to aid in the mapping of subsystems. It was also

shown that a characterized device model allows a designer to explore the different shapes and placement locations (referred to as *footprints*) for each subsystem. Different footprints of a subsystem have distinct resource column arrangement despite the similar number of resources. It was shown that some column arrangements are more preferable for a module than the others, which leads to performance variations across different footprints. However, the method to find the final shape and placement for each of the subsystems in the application was not presented in chapter 4.

Furthermore, the increasingly large applications being ported onto FPGA devices [37] [38][39] [40] leads to a large number of subsystems after the subsystem generation step. As will be shown later in Section 5.2.4, runtime of the mapping step increases significantly with increasing number of subsystems. Hence, to ensure scalability of the proposed techniques, a pre-processing step has been proposed as a first step to identify groups of subsystems with high inter-subsystem communication. The output of this step is used as an initial solution in the subsequent mapping step.

Next, a subsystem mapping methodology has been proposed that leverages the device characterization technique proposed in chapter 4, to find the best location and shape for subsystems of an application on target FPGA architecture, while lowering power dissipation on FPGA interconnect fabric which occurs due to higher inter-subsystem communication on longer routing paths. The main contributions of this chapter are:

- A pre-processing step that identifies subsystem with high inter-subsystem communication to identify them as a group to be mapped in close proximity in the subsequent stage.
- An algorithm to explore different *shapes* for each subsystem to effectively utilize the heterogeneity of FPGAs and lower the intra-subsystem power dissipation, without maintaining a pre-compiled module database.
- Analysis of subsystem locking for rapid convergence of subsystem placement solution.

- Automating architecture-aware subsystem mapping in commercial CAD flows.

The rest of the chapter is organized as follows. In section 5.2, the architecture-aware subsystem mapping methodology is discussed with an extensive evaluation. In section 5.3, the proposed methodology is refined for scalability. The chapter is concluded in section 5.4.

Portions of work described in this chapter have been published in [139] and [123].

5.2 Methodology

Fig. 5.3 Different Shapes and Locations for Mapping Subsystems
(a) Footprint variations on FPGA (b) Performance of footprints

First, the steps involve in mapping subsystems on FPGAs are outlined before discussing the motivation for the pre-processing step. As discussed earlier in Chapter 4, each subsystem can be placed at different locations and exhibit varying shapes on a FPGA device, resulting in different performance metrics. The possible locations and shapes of a subsystem are referred

to as the footprints of the subsystem. The concept of footprint variations is illustrated using the following example. In figure 5.1b, a design is represented as a graph of subsystems. Each node represents a subsystem whereas an edge between node represents the existence of physical connection between two subsystems. A node S_i is annotated with $[l_i, m_i, d_i][\alpha_i]$, which represent area requirement by the module as a tuple (required number of LUTs, BRAMs and DSPs) and intra-subsystem communication respectively. Similarly, an edge between S_i and S_j is annotated with $a'_{i,j}$ to show the inter-subsystem communication between respective subsystems. For each subsystem S_i , 4 footprint variations are generated using a characterized device model for Altera EP4CGX50 series device as explained in chapter 4. The footprint variations for each subsystems are shown in figure 5.3(a) that are marked as $fp_{i,j}$, where j represents a footprint variation. For a given subsystem i , all the footprint variations satisfy the area requirement $[l_i, m_i, d_i]$, but they can have different column arrangements within the footprint, depending upon the position and size of the footprint on the FPGA fabric. Footprint variations for each subsystem are tabulated in the figure 5.3(b) along with their estimated performance (in MHz), obtained using the characterized device model. Footprints are marked in different color codes (shown in the legend of figure 5.3) to highlight their preference level, which is analogous to performance. The footprint variation with highest performance is given the preference level 1.

Given the subsystem graph and the variations of footprints for each subsystem, the following outcomes are intended to achieve:

- (i) Select footprints with higher preference level
- (ii) Footprints belonging to subsystems with higher inter-subsystem communication should be closer to each other on FPGA
- (iii) Selected footprints for each subsystem must not overlap in the final solution

If the inter-subsystem communication is negligible, it is profitable to select the non-overlapping footprints with highest possible preference level from all 4 subsystems. However, if the inter-subsystem communication is considerably high, for instance the communication between S_3 and S_4 in figure 5.1b, it is important to make a trade-off between preference level and close proximity.

Figure 5.3(b) also highlights a potential selection of footprints for each subsystem as $fp_{1,4}$, $fp_{2,2}$, $fp_{3,1}$ and $fp_{4,2}$. According to the figure, footprints $fp_{2,2}$ and $fp_{4,2}$ overlap. It is important to note that typical FPGA architectures are uniform in vertical direction. Shifting a footprint along vertical direction does not alter the column arrangement within the footprint, hence the estimated performance does not change. Therefore, it is possible to shift vertically either of overlapping footprints in order to satisfy the non-overlapping requirement. Shifting will also change the distance between subsystems. Therefore, there is another trade-off to be made between resolving overlaps by vertical movement and selecting non-overlapping footprints. If the overlap is resolved by shifting, it is necessary to decide which footprint should be shifted out of two overlapping footprints, further adding to the complexity of the subsystem mapping process.

In addition, the subsystem generation process as discussed in chapter 3 may produce a large number of subsystems for large applications. Since each subsystem has number of footprints variations, the problem size of footprint mapping can be immense and will require prohibitively long runtime to solve. At the same time, large problem size will increase the chances of converging to an undesirable sub-optimal solution. Many studies have demonstrated that the Ant Colony Optimization (ACO) algorithm is capable of finding better solutions when the problem size grows, as compared to other meta-heuristic algorithms, such as Simulated-Annealing (SA) or Genetic Algorithm (GA) [140] [141]. However, ACO takes longer runtime to produce high quality solutions [142]. Therefore, a two-step ACO based

Fig. 5.4 Two Step Approach to Subsystem Mapping

methodology is proposed in this chapter to map subsystems to footprints. It is noteworthy that selecting the best algorithm for subsystem mapping is not the main focus of this methodology.

1. The first phase is a pre-processing step to find the associativity of subsystems of a given subsystem graph, by analyzing inter-subsystem connections and links. For instance, subsystems S_2 , S_3 and S_4 of subsystem graph in figure 5.4a has relatively high inter-subsystem communication. Therefore, these three subsystems can be identified as a group. During this subsystem grouping process, the footprint information

- (various shapes and locations) is not considered in order to limit the problem size. A communication-aware placement technique is used to find the *relative position* of each subsystem to identify the subsystem groups. The proposed Subsystem grouping methodology is discussed below in section 5.2.1.
2. In the second phase, the footprint variations of each subsystem in the subsystem graph are analyzed for selecting the best shape and location to map each subsystem on the target FPGA. The subsystem grouping information obtained in the first phase is used in reducing the search space of deciding the final footprint locations to map subsystems. For instance, consider the 4 footprints lists shown in figure 5.4b. As discussed in chapter 4, a footprint has information of its shape, location on FPGA and the estimated performance. These footprints lists show the horizontal location of each footprint (x) along with the estimated performance. In the phase 1, it is identified that the subsystems S_2 , S_3 and S_4 should be placed in a closed proximity. That grouping information is taken into the consideration in this step, by promoting footprints $fp_{2,2}$, $fp_{3,1}$ and $fp_{4,2}$ (as highlighted in figure 5.4b) whose horizontal location x is relatively close to each other. Footprint for S_1 ($fp_{1,4}$), on the other hand, is allowed to placed at slightly distant location ($x = 16$). The details of this proposed Subsystem mapping technique is presented later in section 5.2.2.

5.2.1 Inter-subsystem Communication-aware Subsystem Grouping

In this section, the phase 1 (subsystem grouping) of the proposed two-step approach is discussed in detail.

5.2.1.1 Problem Formulation

There are two inputs to the subsystem grouping methodology:

- (i) A subsystem graph $G(V, E)$, where V is the set of vertices which represents subsystems $V = \{S_1, S_2, \dots, S_N\}$ and E is the set of edges which represent inter-subsystem connections. Each vertex $S_i \in V$ is annotated with the area requirement tuple $A_i(l_i, m_i, d_i)$ where l_i , m_i , and d_i represent the required number of CLBs, BRAMs and DSPs respectively, where as an edge $e_{S_i, S_j} \in E$ has a cost value $\alpha_{i,j}$ to represent the inter-subsystem communication between subsystems S_i and S_j .
- (ii) Number of columns (W) and number of rows (H) in the target FPGA along with FPGA column arrangement $F = f_1, f_2, \dots, f_W$, where $f_i = \{l|m|d\}$ represents the resource type of column i on target FPGA (l , m , and d represents CLB, BRAM and DSP respectively).

Given a subsystem graph $G(V, E)$, target FPGA parameters W , H , and the target FPGA resource column lists F , the subsystem grouping problem is to find a horizontal location x_{S_i} for each subsystem $S_i \in V$ such that the rectangular regions $L_v(x_v, y_v, w_v, h_v)$ do not overlap on the FPGA space for each subsystem v where (x_v, y_v) are the FPGA coordinates of the region and (w_v, h_v) are the width and the height of the rectangular region. L_v should meet the following requirements:

1. x_{S_i} is the lower left corner of a rectangular region on FPGA that satisfy the area requirement of the respective subsystem A_{S_i}
2. All rectangular regions are non-overlapping and must fit within the FPGA limits W and H

The subsystem mapping problem is modeled into a Traveling Salesman Problem (TSP) as shown in figure 5.5. First the subsystem graph is rearranged such that the subsystems require only one FPGA resource type. For instance, subsystem S_1 of graph in 5.1b is divided into two subsystems as shown in figure 5.5(a) such that new subsystem S_1 requires only CLBs ($l_{S_1} = 96$) and subsystem $S_{1.1}$ requires only BRAMs. However, in order to maintain S_1 and

Fig. 5.5 (a) Divide Subsystems based on Resource Type (b) Traversal on Point Map

$S_{1.1}$ as one subsystem, very high inter-subsystem communication N to the edge $e_{S_1, S_{1.1}}$ is introduced. In figure 5.5(b), a sample traversal is shown on a point map. Each subsystem has W number of points, and each point represents the event of placing the lower left corner of the subsystem at respective horizontal coordinate, x . For each subsystem, traversal selects a point from the list. However, the points also represent the respective resource column type. For instance, let the FPGA column arrangement be $F = l, m, l, l, m, \dots, l$. Therefore, the points in $x = 2$ are BRAM type. Hence, the subsystems that requires only BRAMs (ex. $S_{1.1}$, $S_{2.1}$) are allowed to choose points at $x = 2$ and $x = 5$.

5.2.1.2 Subsystem Grouping using Ant Colony Optimization

The proposed algorithm for subsystem grouping, and the methodology to identify the subsystems that should be placed in close proximity, are discussed in this sub-section. However, it is noteworthy that the exact location and shapes on target FPGA is not considered in this step. The TSP modeled above is solved using Ant Colony Optimization based algorithm. The

proposed technique uses a pheromone table similar to figure 5.5(b). A cell in the pheromone table, $\tau_{S_i,x}$ represents the likelihood of selecting x column on target FPGA to place the subsystem S_i . The pseudo-code of the proposed ACO based placement methodology is given in algorithm 2. In the main section of algorithm 2, $RUN_ACO1()$, number of solutions are generated by artificial ants in each iteration (line 8). The best solution in each iteration Q_{curr_best} gets the change to update the pheromone table by increasing the pheromone entries relevant to the solution (lines 14-17). In addition, if the Q_{curr_best} is better than the overall best solution Q_{hist_best} , pheromone table is updated with the solution again (line 18-21). Pheromone updates will increase the likelihood of choosing better traversal in subsequent iterations. Once the *stopping_criteria* is met (maximum number of iterations or very low changes in Q_{hist_best}), the algorithm outputs the best solution, Q_{hist_best} .

Algorithm 2 Communication-aware Subsystem Grouping

```

1: procedure RUN_ACO1(G,F)
2:   INIT()
3:    $Q_{hist\_best} \leftarrow Q_{init}$ 
4:    $hist\_best \leftarrow MAX$ 
5:   while stopping_criteria == FALSE do
6:      $Q_{curr\_best} \leftarrow Q_{init}$ 
7:      $curr\_best \leftarrow MAX$ 
8:     for each  $ant \in ANTS$  do
9:        $(seq_{ant}, x_{ant}) \leftarrow TRAVERSE\_ANT(G)$ 
10:       $y_{ant} \leftarrow SELECT\_Y(x_{ant}, G)$ 
11:      if IS_VALID( $y_{ant}$ ) == FALSE then
12:        continue
13:       $cost_{ant} \leftarrow COST(G, x_{ant}, y_{ant})$ 
14:      if  $cost_{ant} < curr\_best$  then
15:         $curr\_best \leftarrow cost_{ant}$ 
16:         $Q_{curr\_best} \leftarrow (seq_{ant}, x_{ant})$ 
17:        UPDATE_PHEROMONE( $seq_{ant}, x_{ant}$ )
18:      if  $cost_{ant} < hist\_best$  then
19:         $hist\_best \leftarrow cost_{ant}$ 
20:         $Q_{hist\_best} \leftarrow (seq_{ant}, x_{ant})$ 
21:        UPDATE_PHEROMONE( $seq_{ant}, x_{ant}$ )
22:      DECAY_PHEROMONE
23:   return  $Q_{hist\_best}$ 

```

Fig. 5.6 Shape of Subsystems

Pheromone Table Initialization In the beginning of the algorithm, all subsystems have equal probability to select any x point that matches its resource type. Therefore, all the entries in the pheromone table are initialized to a small value in $INIT()$ (line 2). This value is empirically found to be 0.2.

Traversing In each iteration, a set of computing agents (artificial ants) traverse through the point map (line 9). Each ant traverses through all the subsystems exactly once by selecting point from the point map. A random number is used to select the subsystem to traverse. Point selection is done based on the pheromone values $\tau_{S_i,x}$. Each ant produces a solution with sequence of subsystems (seq_{ant}) and respective x coordinate for each subsystem x_{ant} . Sequence x_{ant} only provides the horizontal coordinate of each subsystem on target FPGA, and the vertical coordinate (y) of all the subsystems is assumed to be 1. The following technique is used to interpret the shape and the vertical position of each subsystem.

Shape of Subsystems As discussed earlier, subsystems requiring multiple resource types (CLBs and BRAMs) are split into sub-modules in a way that each sub-module contains only one resource type. For instance, as shown in figure 5.5(a), subsystem S_3 with area

requirement $A_{S_3} = (96, 8, 0)$ is divided into two with area requirements of $A_{S_3} = (96, 0, 0)$ and $A_{S_{3,1}} = (0, 8, 0)$ respectively. In this pre-processing phase, finding the shape on FPGA space for a subsystem is not the objective. It has been shown in [87] that shapes with aspect ratios close to 1, help in reducing the wire length within a subsystem. Therefore, the shape of subsystems with CLBs (ex. S_3) is set to have a height(h) to width(w) ratio close to 1 such that the resource requirement is just satisfied. Figure 5.6 shows how the shapes for subsystem S_3 and $S_{3,1}$ are derived. The height and width of S_3 should ideally be 10 in order to have ratio as 1 while satisfying A_{S_3} . However, setting width 10 will include a BRAM column, hence A_{S_3} is not met. Therefore, the width of S_3 is set to 11 to satisfy S_3 . Subsystems with other resources (ex. $S_{3,1}$) are considered as a single column shape.

Resolve Overlaps Selecting a shape as described above may result in overlaps between subsystems. Hence, the affected subsystems are shifted vertically to resolve the overlaps. For instance, let two subsystems S_a and S_b be placed at $x_a = 10$ and $x_b = 15$. If the shape of the subsystems are inferred to be $w = 10$ and $h = 5$ for both cases, an overlap is created. If S_b appears in the *seq_{ant}* before S_a , subsystem S_a is shifted by 5, resulting $y_a = 6$ and $y_b = 1$. In this technique, the vertical position is obtained for all the subsystems (line 10).

Congestion Model Routability evaluation is an important step in a placement algorithm. For instance, placing a large number of components in a smaller area could lead to an excessive requirement for routing resources. Typical routing algorithms avoid highly congested areas and therefore may use longer wires to connect components, leading to the degradation of quality of results (performance and power). In the proposed approach, a bounding box based congestion analysis is incorporated to estimate the routing congestion similar to [72]. In this model, the FPGA space is considered as a 2D array known as a ‘congestion map’. All congestion map elements are initialized to zero. For each inter-subsystem connection in figure 5.1b, a bounding box can be formed as in figure 5.7(a) (Note that, all the bounding

Fig. 5.7 Congestion Model

box are not shown, for ease of explanation). Next, congestion map elements relevant to each bounding box is incremented. Therefore overlapping bounding boxes, reflect high congestion regions. The congestion map relevant to the subsystems in figure 5.7(a) is shown in 5.7(b). The congestion map is updated in each iteration of the algorithm.

Cost Function and Constraints In order to minimize routing power dissipation, this work intends to avoid longer routing wires, especially for high inter-subsystem communication links. Therefore the objective of this technique is to minimize the cost function in equation 5.1. Here, l_e is half perimeter wire length for the bounding box enclosing the subsystems relevant to edge e , whereas α_e is inter-subsystem communication. While minimizing the cost function, a routing congestion parameter C is also considered. For a valid placement solution, its maximum congestion derived above must be less than empirically evaluated C .

$$COST(s) = \sum_{e \in E} l_e \cdot \alpha_e \quad (5.1)$$

Pheromone Update and Decay As discussed earlier, pheromone table entries ($\tau_{S_i,x}$) related Q_{curr_best} and Q_{hist_best} are increased by 10% in order to promote better solutions in the subsequent iterations. It is also noteworthy that ACO mimics the evaporation of natural pheromones such that after each iteration, all the pheromone values in the pheromone table is reduced by a very small amount.

seq_{ant}	x_{ant}	w	h	y
S4	45	9	7	1
S3	59	12	10	1
S3.1	60	1	8	1
S4.1	46	1	7	1
S1	10	11	10	1
S2.2	46	1	8	8
S2	43	8	7	8
S1.1	12	1	3	1

(a) Solution Creation using the Best Traverse Path

(b) Display the Solution on Target FPGA

Fig. 5.8 Example of a Solution from Subsystem Grouping Technique

5.2.1.3 Output of Subsystem Grouping

At the end of the ACO algorithm, each subsystem S_i is assigned a coordinate (x_i, y_i) , indicating its position on the target FPGA. A standard data clustering technique such as Mean Shift Clustering [143] is used next to identify subsystem groups. For example, seq_{ant} and x_{ant} in figure 5.8a shows a traversal solution for the subsystem graph in figure 5.1b. The traversal seq_{ant} and x_{ant} are used in extracting the shape (w, h) and the vertical coordinate (y) of each subsystem using the methods explained earlier. Figure 5.8b shows the created solution on target FPGA. Using the Mean Shift Clustering algorithm, the (x, y) coordinates of each

subsystem in the solution is used to identify the subsystem groups i.e. Group $\{S_1\}$ and Group $\{S_4, S_3, S_2\}$ for this sample application.

The pre-processing step discussed in this section is utilized in the next section for the final subsystem mapping phase.

5.2.2 Mapping Subsystems

The phase 2 of the proposed methodology is discussed in this sub-section.

5.2.2.1 Problem Formulation

There are three inputs required by our methodology:

- (i) A graph $G = \{V, E\}$, where V is the set of vertices which represents subsystems $V = \{S_0, S_1, \dots, S_N\}$ of a given design, and E is the set of edges which represents connections between subsystems. Each vertex $S_i \in V$ has an area requirement of $A_{S_i} = \{l_{S_i}, m_{S_i}, d_{S_i}\}$ and intra-subsystem communication α_{S_i} , whereas an edge $e_{i,j} \in E$ has a cost value $\alpha_{i,j}$ to represent inter-subsystem communication between subsystems S_i and S_j .
- (ii) Subsystem Groups $GP = \{gp_0, gp_1, \dots, gp_m\}$ (identified in section 5.2.1), where each group gp_i subset of nodes $S_i \in G$
- (iii) Set of Footprint lists $L = \{l_{S_0}, l_{S_1}, \dots, l_{S_N}\}$, where each list l_{S_i} contains n number of variations of footprints $l_{S_i} = \{fp_{i,0}, fp_{i,1}, \dots, fp_{i,n}\}$ of subsystem S_i . A footprint can be represented as a tuple $fp_{S_{i,j}} = \{x_{i,j}, w_{i,j}, h_{i,j}, f_{i,j}\}$, where x is the bottom left horizontal coordinate of the footprint on FPGA, w and h are width and height of the footprint and f is the estimated performance of the footprint which can be considered as the preference level of the footprint. Footprint list l_{S_i} is sorted in descending order based on performance $f_{i,j}$.

Here, as discussed earlier, the vertical coordinate y of the footprint can be safely ignored due to the uniformity of FPGA architecture along the vertical direction. The footprint lists are generated using the target FPGA device characterization model, as explained in Chapter 4.

Given a design as a subsystem graph $G = \{V, E\}$, subsystem groups G and footprint list set $L = \{l_{S_0}, l_{S_1}, \dots, l_{S_N}\}$, the subsystem mapping problem is to select a footprint $fp_{i,j}$ from each footprint list l_{S_i} . The footprint selection should minimize a cost function while resolving footprint overlaps. The above problem is formulated as a variant of Traveling Salesman Problem (TSP). First, as shown in figure 5.9b, a *complete graph* $G' = \{V', E'\}$ is created using the graph G . Each node in V' is same as V of G and has a list of footprints that are associated with the respective subsystem. An edge $e'_{i,j}$ in E' represents an event of choosing subsystem S_j after S_i . Therefore, there is an edge from each node to all the others nodes in the complete graph. An edge e' is associated with a cost where lower cost indicates higher priority. An addition, footprints in all the nodes are also carry a cost where footprints with higher preference carry a low cost. The TSP variant aims to (i) find a route to visit each node in G' and (ii) select one footprint from each node to form a set of footprints with minimum cost. The order in which the footprints are placed on FPGA is determined by the sequence of the nodes traversal.

5.2.2.2 Subsystem Mapping using Ant Colony Optimization

The TSP problem modeled in the section above is solved using the popular Ant Colony Optimization (ACO) technique. However, the TSP variant is significantly different from the typical TSP problem because of the costs associated with the nodes in addition to the edges. Therefore, the traditional ACO algorithm needs to be modified in order to be applied to our problem. In the proposed ACO technique, two types of pheromones (probabilistic values) have been used to represent the edge and node costs.

Algorithm 3 Ant Colony Optimization based Algorithm for Subsystem Mapping

```

1: procedure RUN_ACO(  $G, G', L, GP$  )
2:   INIT()
3:    $Q_{hist\_best} \leftarrow Q_{init}$ 
4:    $hist\_best \leftarrow MAX$ 
5:   while  $stopping\_criteria == FALSE$  do
6:      $Q_{curr\_best} \leftarrow Q_{init}$ 
7:      $curr\_best \leftarrow MAX$ 
8:     for each  $ant \in ANTS$  do
9:        $seq_{ant} \leftarrow TRAVERSE\_ANT(G', GP)$ 
10:       $fps_{ant} \leftarrow SELECT\_FOOTPRINTS(G', L, GP, seq_{ant})$ 
11:       $fps_{ant} \leftarrow RESOLVE\_OVERLAPS(fps_{ant})$ 
12:      if  $fps_{ant}.isValid() == FALSE$  then
13:        continue
14:       $cost_{ant} \leftarrow COST(G, fps_{ant})$ 
15:      if  $cost_{ant} < curr\_best$  then
16:         $curr\_best \leftarrow cost_{ant}$ 
17:         $Q_{curr\_best} \leftarrow fps_{ant}$ 
18:         $UPDATE\_PHEROMONE(G', seq_{ant}, fps_{ant})$ 
19:      if  $cost_{ant} < hist\_best$  then
20:         $hist\_best \leftarrow cost_{ant}$ 
21:         $Q_{hist\_best} \leftarrow fps_{ant}$ 
22:         $UPDATE\_PHEROMONE(G', seq_{ant}, fps_{ant})$ 
23:       $DECAY\_PHEROMONE(G')$ 
24:   return  $Q_{hist\_best}$ 

```

```

25: procedure TRAVERSE_ANT( $G', GP$ )
26:    $rand \leftarrow \text{RAND}(1, N)$ 
27:    $seq.append(S_{rand})$ 
28:    $G'' \leftarrow G'$ 
29:   for  $i = 1$  to  $N-1$  do
30:      $gp \leftarrow \text{GET\_GROUP}(seq[i], GP)$ 
31:      $\text{MODIFY\_PHEROMONES}(G'', gp)$ 
32:      $sum_{\tau} \leftarrow \sum \tau_{seq[i], j} \forall j \in G'', j \notin seq$ 
33:      $rand \leftarrow \text{RAND}(0, sum_{\tau})$ 
34:     for all  $j \in G'', j \notin seq$  do
35:        $cummu_{\tau} \leftarrow cummu_{\tau} + \tau_{seq[i], j}$ 
36:       if  $cummu_{\tau} \geq rand$  then
37:          $seq.append(S_{rand})$ 
38:         break
39:   return  $seq$ 
40:
41: procedure SELECT_FOOTPRINTS( $G', L, GP, seq_{ant}$ )
42:    $L' \leftarrow L$ 
43:   for all  $s \in seq_{ant}$  do
44:      $gp \leftarrow \text{GET\_GROUP}(s, GP)$ 
45:      $sum_{\tau'} \leftarrow \sum \tau'_{s, j}, 1 \leq j \leq n, j \in L'_s$ 
46:      $rand \leftarrow \text{RAND}(0, sum_{\tau'})$ 
47:     for  $j = 1$  to  $n$  do
48:        $cummu_{\tau'} \leftarrow cummu_{\tau'} + \tau'_{s, j} j \in L'_s$ 
49:       if  $cummu_{\tau'} \geq rand$  then
50:          $fps_{ant}.append(fp_{s, j})$ 
51:          $\text{MODIFY\_SECOND\_PHEROMONE}(L', gp, fp_{s, j})$ 
52:         break
53:   return  $fps_{ant}$ 
54:
55: procedure RESOLVE_OVERLAPS( $fps_{ant}$ )
56:   for all  $fp_j \in fps_{ant}$  do
57:     for all  $fp_i \in fps_{ant} \mid i < j$  do
58:       if  $fp_i.overlaps(fp_j) == TRUE$  then
59:          $y_j \leftarrow y_j + \text{get\_vertical\_shift}()$ 
60:          $i = 1$ 
61:   return  $fps_{ant}$ 
62:
63: procedure UPDATE_PHEROMONE( $G', seq_{ant}, fps_{ant}$ )
64:   for  $i = 1$  to  $N-1$  do
65:      $\tau_{i, i+1} \leftarrow \tau_{i, i+1} + \epsilon_{\tau}$ 
66:   for all  $fp \in fps_{ant}$  do
67:      $\tau'_{fp} \leftarrow \tau'_{fp} + \epsilon_{\tau'_{fp}}$ 

```

(a) Example of Subsystem Grouping

(b) Complete Graph G' Respective to Group 1

Fig. 5.9 Building a Complete Graph for Groups

Algorithm 3 provides a pseudo code of the proposed ACO-based footprint selection algorithm. $RUN_ACO()$, the main section of algorithm 3, solves the problem in several iterations. In each iteration, a new population of artificial ants, i.e. computing agents, are generated (line 8) and all ants in the population produce a solution. The algorithm allows some ants, who could update the best solution in the current population Q_{curr_best} or who could update the overall best solution Q_{hist_best} , to deposit some pheromone on the corresponding nodes and edges along the solution path. Large amount of pheromones promotes respective path to be used in the next iteration. Once the termination condition is satisfied, Q_{hist_best} is returned as the solution to the footprint placement problem (line 24).

Pheromone Initialization To promote solutions with lower cost, pheromones are introduced to the edges and the nodes as shown in figure 5.9b. An edge in G' , $e'_{i,j}$, carries a pheromone value $\tau_{i,j}$ which shows the likelihood of choosing S_j after S_i . In addition to pheromone representation of typical ACO, a second degree pheromone list is also included for each node in G' , which is represented as $T'_{S_i} = \{\tau'_{fp_{i,1}}, \tau'_{fp_{i,2}}, \dots, \tau'_{fp_{i,n}}\}$. Second degree pheromone $\tau'_{fp_{i,j}}$ represents the likelihood of choosing respective footprint $fp_{i,j}$ for the subsystem S_i . In other words, lists elements in T'_{S_i} represents the pheromone level related to the respective list entry in l_{S_i} .

In $INIT()$ (line 2), the complete graph G' from the input design G is created, and all the pheromones are initialized. The edge pheromones $\tau_{i,j}$ are initialized to have the same value in the beginning. However, the second degree pheromone values $\tau'_{i,j}$ are interpolated based on the performance of the respective footprint. A footprint variation with higher performance (close to preference level 1) is promoted for the selection. The highest and lowest values for $\tau'_{i,j}$ are respectively 0.8 and 0.2 and are found emphatically for better convergence. Therefore, the following initialization is used;

$$\tau'_{i,j} = 0.2 + 0.6 \times \frac{f_{i,j} - f_{i,n}}{f_{i,1} - f_{i,n}} \quad (5.2)$$

It should be noted that l_{S_i} is an ordered list where $fp_{i,1}$ and $fp_{i,n}$ are the footprint with highest and the lowest performance, respectively.

Solution Construction Each ant in the current population (iteration) traverses through each node of G' exactly once and the order of the nodes is referred to as a sequence. Procedure $TRAVERSE_ANT()$ shows how an ant is traversed through all the nodes based on the edge pheromone values ($\tau_{i,j}$). Here, choosing the first node from G' is a random process and chosen node is appended into the sequence list seq (lines 26-27). The grouping information is applied in selecting the next nodes to the sequence. A temporary complete graph, G'' is

updated in every node selection, such that the pheromone values from the last selected node to the nodes from the same group is 50% higher than the original value. For instance selecting S_3 of figure 5.1b to the sequence will increase $\tau_{3,2}$, $\tau_{3,4}$ in G'' as S_2 and S_4 subsystems are in the same group as S_3 . The sum of the pheromone values that are on the edges from the last node of seq to the nodes that are not a part of seq yet, is calculated. A cumulative list of pheromones of the nodes which are not in the list is used for selecting the next node in a random fashion. Nodes which are connected to the last node of seq by an edge with higher pheromone value get a higher chance of being selected (lines 32-37). This process is continued until the seq is build with all the nodes in G' .

Next, the algorithm selects a footprint for each node S_i in seq , from the respective footprint list l_{S_i} . The same technique as the node selection for seq is used in *SELECT_FOOTPRINTS()* as well. However, the random selection of footprints from list l_{S_i} is based on the cumulative spectrum of secondary pheromone $\tau_{i,j}$ (lines 45-50). A footprint promoting technique is used based on the selected footprint for a subsystem. For instance, let the algorithm to choose footprint $fp_{3,10}$ for the subsystem S_3 . According to the grouping information, the second degree pheromone levels are updated in the footprint lists for S_2 and S_4 , such that, pheromones associated with the footprints that are closer to the vertical coordinate of $fp_{3,10}$ are increased by 50%. This selection is repeated until a footprint from each node in seq is chosen and appended to the footprint list of the current ant fps_{ant} . This list is an ordered list similar to seq , where the footprint at the front of the list has the highest priority for being placed on FPGA.

The footprint sequence fps generated by the above process may include overlapping footprints. Therefore, a simple technique is utilized in *RESOLVE_OVERLAPS()* to resolve such overlaps based on the priority level of footprints. A footprint fp is selected from the list fps and it is checked for overlaps only with the footprints whose priority level is higher than fp (lines 56-58). For any overlap detection, the overlap is resolved by moving fp

vertically (line 59). The new position can still be overlapping with another footprint of higher priority. Therefore, the fp should be rechecked from the highest priority footprint (line 60). In order to deal with extreme conditions, where the footprints may be shifted beyond the upper vertical border of the FPGA, such cases are identified as invalid footprint sequences and discarded by the proposed algorithm (lines 12-13).

Cost Function Each valid footprint sequence is evaluated next (line 14 of Algorithm 3). According to equation in chapter 2.3.1, the effective capacitance of nets with higher toggle rate (communication) must be reduced. The capacitance could be minimized by allocating shorter routing wires for such nets. Hence, during the placement methodology, the modules with higher inter-subsystem communication are placed at a close proximity so that during the routing step of CAD flow will promote the links to have shorter routing wires. To achieve our main goal, the algorithm minimizes the following cost function.

$$cost = \sum_{e \in E} \alpha l_e \times (d_e)^k \quad (5.3)$$

where αl_e represents the inter-subsystem communication of the edge e of the graph G and d_e represents the distance between two subsystems associated with the edge e . d_e is calculated as the half perimeter wire-length (HPWL) of the bounding box which encloses both footprints belongs to two subsystems associated with e . Typically, the numerical value of inter-subsystem communication, αl_e , is much greater than that of distance values, d_e . Therefore, a distance weight parameter, k , is employed to balance the numerical values. The value for k is empirically evaluated as 1.6, using various communication and distance values of many benchmark applications.

Pheromone Update and Decay Based on the cost of fps_{ant} , some of the ants get the chance to update the best solution in the current ant population Q_{curr_best} (lines 15-16 of Algo-

rithm 3) as well as to update the best solution that has been observed so far Q_{hist_best} (lines 19-20). The pheromone values are also updated upon an occurrence in Q_{curr_best} or Q_{hist_best} in order to increase the probability of ant choosing better paths. In $UPDATE_PHEROMONES()$, the pheromone values $\tau_{i,j}$ of the solution path edges of seq_{ant} are increased by ϵ_τ , and the respective secondary pheromones of the footprint entries in fps_{ant} by $\epsilon_{\tau'}$. The increment values are given in equations 5.4 and 5.5. The first value is equal to the half of the average edge pheromone values in G' and the latter is equal to half of the average secondary pheromone values in the pheromone list respective to the footprint.

$$\epsilon_\tau = \frac{1}{2} \times \sum_{e' \in E'} \tau_{e'} / (N \times (N - 1)) \quad (5.4)$$

$$\epsilon_{\tau'_{fp}} = \frac{1}{2} \times \sum_{\forall fp_{i,j} \in T'_{S_i}} \tau'_{i,j} / N \text{ where } fp \in l_{S_i} \quad (5.5)$$

After all the ants in the current population finish their traversal, all the edge pheromone values are reduced by a very small constant to mimic the evaporation of pheromones (line 23). In the next iteration, a new ant population is created and all the aforementioned steps are repeated. This process is continued until the stopping criteria, i.e. maximum number of iterations, is satisfied.

5.2.3 CAD Tools Assisted Mapping of Subsystems onto Target FPGA

In this section, the steps involved in implementing the proposed subsystem mapping methodology on a target FPGA device are discussed. For implementation and evaluation of the proposed methodology, Intel's(Altera) Quartus CAD flow [127] is exploited. It should be noted however that an equivalent approach could also be employed with the Xilinx or any other FPGA CAD tools [144].

Subsystem mapping methodology requires two inputs as shown in figure 5.10.

1. A subsystem graph G : Each node in the graph represents a subsystem, and indicates a list of basic computation units that belong to the subsystem. Edges represent inter-subsystem connections and are annotated with the respective inter-subsystem communication frequency. The subsystem generation methodology discussed in chapter 3 can be used to obtain G .
2. Footprint lists L : A list of footprints for each subsystem in G . Each footprint carries parameters in the tuple $\{x, w, h, f\}$, where x represents the footprint's horizontal location, w, h represent the physical dimensions and f represents the estimated F_{max} of the footprint. The footprint lists can be generated using the methodology discussed earlier in chapter 4.

As discussed above in Section 5.2.1, the proposed methodology first pre-processes (in step 1) the input subsystem graph G and produces the subsystem groups GP . Subsystem groups contain a set of subsystems that have high inter-subsystem communication. Next, (in step 2) the subsystem mapping methodology discussed in Section 5.2.2 selects the footprints with highest possible performance for each subsystem in G , while minimizing the distance between footprints that have high inter-subsystem communication between respective subsystems. The output of the methodology is a sequence of non-overlapping footprints, Q . For example, a possible subsystem mapping solution for the the subsystem graph in figure 5.1b can be given as the footprint sequence $[fp_{1,4}, fp_{2,2}, fp_{3,1}, fp_{4,2}]$ as shown in figure 5.3. The subsystem information (basic computation units in each subsystem) are combined with footprint tuple information (x, y, w, h) from Q . It is noteworthy that the vertical location of the footprint (y) is obtained by vertical shifting of footprints as discussed in section 5.2.

The subsystem information is provided to the *LogicLock*[127] floor-planning feature in the Altera Quartus tool. LogicLock enables the designer to restrict a particular section on FPGA to map a set of RTL entities. In chapter 3, subsystems are implemented on target

Fig. 5.10 Implementation of Subsystem Mapping Methodology

FPGA by providing only the names of each subsystem's basic computation units (RTL entities) to the LogicLock. The location and the shape of the LogicLock was decided by the Quartus compiler. However, a footprint sequence Q as discussed in Chapter 4 includes both shape and placement information. Therefore, the LogicLock can be guided with the physical shape and location on target FPGA in order to map subsystems, eliminating the need for the CAD tools to decide the exact location.

5.2.4 Performance Evaluation

In this section, the effectiveness of the proposed two-step subsystem mapping methodology is evaluated. The evaluation is done by measuring power dissipation of the final design after the compilation, as shown in figure 5.10. Power values are obtained using Quartus Powerplay power estimator [133]. A gate level simulation is performed for the placed and routed design to extract the signal toggle details. This information is fed into Powerplay tool to ensure most accurate power estimation.

The performance (F_{max}) of the mapping solutions given by the proposed subsystem mapping methodology is also evaluated. The performance is extracted using the compilation reports generated during the CAD flow.

5.2.4.1 Comparison

The effectiveness of our methodology in lowering power dissipation in large applications is analyzed as follows. The routing power dissipation of benchmark applications, obtained by the proposed methodology, is compared against the routing power obtained by the following techniques:

1. Quartus: The first comparison is made against the commercially available Quartus tool version 16.1. Here, the design is compiled using the Quartus CAD flow that does not consider any communication information of the design.
2. C3: Second work-flow used for comparison is based on the techniques proposed earlier in chapter 3. In this technique, subsystems created using a communication-aware design partitioning technique is used to minimize power dissipation through minimization of intra-subsystem communication. This methodology does not produce the placement and shape of the subsystem mapping on target FPGA. These parameters are decided by Quartus flow.

In total, 8 benchmark applications are used for the evaluation. The test applications are created as discussed in chapter 3.3.2 to demonstrate the properties of large applications, using existing benchmark suites such as Polybench [134]. The methodology is also tested on three different FPGAs from two families: EP4CGX50 and EP4CGX110 from Cyclone 4 family, and EP4SGX70 from Stratix 4 family.

Secondly, the effectiveness of the ACO-based subsystem grouping technique (step 1) proposed in section 5.2.1 in pre-processing solutions, is evaluated. The runtime of subsystem mapping methodology with and without incorporating the subsystem grouping is compared. Finally, the performance (F_{max}) of designs with subsystems is compared against the performance achieved by default Quartus compilation without subsystems.

5.2.4.2 Results and Discussion

Table 5.1 shows the percentage reduction in routing power achieved using our methodology with respect to default Quartus compilation. The proposed methodology outperforms Quartus by a significant margin, reducing routing power on average by over 15%. The reduction ranges from 9.6% to as high as 25.8%. This has been achieved by effectively ensuring shorter interconnect wires for high communication links as well as considering the heterogeneity of the FPGA.

In addition, figure 5.11 also shows an average improvement of about 11.0% (ranging 6.9% to 16.0%) over the communication-aware subsystem generation technique in chapter 3. The subsystem generation technique, *only* considers the intra-subsystem communication. However, depending on the applications, there can still be links with high inter-subsystem communication. Hence, inter-subsystem communication-aware subsystem placement, as proposed in this chapter, is critical to achieve further power savings.

It is important to note that the proposed methodology does not change application characteristics or the compilation flow. Hence, the FPGA resource consumption as well as

static power consumption remain unchanged. It has been observed that the routing power is one of the dominant component in the total power dissipation for most applications. For instance, routing power dissipation of all but one benchmark applications tested here, ranges from 33% to 45% of total power dissipation. Only the *gesummv'* application has a relatively low routing power dissipation of about 15% of its total power. On average, the proposed methodology has lowered the total power reduction by 6% as shown in the last column of Table 5.1.

Table 5.1 Routing Power Reduction over Quartus CAD Flow

Application	Device	Percentage Power Reduction	
		Routing Power	Total Power
Synth1	EP4CGX50	17.0%	5.9%
Synth2	EP4CGX50	25.8%	6.8%
Synth3	EP4CGX50	18.9%	6.2%
atax/	EP4CGX110	10.6%	6.7%
bicg/	EP4CGX110	12.0%	6.3%
gesummv/	EP4SGX70	9.6%	2.7%
cholesky/	EP4CGX110	10.9%	6.3%
symm/	EP4CGX110	10.7%	4.7%

Fig. 5.11 Routing Power Reduction Over the Methodology in Chapter 3

Next, the effectiveness of the proposed subsystem grouping strategy is discussed. In table 5.2, the runtime of the proposed methodology is presented with and without integrating the subsystem grouping technique in section 5.2.1. The average runtime of our footprint mapping methodology without grouping technique is 759 seconds ranging from 491 to 1007 seconds. Average runtime of the two step methodology (with grouping) is considerably decreased to 214 seconds, by over 70% when the solutions obtained from subsystem grouping technique is used in reducing the subsystem mapping problem size. As shown in figure 5.12, integrating grouping information in subsystem mapping has not changed the effectiveness of the proposed footprint mapping in reducing routing power, despite the considerably low runtime.

Table 5.2 Runtime Analysis of ACO Algorithm

Application	Runtime of the Technique (/sec)		Percentage Reduction
	Mapping w/o Grouping	Mapping w Grouping	
Synth1	1007	254	74.8%
Synth2	969	235	75.7%
Synth3	658	193	70.7%
atax'	491	170	65.4%
bicg'	669	204	69.5%
gesummv'	795	212	73.3%
cholesky'	693	218	68.5%
symm'	824	221	73.2%

Figure 5.14 shows the performance of applications achieved using default Quartus compilation, and using our subsystem mapping methodology. It is evident that our methodology improves F_{max} while reducing power dissipation. Test applications that are chosen for evaluation demonstrate the qualities of large application, resulting default Quartus compilation to produce solutions with degraded performance and power. On the other hand, typical MDM techniques discussed in chapter 2 improves the design productivity but inferior in performance as compared to the non-MDM flows. The reason for low performance is because creating modules (subsystems) removes the global information of the application, hence the

Fig. 5.12 Effect of Grouping Technique on Routing Power

Fig. 5.13 Routing Power Reduction due to Subsystem Mapping over Quartus

Fig. 5.14 Effect of Grouping Technique on Performance (F_{max})

optimization across subsystems cannot be performed. However, the proposed methodology uses inter-subsystem connection details to preserve overall performance, while optimizing power dissipation.

5.3 Subsystem Locking for Rapid Convergence

The ACO based technique discussed in section 5.2 finds a set of footprints by selecting one footprint from each subsystem footprint list. As the application size grows, the number of subsystems also increases, causing the ACO algorithm to consider a large number of footprint list entries. Typically, ACO requires long runtime to converge to an optimal solution and the probability of converging to a non-optimal solution increases with the problem size [142].

Figure 5.15 shows an example for converging to a sub-optimal mapping solution with an FPGA floorplan with mapped subsystem footprints. According to the figure, footprints of subsystems S_6 to S_{12} are placed tightly as a group while the rest of the footprints are placed away from this group. This limits the opportunity for global optimization to reduce the overall communication for the entire application. Therefore in this section, a technique

that improves the quality of the solution of ACO based subsystem mapping methodology is discussed.

Fig. 5.15 Suboptimal Mapping Solutions

As discussed in section 5.2.4, it is evident that the proposed technique efficiently reduces power dissipation. It is achieved by mapping subsystems with high inter-subsystem communication into nearby footprints. For instance, figure 5.16(a) shows the subsystem graph of a sample application. The respective footprint mapping solution is shown in figure 5.16(b). As highlighted (A and B) in the floorplan, some of the footprints are mapped very closely because of very high inter-subsystem communication as shown in the graph. The ability of the proposed technique to map subsystems with high inter-subsystem communication onto closer footprints is exploited by running the ACO algorithm iteratively in order to converge to a better solution. After the first run of the ACO algorithm, the footprint mapping solution on the target FPGA is analyzed and hence the footprints, which are placed in close proximity, are identified. Such footprints are locked in order to preserve the close placement of subsystems, and the step 2 of subsystem mapping technique is re-run. These steps are repeated until there are no subsystems to be locked.

(a) Design Graph

(b) Closely Placed Subsystems

Fig. 5.16 Subsystem Locking

5.3.1 Identify Subsystem for Locking

The candidate subsystems that are suitable for locking are identified using the following criteria. First, the footprints that are placed in a close proximity are observed. Then, a rectangular bounding box (*BB*) is defined around such footprints as shown in the figure 5.16(b). However, the area covered by a bounding box should not be much larger than the total of the area covered by the individual footprints. In other words, locked footprint area should not include large empty white spaces. The following relationship is used to evaluate the candidate subsystems;

$$\sum_{fp_i \in BB} h_i \times w_i \geq (\delta_{lock} \times \text{Area of the } BB) \quad (5.6)$$

Here the constant $\delta_{lock} (< 1)$ is found empirically and is close to 1. Identification of subsystems for locking step is shown in *LOCK()* in Algorithm 4 (line 5), along with overall flow of the mapping algorithm. The subsystem grouping is performed (line 1) in order to have an initial solution, and then, footprint pruning for all the footprint lists *L* (line 2) is done based on the initial solution. Next, the ACO is run iteratively, until subsystem locking is possible (Lines 4 through 10). Once the candidate subsystems for locking are identified, the respective subsystems in the subsystem graph *G* are updated such that the respective subsystems to a locked footprints are merged, and the edges in *G* are updated with new connections from/to locked subsystems and with updated inter-subsystem communication. The updated subsystem graph is noted as G_{new} and accordingly a new complete graph G'_{new} is also created. Subsystem locking step also requires regeneration of footprints list *L* for the locked subsystems, and is discussed in the next section.

Algorithm 4 Overall Execution of Mapping Algorithm (with Subsystem Locking)

```

1:  $Q_{initial} \leftarrow \text{GROUPING}(G)$ 
2:  $L \leftarrow \text{PRUNE\_FOOTPRINTS}(L, Q_{initial})$ 
3: while true do
4:    $Q \leftarrow \text{RUN\_ACO}(G, G', L)$ 
5:    $(G_{new}, G'_{new}, L_{new}) \leftarrow \text{LOCK}(Q)$ 
6:   if  $L_{new} == L$  then
7:     break
8:    $G \leftarrow G_{new}$ 
9:    $G' \leftarrow G'_{new}$ 
10:   $L \leftarrow L_{new}$ 
11: return  $Q$ 

```

5.3.2 Footprint List of a Locked Subsystem

As shown in section 5.2, the proposed ACO algorithm takes a footprint list l_{S_i} for each subsystem S_i . Since the locked subsystems, for instance A and B in figure 5.16, act as a single subsystem in the next run of ACO, it is necessary to generate new footprint lists for the locked subsystem. The following technique is used to generate new footprint.

A footprint list entry of a subsystem S_i , $fp_{S_{i,j}}$ has the tuple format $\{x_{i,j}, w_{i,j}, h_{i,j}, f_{i,j}\}$ where the tuple elements represent bottom left horizontal coordinate of the footprint on FPGA, width and the height of the footprint and the estimated performance. It is important to note that the column arrangement of the locked region is preserved, as the two-step ACO algorithm has chosen best footprints for the respective subsystems. For instance, the column arrangement of the locked regions in figure 5.17 is shown with gaps. The preservation is done by only the populating the footprint list only with footprints with the same column arrangement. For instance, column arrangement of the locked region A in figure 5.17a is not found anywhere else on the target FPGA. Whereas, for B in 5.17b, an exact column arrangement at another location on the FPGA is shown. Therefore, the footprint list l_A is set as $\{fp_{A,1}\}$ where $fp_{A,1} = \{26, 23, 27, f_A\}$, and l_B is set as $\{fp_{B,1}, fp_{B,2}\}$ where $fp_{B,1} = \{57, 18, 33, f_B\}$ and $fp_{B,2} = \{18, 18, 33, f_B\}$. Since the footprint selection for the enclosed subsystems has been done, the values for f_A and f_B are not required.

Fig. 5.17 Footprint List for Merged Footprints

5.3.3 Performance Evaluation

Table 5.3 Routing Power Reduction with Subsystem Locking

Application	Routing Power (/mW)			Overall Routing Power Reduction Over Quartus
	Quartus	Mapping w/o Locking	Mapping w Locking	
Synth1	85.87	71.46	65.36	23.9%
Synth2	88.68	67.85	60.84	31.4%
Synth3	88.06	70.52	65.3	25.8%
atax'	320.01	287.52	278.13	13.1%
bicg'	332.62	292	282.99	14.9%
gesummv'	114.85	103.71	103.71	9.7%
cholesky'	239.81	214.44	214.44	10.6%
symm'	146.9	131.1	131.1	10.8%

The further routing power reduction occur due to subsystem locking is evaluated and shown in table 5.3. The comparison is done against Quartus compilation and the two-step subsystem mapping technique (Mapping w/o locking), which is discussed in section 5.2. It should be noted that some of the applications may not produce footprints that are eligible for subsystem locking. For instance, subsystem locking is not possible with the applications gesummv', cholesky' and symm'. However, the technique has further reduced the routing

Fig. 5.18 Effect of Subsystem Locking Technique on Routing Power

Fig. 5.19 Routing Power Reduction due to Subsystem Locking over Quartus

power by 6.5% as an average, as compared to two-step approach. Therefore, by integrating subsystem locking techniques with two-step mapping technique, the routing power dissipation has been reduced by over 17% as compared to Quartus CAD flow compilation.

5.4 Summary

In this chapter, a novel method for target architecture-aware subsystem mapping is proposed. It relies on the subsystems, generated using the methodology in Chapter 3, and the list of footprints for each subsystem, generated using the technique proposed in Chapter 4. The first step of this methodology uses a pre-processing stage to identify groups of subsystems that should be placed in close proximity to minimize the overall inter-subsystem communication. This step helps to reduce the runtime of the subsequent placement algorithm without any loss in the quality of the solution.

Next, the proposed placement algorithm employs a modified version of Ant Colony Optimization to place subsystem footprints (obtained from Chapter 4) of a large FPGA design while minimizing long distance communication. It considers different footprint variations for each subsystem and selects the most appropriate footprints that can utilize the heterogeneous resource column arrangement of modern FPGAs efficiently for better allocation of intra-subsystem communication links. It also ensures that the selected footprint locations minimize the distance between subsystems with high inter-subsystem communication. The proposed methodology reduces the routing power dissipation by about 16% with respect to default Quartus compilation and by about 11% as compared to a technique that only considers intra-subsystem communication during the subsystem creation stage.

Lastly, a subsystem locking technique proposed in Section 5.3 to accelerate the convergence of the proposed ACO based subsystem mapping algorithm. The technique has shown converging to better mapping solutions that further reduce routing power dissipation by 6.5%.

Chapter 6

Framework for Design Space

Exploration

6.1 Introduction

Over the years, the advancements in process technology have lead FPGA vendors to produce FPGAs with multi-million logic cells and thousands of hard IP cores such as BRAMs and DSP Blocks [13] [12]. As a result, modern FPGAs have been used for increasingly large applications [14] [38] [39] [40] [41]. State-of-the-art CAD have been evolved to cater the complexity of mapping such large applications [83] [84]. Despite of existing CAD tools' superiority in mapping small and medium scale designs, state-of-the-art CAD flows takes hours to days to compile large applications and produces mapping solutions with degraded QoS.

An experiment is conducted in order to demonstrate the increase in compilation time when the application size is increased. Series of test applications in different sizes (different FPGA resource requirement) were made using real benchmark applications, such that a test application is comprised of number of benchmark applications(kernels) as discussed in section 3.3.2 and 5.2.4. Each application is compiled using Intel(Altera) Quartus v17.1 (setting

the target FPGA as the largest FPGA in Intel Stratix V family [20]) and the compilation time is plotted against the test application size as shown in figure 6.1. According to the plot, it is evident that applications that require below 80k of Adaptive Logic Modules (ALMs) [145] are compiled within an hour. Typically, the placement and routing step in CAD flow for such applications does not fail. Hence the linear increase in compilation time can be observed. These applications are referred as *small to medium scale* applications. On the other hand, quadratic growth in compilation time is observed for applications larger than 80k ALMs. Compilation process for such applications goes through multiple iterations in placement and routing, hence identified as *large scale* applications. Each iteration of placement and routing takes longer runtime than previous iteration, leading exponential runtime increase. As discussed in section 2.4, design verification, timing closure and satisfying design constraints require multiple compilations of the designs. As a result, typical design cycle will require greater effort and design expertise when the applications are large. Therefore, prolonged CAD runtime adversely affects the overall design productivity and time-to-market.

Fig. 6.1 Variation of Compilation Time w.r.t. to Application Size

Proposed power and performance efficient techniques to map large designs into FPGAs are discussed in previous chapters. These techniques partition large applications into smaller

subsystems and rely on CAD tools' capability in mapping small to medium scale applications. Therefore they can leverage on the state-of-the-art CAD tools, regardless of the inefficiencies. Commercial CAD flows such as Intel(Altera) Quartus Prime [66] or Xilinx Vivado [1] provide tools, like Quartus Incremental Tool Flow [127] or Vivado Hierarchical Design [144] in order to create subsystems and place them. However, these tools do not identify subsystems automatically, and involves considerable amount of designer expertise in order to utilize them. Therefore, in this chapter, a framework, which integrates the techniques that are discussed in previous chapters, is proposed. This framework also can be used for exploring different mapping solutions by changing the criterion for partitioning large applications into smaller subsystems.

The rest of the chapter is organized as follows. Section 6.2 outlines the methodology for developing the framework for subsystem mapping, while the performance evaluation is presented in section 6.2.2. A technique to improve subsystem mapping solutions, and the methodology to integrate it to the framework is discussed in section 6.3. Next, design space exploration using the proposed framework is discussed in section 6.4 followed by a conclusion in section 6.5.

6.2 Methodology

Figure 6.2 shows the organization of the framework which incorporates proposed techniques discussed in chapter 3-6. Each technique is highlighted as a section in the figure, and can be implemented using commercial CAD tools. The sections in the framework are briefly as follows.

Step 1 in figure 6.2 shows the proposed technique in chapter 3. A subsystem generation technique is proposed in chapter 3, in order to partition a large application into smaller subsystems. The technique takes an RTL code as the input. RTL entities are considered

Fig. 6.2 Framework for Subsystem Mapping

as basic computation units. Users input two parameters; (i) resource constraint - to decide maximum area a subsystem can occupy on the FPGA (ii) connection density constraint - to limit maximum number of RTL connections in a unit FPGA area. A partitioning algorithm is used for generating subsystems. It partitions a large application based on the communication between basic computation units, while maximizing the intra-subsystem communication and minimizing the inter-subsystem communication.

Step 2 in figure 6.2 shows the output of the technique in chapter 4. A device characterization technique was proposed in order to estimate the performance of a subsystem if it is mapped into a given footprint (a rectangular FPGA area with specific location and dimension on FPGA). Proposed technique uses large number of training data of different applications and different footprints for onetime target device characterization process.

Step 3 in figure 6.2 shows the proposed technique in chapter 5. The technique takes a subsystem graph and a list of candidate footprints for each subsystem as inputs. Proposed technique maps subsystems by selecting the footprints with best shape and location for each subsystem. The overall mapping is a two-step Ant Colony Optimization (ACO) process. In the first step (Subsystem grouping), the subsystems are grouped based on the inter-subsystem communication as discussed in section 5.2.1. Subsystem in a particular group should be placed at a close proximity. The subsystem group information along with footprint list are provided to the step 2 (Subsystem Mapping), in order to select the best set of footprints for the subsystems. For faster convergence, step 2 of the technique is run iteratively while locking profitable subsystems in each iteration.

6.2.1 Implementation of the Framework for Subsystem Mapping

In this section, the methodology in implementing the proposed framework is discussed in detail. All the techniques that are briefed in section 6.1 can be implemented in any

vendor specific state-of-the-art CAD flow. Intel and Xilinx account for 89% of total FPGA market share [146]. Therefore, the framework is developed to be compatible with the most popular commercial vendor tools, Intel(Altera) Quartus Prime [66] and Xilinx Vivado [1]. Implementation of the framework using vendor specific CAD flows, Altera Quartus [66] [127] and Xilinx Vivado [144], is discussed in next sections.

6.2.1.1 Subsystem Generation

The input to the framework is a Register Transfer Level (RTL) code of a design. The parameters such as area of RTL design entities (basic computation units) and communication frequencies between entities are extracted by performing quick RTL synthesis process followed by RTL Simulation in the respective CAD flow. For Quartus, area of basic computation units is extracted after *Analysis and Synthesis* process, whereas for Vivado the process in *Synthesis*. Similarly, *RTL synthesis* or *Post-Synthesis Functional Simulation* is performed in Quartus and Vivado respectively, in order to extract the communication frequency. RTL code is processed using a script to identify the computation units and memory units and the connections between them.

Computation and memory units are grouped to form subsystems based on the communication between them. The size of a subsystems is limited to have profitable subsystems which accumulates computation and memory units to maximize intra-subsystem communication while inter-subsystem communication is minimized. The subsystem generation algorithm is implemented using C language, and the area, communication and connection input details are taken using tabulated text files. The output of the subsystem generation process is a text file with a list of subsystems $\{S_0, S_1, \dots, S_N\}$ and respective basic computation units belongs to each subsystem. The output also specify the communication frequencies between subsystems.

6.2.1.2 Candidate Footprint List Generation

As discussed in chapter 4, footprint list generation process requires subsystem specific parameters; number of connections (c_i) and default critical path(cp_i) for each subsystem S_i . The subsystem information obtained in the previous step is therefore passed to Design Partitioning tool in Quartus CAD flow. This tool isolates and separately compiles each subsystem. After the incremental compilation, the timing report and the fitter report is referred to extract cp_i and c_i , respectively. The area requirement (l_i, m_i, d_i) of all the subsystems is also known from the first step.

A candidate footprint (a foot print before performance estimation) has 3 parameters; bottom left horizontal position of the rectangular shape on target FPGA (x), width (w) and height (h) of the rectangular region. For an arbitrary location x_i on the target FPGA, a list of rectangular shapes on the FPGA that satisfies the area requirement (l_i, m_i, d_i) can be derived. The same idea is extended to obtain comprehensive set of candidature footprints, by changing x_i locations. This process is followed for all subsystems S_i to get respective candidature footprints in the form of $\{(x_{i,0}, w_{i,0}, h_{i,0}), (x_{i,1}, w_{i,1}, h_{i,1}) \dots\}$. x_i locations are selected with intervals to generate candidature footprints instead of generating for all possible x_i , in order to avoid generating large number of candidate footprints.

Next, the subsystem parameters and corresponding candidate footprint list is provided to the characterized device model of the target FPGA, which is generated using the proposed technique in section 4 and actual footprint list for each subsystem is obtained (L). These lists are similar to candidate footprint lists but annotated with their estimated performance (f). Footprint lists are also sorted in descending order of estimated performance.

6.2.1.3 Subsystem Grouping

The subsystem information obtained from the subsystem generation step is provided to the subsystem grouping technique (Step 1 of ACO base subsystem mapping) proposed in section

5.2.1. Subsystem information also includes the communication between subsystem. This inter-subsystem communication is considered to form subsystem groups, and subsystems in each group should be mapped in close proximity on target FPGA.

6.2.1.4 Subsystem Mapping

The subsystem grouping technique pre-process the list of subsystems and form groups. Grouping information is provided to the subsystem mapping technique (Step 2 of ACO based subsystem mapping), along with the footprint lists generated using pre-characterized device model. Subsystem mapping technique then selects best set of footprints to map each subsystem. Output of this step consists of two parts.

- Subsystem composition : A List of basic computation units that are included in a subsystem. This output is similar to what is obtained during subsystem generation.
- Subsystem footprints : The shape and the location of each subsystem on target FPGA.

6.2.1.5 Subsystem Locking

A subsystem locking technique is proposed in section 5.3 for faster convergence of the subsystem mapping technique. The shapes and locations of the selected footprints by the subsystem mapping technique are analyzed in this step, and well organized footprints on the target FPGA are identified. Identified footprints are then considered as a locked region . The subsystem mapping technique is re-iterated while keeping shapes and horizontal locations of the identified subsystems are unchanged.

6.2.1.6 Compilation

Output of the previous step is the subsystem lists and their footprints on FPGA. In order to implement these footprints on target FPGA, LogicLocks in Quartus or PBlocks in Vivado is used. LogicLocks/PBlocks implements the actual subsystems and footprints on the target

FPGA fabric as floorplaning. After that, the compilation process in respective CAD flow is invoked to get the programmable bitcode. i.e. *Compile* for Quartus and *Implement* for Vivado.

6.2.2 Performance Evaluation

In this section, the runtime of the proposed framework in mapping large applications to FPGAs is evaluated. The evaluation is done using both Intel Quartus Prime CAD flow and Xilinx Vivado flow on the respective FPGA devices. In sections 3.3.2 and 5.2.4, test applications that mimics the properties of large application are used. The same strategy is utilized to develop new set of applications in order to evaluate the runtime. Table 6.1 shows the resource requirement of test applications in ALMs, BRAMs and DSPs. Applications are categorized into A through E, based on requirement for three resource types. Category A has the highest resource requirement and E has the lowest.

Table 6.1 Test Application Details

Test Application	Resource Requirement			Application Size Category
	ALMs	BRAMs	DSPs	
Synth2_5se	123642	1120	960	A
Synth3_5se	83630	960	960	B
Synth1_5se	67529	640	640	C
Atax_5se	56358	640	512	D
Bicg_5se	43803	512	512	E

Table 6.2 shows the runtime of existing CAD flows and the proposed framework flow for each test application. The proposed framework shows upto 50% compilation time reduction for the first three applications. However, for the latter two applications, the test results demonstrate a longer compilation time as compared to the existing CAD flow. As shown in figure 6.2, the proposed framework act as a plug-in to existing CAD flows and relies on them to obtain the final mapping on target FPGAs. For small and medium scale applications, the framework runtime overhead is significant as compared to the CAD compilation time. Therefore, for such applications, the framework is less effective to lower the compilation

time. However, the framework shows greater runtime reduction for large applications. Figure 6.3 plots the compilation time against the test application size. It shows that the compilation time of the framework flow is lower than the existing CAD flow, when the application size is greater than about 70k of ALMs (roughly 190k logic elements). According to experiment results shown in figure 6.1, applications below 70k ALMs approximately belongs to small and medium scale category.

In addition, three more test applications are tested as shown in table 6.3. These applications consumes about 80% of FPGA resources from the largest FPGA in Intel(Altera) Stratix V [20] family. Due to the immense resource requirement of these applications, the existing CAD flows failed to perform a successful compilation. However, the proposed framework has been successful in compiling these extreme cases, highlighting its usefulness in improving the computability of existing CAD flows.

Table 6.2 Comparison of Compilation Time

Test Application	Application Size Category	Runtime (Hours)		Speed-up
		Existing CAD Flow	Proposed Framework	
Synth2_5se	A	31.5	15.72	50.1%
Synth3_5se	B	7.52	5.82	22.6%
Synth1_5se	C	6.4	5.05	21.1%
Atax_5se	D	1.78	1.85	-3.9%
Bicg_5se	E	1.12	1.22	-8.9%

Table 6.3 Compilation Time for Extreme Cases

Test Application	Resource Requirement			Runtime(Hours)	
	ALMs	BRAMs	DSPs	Quartus	Framework
Synth2_5se_L	127541	1120	960	Fail	16.35
Synth3_5se_L	115750	1040	960	Fail	13.69
Synth1_5se_L	114717	1040	1040	Fail	14.02

Figure 6.4 compares the routing power dissipation of the mapping solutions obtained using the proposed framework with routing power of the mappings of Quartus flow. Our frame-

Fig. 6.3 Increase in Compilation Time with Application Size

Fig. 6.4 Comparison of Routing Power Dissipation

work has shown average routing power reduction of 14.7%, ranging from 9.6% (*Atax_5se*) to 25.9% (*Synth2_5se*) as compared to the Quartus flow. In addition, the performance analysis in figure 6.6 that the proposed framework improves performance (F_{max}) by about 5% as an average, as compared to Quartus flow.

Fig. 6.5 Routing Power Reduction over Quartus

Fig. 6.6 Comparison of Performance

6.3 Subsystem Splitting for Optimal Placement

In section 6.2, a framework that integrates all the techniques proposed in this thesis is presented. Despite of the effectiveness of the proposed framework, it is observed that some of the footprints obstruct better mapping of footprints, leading un-utilized space between subsystems. For instance, consider the solution in figure 6.7 which is obtained by the framework for the test application *Atax_5se*. The footprint S_2 in the mapping solution is wider than the empty space below the footprint, leaving the FPGA area *below* the subsystem unused. It is important to note that, the subsystem mapping and subsystem approach is a deterministic approach to converge to the best solution for given subsystems. Hence, To overcome this issue, mapping subsystems to different footprints (with different size and placement) does not guarantee a performance and power optimized mapping.

Fig. 6.7 Unused FPGA Resource During Subsystem Mapping

6.3.1 Splitting Subsystem by Subsystem Re-generation

In order to overcome such footprints, a technique that alters the composition of the subsystems is proposed. These subsystems are reiterated through subsystem generation process in the proposed framework with lower area constraint to create smaller subsystems. This step is referred to as *splitting*. Footprints respective to the new (smaller) subsystems are generated using characterized device model as explained in chapter 4. The original mapping problem is then altered by adding new subsystems to the subsystem graph G and accordingly the complete graph G' . Suitable footprints for splitting are identified as discussed follows.

As shown in figure 6.7, footprint obstruction typically happens due to odd-shape footprints. Hence, a simple aspect ratio based technique is developed. First, the average aspect ratio (AR) of all the footprints is calculated as follows.

$$AR_{S_i} = \frac{\max(w_{S_i}, h_{S_i})}{\min(w_{S_i}, h_{S_i})} \quad (6.1)$$

where w_{S_i} and h_{S_i} represent the width and the height of the footprint S_i respectively. If the average and standard deviation of aspect ratios of the footprints are found $\overline{AR}$ and σ_{AR} respectively, the odd footprints are then identified using the following definition.

$$oddfootprint = \begin{cases} 1, & \text{if } AR_{S_i} > \overline{AR} + \sigma_{AR} \\ 0, & \text{otherwise} \end{cases} \quad (6.2)$$

Next, the whitespace area below each odd footprints is calculated. The odd footprint with largest whitespace area below the footprint is identified as the an eligible footprint for the splitting process. For instance, the footprint S_2 of the subsystem mapping in figure 6.7 is identified as an eligible footprint.

Figure 6.8 shows the addition of proposed splitting technique in the framework. Eligible subsystems are first processed through the subsystem generation step at a tighter area

constraint in order to re-generate smaller subsystems. The candidate footprint list for re-generated new subsystems are then obtained using the characterized device model. New footprint list and the subsystem information is then provided to subsystem generation and mapping (two-step subsystem mapping approach) techniques in order to select the best set of footprints for re-generated subsystems. It should be noted that, footprints generated in the earlier iterations are not changed.

6.3.2 Performance Evaluation

As discussed in chapter 2, state-of-the art CAD flows takes hours to compile large applications, and often the mapping solutions are with degraded performance and power. Therefore, the evaluation investigates three parameters; (i) Tool runtime (ii) Power dissipation and (iii) Performance (F_{max}). The results obtained from the integrated framework is compared against the commercial CAD flows. Power values are taken using Quartus PowerPlay power estimation. In order to obtain the most accurate values, the estimator is fed with signal interactions data of the compiled design by performing a gate level simulation as discussed in section 3.3 and 5.2.4. Performance of each design is extracted from the CAD timing report after a successful compilation.

Table 6.4 compares the Quartus compilation time and the runtime of the proposed framework. The runtime of the framework is dominated by the time taken for perform compilation using existing CAD flows. Therefore, addition of splitting technique (as shown in figure 6.8) does not increase the runtime of the framework significantly. Therefore, it is evident that integrated framework is drastically reduces the compilation time of existing CAD flows, especially for the large applications as shown in figure 6.9.

In addition, figure 6.10 shows the routing power dissipation of mapping solutions obtained by (i) Quartus flow (ii) proposed framework without splitting process and (iii) modified framework with splitting. According to the figure, the splitting process has further reduced

Fig. 6.8 The Integrated framework with Subsystem Splitting

Table 6.4 Comparison of Compilation Time

Test Application	Application Size Category	Runtime (Hours)		Speed-up
		Existing CAD Flow	Proposed Framework	
Synth2_5se	A	31.5	15.72	50.1%
Synth3_5se	B	7.52	5.95	20.9%
Synth1_5se	C	6.4	5.22	18.4%
Atax_5se	D	1.78	1.9	-6.7%
Bicg_5se	E	1.12	1.22	-8.9%

Fig. 6.9 Increase in Compilation Time with Application Size

routing power dissipation by 6%, as compared to the framework without splitting process. However, the splitting technique has not found eligible subsystems for the subsystem regeneration process in applications *Synth2_5se* and *Bicg_5se*. Therefore, the power dissipation for these applications remains the same. The overall average routing power reduction is reported as 18% for the test applications, whereas the maximum reduction is as high as 24.9% as compared to the Quartus flow. Comparison of performance (F_{max}) of the framework mapping solutions and that of Quartus CAD flow is presented in figure 6.12. According to the figure, framework mapping solutions show an average of 6% performance improvement over Quartus. It is also noteworthy that applications *Atax_5se* and *Bicg_5se* show a better

Fig. 6.10 Comparison of Routing Power Dissipation

Fig. 6.11 Routing Power Reduction over Quartus

Fig. 6.12 Comparison of Performance

power and performance over Quartus CAD flow, despite of the longer runtime as shown in table 6.4. It shows that small to medium scale applications are also benefited from the proposed methodology especially for reducing power.

In section 3.3, the steps to measure power using Quartus PowerPlay [133] is discussed. For an accurate power measurement, toggle rate data, which is extracted from a gate-level simulation of both Quartus mapping and the framework mapping solutions, is provided to PowerPlay. During the power measurement, the gate-level simulations are run at the same running frequency (clock frequency) for both designs. However, according to figure 6.12, mapping solutions from the proposed framework can be run at a higher clock frequency. Running at a higher clock frequency helps in saving energy that dissipates due to the static power component. It is observed that an average energy savings about 23.1% can be obtained as compared to Quartus, by running the mapping solutions at respecting F_{max} shown in figure 6.12.

Fig. 6.13 Integrated Framework Assisted Design Space Exploration

6.4 Design Space Exploration

In the previous section, it is shown that integrating the techniques proposed in chapter 3 through chapter 5 into a framework leads to better runtime, power dissipation and compilation time for large applications. In this section, a design space exploration methodology that leverages on the automated framework is presented.

As shown in figure 6.2, subsystems are generated with an area constraint and a connection density constraint. These subsystems are then being used in the subsequent steps in the framework flow, in order to achieve better power-performance results as compared to the existing state-of-the-art CAD flow. As shown in figure 6.13, the generation of subsystems can be influenced to produce different sets of subsystems for a given application. These new sets of subsystems will lead to a different power-performance results. Hence, a designer can select a suitable option based on the requirements.

The proposed design space exploration methodology is demonstrated using the same test application set that is used in chapter 3. For each application, multiple subsystem sets (design points) are generated by varying area constraint and congestion constraints. These parameters affect the each subsystem composition and number of subsystems per application. The subsystems are then send through the rest of the steps in the framework. The final subsystem mapping of each case is then compiled using existing CAD flow to get the target FPGA mapping. The timing reports are referred to extract the performance (F_{max}) of the solution, and the same methodology in chapter 3.3 and 5.2.4 is utilized to obtain the power dissipation. Power and Performance results of each design point are plotted as shown in figures 6.14 through 6.18. In addition to the design points that are obtained by changing constraints, the power-performance results for default CAD (Quartus) compilation are also plotted.

Table 6.5 Design Points for Synth1

Design Point	Constraints				No. of Subsystems
	CLBs	BRAMs	DSPs	Congestion	
1	256	8	8	1.2	8
2	512	12	12	1.8	5
3	256	8	8	1.8	7
4	512	12	12	1.2	6

Fig. 6.14 Design Space Exploration for Synth1 Application

Table 6.6 Design Points for Synth2

Design Point	Constraints				No. of Subsystems
	CLBs	BRAMs	DSPs	Congestion	
1	256	8	8	1.2	8
2	256	8	8	1.8	7
3	512	12	12	1.8	5
4	512	12	12	1.2	6

Fig. 6.15 Design Space Exploration for Synth2 Application

Table 6.7 Design Points for Synth3

Design Point	Constraints				No. of Subsystems
	CLBs	BRAMs	DSPs	Congestion	
1	256	8	8	1.2	8
2	512	12	12	1.8	5
3	256	8	8	1.8	7
4	512	12	12	1.2	6

Fig. 6.16 Design Space Exploration for Synth3 Application

Table 6.8 Design Points for Atax'

Design Point	Constraints				No. of Subsystems
	CLBs	BRAMs	DSPs	Congestion	
1	256	4	4	1.2	12
2	512	8	8	1.8	7
3	256	4	4	1.8	11
4	512	8	8	1.2	8

Table 6.9 Design Points for Bicg'

Design Point	Constraints				No. of Subsystems
	CLBs	BRAMs	DSPs	Congestion	
1	256	4	4	1.2	10
2	256	4	4	1.8	9
3	512	8	8	1.2	8
4	512	8	8	1.8	6

According to the figures 6.14 to 6.18, it is evident that the integrated framework can be used to obtain different subsystem sets (design points) with different performance-power metric. For each application, it is possible to identify a power-efficient design point, as well as a performance maximized design point. It is also noteworthy that all the design points that

Fig. 6.17 Design Space Exploration for Atax' Application

Fig. 6.18 Design Space Exploration for Bicg' Application

are obtained for each application show performance-power metric as compared to Quartus. Hence, the integrated framework can be utilized in selecting a suitable subsystem mapping based on design requirements.

6.5 Summary

This chapter presents an automated framework which integrates the methodologies discussed in previous chapters. The methodologies do not require modifications in CAD algorithms and can be used on top of any vendor-specific CAD flow. The framework also allows rapid design space exploration by setting different area constraints during subsystem generation and hence allowing to obtain different numbers of subsystems with different sizes. The framework is implemented on both Intel (Altera) and Xilinx CAD flows and tested against the latest versions of respective CAD flows for mapping on latest FPGA device families. Experimental results demonstrate that the framework can be leveraged upon to reduce routing power by over 22% while improving performance by 8%. The solutions are about 23% more energy-efficient than the commercial CAD mapping solutions. Unlike typical CAD flows which exponentially increase the compilation time as the application size increases, the proposed framework shows a linear increase in tool runtime. Results have demonstrated a reduction over 40% as compared to state-of-the-art CAD. The proposed framework also demonstrates high compilability in compiling large applications by successfully producing mapping solutions for extreme cases where commercial CAD flows fail to compile. This chapter also presents the methodology for using the proposed integrated framework in design space exploration.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis several novel techniques have been proposed to map large-scale systems to high-density FPGAs. The proposed techniques also leverage on state-of-the-art FPGA CAD tools flows for mapping large applications to achieve high performance and energy-efficient solutions. Unlike existing commercial tools, the proposed techniques allow for deterministic runtime across different FPGA vendors and device families. It has been shown that existing CAD tools are only capable of producing high quality compilation results for small to medium scale applications. Moreover, their performance degrade severely when mapping large designs, often leading to suboptimal compilation results and prohibitively long runtimes.

The proposed communication-aware subsystem generation technique in Chapter 3 partitions a large system at the RTL level into several smaller systems (referred to as subsystems) and relies on existing CAD tools to compile these smaller subsystems in a systematic manner. In particular, the proposed technique considers signal toggle rates (referred to as communication) of an application during the subsystem generation process to guarantee power-efficient solutions while also optimizing for performance, especially for large applications. A rapid simulation-based technique has been developed to extract accurate communication frequency

(number of toggles in RTL signals/connections) between RTL entities (referred to as basic computation units) of a given application by relying on a commercial simulator, Modelsim. The rapid simulation at the RTL level for extracting the communication information is considered during the subsystem generation step to maximize intra-subsystem communication that leads to power and performance efficient designs. An upper bound on the resources (LUT, BRAMS, etc.) for each subsystem is imposed to limit the subsystem size since larger subsystems are less effective in keeping basic computation units in close proximity. In order to ensure a desirable balance between intra-subsystem communication costs and the corresponding subsystem size, the number of RTL connections per unit area is limited to prevent routing congestions within a subsystem. The subsystems generated in this way can readily benefit from existing CAD flows to generate the final bit-stream for FPGA device configuration. The proposed methods have ensured that the basic computation units with high communication are placed in close proximity on FPGA, leading to significant power savings. The proposed technique was evaluated using the widely used commercial Intel (Altera) Quartus CAD flow on several test applications that are developed based on benchmark applications from the widely used Polybench benchmark suite to mimic large applications and led to a performance improvement of over 60% in the performance (F_{max}), while also reducing the overall energy consumption by nearly 20%.

Architectural details of the target FPGA device, such as various types of resources arranged in specific columns and different types (lengths) of routing wire channels are modeled using a variety of shapes (height and width) and location in the rectangular region of an FPGA (referred to as a footprint). This strategy also ensures that the clock network design of the target FPGA is also captured during the device characterization process. In addition to the architecture, subsystem specific information is also modeled using the critical path and routing resource requirement on the target FPGA. An automated technique is developed to generate a large number of training data points by exhaustively changing the shapes and

locations of footprints and obtaining the respective F_{max} for a number of training subsystems. In order to efficiently characterize the heterogeneous resources available in modern FPGAs during mapping of subsystems on target FPGAs, an Artificial Neural Network (ANN) based technique is proposed to represent high-density FPGAs. The proposed methodology relies on the training the ANN with target FPGA architecture through supervised learning. The ANN model was built using this large amount of training data by relying on the ANN toolbox in MatLab. An empirical study is carried out to select the most suitable learning algorithm (Resilient Back-propagation) and the appropriate number of hidden layers. Next, the trained model is used to estimate the performance (F_{max}) of a given subsystem at any location on the target FPGA. This one-time characterization of a target FPGA obviates the need to store a large module library in existing state-of-the-art modular design methodology (MDM) techniques that involves compiling each module to be placed and routed for every possible location and storing in a database. The proposed technique also supports multi-shaped subsystems, whereas typical MDM techniques only operate on fixed shapes. The proposed technique has been extensively evaluated using 26 applications from the popular Polybench benchmark suite and synthetic benchmarks on 4 FPGA devices from different device families and shows an average error of less than 4% in the estimated performance.

The subsystem mapping problem, described in Chapter 5, is modeled into a traveling salesman problem and a two-step Ant Colony Optimization (ACO) based technique is employed to solve it. In the first step, subsystems are grouped based on the inter-subsystem communication, in order to identify subsystems to be placed in close proximity. The grouping information is then used in the second step to select the most beneficial footprints from each subsystem's candidate footprint list obtained using the characterized target device model. Footprints with the highest estimated performance are selected while keeping footprints of subsystems with high inter-subsystem communication close to each other. The two-step approach ensures notable reduction in the algorithm runtime, without degrading the

quality of solution. It is noteworthy that the first step of grouping does not completely eliminate any solution. Instead, it promotes profitable solutions by increasing their probability values (pheromone) while ensuring inadvertent long interconnects do not degrade compute performance. Therefore, even a sub-optimal grouping solution does not impact the quality of the mapping solutions produced by the second step. To aid in the faster convergence of the algorithm, subsystems exhibiting tight integration are locked (or frozen) at the end of each iteration. The proposed technique also estimates the routing congestion in each iteration due to over packing of footprints in any region. It was observed that such routing congestion were mainly due to unusually longer routing wires for inter-subsystem connections. Experimental results show that the subsystem mapping technique can outperform Quartus CAD flow, with over 17% of routing power reduction without impeding performance (F_{max}).

The various techniques proposed in this thesis have been integrated to aid in the mapping of a large application onto modern heterogeneous FPGAs. The proposed framework allows for mapping large systems in deterministic time, lower power consumption and improve compute performance. The framework can support commercial Intel (Altera) and Xilinx CAD flows to benefit from the block-based design options (such as LogicLock in Altera Quartus, PBlocks in Xilinx Vivado) for mapping subsystems. This integrated framework eliminates the reliance on design expertise and manual intervention typically required for using commercial CAD tools. A subsystem splitting technique has also been proposed to improve the FPGA area utilization by splitting subsystems when necessary to ensure tight integration. This approach has been shown to fill any area gaps that remain among subsystems. Experimental results demonstrate that the framework is capable of reducing routing power by over 18% while also improving performance by 8%. The resulting solutions are over 23% more energy-efficient than the commercial CAD mapping solutions. Unlike existing CAD flows, in which the compilation time increases exponentially with increasing application size (due to multiple iterations in placement and routing step), the proposed

framework shows a near-linear increase in tool runtime. Runtime reduction of over 50% has been observed for large applications, as compared to state-of-the-art commercial CAD tools. The proposed framework can be effectively deployed to achieve deterministic compilation runs when mapping very large applications.

A novel use of this integrated framework to facilitate design space exploration has also been proposed. It relies on the generation of subsystems of different sizes by varying the resource and connection density constraints during the subsystem generation process. The new set of subsystems are then mapped to obtain a set of power, delay and area metrics. This process is repeated to facilitate the design space exploration to provide a range of design choices. Finally, the proposed methods can also be applied to a specific region of a large system to enable selective tuning as required.

Reconfigurability and the large capacity of modern high density FPGAs have attracted fast evolving domains such as Cloud Computing [147], Artificial Intelligence (also Machine Learning and Artificial Neural Networks) [37], 5G [148] and Autonomous Vehicles [149]. Cloud computing market is expected to grow to USD 623.3 billion by 2023 at a Compound Annual Growth Rate (CAGR) of 18.0%, whereas machine learning and autonomous vehicles markets are expected to grow to USD 8.81 billion at CAGR 44.1% by 2022 and USD 556.67 billion at CAGR of 39.47% by 2026, respectively [150][151][152]. Typically, the applications from these domains require massive data processing throughput [147], hence require extreme FPGA sizes. To cater this rapidly evolving large-scale application mapping, efficient FPGA CAD has become crucial. Therefore, the proposed techniques complement the advancements in FPGA CAD. In addition, these techniques minimize the manual interventions and requirement for design expertise that often involve in large-scale application mapping, hence encouraging the industries to map more applications to FPGAs.

7.2 Future Research Directions

1. Communication-aware subsystem generation from C/C++ applications

Chapter 3 describes the subsystem generation methodology for large applications given the RTL code. However, FPGA based designs using RTL code is a cumbersome process, especially due to testing and verification challenges. It would be appropriate to extend this to allow for C/C++ applications to be supplied as inputs by introducing a High-Level Synthesis (HLS) stage. It should be possible to convert the C/C++ application to an intermediate representation (IR) produced by an architecture-independent compiler process, for instance, LLVM, to identify basic computation units. For example, basic blocks and variables in the LLVM IR can be defined as computation units and memory units, respectively. Profiling at the IR level can be used to identify the execution frequency of each basic block. This in turn can provide a means to extract the communication frequency between the computation units and the connected memory units as needed by the subsystem generation technique proposed in Chapter 3. An HLS tool like LegUp can then be leveraged to translate basic blocks in LLVM IR into RTL entities.

2. FPGA architecture-aware subsystem generation

The subsystem generation technique proposed in chapter 3 currently does not consider dependencies of target FPGA. It operates on two constraints, namely, an area constraint and connection density. Noting that modern FPGAs continues to offer different logic block variants and BRAM structures, it would be appropriate to consider architecture-specific characteristics during the subsystem generation process. This should pave the way for accurate estimation of the required number of resources such as BRAMs and routing channel capacity.

3. FPGA Architecture aware congestion modeling

In Chapter 5, a bounding box based technique was proposed to model the routing-congestion characteristics of a generic platform. However, different routing network topologies and channel capacities can be seen across different FPGA Platforms. For instance, Xilinx Virtex 4 series devices are composed of horizontal and vertical routing wires, whereas Virtex 5 series devices consist of an additional diagonal type of routing wires [115]. Virtex 5 has also a minimized the number of hops during routing to lower the routing resource utilization. It would be good to explore if such advances to lower the utilization of routing resources can be incorporated into the model to better facilitate the subsystem grouping process.

4. Enhancing performance estimation of subsystems

At present, the performance estimation method relies on the mapping of 26 benchmark applications onto templates of varying shapes at all possible locations on FPGA. The training data so generated has also led to an ML based method for rapid estimation. It would be of interest to explore if the quality of the 26 benchmark applications can be further improved to enhance the accuracy of the performance estimation of subsystems on target FPGA platform. It is envisaged that applications exhibiting a range area and resource requirements can be adopted to enhance the characterization of the target FPGA platform.

5. Accelerating Ant Colony Optimization for subsystem mapping

In chapter 5, an ACO based subsystem mapping technique was proposed to select the most beneficial footprint for each subsystem from a long list of candidate footprints. It is envisaged that the ACO algorithm can be supplied with a limited set of footprints for each subsystem to begin with to encourage rapid convergence. This can provide for an iterative method to invoke successive ACO invocations that can be terminated when no significant improvements are possible.

6. Prioritizing the mapping of subsystem groups

Subsystem grouping technique proposed in chapter 5 identifies and allocates subsystems with high inter-subsystem communication into distinctive groups. It would be beneficial to order the mapping of each group of subsystems based on the significance of each group in terms of potential savings in communication costs. It is envisaged that prioritizing the mapping of a group exhibiting higher savings will provide for higher chances of facilitating tighter integration of its subsystems.

7. Selective design space exploration

An automated framework has been proposed in Chapter 6 to facilitate design space exploration. The proposed methodology for design space exploration can be extended to restrict it to a specific portion(s) of the main system. This will provide for designer flexibility to help target the design space exploration to meet the constraints of a subsystem/module without affecting the rest of the system. This should pave the way for meeting different optimization goals (timing closure or power reduction) in various sections of a large design.

References

- [1] Xilinx. <https://www.xilinx.com/>.
- [2] Intel Corporation. Intel. <https://goo.gl/kTtcv6>, 2019.
- [3] Stephen M Trimberger. Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. *Proceedings of the IEEE*, 2015.
- [4] Artificial Intelligence: Where FPGAs Surpass GPUs. <https://eecatalog.com/fpga/2017/10/24/artificial-intelligence-where-fpgas-surpass-gpus/>.
- [5] Marcel Gort et al. Design re-use for compile time reduction in FPGA high-level synthesis flows. In *Field-Programmable Technology (FPT), 2014 International Conference on*, 2014.
- [6] Huimin Bian et al. Towards scalable placement for FPGAs. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, 2010.
- [7] James Coole et al. BPR: fast FPGA placement and routing using macroblocks. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2012.
- [8] Adrian Ludwin et al. Efficient and deterministic parallel placement for FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2011.
- [9] Kevin E Murray et al. Titan: Enabling large and complex benchmarks in academic CAD. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, 2013.
- [10] MarketsandMarkets Research Private Ltd. Embedded System Market by Hardware, Software, System Size, Functionality, Application, Region - Global Forecast to 2025. <https://bit.ly/3d53XNr>, 2020.
- [11] Christina A Hacker and Alain C Diebold. 2012 updates to the international technology roadmap for semiconductors (itrs) metrology chapter. Technical report, 2013.
- [12] Intel Corporation. Intel Stratix 10 GX/SX Device Overview. <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10-overview.pdf>, 2019.

- [13] Xilinx Inc. UltraScale Architecture and Product Data Sheet: Overview. https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf, 2019.
- [14] Ruizhe Zhao, Ho-Cheung Ng, Wayne Luk, and Xinyu Niu. Towards efficient convolutional neural network for domain-specific applications on FPGA. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018.
- [15] Microsoft. Project Brainwave. <https://www.microsoft.com/en-us/research/project/project-brainwave/#!microsoft-blogs>, 2019.
- [16] Pentek, Inc. Growth in Logic Cells in Xilinx Virtex Devices. https://www.pentek.com/pipeline/20_2/v7.cfm, 2019.
- [17] Umer Farooq, Zied Marrakchi, and Habib Mehrez. FPGA architectures: An overview. In *Tree-based heterogeneous FPGA architectures*, pages 7–48. Springer, 2012.
- [18] Xilinx Inc. Virtex-4 Family Overview. https://www.xilinx.com/support/documentation/data_sheets/ds112.pdf, 2010.
- [19] Xilinx Inc. 7 Series FPGAs Data Sheet: Overview. https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf, 2018.
- [20] Intel Corporation. Stratix V Device Overview. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-v/stx5_51001.pdf, 2015.
- [21] Kevin E Murray and Vaughn Betz. HETRIS: Adaptive floorplanning for heterogeneous FPGAs. In *2015 International Conference on Field Programmable Technology (FPT)*, pages 88–95. IEEE, 2015.
- [22] Xilinx Inc. UltraRAM: Breakthrough Embedded Memory Integration on UltraScale+ Devices. https://www.xilinx.com/support/documentation/white_papers/wp477-ultraram.pdf, 2016.
- [23] Altera Corporation. Overview for the Stratix IV Device Family. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-iv/stx4_siv51001.pdf, 2016.
- [24] Intel Corporation. CycloneII FPGAs. <https://www.altera.com/products/fpga/cyclone-series/cyclone-ii/support.html>, 2019.
- [25] W James MacLean. An evaluation of the suitability of FPGAs for embedded vision systems. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops*, pages 131–131. IEEE, 2005.
- [26] Babak Falsafi, Bill Dally, Desh Singh, Derek Chiou, J Yi Joshua, and Resit Sendag. Fpgas versus gpus in data centers. *IEEE Micro*, 37(1):60–72, 2017.

- [27] Fernanda Kastensmidt and Paolo Rech. FPGAs and parallel architectures for aerospace applications. In *Soft Errors and Fault-Tolerant Design*. Springer, 2016.
- [28] Shanker Shreejith and Suhaib A Fahmy. Smart network interfaces for advanced automotive applications. *IEEE Micro*, 38(2):72–80, 2018.
- [29] Fynn Schwiegelshohn, Lars Gierke, and Michael Hübner. Fpga based traffic sign detection for automotive camera systems. In *2015 10th international symposium on reconfigurable communication-centric systems-on-chip (ReCoSoC)*, pages 1–6. IEEE, 2015.
- [30] Guy Wassi, Sylvain Iloga, Olivier Romain, and Bertrand Granado. Fpga-based real-time mfcc extraction for automatic audio indexing on fm broadcast data. In *2015 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–6. IEEE, 2015.
- [31] Issam Bouganssa, Mohamed Sbihi, and Mounia Zaim. Implementation on a fpga of edge detection algorithm in medical image and tumors characterization. In *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*, pages 59–64. IEEE, 2016.
- [32] Pál Varga, László Kovács, Tamás Tóthfalusi, and Péter Orosz. C-gep: 100 gbit/s capable, fpga-based, reconfigurable networking equipment. In *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6. IEEE, 2015.
- [33] Elisabeth Glocker, Qingqing Chen, Ulf Schlichtmann, and Doris Schmitt-Landsiedel. Emulation of an asic power and temperature monitoring system (etpmon) for fpga prototyping. *Microprocessors and Microsystems*, 50:90–101, 2017.
- [34] Ahmet Can Mert, Ercan Kalali, and Ilker Hamzaoglu. An fpga implementation of future video coding 2d transform. In *2017 IEEE 7th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*, pages 31–36. IEEE, 2017.
- [35] Weikang Qiao, Jieqiong Du, Zhenman Fang, Michael Lo, Mau-Chung Frank Chang, and Jason Cong. High-throughput lossless compression on tightly coupled CPU-FPGA platforms. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018.
- [36] Apple announces all-new redesigned Mac Pro. <https://www.theverge.com/2019/6/3/18646424/apple-mac-pro-redesign-new-specs-features-photos-wwdc-2019>.
- [37] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

- [38] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016.
- [39] Xiaofan Zhang, Xinheng Liu, Anand Ramachandran, Chuanhao Zhuge, Shibin Tang, Peng Ouyang, Zuofu Cheng, Kyle Rupnow, and Deming Chen. High-performance video content recognition with long-term recurrent convolutional network for FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017.
- [40] Yuran Qiao, Junzhong Shen, Tao Xiao, Qianming Yang, Mei Wen, and Chunyuan Zhang. FPGA-accelerated deep convolutional neural networks for high throughput and energy efficiency. *Concurrency and Computation: Practice and Experience*, 2017.
- [41] Amir Yazdanbakhsh, Michael Brzozowski, Behnam Khaleghi, Soroush Ghodrati, Kambiz Samadi, Nam Sung Kim, and Hadi Esmailzadeh. Flexigan: An end-to-end solution for fpga acceleration of generative adversarial networks. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'18)*, 2018.
- [42] Aldec, Inc. FPGA vs GPU for Machine Learning Applications: Which one is better? <https://www.aldec.com/en/company/blog/167--fpgas-vs-gpus-for-machine-learning-applications-which-one-is-better>, 2019.
- [43] Eugene Kim, Business Insider US. 3 reasons why Intel pulled the trigger on a massive 16.7 billion deal, its biggest ever. <https://www.businessinsider.sg/intel-closes-altera-deal-2015-12/?r=US&IR=T>, 2015.
- [44] Shawn Knight , Techspot. Microsoft's Project Catapult is why Intel bought FPGA-maker Altera for 16.7 billion last year. <https://www.techspot.com/news/66461-microsoft-project-catapult-why-intel-bought-fpga-maker.html>, 2016.
- [45] Intel Corporation. Intel Xeon Processors. <https://www.intel.com/content/www/us/en/products/processors/xeon.html>, 2019.
- [46] Intel Corporation. Intel FPGA Acceleration Hub. <https://www.intel.com/content/www/us/en/programmable/solutions/acceleration-hub/overview.html>, 2019.
- [47] Microsoft. Deploy a model as a web service on an FPGA with Azure Machine Learning service. <https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-deploy-fpga-web-service>, 2019.
- [48] Inc. Amazon Web Services. Amazon Web Services. <https://aws.amazon.com>, 2019.
- [49] Baidu FPGA cloud serve. <https://cloud.baidu.com/product/fpga.html>.

- [50] Cloud Tensor Processing Units (TPUs). <https://cloud.google.com/tpu/docs/tpus>.
- [51] Microsoft: FPGA Wins Versus Google TPUs For AI. <https://www.forbes.com/sites/moorinsights/2017/08/28/microsoft-fpga-wins-versus-google-tpus-for-ai/#26549b8b3904>.
- [52] MarketWatch. Advanced Driver Assistance System (ADAS) Market – 2019 Trends, Size, Share, Growth Insight, Competitive Analysis, Statistics, Regional, And Global Forecast To 2023. <https://on.mktw.net/2mBf244>, 2019.
- [53] Seunghyuk Choi, Florian Thalmayr, Florian Thalmayr and Florian Weig. Advanced driver-assistance systems: Challenges and opportunities ahead. <https://www.mckinsey.com/industries/semiconductors/our-insights/advanced-driver-assistance-systems-challenges-and-opportunities-ahead>, 2016.
- [54] Why FPGAs are a Good Fit for Automotive Driver Assistance Systems. <https://electronics360.globalspec.com/article/4506/why-fpgas-are-a-good-fit-for-automotive-driver-assistance-systems>.
- [55] Cyber Security is the No.1 Concern for Connected Devices on the Network Edge. <https://www.microsemi.com/product-directory/fpga-soc/1738-security>.
- [56] Deming Chen, Jason Cong, Peichen Pan, et al. FPGA design automation: A survey. *Foundations and Trends® in Electronic Design Automation*, 1(3):195–330, 2006.
- [57] Jason H. Anderson and Tomasz S. Czajkowski. Computer-Aided Design for FPGAs: Overview and Recent Research Trends. <https://janders.eecg.utoronto.ca/pdfs/chap.pdf>, 2012.
- [58] Jason Cong and Yuzheng Ding. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1994.
- [59] Stephen Jang, Billy Chan, Kevin Chung, and Alan Mishchenko. WireMap: FPGA technology mapping for improved routability. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, 2008.
- [60] Vaughn Betz and Jonathan Rose. Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size. In *Proceedings of CICC 97-Custom Integrated Circuits Conference*, 1997.
- [61] Alexander Marquardt, Vaughn Betz, and Jonathan Rose. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. In *FPGA*, 1999.
- [62] Vaughn Betz et al. VPR: A new packing, placement and routing tool for FPGA research. In *International Workshop on Field Programmable Logic and Applications*, 1997.

- [63] Shilpa Bhoj and Dinesh Bhatia. Thermal modeling and temperature driven placement for FPGAs. In *2007 IEEE International Symposium on Circuits and Systems*, pages 1053–1056. IEEE, 2007.
- [64] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.
- [65] Alexander Marquardt, Vaughn Betz, and Jonathan Rose. Timing-driven placement for FPGAs. In *Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*, 2000.
- [66] Intel Corporation. Intel Quartus Prime Software Suite. <https://www.intel.sg/content/www/xa/en/software/programmable/quartus-prime/overview.html>, 2019.
- [67] Tzu-Hen Lin et al. An efficient and effective analytical placer for FPGAs. In *Proceedings of the 50th Annual Design Automation Conference*, 2013.
- [68] Gang Chen and Jason Cong. Simultaneous timing driven clustering and placement for FPGAs. In *International Conference on Field Programmable Logic and Applications*, 2004.
- [69] Amit Singh, Ganapathy Parthasarathy, and Malgorzata Marek-Sadowska. Efficient circuit clustering for area and power reduction in FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2002.
- [70] Phillip Christie and Dirk Stroobandt. The interpretation and application of rent’s rule. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(6):639–648, 2000.
- [71] Ganapathy Parthasarathy, Malgorzata Marek-Sadowska, Arindam Mukherjee, and Amit Singh. Interconnect complexity-aware FPGA placement using Rent’s rule. In *Proceedings of the 2001 international workshop on System-level interconnect prediction*, pages 115–121. ACM, 2001.
- [72] Yue Zhuo et al. A congestion driven placement algorithm for FPGA synthesis. In *Field Programmable Logic and Applications, 2006. FPL’06. International Conference on*, 2006.
- [73] Ali Asghar and Husain Parvez. An improved diffusion based placement algorithm for reducing interconnect demand in congested regions of FPGAs. *International Journal of Reconfigurable Computing*, 2015:8, 2015.
- [74] Pongstorn Maidee, Cristinel Ababei, and Kia Bazargan. Fast timing-driven partitioning-based placement for island style FPGAs. In *Proceedings of the 40th annual design automation conference*, pages 598–603. ACM, 2003.

- [75] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181. IEEE, 1982.
- [76] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: applications in VLSI domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):69–79, 1999.
- [77] Marcel Gort et al. Analytical placement for heterogeneous FPGAs. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, 2012.
- [78] Shih-Chun Chen and Yao-Wen Chang. Fpga placement and routing. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 914–921. IEEE Press, 2017.
- [79] Larry McMurchie and Carl Ebeling. PathFinder: a negotiation-based performance-driven router for FPGAs. In *Reconfigurable Computing*, pages 365–381. Elsevier, 2008.
- [80] Chandra Mulpuri and Scott Hauck. Runtime and quality tradeoffs in fpga placement and routing. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, pages 29–36. ACM, 2001.
- [81] Chris C Wang et al. Scalable and deterministic timing-driven parallel placement for FPGAs. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, 2011.
- [82] Matthew An et al. Speeding up FPGA placement: Parallel algorithms and methods. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, 2014.
- [83] Intel Corporation. Intel Quartus Prime Pro Edition User Guide: Design Compilation). <https://www.intel.com/content/www/us/en/programmable/documentation/zpr1513988353912.html#jbr1444414867665>, 2019.
- [84] Junqi Yuan, Jialing Chen, Lingli Wang, Xuegong Zhou, Yinshui Xia, and Jianping Hu. ARBSA: Adaptive Range-Based Simulated Annealing for FPGA Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
- [85] Lei Cheng and Martin DF Wong. Floorplan design for multimillion gate FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2795–2805, 2006.
- [86] Pritha Banerjee, Susmita Sur-Kolay, and Arijit Bishnu. Floorplanning in modern FPGAs. In *20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07)*, pages 893–898. IEEE, 2007.

- [87] Maogang Wang et al. Multi-million gate FPGA physical design challenges. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, 2003.
- [88] Li Shang et al. Dynamic power consumption in VirtexTM-II FPGA family. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, 2002.
- [89] Altera Corporation. Reducing Power Consumption and Increasing Bandwidth on 28-nm FPGAs). <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01148-stxv-power-consumption.pdf>, 2012.
- [90] Tim Tuan et al. A 90nm low-power FPGA for battery-powered applications. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, 2006.
- [91] Ricardo Gonzalez, Benjamin M Gordon, and Mark A Horowitz. Supply and threshold voltage scaling for low power CMOS. *IEEE Journal of Solid-State Circuits*, 32(8):1210–1216, 1997.
- [92] Fei Li, Yan Lin, Lei He, and Jason Cong. Low-power FPGA using pre-defined dual-V_{dd}/dual-V_t fabrics. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 42–50, 2004.
- [93] Aman Gayasen, K Lee, Narayanan Vijaykrishnan, Mahmut Kandemir, Mary Jane Irwin, and Tim Tuan. A dual-v_{dd} low power fpga architecture. In *International Conference on Field Programmable Logic and Applications*, pages 145–157. Springer, 2004.
- [94] Intel. AN 711: Power Reduction Features in Intel Arria 10 Devices. <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an711.pdf>, 2020.
- [95] Ibrahim Ahmed, Linda L Shen, and Vaughn Betz. Becoming more tolerant: Designing FPGAs for variable supply voltage. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2019.
- [96] Shuze Zhao, Ibrahim Ahmed, Armina Khakpour, Vaughn Betz, and Olivier Trescases. A robust dynamic voltage scaling scheme for FPGAs with IR drop compensation. In *2017 IEEE Applied Power Electronics Conference and Exposition (APEC)*, pages 2939–2944. IEEE, 2017.
- [97] Joshua M Levine, Edward Stott, and Peter YK Cheung. Dynamic voltage & frequency scaling with online slack measurement. In *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*, pages 65–74, 2014.
- [98] Qiang Wang, Subodh Gupta, and Jason H Anderson. Clock power reduction for virtex-5 fpgas. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 13–22, 2009.

- [99] Safeen Huda, Muntasir Mallick, and Jason H Anderson. Clock gating architectures for fpga power reduction. In *2009 International Conference on Field Programmable Logic and Applications*, pages 112–118. IEEE, 2009.
- [100] Quartus II Handbook Volume 3: Verification. (2015, May 04). https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/qts_qii5v3.pdf.
- [101] Xilinx Inc. Xilinx Power Estimator (XPE). <https://www.xilinx.com/products/technology/power/xpe.html>, 2019.
- [102] Deming Chen et al. Low-power high-level synthesis for FPGA architectures. In *ISLPED*, 2003.
- [103] Xilinx Inc. Proven Power Reduction with Xilinx UltraScale FPGAs). https://www.xilinx.com/support/documentation/white_papers/wp466-proven-ultrascale-power-leaders.pdf, 2015.
- [104] Intel Corporation. INTEL CYCLONE 10 LP FPGA. <https://www.intel.sg/content/www/xa/en/products/programmable/fpga/cyclone-10/lp.html>, 2019.
- [105] Intel Corporation. Stratix Series FPGA Low Power Consumption Features). <https://www.intel.com/content/www/us/en/programmable/products/fpga/features/stx-power-about.html>, 2019.
- [106] Nikil Mehta and André DeHon. Low-Power Techniques for FPGAs. In *Low-Power Variation-Tolerant Design in Nanometer Silicon*. 2011.
- [107] Hao Li, Srinivas Katkoori, and Wai-Kei Mak. Power minimization algorithms for LUT-based FPGA technology mapping. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2004.
- [108] Amir H Farrahi and Majid Sarrafzadeh. FPGA technology mapping for power minimization. In *International Workshop on Field Programmable Logic and Applications*, 1994.
- [109] Jason Helge Anderson and Farid N Najm. Power-aware technology mapping for LUT-based FPGAs. In *2002 IEEE International Conference on Field-Programmable Technology, 2002.(FPT). Proceedings.*, 2002.
- [110] Julien Lamoureux and Steven JE Wilton. On the interaction between power-aware FPGA CAD algorithms. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, 2003.
- [111] Siobhán Launder, Colin Doyle, and Wesley Cooper. Switching-activity directed clustering algorithm for low net-power implementation of FPGAs. In *International Workshop on Power and Timing Modeling, Optimization and Simulation*, 2005.

- [112] Marvin Tom and Guy Lemieux. Logic block clustering of large designs for channel-width constrained FPGAs. In *Proceedings of the 42nd annual Design Automation Conference*, 2005.
- [113] Julien Lamoureux and Wayne Luk. An overview of low-power techniques for field-programmable gate arrays. In *2008 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 338–345. IEEE, 2008.
- [114] Subodh Gupta, Jason Anderson, Linda Farragher, and Qiang Wang. Cad techniques for power optimization in virtex-5 fpgas. In *2007 IEEE Custom Integrated Circuits Conference*, pages 85–88. IEEE, 2007.
- [115] Petar Borisov Minev and Valentina Stoianova Kukenska. The virtex-5 routing and logic architecture. *Annual Journal of Electronics, Technical University of Sofia*, 3:107–110, 2009.
- [116] Yaska Sankar et al. Trading quality for compile time: Ultra-fast placement for FPGAs. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, 1999.
- [117] Xilinx Inc. FPGA Design Flow Overview. https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm, 2008.
- [118] Russell Tessier. Fast placement approaches for FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2002.
- [119] Christopher Lavin et al. HMFlow: Accelerating FPGA compilation with hard macros for rapid prototyping. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, 2011.
- [120] Tannous Frangieh et al. A design assembly framework for FPGA back-end acceleration. In *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, 2012.
- [121] Farnaz Gharibian et al. Identifying and placing heterogeneously-sized cluster groupings based on FPGA placement data. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, 2014.
- [122] Marco Rabozzi et al. Floorplanning for partially-reconfigurable fpga systems via mixed-integer linear programming. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, 2014.
- [123] Kalindu Herath, Alok Prakash, Udaree Kanewala, and Thambipillai Srikanthan. Communication-aware Module Placement for Power Reduction in Large FPGA Designs. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018.

- [124] Kevin Lee et al. Shape exploration for modules in rapid assembly workflows. In *ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on*, 2015.
- [125] Fubing Mao et al. Dynamic module partitioning for library based placement on heterogeneous FPGAs. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2017 IEEE 23rd International Conference on*, 2017.
- [126] Kalindu Herath et al. Performance Estimation of FPGA Modules for Modular Design Methodology Using Artificial Neural Network. In *Applied Reconfigurable Computing, Architectures, Tools, and Applications*, 2018.
- [127] Altera Corporation. Increasing Productivity with Quartus II Incremental Compilation. <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01062-quartus-ii-increasing-productivity-incremental-compilation.pdf>, 2008.
- [128] Xilinx. Hierarchical Design Methodology Guide. https://www.xilinx.com/support/documentation/sw_manuals/xilinx12_2/Hierarchical_Design_Methodology_Guide.pdf, 2010.
- [129] Julien Lamoureux et al. On the interaction between power-aware FPGA CAD algorithms. In *2003 IEEE/ACM international conference on Computer-aided design*, 2003.
- [130] Huimin Bian et al. Towards scalable placement for FPGAs. In *FPGA*, 2010.
- [131] Kalindu Herath, Alok Prakash, Jiang Guiyuan, and Thambipillai Srikanthan. Communication-aware partitioning for energy optimization of large FPGA designs. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017.
- [132] Mentor. ModelSim. <https://www.mentor.com/products/fpga/verification-simulation/modelsim/>, 2019.
- [133] Intel Corporation. PowerPlay Power Analyzer Support Resources. 2019.
- [134] Louis-Noël Pouchet. Polybench: The Polyhedral Benchmark Suite, 2015.
- [135] Mechelle Gittens, Yong Kim, and David Godwin. The vital few versus the trivial many: examining the Pareto principle for software. In *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, volume 1, pages 179–185. IEEE, 2005.
- [136] Christopher Lavin et al. Improving clock-rate of hard-macro designs. In *Field-Programmable Technology (FPT), 2013 International Conference on*, 2013.
- [137] The MathWorks, Inc. Neural Network Toolbox. <https://www.mathworks.com/products/neural-network.html>, 2019.

- [138] Altera Corporation. AN 584: Timing Closure Methodology for Advanced FPGA Designs. <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an584.pdf>, 2014.
- [139] Kalindu Herath et al. Ant Colony Optimization based Module Footprint Selection and Placement for Lowering Power in Large FPGA Designs. In *2018 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2018.
- [140] N Kumbharana and Gopal M Pandey. A comparative study of ACO, GA and SA for solving travelling salesman problem. *International Journal of Societal Applications of Computer Science*, 2(2):224–228, 2013.
- [141] Anna Maria Sri Asih, Bertha Maya Sopha, and Gilang Kriptaniadewa. Comparison study of metaheuristics: Empirical application of delivery problems. *International Journal of Engineering Business Management*, 9:1847979017743603, 2017.
- [142] Hosam HA Mukhairez and Ashraf YA Maghari. Performance comparison of simulated annealing, GA and ACO applied to TSP. *Performance Comparison of Simulated Annealing, GA and ACO Applied to TSP*, 6(4), 2015.
- [143] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.
- [144] Xilinx Inc. Vivado Design Suite User Guide-Hierarchical Design. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug905-vivado-hierarchical-design.pdf, 2014.
- [145] Intel Corporation. Adaptive Logic Module (ALM) Definition. https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/reference/glossary/def_alm.htm, 2017.
- [146] eMedia Asia Ltd. Xilinx takes the lead in 2016 FPGA land. <https://www.eetindia.co.in/news/article/xilinx-takes-the-lead-in-2016-fpga-land>, 2019.
- [147] Microsoft. Project Catapult. <https://www.microsoft.com/en-us/research/project/project-catapult/>, 2020.
- [148] Vinay Chamola, Sambit Patra, Neeraj Kumar, and Mohsen Guizani. Fpga for 5g: Re-configurable hardware for next generation communication. *IEEE Wireless Communications*, 2020.
- [149] Inc Avnet. Xilinx Automotive Solutions.
- [150] MarketsandMarkets Research Private Ltd. Cloud Computing Market. <https://www.marketsandmarkets.com/Market-Reports/cloud-computing-market-234.html>.
- [151] MarketsandMarkets Research Private Ltd. Machine Learning Market . <https://bit.ly/3exglXn>.
- [152] Valuates Reports. Autonomous Vehicle Market. <https://bit.ly/3h1E976>.