Rapid Overlay Builder for Xilinx FPGAs

by

Xi Yue

B.A.Sc., University of Toronto, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REUIQEMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

The University of British Columbia

(Vancouver)

October 2014

Abstract

A field-programmable gate array (FPGA) is a type of programmable hardware, where a logic designer must create a specific hardware design and then "compile" it into a bitstream that "configures" the device for a specific function at power-up. This compiling process, known as place-and-route (PAR), can take hours or even days, a duration which discourages the use of FPGAs for solving compute-oriented problems. To help mitigate this and other problems, overlays are emerging as useful design patterns in solving compute-oriented problems. An overlay consists of a set of compiler-like tools and an architecture written in a hardware design language like VHDL or Verilog. This cleanly separates the compiling problem into two phases: at the front end, high-level language compilers can quickly map a compute task into the overlay architecture, which is now serving as an intermediate layer. Unfortunately, the back-end of the process, where an overlay architecture is compiled into an FPGA device, remains a very time-consuming task. Many attempts have been made to accelerate the PAR process, ranging from using multicore processors, making quality/runtime tradeoffs, and using hard macros, with limited success. We introduce a new hard-macro methodology, called Rapid Overlay Builder, and demonstrate a run-time improvement up to 22 times compared to a regular unaccelerated flow using Xilinx ISE. In addition, compared to prior work, ROB continues to work well even with high logic utilization levels of 89%, and it consistently maintains high clock rates. By applying this methodology, we anticipate that overlays can be implemented much more quickly and with lower area and speed overheads than would otherwise be possible. This will greatly improve the usability of FPGAs, allowing them to be used as a replacement for CPUs in a greater variety of applications.

Preface

Research was conducted with insight and efforts from Dr. Guy Lemieux and Dr. Dirk Koch. The Rapid Overlay Builder methodology was built on a CAD tool called GoAhead co-developed by Dr. Koch. The VHDL code for the floating-point operations used in the complex PE was prepared and experiments presented in Section 4.2 were performed by Dr. Koch.

Table of Contents

List of Tables

List of Figures

Glossary

ALU Arithmetic Logic Unit

BRAM Block Random Access Memory

CAD Computer Aided Design

CGRA Coarse-Grained Reconfigurable Array

CLB Configurable Logic Block

CPU Central Processing Unit

DDR Double Data Rate

DSP Digital Signal Processing

FPGA Field-programmable Gate Array

ISA Instruction Set Architecture

LUT Look-Up Table

PAR Place-And-Route

PCIe Peripheral Component Interconnect Express

PE Processing Element

RTL Register-Transfer Level

Acknowledgement

I would like to sincerely thank my supervisor Dr. Guy Lemieux for his guidance and patience throughout the program. Without his insight and support, I would not be completing the degree and writing this thesis.

I would also like to thank Dr. Dirk Koch for his monumental support and input to this thesis. This thesis would have been impossible without his knowledge and expertise in this research area.

Finally, special thanks goes to my parents and my wife Wen for always supporting me in all aspects over the years.

Chapter 1

Introduction

1.1 Motivation

Modern FPGA devices contain over 1 million LUTs and over 1000 dedicated memory and multiplier blocks. As circuits continue to scale up, the long place-and-route process required by the CAD tools forms a growing concern. To address this issue, vendor tools and other related CAD research accelerate the compilation process using different techniques, including parallel compilation [1-8], design partitions [9], netlist preservation [1-2] and trading circuit performance [10] for faster compilation. However, the speedup provided by these approaches is still limited.

Recently, a new design flow that uses overlay architectures has emerged to solve reconfigurable problems. Overlay architectures, which are pre-compiled circuits that are reconfigurable themselves, provide a higher level of abstraction to the hardware designers. The new design flow directly maps applications to the overlay architectures using custom tools or compilers, and therefore eliminates the long place-and-route times, which are found in the traditional hardware design flow. This significantly boosts the design

productivity for users by allowing shorter turnaround times, which can be found in software development.

Despite the fact that overlay architectures enable a more interactive and portable application development process, implementing an overlay architecture in an FPGA device still suffers from long place-and-route times [11]. With these long place-and-route times, overlays cannot be as nimble and dynamic through the use of *specialization*, where the overlay architecture is highly customized to the needs of the application. As an extreme example, it prevents users from generating a new architecture implementation each time they change their algorithm, even though that may be beneficial to the overall result.

A number of previous research efforts accelerate the place-and-route process in a component-based design fashion, which fits the overall profile of overlay architectures. One of the most notable techniques employed in this research area is module relocation. Module relocation is a technique of relocating netlists of a pre-built module in other locations of the device, and is a technique that is commonly applied on Xilinx FPGAs. By preserving and reusing previous CAD efforts, the place-and-route problem size is reduced and the process is accelerated. While some of the related research [12] relocates post-place netlists, others [13][14][15] relocate post-route netlists, which are also known as hard macros in a Xilinx context. For example, in HMFlow[13], a simulated annealing macro placer is developed and swaps hard macros using module relocation to achieve better placement results. Although [25] obtained significant speedups in the place-and-route process, the compilation flow cannot guarantee the speedups when logic utilization is above 50% and the resulted clock rates can only achieve 75% of the clock

rates resulted from Xilinx tools. In addition to the research efforts that employed module relocation, a bottom-up compilation flow [11] utilizes partitioning and floorplanning with pre-built modules to accelerate the place-and-route process while high clock rates are maintained. However, the highest logic utilization level reported in [11] is 70% and it is not clear whether the speedups can be maintained when circuits continue to scale. While reusing pre-built modules significantly reduces the placement time, the time consumption in the routing process dominates in the compilation flow of [11][13].

1.2 Approaches

To overcome these limitations, this thesis presents the Rapid Overlay Builder (ROB) methodology for Xilinx FPGAs. ROB accelerates the place-and-route process of building overlays that can be floorplanned into a set of adjacent modules. Most overlays fall into this category, containing a high degree of repetition and regularity. For example, array-based coarse-grained reconfigurable architectures (CGRAs) are ideal in that they have a regular layout and they replicate a similar (but not necessarily identical) processing element (PE) at each site.

ROB is a component-based design methodology, including a set of scripts, tools, and know-how, that interacts with the Xilinx ISE toolchain. To obtain fast place and route speeds, it takes advantage of three key underlying techniques: (1) module relocation, (2) module variants, and (3) stitching modules by zipping. Module relocation compiles a module into a hard macro; it can usually be relocated almost anywhere vertically with little or no additional CPU effort. Module variants (or design alternatives) are modules with the exact same functionality but mapped to a different resource footprint. This

assists horizontal relocation in the presence of heterogeneous columns found in modern FPGAs. Zipping is a routerless method of stitching adjacent modules with zero overhead, such that their interconnect aligns perfectly without any extra logic, switches, or wires. Zipping also allows very tight packing of adjacent modules. All three techniques are utilized by ROB to reduce place-and-route effort. Although not presently done in ROB, the decomposition above also allows for easy parallelization across multiple workstations.

1.3 Contributions

This thesis utilizes pre-existing techniques including module relocation, module variants and a routerless stitching mechanism in the ROB methodology. The contributions of this thesis are summarized as follows:

1. Obtaining scalable speedups in building CGRA designs
2. Achieving high logic utilization level with scalable speedups
3. Maintaining consistent and high clock rates of CGRA designs

1.4 Thesis Organization

The remainder of this thesis is organized as follow. Chapter 2 presents background information of overlay architectures and related technology employed in the ROB methodology. Chapter 3 describes a homogeneous CGRA using integer-only PEs that is used in a case study and a heterogeneous CGRA with floating-point capabilities. Chapter 4 describes the ROB methodology in further details. Chapter 5 compares the results from

Xilinx ISE and the ROB methodology. Chapter 6 lists the limitations of this thesis and some future work. Lastly, Chapter 7 presents conclusions of the thesis.

Chapter 2

Background

This chapter first presents an overview of overlay architectures. Then, related previous work on accelerating the place-and-route process will be described, with their limitations. Next, this chapter gives background information of the target FPGA device and defines terminology being used throughout this thesis. Finally, the previously known techniques that the Rapid Overlay Builder (ROB) methodology employs will be presented and similar tool flows will be described with their limitations.

2.1 Overlay Architectures

Overlay architectures can be thought of as pre-compiled circuits that are reconfigurable themselves. The overlay provides a higher level of abstraction to application developers than just “raw gates” provided by an FPGA. More precisely, an overlay is a framework, consisting of custom tools or compilers as well as an RTL description of the architecture, which transforms a general-purpose FPGA into a compute-oriented structure. The overlay tools provide two productivity boosts for users: they tend to run quickly, like software

compilers, and they enable the use of a more programmer-friendly language. They can also enable portability across devices, device families, and vendors.

Just as an FPGA loads a *configuration bitstream* containing an implementation of the overlay RTL, the overlay itself must load an *application bitstream* generated by the overlay tools. The second step is called *personalization* in this thesis.

A number of overlay architectures have been proposed, including ZUMA [16][17], iDEA [18], MXP [19], Octavo [20], VLIW-SCORE [21], Soft-CGRA [22], and Mesh-of-FUs [23]. These overlays suffer from very long place-and-route times required to implement the architecture in the FPGA. For Altera FPGAs, design partitioning was found to help quality of results from CAD, but not mapping time [9].

A common feature of all these overlays is the repetition of a *processing element* or PE across the device. The PEs can either be homogeneous or heterogeneous. In some cases, the PEs can be *specialized*, where some of the PE flexibility is removed to save area and/or improve delay.

The overlay architecture studied in this thesis is an array-based CGRA, where processing elements (PEs) communicate only with their nearest neighbours. This scheme is often used in related work [24]. Consequently, the ROB methodology can be easily applied to these CGRA architectures. Other overlay implementations that use a floorplan to lock the position of adjacent modules will also work.

2.2 Fast Compilation with FPGAs

As FPGA capacity continues to scale up, the long place-and-route (PAR) process required by the CAD tools forms a growing concern. Although overlay architectures emerge as a way to boost design productivity, placing and routing overlay architectures themselves still takes a long time to complete [11]. This section reviews prior work and their limitations on accelerating the PAR process.

2.2.1 Parallel Compilation

One promising solution to long PAR times is to parallelize CAD algorithms. By employing multiple processor cores, PAR problems are divided into smaller problems that can be solved concurrently. There have been a number of prior work that proposed parallel algorithms for placement [1-6] and routing [7-8]. Although Altera and Xilinx enabled the capabilities of parallelizing the PAR process, most of the prior work were done using the VPR framework due to limited access to the proprietary PAR tools from the vendors. Despite the fact that parallel compilation can lead to reasonable speedups, the resulted size of the bounding boxes and clock rates might be degraded, especially when a PAR problem is highly parallelized. Careful and comprehensive experiments need to be performed in order to understand the tradeoff between CAD time speedups, logic utilization levels and clock rates.

2.2.2 Netlist Preservation

Incremental compilation of design partitions is another approach to accelerate the PAR process by reducing the problem size. Instead of a flat compilation of the entire design, vendor tools provide options to utilize netlists of modules that were compiled before

[1][2]. The post-routed netlists of one module is known as a *hard macro* in Xilinx context. Despite the high degree of repetition of resources present on FPGA devices, vendor tools have very limited support in relocating pre-built hard macros. Therefore, a PAR process is required for instantiating the same module in other locations of the FPGA devices.

2.2.3 Trading Circuit Performance

While most CAD research focused on improving circuit performance, some research worked on trading circuit performance for fast PAR runtimes. By accelerating the PAR process, debug cycles will be shortened, which helps with improving productivity of hardware designers. Mulpuri [10] examined the tradeoffs between routing quality and PAR runtimes. With a 27% degradation of circuit performance, the PAR process obtained a 3x speedup. HMFlow [25] utilized hard macros to accelerate the PAR process by 30x, while the clock rates remain 75% of the clock rate achieved by the vendor tool. Unlike these research efforts, this thesis focuses on accelerating the PAR process, while maintaining high clock rates with vendor tool standards.

2.3 Targeted FPGA Device

Modern FPGA devices contain over 1 million LUTs and over 1000 dedicated memory and multiplier blocks, providing heterogeneous types of underlying resources to meet the demand of hardware designers. These heterogeneous resources with their own physical sizes are unevenly distributed across the FPGA devices.

The targeted device in this thesis is a Xilinx XC6VLX240T-FF1156 from an ML605 board. There are four different types of resources in the targeted device: DSP block (D),

BRAM block (B), Slice-M featured CLB (M) and Slice-L featured CLB (L). DSP blocks contain dedicated multipliers and BRAM blocks contain dedicated memory. The Slice-M featured CLBs contain LUTs that can alternatively be configured as memory or shift registers, where as Slice-L featured CLBs can only be used as logic. On the targeted Virtex 6 device, the height of one DSP block is the same as the height of one BRAM block and equivalent to the height of five CLBs. It is also important to know that the height of a clock region on the device is equivalent to the height of 40 CLBs. In this thesis, none of the implemented modules will span across the boundary of a clock region. This implementation allows a better support in partial reconfiguration, which can be explored in future work.

A placed and routed module is called a *pre-built* module in this thesis. For a given pre-built module, this thesis defines its *footprint mask* as the set of underlying resources (columns) used by the module. For example, if a module is implemented on a device and utilizes one DSP column, one Slice-M featured CLB column, one Slice-L featured CLB column and one BRAM column from left to right, then the corresponding footprint mask is {D, M, L, B}, and the footprint mask width is 4.

2.4 Related Technology Overview

In this section, an overview of previously known techniques that will be employed by the ROB methodology is first presented. Lastly, this section compares the ROB methodology

with other similar component-based design flows, the limitations of which will then be detailed.

2.4.1 Module Relocation

Component-based system design using module relocation is an efficient technique employed in this thesis to accelerate the PAR process. Although Xilinx vendor tools do not support module relocation well, there are various works reported to support this feature. The earliest attempts on supporting module relocation were done by relocating bitstreams using custom CAD tools. These tools, including PARBIT [26], REPLICA [27] and REPLICA2Pro [28], modify addresses within the bitstream to enable relocation. These CAD tools only provide support for legacy Xilinx devices, and unlike post-routed netlists, the relocated bitstream is not timing verifiable. In [29], a methodology is proposed for enhancing relocation flexibility by not using the primitives of certain resource columns. Those columns can then act as a wildcard for module placement because the routing fabric is identical for logic columns, memory columns and multiplier columns on Xilinx FPGAs. Unfortunately, the skipped columns become stranded resources that cannot be utilized.

Recent work on module relocation is based on hard macros at the netlist level [12-15]. They utilize custom CAD tools that calculate a set of valid placements that follow the footprint mask requirements. In prior work, some macro placers [13][14][15] are able to preserve post-routed netlists during relocation, while [12] only preserves the placement during this process. Fitting modules into predefined bounding boxes typically results in both internal and external fragmentation. For HMFlow, the authors report that rapid compilation cannot be guaranteed when the logic utilization exceeds 50% [13].

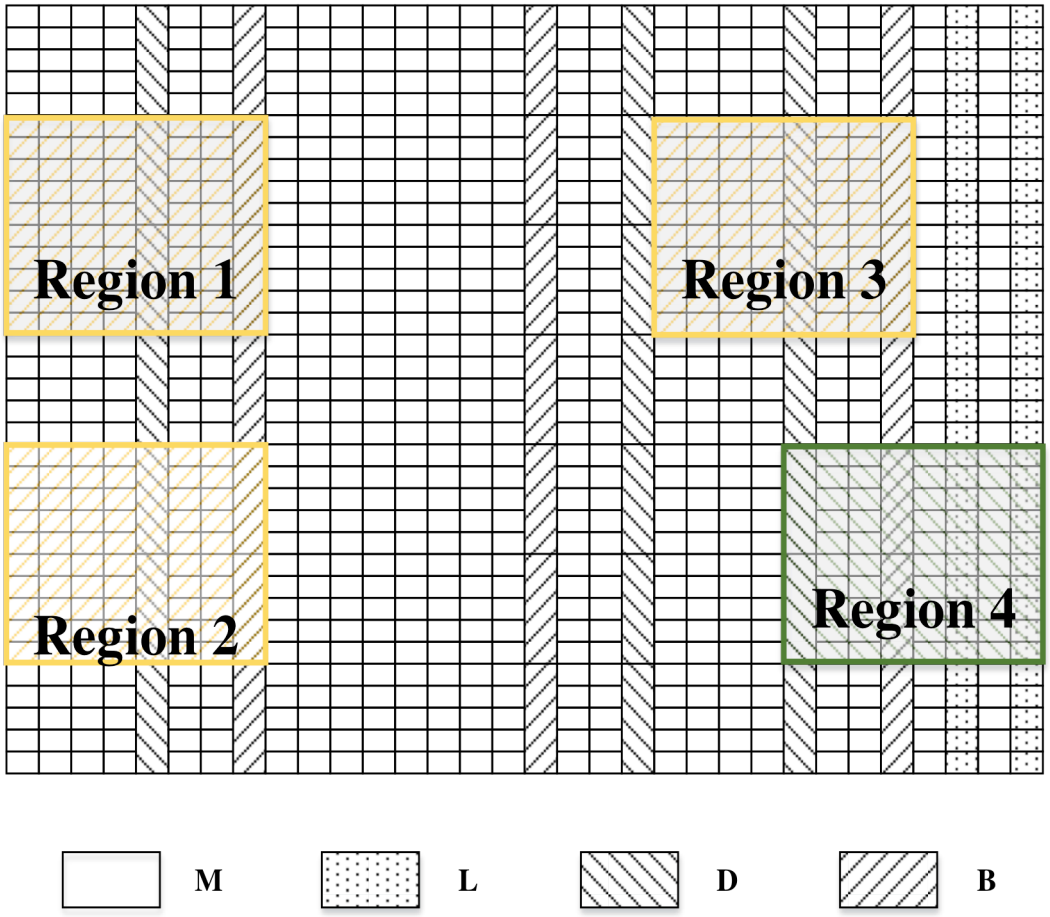

Figure 2.1: Valid Module Relocation Scheme

2.4.2 Module Variants

Theoretically, a pre-built module is relocatable to locations with an identical resource footprint. As shown in Figure 2.1, Region 1 corresponds to the bounding box of a module with footprint mask {M, M, M, M, D, M, M, B}. The *footprint mask width* of Region 1 is 8. In Xilinx FPGAs, Region 1 can be vertically relocated to Region 2, or horizontally relocated to Region 3. In general, however, the number of compatible horizontal placement sites is quite restricted. For example, Region 1 cannot be relocated to Region 4, because the destination footprint mask {D, M, M, B, M, L, M, L} differs. Instead, to

utilize Region 4, the module needs to be re-placed and re-routed within an appropriate bounding box [30]. After this, the new module instance in Region 4 is created, which we call a *variant* of the original module. The ROB methodology creates a set of *module variants* to increase horizontal relocation flexibility. It also attempts to reduce the number of variants required.

In the context of the CGRA that this thesis studied, the module variants also known as PE variants, are utilized in the ROB methodology. The post-routed netlists of a PE variant constrained in a bounding box defined by a floorplan is called a *PE tile*. The targeted device in this thesis is a Xilinx XC6VLX240T-FF1156 from an ML605 board. Its footprint mask has 101 columns, as shown in Figure 2.2. The left and right sides of the device have similar footprint masks, which can be exploited to reduce the number of required PE variants by half. Building a CGRA using a set of PE variants not only lowers the external fragmentation, but also helps with achieving consistent clock rates of the CGRA.

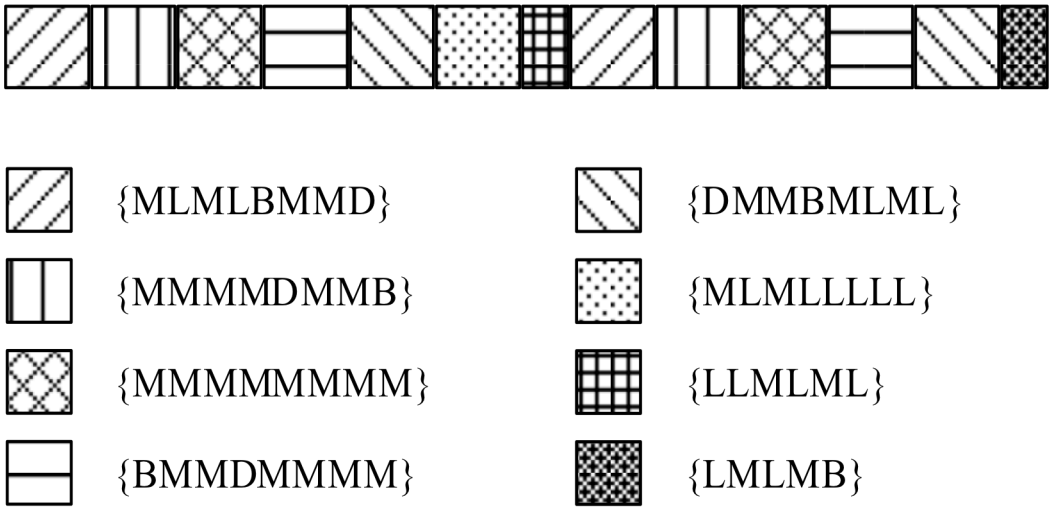

Figure 2.2: Virtex-6 VLX240T Resource Footprint Mask

2.4.3 Routerless Stitching

To stitch instantiated modules together, a routing process is usually required. To reduce overhead, instead of invoking the vendor's routing tool, some prior work [4][6] developed custom routers for the stitching process. Custom routers were required because the vendor routing tool only takes NCD format netlists as input, whereas these tools work with hard macros described by XDL format netlists. Conversion from XDL to NCD to complete this operation is problematic because (1) it flattens hard macros and (2) routes inside hard macros are not guaranteed to remain intact during the stitching process.

This thesis tackles the stitching process in a different way such that the routing step can be eliminated. One way to achieve this routerless objective is to use bus macros [31]. However, this option was not considered here for three reasons (1) bus macros need considerable extra logic, (2) they add extra latency, and (3) bus macros would have impacted the placement flexibility (e.g. they cannot be placed on BRAM columns).

The stitching mechanism used in this thesis is a method we call zipping [32]. This mechanism enables direct module-to-module communication without any logic overhead and allows higher logic utilization in return. In the ROB methodology, fitting modules into bounding boxes and placing them adjacently provides locality that allows for short, predefined routes. This results in zero area and delay overhead on the connections. As the routing process is replaced by simple netlist manipulation, the stitching process is accelerated.

2.4.4 Related Research Comparison

Table 2.1 compares prior work that accelerated the PAR process in a component-based design fashion.

CAD Features and Results	HMFlow [25]	Bottom-up Compilation [11]	ROB Methodology
Platform	Xilinx	Altera	Xilinx
Module Relocation	post-routed netlists	N/A	post-routed netlists
Module Variants	no	no	yes
Stitching Process	custom router	vendor's router	routerless
Speedups	30x	33x	22x
Logic Utilization Achieved	50%	70%	89%
Clock Rates	0.75x	1.02x	1.37x

Table 2.1: Component-based Design Related Research Comparison

HMFlow [25] utilized hard macros to accelerate the PAR process and is able to obtain a speedup up to 30x. However, the speedup is not guaranteed when a design has a logic utilization rate above 50% [13]. This is one drawback of the hard macro placer developed in HMFlow. In HMFlow, before the placement process occurs, the tool computes a set of valid locations for hard macros that are being placed on the device. A simulated annealing placer is then invoked to place the hard macros. Unlike a traditional placer swapping primitive instances, the placer swaps entire hard macros, including the primitive instances and routed nets inside the hard macros, to achieve better placement results. Because hard macros have irregular sizes and different aspect ratios, the external fragmentation has to remain high to allow unutilized area for the placer to swap hard macros. In addition, HMFlow can only achieve clock rate that is 75% of the clock rate achieved by the vendor tool.

Bottom-up compilation [11] utilized circuit partitioning and floorplanning to achieve high scalability of large custom mesh-of-functional-units overlays. This compilation flow accelerated the PAR process by stitching post-routed tiles that were implemented offline,

obtaining a 33x speedup. However, unlike ROB, the post-routed tiles proposed in this flow are not relocatable. To make the post-routed tiles portable on an FPGA device, this compilation flow has to build all possible tile variants and to store these variants in the tile library. In addition, the stitching process is handled by the vendor's routing tool, which can take 35 minutes to route a design with a 70% utilization rate. Instead, the ROB methodology employs zipping to accelerate this process by simple netlist manipulation.

It is important to understand that all speedups in Table 2.1 are achieved by excluding the time for building initial modules/PEs. In addition, although we include the XDL conversion time to calculate the speedup of the methodology, it was not explicitly stated in [25] whether the XDL conversion time was also excluded.

2.5 Summary

This chapter presents the overview of overlay architectures and illustrates the advantage of the use of overlay architectures. Related previous research on accelerating the PAR process using traditional methods of parallel compilation, netlist preservation and trading circuit performance were described, with their limitations. In addition, we combine module relocation, module variants, and routerless stitching into the ROB methodology to achieve ultra-fast compilation speeds. Meanwhile, The ROB methodology gets high run-time speedups that have been proven with logic utilization levels up to 89%. Such a high level of utilization is usually very difficult for most tools to achieve. In comparison, HMFlow has difficulty exceeding 50% and Bottom-up Compilation Flow was not tested beyond 70% utilization. Lastly, the clock rates resulted from the ROB methodology are consistent and higher than other similar tool flows described in this section.

Chapter 3

CGRA Architecture

The focus of this thesis is to accelerate the compilation process of building overlays that have some regularity and repetition. To provide a concrete demonstration, this chapter presents two CGRA architectures to be used in developing the ROB methodology. The first architecture uses a simple PE; each simple PE can perform a common set of integer operations. The second architecture uses complex PEs that include both integer and floating-point operations. Furthermore, the first architecture is always homogeneous, where all PEs are identical, whereas the second architecture will apply specialization to its columns; each column of complex PEs in the CGRA will be homogeneous and support only a subset of the total operations. Applying this to other overlays should require only minor adaptations.

3.1 Homogeneous CGRA with Simple PEs

The first CGRA chosen for the case study consists of a homogeneous 2D array of simple processing elements (PEs); these PEs are called simple because they only support integer operations. The CGRA architecture and PE structure is shown in Figure 3.1.

Each PE communicates with each nearest neighbor through a register labeled N, S, E and W. Each communication channel (direction) consists of a 32-bit input bus and 32-bit output bus. These buses are used to send operands and results between PEs. Input and output buses of the PEs at the outer edge of the CGRA are connected in a loop-back manner. In addition, an input personalization bus and an output personalization bus are used for streaming *application bitstream* data along the PEs in a daisy-chained manner. To keep the personalization buses compact, they are only 4 bits wide plus an extra "enable" bit.

Each PE also has a local register labeled R in the figure for holding intermediate results. The ALU, which is capable of an assortment of integer operations (shifting, addition, subtraction, multiplication and some bit manipulations but not division), takes its operands from any of the N, S, E, W, or R registers and writes back to R and/or any other register(s) in one cycle. In addition, a crossbar may concurrently route data across the PE as long as there are no destination conflicts.

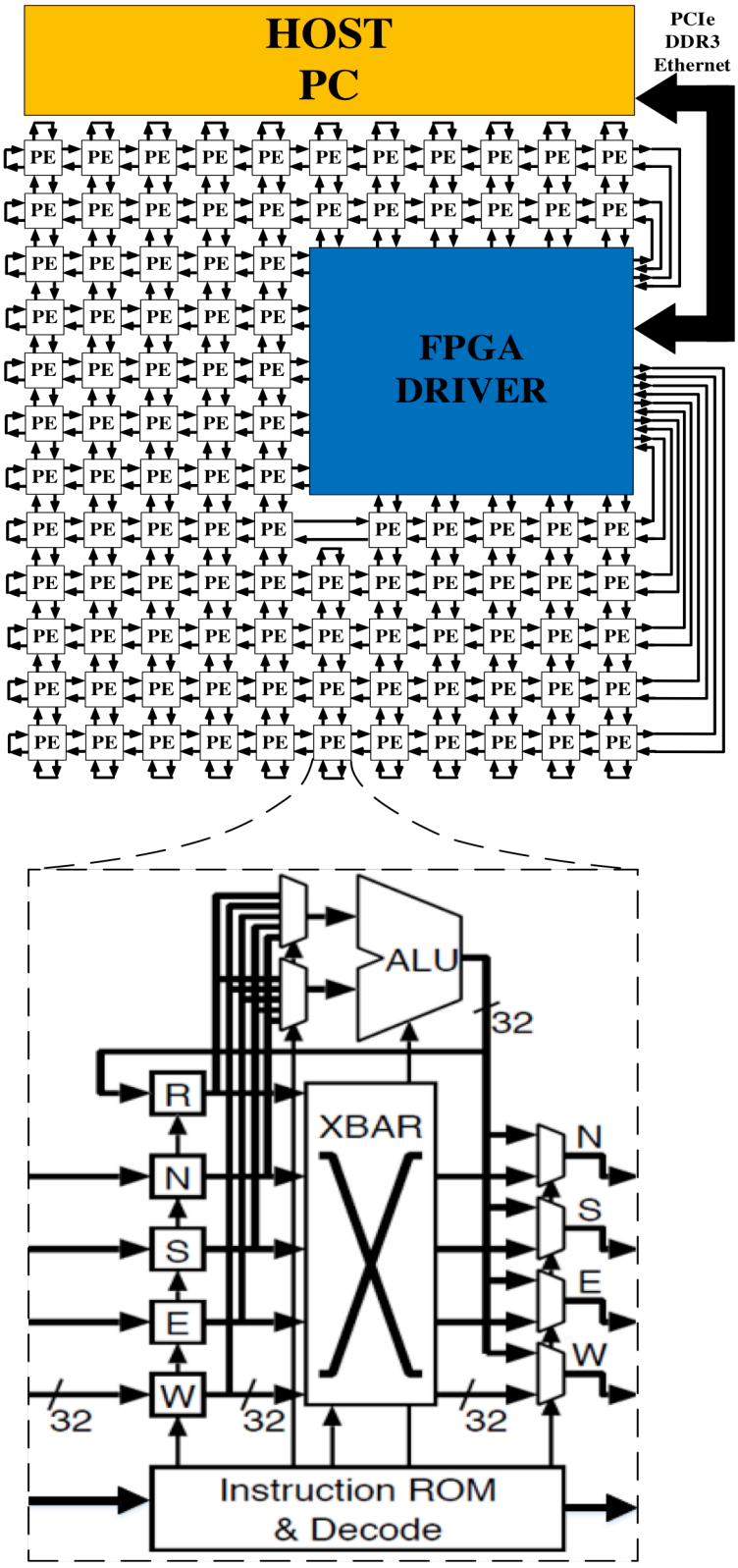

Figure 3.1: 101-PE CGRA Design with FPGA Driver

3.2 FPGA Driver

The entire CGRA is designed to communicate with DDR3, Ethernet, and a PC host over PCIe. Hence, a large, predesigned IP block called the FPGA Driver [33] is used to connect these I/O devices. The FPGA Driver occupies 8114 logic slices, which is an area equivalent to 30 simple PEs in the homogeneous CGRA. To meet timing requirements, it is constrained in the right side of the device for direct access to the I/O pins for Ethernet, PCIe and DDR3 memory. The FPGA Driver is responsible for: (1) streaming data between PCIe, the on-board DDR3 memory and the CGRA overlay, (2) carrying out the personalization process using the application bitstream, (3) enabling the partial reconfiguration capability. The FPGA Driver feeds *application data* and the *personalization bitstream* into the CGRA from the rightmost PEs of each row, and collects the computation results from the CGRA.

3.3 Heterogeneous CGRA with Complex PEs

To demonstrate a more complex usage scenario, a *customizable heterogeneous CGRA* using complex PEs with both integer and floating-point capabilities is also built. However, since a complex PE which supports all floating-point operations is quite large (812 slices), we will apply specialization to columns of PEs in Chapter 5 to save area. This specialization will involve forcing a column of PEs to support just one floating-point operation, but varying which operation is supported from column to column. This results in a heterogeneous CGRA architecture.

When applying specialization to the CGRA, it is assumed that a fully general-purpose PE is often underutilized in a CGRA that is running a specific application. Hence, by analyzing an application (or a domain), designers can not only determine whether some instructions go completely unused, but also determine the appropriate mixture among the remaining instructions. In addition, the designers must also consider the parallelism profile of these instructions in the application, and provide enough concurrency for each. Bearing this all in mind, it is beneficial for designers to create a heterogeneous set of specialized PEs for the CGRA that supports the required application. These heterogeneous PEs must ultimately be placed in the CGRA. This is partially analogous to ISA subsetting [34] where a CPU is customized to only provide the instructions that are needed by an actual program it is supposed to run.

Specialized PE	**Logic Slices**		**DSP Blocks**	
	PE Variant 1	**PE Variant 2**	**PE Variant 1**	**PE Variant 2**
FADD	260	270	0	0
FSUB	267	264	0	0
FDIV	279	261	0	0
FMUL	312	306	3	3
FCONV	249	246	2	2
ALU	152	136	3	3
Complex PE	812	812	6	6

Table 3.1: Resource Breakdown of Specialized PEs and the Complex PE

Table 3.1 shows the resource breakdown of the specialized PEs and the complex PE. In the experiments, it is found out that all of the specialized PEs can fit into a tile that is 10 columns x 20 rows. In contrast, the complex PE requires a tile that is 20 columns x 30 rows.

A customized CGRA example is shown in Table 3.2, where the CGRA consists of 8 specialized PE columns and 12 specialized PE rows. All specialized PEs in the same PE column are identical.

Column #	1	2	3	4	5	6	7	8
Specialized PE	FADD	FSUB	FDIV	FMUL	FCONV	ALU	FADD	FSUB

Table 3.2: A Customized CGRA Example

In this thesis, we do not concern ourselves with precisely how one determines the mixture of these PEs. It is simply enough to assume that some type of specialization must be applied, where each column of PEs may contain a PE design that has been uniquely specialized relative to other columns.

The main advantage of a CGRA with specialized PEs is better device utilization, allowing either a larger CGRA (in terms of PE tiles), or the ability to implement a given CGRA on a smaller FPGA device. In addition, the specialized PEs offer the potential for higher clock speeds.

It is important to understand that such customization of CGRA is best done after mapping an application to the CGRA. This is because designers only know the application needs after the mapping phase. This suggests that one cannot simply prebuild the customized overlay before the application. With conventional place and route approaches, an architecture customization like this results in long compilation time, making such a practice infeasible. In contrast, the stitching mechanism employed in the ROB methodology enables architectural customization with the rapid software-like compilation times being preserved.

The customization process is detailed in Chapter 4.

Chapter 4

Rapid Overlay Builder

This chapter presents the Rapid Overlay Builder methodology, or ROB for short. The use of this methodology will first be demonstrated as a case study to build the homogeneous CGRA with simple PEs described in Section 3.1. Then, the CGRA customization process will be presented using the heterogeneous CGRA with complex PEs described in Section 3.3.

4.1 Methodology

There are seven major tasks needed to build a CGRA in ROB. Out of the seven tasks, only the first two tasks presently require manual engagement from the users, while scripts have automated the other five tasks. While the first two tasks are also ultimately scriptable, they are not yet automated due to time limitations. The seven tasks are:

1. Resource budgeting for a PE tile
2. Floorplanning for CGRA designs

3. Placing and routing initial PE variants using Xilinx ISE
4. Extracting PE tiles from initial PE variants
5. Relocating PE tiles on the device
6. Interconnecting adjacent PE tiles
7. Interconnecting between the CGRA design and the FPGA Driver

Below, these seven tasks are covered in greater details. However, to help better understanding the methodology and the results, this section will first review the key feature of the target FPGA device.

4.1.1 Targeted FPGA Device

The targeted FPGA device in this thesis is a Xilinx XC6VLX240T-FF1156 from an ML605 board. As previously described in Section 2.3, four different types of resources present in the targeted device: DSP block (D), BRAM block (B), Slice-M featured CLB (M) and Slice-L featured CLB (L). The resource footprint mask of the device, representing the set of underlying resources of the device, is shown in Figure 4.1.

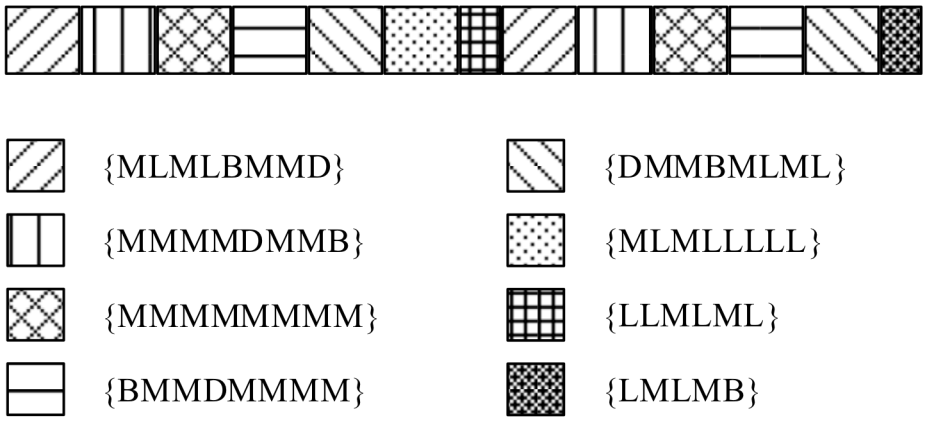

Figure 4.1: Virtex-6 VLX240T Resource Footprint Mask

In order to understand the process of floorplanning and defining PE variants, it is important to know the left and right sides of the device have similar footprint masks, which is shown in Figure 4.1. This similarity can be exploited to reduce the number of required PE variants by half. In addition, the height of a clock region is equivalent to the height of 40 CLBs, whereas the height of a DSP block and a BRAM block is equivalent to the height of 5 CLBs. To simplify the shape of a PE tile, we define a PE tile has to be rectangular and the height of a PE tile has to be a multiple of 5 CLBs.

Although this target FPGA device was chosen for this thesis, the ROB methodology should work on other FPGA devices with minor adaptions.

4.1.2 Resource Budgeting for a PE Tile

In order to floorplan the CGRA properly, an initial place-and-route process of a PE tile is required to obtain the set of required resources as a reference. The place-and-route process is a standard Xilinx ISE compilation run and does not require any constrained floorplan of the PE tile. However, a PE design may require heterogeneous types of resources. Later on in our case study, we will find that some resources such as hard multiplier blocks may not always be readily available nearby when floorplanning. In such a case, the PE tile can use more logic slices by use of a soft multiplier. Therefore, the PE tile sometimes needs to be built with different synthesis options, including whether to use hard multiplier blocks and memory blocks. This provides a comprehensive reference designs with different resource

Synthesis Option #	Logic Slices	Multiplier Blocks
1	230	2
2	304	N/A

Table 4.1: Resource Requirement Reference for a PE Tile

demands and that can add flexibility in floorplaning the PE variants later. With a comprehensive reference set of required resources for the PE tile, designers have a preliminary understanding of the size of the PE tile with different resource footprints on the device.

Table 4.1 shows the reference set of required resource of the integer-only PE design with different synthesis options. Synthesis results for utilizing memory blocks are not shown in the table, because the PE design does not utilize any memory blocks. After removing the resources reserved for the FPGA Driver from the target FPGA, there are a total of 26,160 logic slices and 304 multiplier blocks available.

4.1.3 Floorplanning for CGRA Designs

After reserving room for the FPGA Driver, Table 4.2 shows floorplan alternatives consisting of PE tiles with different aspect ratios that accommodate about 230 logic slices (115 CLBs). It also gives the corresponding external fragmentation (leftover CLBs) after instantiating the maximum number of PE tiles on the device. In some of the floorplan alternatives, after defining bounding boxes that include DSP blocks, it is found that the

Floorplan Option #	PE Width (CLBs)	PE Height (CLBs)	Number of PEs	External Fragmentation (CLBs)
1	24	5	93	1920
2	12	10	95	1680
3	8	15	102	720
4	6	20	101	840
5	5	25	78	3330
6	4	30	81	3360

Table 4.2: Floorplan Alternatives

logic resources left over can also be sufficient to build a logic-only PE tile. In such a case, logic-only PE tiles are built to lower the external fragmentation.

It is shown in Table 4.2 that floorplan options #3 and #4 yield the lowest external fragmentation. Floorplan option #4 is adopted because the resulting PE tiles will be half the height of a clock region. Hence, none of the PE tiles will span across the boundary of a clock region. This also allows the floorplan to be used in a dynamically reconfigurable system, which can be explored in future work.

The chosen floorplan consists of 11 PE tiles in the horizontal direction and 12 PE tiles in the vertical direction. Since the PE tiles are relocatable in the vertical direction, only one PE variant is needed for every column of PE tiles. Furthermore, with the feature similarity found on the left and right side of the device, only 6 variants of each Simple PE are required across the 11 PE columns. Figure 4.2 presents the footprint masks of all 6 PE variants utilized in the floorplan. With this set of PE variants, CGRAs of 8 different sizes

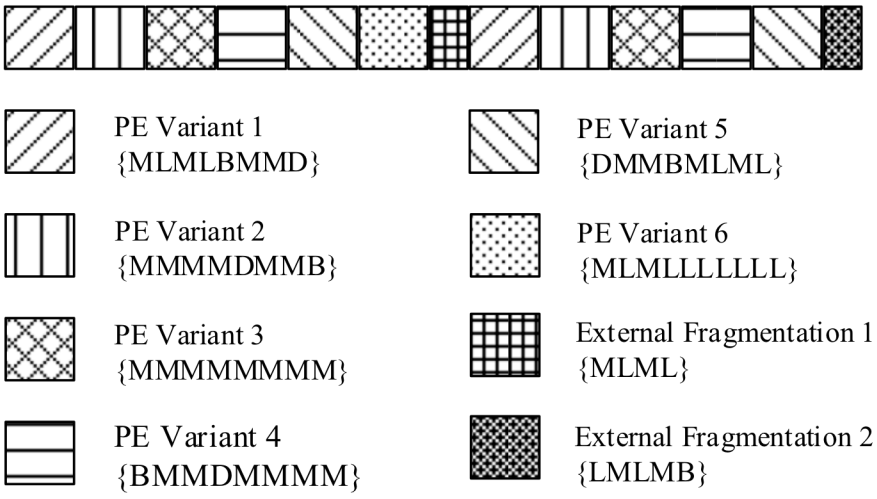

Figure 4.2: PE Variants with Footprint Masks

are built in a case study to examine the scalability of the ROB methodology, as shown in Figure 4.3.

After this step, all the tasks in the ROB methodology are automated by scripts.

Figure 4.3: CGRA Designs with 8 Different Sizes

4.1.4 Placing and Routing Initial PE Variants

Before placing the routing the initial PE variants, the ROB methodology reserves dedicated area for the *connection anchors* to produce a *PE tile*, according to the predefined floorplan. A PE tile is a bounding-box constrained region (of size PE Width x PE Height) that contains all of the PE logic and routing. One PE tile will be produced for each PE variant. The connection anchors are temporary appendages that will be discarded later on.

For each PE tile, a set of connection anchors is required on each of the four sides of the rectangular PE tile. A connection anchor is a pre-built hard macro that will connect the signals for communication between adjacent PE tiles, each one forming one half of the interface for zipping later. When PE tiles are abutted, it is important for interconnect

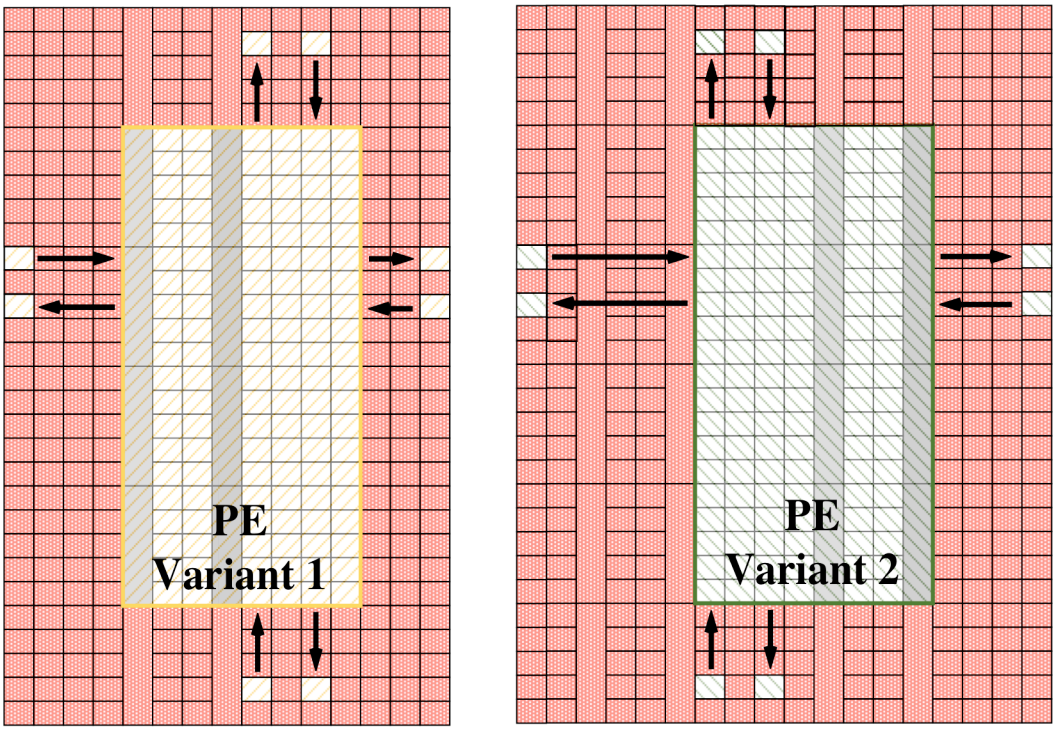

Figure 4.4: PE Tiles and Connection Anchors Allocation

between adjacent PE tiles to lie in a straight line. Otherwise, a wasteful "dogleg" shape connection would be required. To avoid "doglegs", the layouts of the connection anchors on the opposite sides of one PE tile must physically correspond to each other, as shown in Figure 4.4. One limitation of this process is that the connection anchors have to be allocated on the odd columns of the FPGA device, due to low-level implementation issue. More precisely, the routing resources on the odd columns are different than the resources on the even columns. This means the primitives placed on the odd columns cannot be relocated to the even columns.

In early experiments, it was found that ISE would sometimes utilize resources outside the PE tile, even though the PE tile was physically constrained by an area group constraint. To prevent this behavior, any resource that is not in the reserved area of the connection anchors and the PE tile is prohibited for placement by patching the user constraint file with "PROHIBIT" constraints.

Another issue found was that the routing is more congested at the boundary of the PE tile. This congestion manifests itself as longer route times in ISE and lower clock frequency. This is because logic resources at the border have access to fewer wires for routing than the logic resources that are located in the center of the PE. To address this

Prohibition Methodology	**Routing Time (seconds)**
No prohibited areas	332
Top row prohibited	132
Bottom row prohibited	121
Top and bottom row prohibited	91

Table 4.3: Prohibition Methodology for PE Variant #1

issue, the ROB methodology offers options to prohibit some logic resources at the PE tile boundary for placement. This results in a better ratio of routing wires per logic resource at the PE boundary at the cost of unused logic resources (i.e. higher internal fragmentation). However, there must still be sufficient logic remaining in the PE tile for the variant to fit after these constraints are applied. In the case study, four strategies for PE Variant #1 were investigated and the results are presented in Table 4.3. It was found that blocking the top and bottom rows inside the PE tile from placement gets a faster routing time in ISE. Based on this, the top and bottom rows were prohibited for all PE variant tiles in this thesis.

Once the physical constraints of the PE variants are set, every PE variant needs to be placed and routed using ISE. Since the compilation process is independent for every variant, this process can be trivially parallelized across workstations. In the case study, a sequential process of building the initial PE variants took 48 minutes to complete, while a parallelized build process took 18 minutes using a workstation with 4 CPU cores. Depending upon the precise overlay design and usage, it may be possible to precompute this initial PE build time so it is not observed by users. Resource utilization summary and the build time for each PE variant are shown in Table 4.4.

PE Variant Footprint Mask	Logic Slices	Multiplier Blocks	PE Variant Build Time (minutes)
{MLMLBMMD}	216	2	9
{MMMMDMMB}	211	2	7
{MMMMMMMM}	281	N/A	8
{BMMDMMMM}	208	2	7
{DMMBMLML}	216	2	8
{MLMLLLLLL}	290	N/A	9

Table 4.4: Resource Summary of PE Variants

A placed and routed PE variant is shown in Figure 4.5, where cyan wires represent the clock network and dark blue wires represent everything in the PE variant except for the clock signal. In the figure, four sets of connection anchors, which represent the IO ports for the data bus, are clearly shown on four sides of the PE tile. The connection anchors for the personalization bitstream are located at the bottom of the PE tile. These connection anchors will be discarded in the next step.

4.1.5 Extracting PE Tiles from Initial PE Variants

Once the PE variants are placed and routed, the NCD netlists of the PE variants are automatically converted to XDL netlists by scripts. With the XDL netlists, the PE

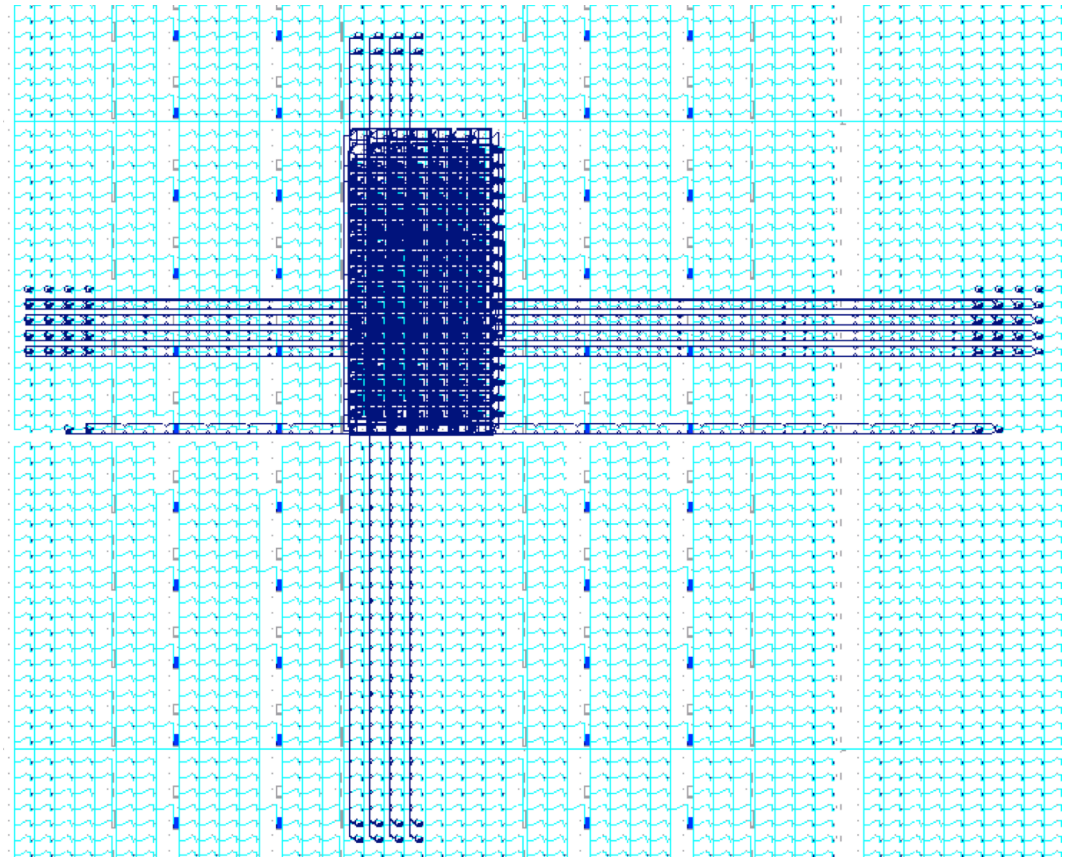

Figure 4.5: One Placed and Routed PE Variant

variants can be cut out along the boundary of the PE tile, leaving the data bus wires as floating antennas, as shown in Figure 4.6. The cut interconnect wires will be used for zipping together adjacent tiles. These XDL representations are stored in a pre-built PE tile library. The library of pre-built PE tiles will be used for assembling the final CGRA.

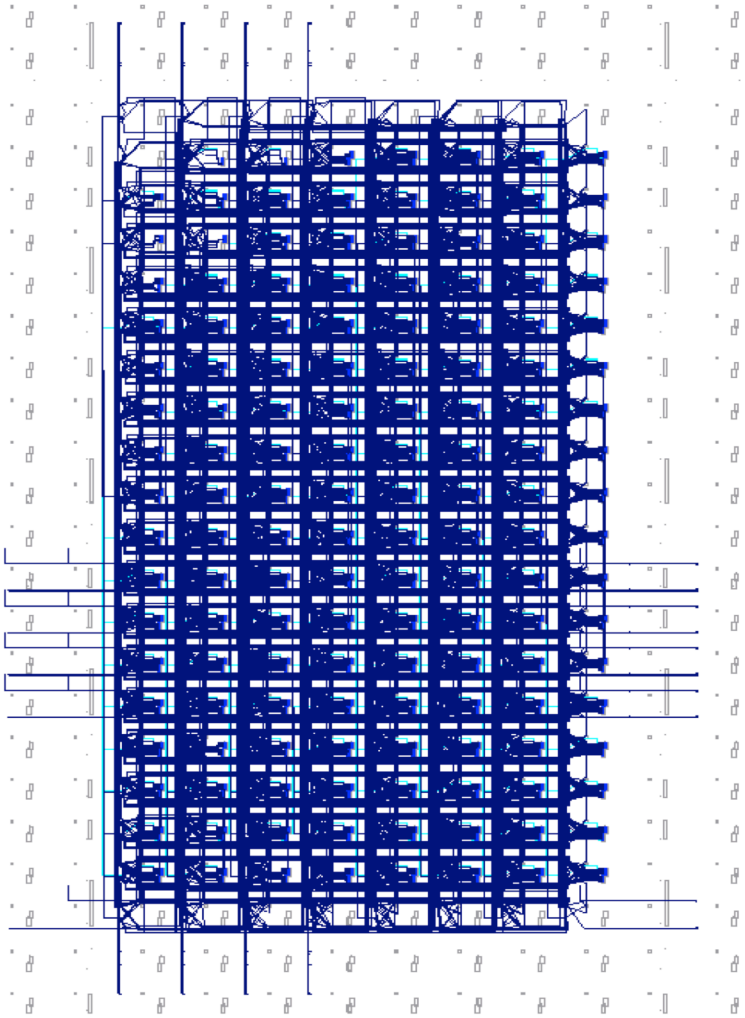

Figure 4.6: PE Tile Extracted from a PE Variant

4.1.6 Relocating PE Tiles on the Device

In the case study, every pre-built PE tile spans 20 rows of CLBs, which is half the height of a clock region. Five PE tiles have a footprint mask width of 8 and one PE tile, which contains only logic resources, has a footprint mask width of 10. The PE tiles from the library are relocated and instantiated according to a predefined floorplan.

```
# Instantiating PE tiles starting at INT_X9
Set Variable=module_top Value="220";
SetLabel LabelName=LoopHead_1;
AddBlockToSelection UpperLeftTile=INT_X9Y[%module_top%-1]
LowerRightTile=INT_X9Y[%module_top%-1];
Set Variable=module_top Value=[%module_top%-20];
GotoLabel LabelName=LoopHead_1 Condition=%module_top%>170;
AddInstantiationInSelectedTiles = PE_Variant_1;

# Instantiating PE tiles starting at INT_X17
Set Variable=module_top Value="220";
SetLabel LabelName=LoopHead_2;
AddBlockToSelection UpperLeftTile=INT_X17Y[%module_top%-1]
LowerRightTile=INT_X17Y[%module_top%-1];
Set Variable=module_top Value=[%module_top%-20];
GotoLabel LabelName=LoopHead_2 Condition=%module_top%>170;
AddInstantiationInSelectedTiles = PE_Variant_2;

# Instantiating PE tiles starting at INT_X25
Set Variable=module_top Value="220";
SetLabel LabelName=LoopHead_3;
AddBlockToSelection UpperLeftTile=INT_X25Y[%module_top%-1]
LowerRightTile=INT_X25Y[%module_top%-1];
Set Variable=module_top Value=[%module_top%-20];
GotoLabel LabelName=LoopHead_3 Condition=%module_top%>170;
AddInstantiationInSelectedTile = PE_Variant_3;
```

Figure 4.7: Script Employed for Relocating PE Tiles

The most essential part of the script that is employed in this process is shown in Figure 4.7. In the script, the designer is instantiating a total of 9 PE tiles by relocating 3 pre-built PE tiles into a set of specific locations. The resulted circuit is shown in Figure 4.8 and it can be observed that each PE column is created by instantiating the same pre-built PE tiles multiple times.

4.1.7 Interconnecting Adjacent PE Tiles

In this case study, adjacent PE tiles are placed next to each other. By design, the floating

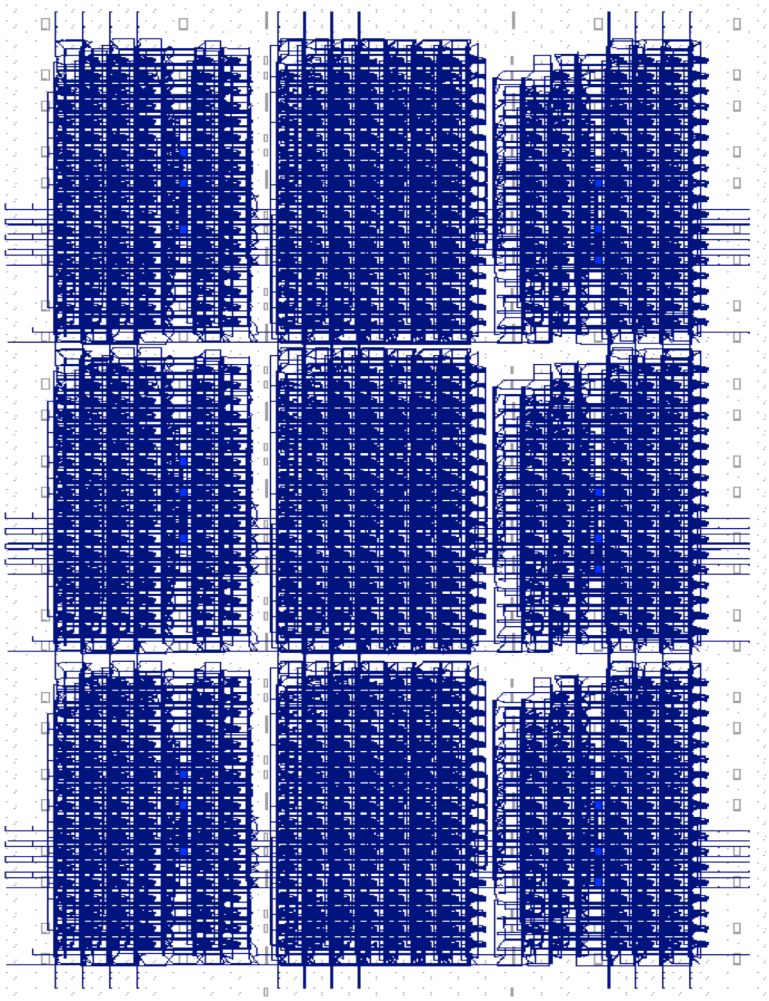

Figure 4.8: Module Relocation and Instantiation

interconnect located along the zipping boundary of each tile perfectly aligns with each adjacent tile, so no additional routing is needed. Not having to use the router saves time and also avoids the potential for allocating additional logic or routing resources to form connections between mismatched components. This also helps achieve a size-independent clock rate for the overlay.

4.1.8 Interconnecting the CGRA Design and the FPGA Driver

After all the PE tiles are stitched together, the CGRA and the FPGA Driver are also automatically stitched together using similar netlist manipulation. Figure 4.8 presents the fully placed and routed CGRA system. The FPGA Driver was floorplanned and built in such a way that all of the IO ports of the FPGA driver correspond with its adjacent PE tiles. Since the FPGA Driver is a common part of the design and can be fit in the compilation process as a hard macro partition, the corresponding compile time is not included in the standard ISE flow nor the ROB methodology. Therefore, the CAD time for building the FPGA Driver is excluded from all experimental results presented in Chapter 5.

In the present CGRA system, the FPGA Driver is the only interface enabled in the ROB methodology to control data flow inside the CGRA and to carry out the personalization process of propagating the application bitstream to the PEs.

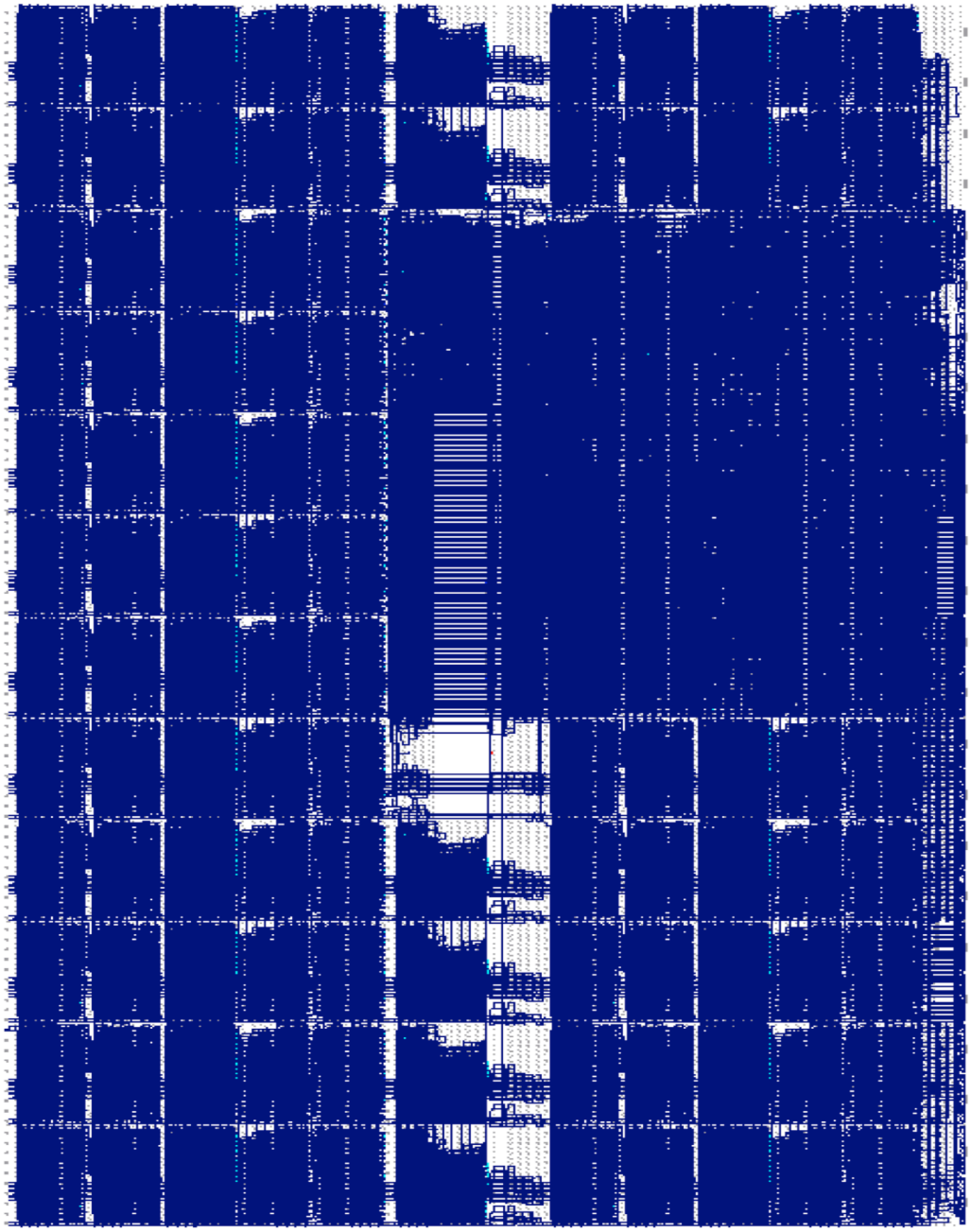

Figure 4.9: Physical Implementation of a 101-PE CGRA System

4.2 Application: CGRA Customization

In demonstrating the ROB methodology so far, a homogeneous array consisting of simple PE tiles was placed and routed. To demonstrate a more complex usage scenario, building a CGRA using PEs with both integer and floating-point capabilities is now considered. Such a general-purpose PE can dramatically simplify application development, but it comes with an unreasonably high resource requirement, as some PE functionality might commonly be left unused. Therefore, this section presents a case study below that customizes CGRA designs by applying specialization to the PEs using the ROB methodology.

4.2.1 PE Specialization

The overall architecture shares some concepts of the PACT-XPP CGRA architecture [35] with a customizable ALU. In the case study, the specialized PEs provide an integer ALU or single-precision floating-point instructions.

A reference "complex PE" that features all supported operations was first implemented. Because of the PE performance and the CAD-tool times improve when defining area constraints for PEs (see Chapter 4), bounding boxes (Xilinx area group constraints) were defined for the placement of the primitives. The complex PE implementation uses 812 logic slices and 6 DSP blocks, while the clock speed was 51.4 MHz. With the size of the complex PE, the chosen Virtex-6 device in the case study can accommodate at most 24 PEs in the CGRA.

Since this complex PE is very large and flexible, specializing the complex PE by ISA subsetting is considered to be effective in reducing the resource requirement for PEs. Specialized PEs, where each contains just one floating-point instruction, are placed and routed in the case study. The area results are shown in Table 4.5. The PE that provides the FMUL (floating-point multiply) operation takes most logic resources, requiring 312 logic slices and 3 DSP blocks.

Specialized PE	Logic Slices		DSP Blocks	
	PE Variant 1	PE Variant 2	PE Variant 1	PE Variant 2
FADD	260	270	0	0
FSUB	267	264	0	0
FDIV	279	261	0	0
FMUL	312	306	3	3
FCONV	249	246	2	2
ALU	152	136	3	3
Complex PE	812		6	

Table 4.5: Resource Utilization of Specialized PEs

For this case study, all PEs are constrained and built to have the same tile size, so that the PEs can be reusable in the same CGRA floorplan scheme. This may introduce internal fragmentation, but it simplifies external tools and limits external fragmentation. The bounding box for each tile provides 320 logic slices, 4 DSP blocks and 4 BRAM blocks. To exploit the module relocatability and to reduce the number of required variants, modules using Slice-M logic are mapped to columns of Slice-L logic. With the specialized PEs, the Virtex-6 device can accommodate 76 PEs with just two variants of each PE. This is three times more PEs than the homogeneous case with much-larger complex PEs. The footprint masks of the two variants are {M, L, M, L, B, M, M, D, M, M} and {M, M, D, M, M, B, M, L, M, L}.

Specializing PEs can not only benefit from a reduced PE size, but might also improve the overall clock rate of the CGRA. By reducing the size of the fully featured PE, the PE becomes less complex and the critical path delay is shortened. Since the slowest PE tile determines the overall clock rate of the CGRA system, it is best to optimize the timing performance of each specialized PE tile. The component-based design flow presented in the ROB methodology provides such an option for designers, whereas the conventional Xilinx ISE compilation flow can only optimize timing performance to the bulk CGRA system. With a vendor tool called SmartXplorer, designers can run the PAR process multiple times with different implementation strategies to achieve timing closure individually for each specialized PE. Such an exploration process can also be fully parallelized to reduce the overall runtime by utilizing multiple processor cores to run individual strategies simultaneously.

In addition to potentially improving the clock rate of the CGRA, the ROB methodology can also improve the predictability of the clock rate in the final physical implementation. This means the ROB methodology can implement size-independent CGRA designs with a consistent clock rate, as long as the same set of PE variants is used. Related experimental results are detailed in Chapter 5.

4.2.2 CGRA Customization

Once the specialized PE tiles are built, designers can instantiate the specialized PE tiles according to the application mapping results. This process is called *CGRA customization* in this thesis. Although the application mapping process is not in the scope of this thesis,

instantiating specilized PE tiles accordingly can ultimately be scripted and run automatically.

Conventionally, with a complex PE that is without any type of specialization, the CGRA system only needs to be built once. However, such an implementation has its limitations in computation capacity as well as the circuit performance as described previously. CGRA customization using specialized PEs overcomes both limitations and promises higher computation throughput. One reasonable cost of CGRA customization is that the application mapping process needs to be tailored that at most one floating-point operation can be assigned to a specialized PE. This is because all pre-built specialized PEs are capable of executing at most one floating-point operation.

The CGRA customization process needs to be done whenever a change is made to the application. With the conventional Xilinx ISE compilation flow, this means re-implementing the CGRA by running the long PAR process, which takes hours to finish. With the stitching mechanism employed in the ROB methodology, the time of the re-implementation process is reduced to minutes.

As compelling as the ROB methodology currently seems, it can be further developed to support partial reconfiguration. This enables designers to change PE tiles dynamically by downloading the partial bitstream while the rest of the CGRA system continues to operate without interruption. Such an advanced use-case is promising, but it is also outside of the scope of this thesis.

4.3 Summary

This chapter detailed the ROB methodology in seven tasks, including (1) resouce budgeting, (2) floorplanning, (3) initial PE building, (4) PE tile extracting, (5) PE tile instantiating, (6) interconnecting adjacent PE tiles and (7) interconnecting the CGRA with the FPGA Driver. The ROB methodology was first used to build a homogeneous CGRA with Simple PE. The CGRA customization process presented as an application of ROB, was also described in this chapter.

Chapter 5

Results

5.1 Introduction

This chapter presents the experimental results of building the homogeneous CGRA and the customized heterogeneous CGRA described in previous chapters. All the experiments use a Dell Workstation T5500, which features an Intel Xeon 3.33GHz quad-core processor and 8GB RAM. Xilinx ISE 14.7 was used under Windows 7 (64-bit). The targeted FPGA device was a Xilinx XC6VLX240T-FF1156 with a -1 speed grade on an ML605 development board.

5.2 Homogeneous CGRA Results

This section first presents the effort in generating HDL code and accelerating the compilation process of building homogeneous CGRA designs in the standard Xilinx ISE flow. A comparison of the elapsed CAD time, logic utilization levels and clock rates resulting from the standard Xilinx ISE flow and the ROB methodology in building CGRAs of different sizes will then be given.

5.2.1 Standard Xilinx ISE Flow

In this thesis, a number of experiments were conducted to compare the performance of the ROB methodology with the standard Xilinx ISE flow in building CGRAs. In all cases, the architectures used are the baseline homogeneous CGRAs that support integer-only instructions with the simple PE.

Before compiling a CGRA design using ISE, the HDL code representing the CGRA and the UCF representing the physical constraints of the CGRA need to be prepared. Handcrafting the HDL code takes a significant amount of time, since a large amount of signals for communication need to be properly instantiated and port mapped. Preparing the UCF requires designers to manually draw bounding boxes representing the PE tiles in PlanAhead. When conducting a new experiment with CGRAs of different sizes, a fair amount of changes still need to be applied to the HDL code and UCF. This is a tedious and error prone process. To improve productivity, several scripts are written to automatically generate the HDL file and the UCF for a rectangular CGRA of any chosen size.

In the case study, several floorplan methodologies were attempted in order to explore the best way of compiling CGRA designs using the standard Xilinx ISE flow. The floorplan methodologies can be categorized into two different kinds:

1) *k* PEs were grouped into a physical region (area group), where *k = 1 ... 8* PEs

2) None of the PEs are physically constrained

The code-generator script described earlier can only generate the UCF for the floorplan that every PE resides in its independent physical region. Grouping PEs into physical

regions cannot be easily accomplished using scripts, since the floorplan becomes irregular when PEs are grouped. Therefore, grouping PEs together is a manual process that modifies the UCF.

It is important to understand that the only common constraint in these floorplan methodologies is that the region reserved for the FPGA Driver is prohibited for placement.

In the experiments, it was found that the time consumption in the placement process dominated the total CAD time. Figure 5.1 shows the elapsed CAD time in building a 101-PE CGRA. It is shown in the figure that the CGRA can be placed and routed with minimum CAD time when the design is physically partitioned into regions of 4 PE modules.

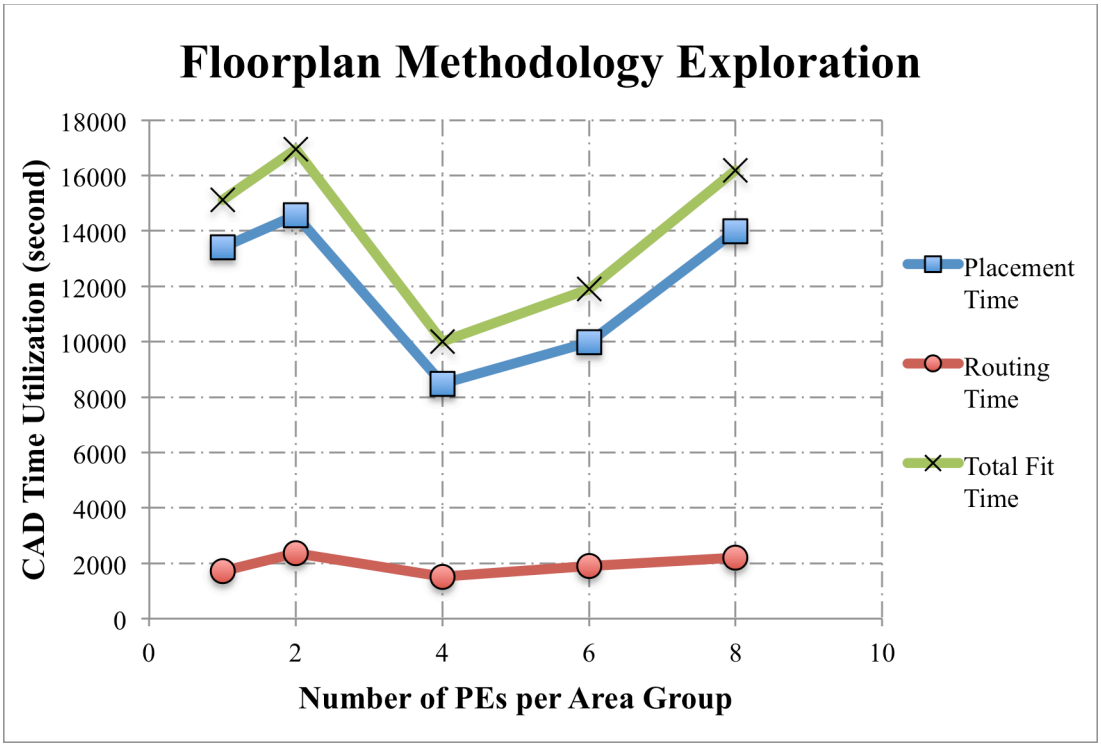

Figure 5.1: Exploration for Optimal Physical Partitioning

CGRAs without any physical constraints were much more difficult to build. Constraining the area where the FPGA Driver locates leaves the CGRA a "C" shape for implementation. The irregular "C" shape dramatically increases the difficulty for ISE to find the optimal placement solution. The poor placement result instantly creates a difficult routing problem, which leads to an extremely long routing time. For example, Figure 5.2 shows the routing progress of a CGRA that filled only 31% of the device (plus a prohibited region over the FPGA Driver area). After routing the CGRA for more than 5 days, the number of unrouted nets is approaching 115,000 nets so slowly that further progress appears unlikely. For remaining experiments that use ISE only, we always use a floorplan with physical regions that hold 4 PEs in each partition.

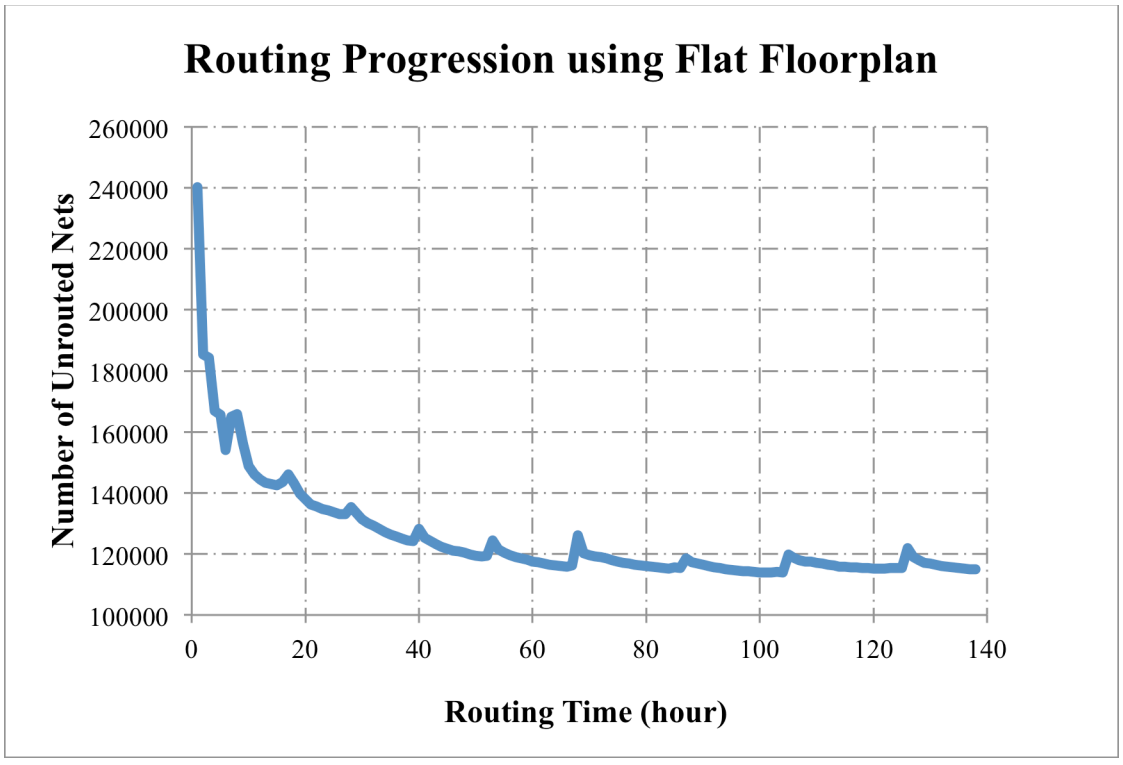

Figure 5.2: Routing Progression using Flat Floorplan

5.2.2 CAD Time Comparison

To explore the CAD tool scalability in circuit size, CGRAs with eight different sizes are built, using the floorplans previously shown in Figure 4.3. In addition, our evaluation considers two user scenarios: (1) user builds CGRAs from scratch; (2) user builds CGRAs with pre-built PE tiles.

The elapsed CAD time for building the homogeneous CGRA is shown in Tables 5.1. In Table 5.1, the elapsed CAD time for both the standard Xilinx ISE, using 4 PE modules per region, is compared to that obtained using our new ROB methodology. However, since the FPGA Driver is a common part of the design and can be fit in the compilation process as a hard macro partition, the corresponding compile time is not included in the standard ISE flow nor the ROB methodology.

The process of building initial simple PE tiles takes 18 minutes. The time for implementing initial PE tiles is included in the total CAD time of the ROB methodology. In user scenario (1), the ROB methodology obtains a speedup from 2x to 5x in implementing CGRA designs compared to the standard ISE tool flow.

In user scenario (2), the time to build the initial PE tiles is excluded. This corresponds to a usage case where they have been pre-built, or cases where the initial PE tile build time can be amortized over a sufficiently large number of different CGRA builds. In these cases, the speedup of the ROB methodology increases up to 22x.

In all cases, the time consumed in the initial PE building process dominates the total runtime in user scenario (1), whereas the time consumed in the XDL conversion process dominates the total runtime in user scenario (2).

CGRA Size	New ROB Methodology Time (seconds)				Standard Xilinx ISE CAD Time (seconds)				Speedup	
	Initial PE Building	Stitching	XDL Conversion	Total Time	Synthesis and Translation	Placement	Routing	Total Time	User Scenario 1	User Scenario 2
18 PEs	1080	40	69	1189	572	1578	247	2397	2.0x	22.0x
41 PEs	1080	56	210	1346	1210	1955	477	3642	2.7x	13.7x
49 PEs	1080	78	277	1435	1270	2432	703	4405	3.1x	12.4x
57 PEs	1080	81	377	1538	1197	3911	631	5739	3.7x	12.5x
65 PEs	1080	88	449	1617	1006	3939	638	5583	3.5x	10.4x
77 PEs	1080	106	695	1881	1137	7896	734	9767	5.2x	12.2x
89 PEs	1080	113	844	2037	1253	7614	831	9698	4.8x	10.1x
101 PEs	1080	125	1054	2259	1419	8483	1086	10988	4.9x	9.3x

Table 5.1: CAD Time Comparison – Simple PE

To evaluate the scalability of the ROB methodology, Table 5.2 shows the average time for building one PE on the device. The build time per PE can be further broken down to the average time for stitching a PE tile with its adjacent tiles as well as the average time to convert the XDL netlists of a PE to NCD format. In Table 5.2, it clearly shows that the ROB methodology scales well and the XDL conversion process scales poorly with the CGRA size. This results in the increasing build time per PE as the CGRA size scales up in user scenario (2).

CGRA Size	Stitching (seconds)	XDL Conversion (seconds)	Total Time (seconds)	
			User Scenario 1	User Scenario 2
18 PEs	2.2	3.8	66.1	6.1
41 PEs	1.4	5.1	32.8	6.5
49 PEs	1.6	5.7	29.3	7.2
57 PEs	1.4	6.6	27.0	8.0
65 PEs	1.4	6.9	24.9	8.3
77 PEs	1.4	9.0	24.4	10.4
89 PEs	1.3	9.5	22.9	10.8
101 PEs	1.2	10.4	22.4	11.7

Table 5.2: Build Time per PE

5.2.3 XDL Conversion Limitation

While exploring the scalability of the ROB methodology, it was found that the XDL conversion process dominates the time consumption in building CGRAs with pre-built PE tiles. To understand this bottleneck in the methodology, this subsection presents further analysis of the XDL conversion process.

It is important to understand that a fully placed-and-routed design output from the ROB methodology is an XDL netlist. In order to generate bitstream, the XDL netlist has to be converted back to an NCD netlist. This process is done entirely by Xilinx tools. Figure 5.3 shows the time consumption of the XDL conversion process and its proportion of the total time in building CGRAs. It shows that when scaling the CGRA size, the portion of the time consumed in the XDL conversion process increases from 63% to 89% for the simple-PE CGRAs. This indicates the poor scalability of the process in circuit size. It is unknown whether Xilinx can improve this runtime, but doing so would be highly advantageous for users of ROB.

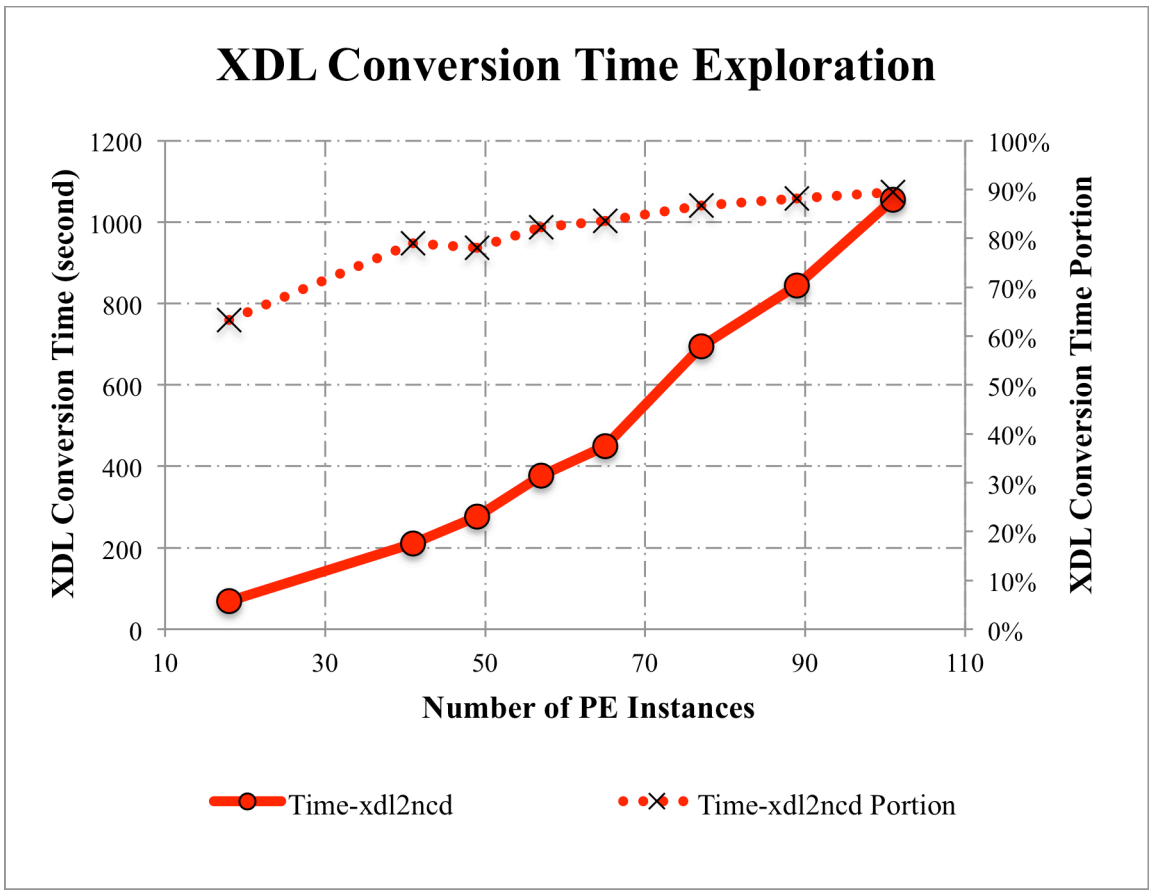

Figure 5.3: XDL Conversion Time Exploration

5.2.4 Utilization and Clock Rate Comparison

In addition to speeding up the process in building CGRAs, maintaining high logic utilization levels and clock rates is also a goal of the ROB methodology. Figure 5.4 illustrates that the logic utilization levels resulting from the ROB methodology are constantly lower than ISE in building the simple-PE CGRAs. This is because the initial PE tiles were implemented with their top and bottom rows of the resource in the bounding box prohibited for placement, as described in Chapter 4. The prohibition constraints forces ISE to apply a more compact slice packing that utilizes less logic primitives.

Figure 5.4 also shows that clock rates from the ROB methodology are consistent and always higher than ISE in building CGRAs. When increasing the size of CGRAs, the F_{max} from ISE fluctuates between 88.1 MHz and 115.8 MHz, while the F_{max} from the ROB

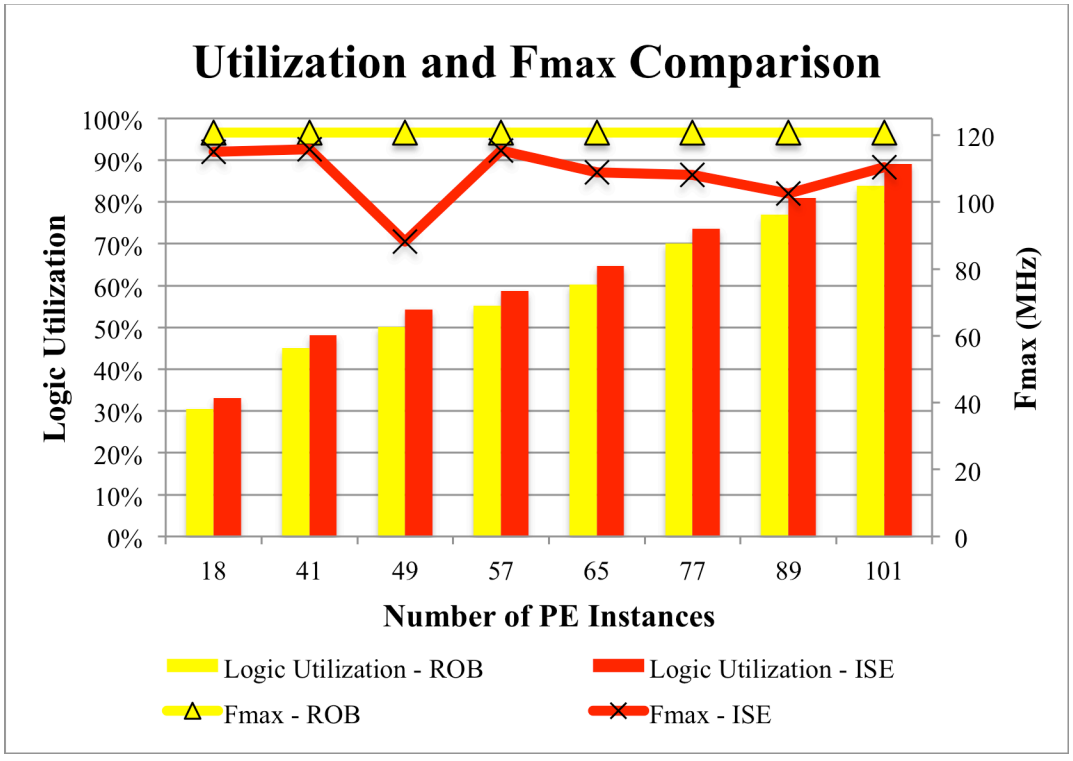

Figure 5.4: Utilization and Fmax Comparison – Simple PE

methodology stays at 120.7 MHz. This shows how the ROB methodology is able to provide consistent timing performance.

5.3 Heterogeneous CGRA Results

For our heterogeneous CGRA results using complex PEs, the resource breakdown, the achieved clock frequencies and the full tool run-time to build each PE variant are listed in Table 5.3.

The complex PE, listed in the bottom row of the table as a reference, requires a tile 20 columns x 30 rows. In contrast, each of the specialized PEs can fit into a tile just 10 columns x 20 rows in size. The specialized PEs span a wide range of functions, where each one implements either a single floating-point operation, or an integer ALU.

The different footprints for a complex PE and specialized PE are illustrated in Figure 5.5 (using the same scaling level). After considering external fragmentation, a homogeneous CGRA can only support up to 24 complex PEs. However, a customized CGRA using specialized PEs can accommodate up to 76 specialized PEs, which is an increase of 3.0x in capacity.

Specialized PE	Logic Slices		DSP Blocks		F_{max} (MHz)		t_{CAD} (seconds)	
	PE Variant 1	**PE Variant 2**	**PE Variant 1**	**PE Variant 2**	**PE Variant 1**	**PE Variant 2**	**PE Variant 1**	**PE Variant 2**
FADD	260	270	0	0	97.0	95.4	602	506
FSUB	267	264	0	0	103.4	96.3	594	497
FDIV	279	261	0	0	81.1	83.4	591	555
FMUL	312	306	3	3	76.1	79.5	641	625
FCONV	249	246	2	2	92.1	94.6	564	425
ALU	152	136	3	3	118.7	118.7	444	372
Complex PE	812	812	6	6	51.4	58.9	573	534

Table 5.3: Resource Breakdown, Achieved Clock Speed and Tool Run-times of Specialized PEs

Not only does the area efficiency improve significantly, there is also an increase in the maximum clock frequency. The slowest PE variant will determine the maximum clock frequency of the whole (heterogeneous) CGRA. In the CGRA with specialized PEs versus the complex PE, this is 76.1 MHz versus 51.4 MHz, an increase of nearly 50%. When combined with three times as many PE tiles, the net increase in peak performance is nearly 4.5x.

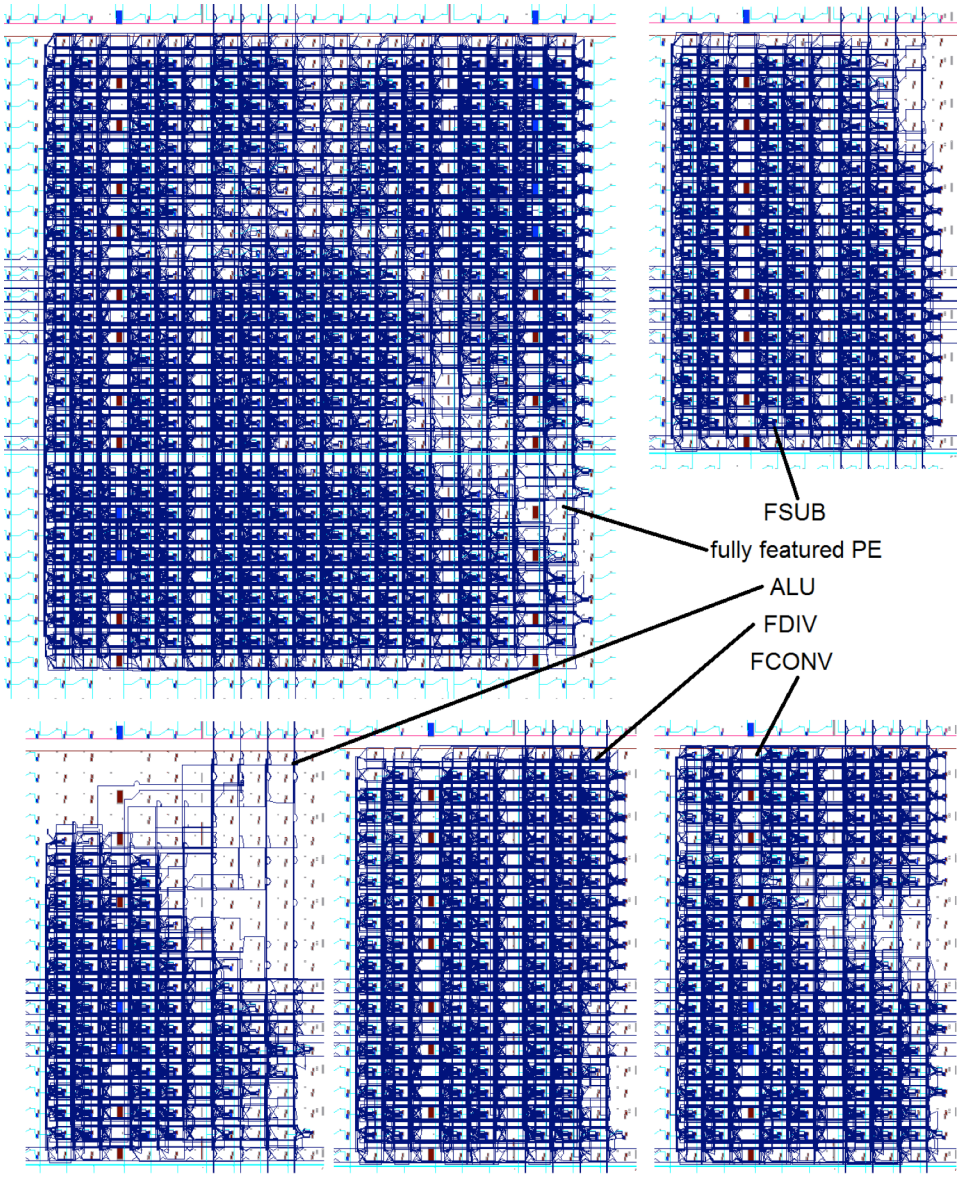

Figure 5.5: Fully Featured Complex PE and Specialized PEs

5.4 Summary

This chapter first presented and compared elapsed CAD times, resource utilization levels and clock rates resulted from the ROB methodology and the Xilinx ISE flow. The ROB methodology can obtain a speedup for up to 22x in building CGRA designs, compared to the standard ISE flow. It was identified that the CAD times resulted from the ROB methodology is scalable with the CGRA size. While ROB obtained considerable speedups in building CGRAs, the bottleneck of obtaining further speedups lies in the XDL conversion process. In addition, a speedup of 9.3x can still be obtained when logic utilizations reaches 89%. Lastly, the clock rates of the built CGRAs resulted from ROB are very consistent, size-independent and constantly higher than the ISE flow. As an application of the ROB methodology, we demonstrated a CGRA customization process that utilized specialized PEs to save resources and to improve timing performance of the CGRA. This confirms the efficiency of the ROB methodology in building CGRA designs.

Chapter 6

Future Work

This chapter presents the limitations of the thesis and the ideas of improving the ROB methodology that can be implemented in future work.

6.1 Tool Flow Automation

As described in Chapter 4, the ROB methodology can be divided into seven incremental tasks. Out of the seven tasks, the first two tasks, which were not made to be automatic, are (1) resource budgeting for a reference PE tile, and (2) floorplanning for CGRA designs. To automate these two tasks, it is important to first understand what the difficulties are in the automation process.

The goal of the resource budgeting process is to help designers to have a preliminary understanding of the resource requirement for a PE tile that can be used as a reference in during the floorplanning phase. However, several factors, including synthesis options, IP utilization options, physical constraints as well as timing requirement specified by users,

might affect the resource requirement of a PE. These factors create a huge exploration space for the users. It is important to understand that optimizing area of a PE tile may not be the only interest for the users in the use of the ROB methodology. Some users might also need options to optimize the timing and power performance of a PE tile. Therefore, experiments of careful and comprehensive sweep of these factors are required to find out what suits the users best. In future work, these experiments should be automated for the users to build the reference PE tile.

Once the reference PE tiles are built and chosen, floorplanning for CGRA designs is the next step to be automated. To effectively reduce the design space, it is important to understand that floorplanning the top PE row is decisive to the entire floorplan. This is because the heights and widths of PE tiles within the same PE column are the same. This simplifies the floorplanning problem from 2-dimension to 1-dimension. Furthermore, as the left and right sides of the device have similar footprint masks, the design space is further reduced by half. The next step in the floorplanning process is to determine the height of the PE tiles. With a different choice of heights, the widths of the PE tiles will also be different so as to provide sufficient resources within the tile. This process can be automated to calculate the corresponding external fragmentation with different choices of heights after instantiating a maximum number of bounding boxes, where each bounding box provides sufficient resources for one PE tile. If no additional requirement of the PE tiles is specified from the user, the floorplan candidate with the minimum external fragmentation will then be chosen.

6.2 Netlist Conversion Limitation

In the ROB methodology, the output is XDL format netlists. Since Xilinx tool only accepts NCD netlists as input, the XDL netlists has to be converted to the NCD netlists first in order to generate bitstream using the vendor tool. However, the netlist conversion process dominates the runtime, and is the bottleneck of achieving further speedups in the flow.

One idea to accelerate the process is to abandon the original flow of converting XDL netlists back to NCD netlists for bitstream generation. Instead, a methodology may be developed to generate bitstream directly from the XDL netlists. However, the practice of this idea might not be feasible because of the proprietary nature of the bitstream generation process.

Another idea is to exploit parallelism to accelerate the netlist conversion process. Although this process is entirely done by the Xilinx tool, it may still be possible to solve the problem in a divide-and-conquer fashion. To do that, the XDL netlists need to be divided into multiple smaller parts first. The smaller XDL files are then converted to NCD files in parallel. Next, the smaller NCD files need to be merged together. In practice, it is already known that, XDL netlists of a circuit can be physically divided up into parts and each part of these XDL netlists can manage to be converted to NCD netlists in our experiments. The only uncertainty of this approach is the possibility of merging smaller NCD files into one complete NCD file. This problem can be further investigated in future work.

6.3 Partial Reconfiguration Capability

To ultimately avoid the netlist conversion process, the ROB methodology can be further developed to support partial reconfiguration, which is a capability of downloading partial bitfiles for one module from the host platform while the rest of the system operates without interruption.

Although the ROB methodology can zip pre-built PEs very fast, the netlist conversion process is inevitable and is the major obstacle that limits the speedup of the methodology. With the capability of partial reconfiguration, users will no longer concern about the CAD time of building CGRA designs. Instead of zipping PE tiles statically, the bitstreams of the PE tiles are generated and stored in the host platform. According to the demand of the user, the bitstreams of the PE tiles are invoked, downloaded and reconfigured in the FPGA fabric. The FPGA Driver instantiated in the CGRA system not only feeds application data and the personalization bitstream into the CGRA, but also communicates with the host platform rapidly, which enables rapid partial reconfiguration. This capability can be further developed in future work.

6.4 Bitstream Verification

Although bitstream can be generated from the fully placed and routed CGRA system, the CGRA system has not been verified on an actual device whether it functions correctly. In future work, the output design from the ROB methodology should be treated with some level of skepticism and the bitstream need to be verified on an actual device.

Chapter 7

Conclusions

This thesis presents the Rapid Overlay Builder (ROB) that efficiently builds CGRA designs on Xilinx FPGAs. Unlike the traditional place-and-route process, the ROB methodology accelerates the process of building CGRA designs by utilizing and relocating pre-built PE tiles according to predefined floorplans. To improve the relocatability of PE tiles and logic utilization levels, PE variants that provide the same functionality but are built on different resource footprints are introduced. To further accelerate the building process, a routerless stitching mechanism that we call zipping is employed such that the interconnections between adjacent PE tiles are established without any logic overhead and without any additional routing step.

In this thesis, two CGRA architectures are employed to demonstrate the use of the ROB methodology. The first one is a homogenous array of simple PEs that support integer operations only. The second one is a heterogeneous array of specialized complex PEs that support both integer and floating-point capabilities. This thesis first demonstrates the ROB methodology as a case study to build the homogeneous CGRA. The case study details major steps required in the building process. These steps include (1) resource budgeting

for PE tiles, (2) floorplanning for the CGRA design, (3) initial PE variants building, (4) PE tiles extracting, (5) instantiating relocatable PE tiles on the device, (6) interconnecting between adjacent PE tiles, and (7) interconnecting the FPGA Driver with the CGRA design. The heterogeneous CGRA is demonstrated as an application of the ROB methodology, which utilized specialized PEs for customizing CGRA designs. The CGRA customization process reveals the use of the ROB methodology in saving resources and maximizing clock speeds, which in return confirms the efficiency of building CGRA designs using the ROB methodology.

In experiments of the ROB methodology, it is found that the ROB can obtain a speedup for up to 22x in building CGRA designs, compared to the standard Xilinx ISE flow. It is also identified that the CAD times resulted from the ROB methodology is scalable with circuit size, even when the logic utilization level reaches 89%. The bottleneck of obtaining further speedups lies in the netlist conversion process, which is entirely done by Xilinx tools. The clock rates of the CGRAs resulted from ROB are very consistent, size-independent and constantly higher than the ISE flow. In the best-case scenario, the clock rate resulted from ROB can be 1.37x higher than that can be achieved by the ISE flow. In experiments of customizing CGRA designs, the ROB methodology utilizes specialized PEs that triples the number of general-purpose PEs that would be instantiated on the targeted device. In addition, the clock rates of the specialized PEs are improved by 50% compared to the general-purpose PE. With the efficient builds of CGRA designs, the ROB methodology promises to yield higher computation throughput.

Limitations of the ROB methodology are revealed in Chapter 6. Future work can be done to fully automate the ROB methodology, to accelerate the netlist conversion process,

and to enable the partial reconfiguration capability so that the CGRA system can evolve during runtime according to the needs of users.

By applying the ROB methodology presented in this thesis, we anticipate that overlays can be implemented more quickly and with lower area overhead and more consistent clock rates.

Bibliography

[1] Altera Corporation, “Quartus II 14.0 Handbook,” http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf, 2014

[2] Xilinx Corporation, “Command Line Tools User Guide, ” http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/isehelp_start.htm, 2013

[3] J.B. Goeders, G.G.F. Lemieux, S.J.E. Wilton, "Deterministic Timing-Driven Parallel Placement by Simulated Annealing Using Half-Box Window Decomposition," Reconfigurable Computing and FPGAs (ReConFig), Nov. 2011

[4] M. An, J.G. Steffan, V. Betz, "Speeding Up FPGA Placement: Parallel Algorithms and Methods," Field-Programmable Custom Computing Machines (FCCM), May 2014

[5] J.B. Goeders, G.G.F. Lemieux, S.J.E. Wilton, "Deterministic Timing-Driven Parallel Placement by Simulated Annealing Using Half-Box Window Decomposition," Reconfigurable Computing and FPGAs (ReConFig), Nov. 2011

[6] M. An, J.G. Steffan, V. Betz, "Speeding Up FPGA Placement: Parallel Algorithms and Methods," Field-Programmable Custom Computing Machines (FCCM), May 2014

[7] M. Gort, J.H. Anderson, "Deterministic multi-core parallel routing for FPGAs," Field-Programmable Technology (FPT), Dec. 2010

[8] Y.O.M. Moctar, P. Brisk, "Parallel FPGA routing based on the operator formulation," Design Automation Conference (DAC), June 2014

[9] C.E. LaForest and J.G. Steffan, "Maximizing Speed and Density of Tiled FPGA Overlays via Partitioning," Field-Programmable Technology (FPT), Dec. 2013

[10] C. Mulpuri and S. Hauck, “Runtime and quality tradeoffs in FPGA placement and routing,” Field Programmable Gate Arrays (FPGA), 2001

[11] D. Capalija, T. Abdelrahman, “Tile-based Bottom-up Compilation of Mesh-of-FUs FPGA Overlays,” Field-Programmable Logic and Applications (FPL), Sept. 2014

[12] R. Tessier, "Fast Placement Approaches for FPGAs," ACM Trans. Des. Autom. Electron. Syst., vol. 7, no. 2, pp. 284-305, 2002.

[13] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "HMFlow: Accelerating FPGA Compilation with Hard Macros for Rapid Prototyping," Field-Programmable Custom Computing Machines (FCCM), May 2011

[14] C. Beckhoff, D. Koch, and J. Torresen, "GoAhead: A Partial Reconfiguration Framework", Field-Programmable Custom Computing Machines (FCCM), Apr. 2012

[15] S. Korf, D. Cozzi, M. Koester, J. Hagemeyer, M. Porrmann, U. Ruckert, and M.D. Santambrogio, "Automatic HDL-Based Generation of Homogeneous Hard Macros for FPGAs," Field-Programmable Custom Computing Machines (FCCM), May 2011

[16] A. Brant, "Coarse and Fine Grain Programmable Overlay Architectures for FPGAs," Master's thesis, Dept of ECE, University of BC, Nov. 2012

[17] A. Brant and G.G.F. Lemieux, "ZUMA: An Open FPGA Overlay Architecture," Field-Programmable Custom Computing Machines (FCCM), Apr. 2012

[18] H.Y. Cheah, S.A. Fahmy, and D.L. Maskell, "iDEA: A DSP Block Based FPGA Soft Processor," Field-Programmable Technology (FPT), Dec. 2012

[19] A. Severance, and G.G.F. Lemieux, "Embedded supercomputing in FPGAs with the VectorBlox MXP Matrix Processor," Hardware/Software Codesign and System Synthesis (CODES+ISSS), Sept. 2013

[20] C.E. LaForest, and J.G. Steffan, "Octavo: an FPGA-centric Processor Family," Field Programmable Gate Arrays (FPGA), Feb. 2012

[21] N. Kapre, and A. DeHon, "VLIW-SCORE: Beyond C for Sequential Control of Spice FPGA Acceleration," Field-Programmable Technology (FPT), Dec. 2011

[22] C. Liu, C.L. Yu, and H.K.H. So, "A Soft Coarse-Grained Reconfigurable Array Based High-level Synthesis Methodology: Promoting Design Productivity and Exploring Extreme FPGA Frequency," Field-Programmable Custom Computing Machines (FCCM), p. 228, Apr. 2013

[23] D. Capalija and T.S. Abdelrahman, "A High-performance Overlay Architecture for Pipelined Execution of Data Flow Graphs," Field-Programmable Logic and Applications (FPL), pp. 1–8, Sept. 2013

[24] B. De Sutter, P. Raghavan, and A. Lambrechts, "Coarse-Grained Reconfigurable Array Architectures," In Handbook of Signal Processing Systems, pp. 553-592, 2013.

[25] C. Lavin, B. Nelson, B. Hutchings, "Improving clock-rate of hard-macro designs," Field-Programmable Technology (FPT), Dec. 2013

[26] E.L. Horta, J.W. Lockwood, D.E. Taylor, and D. Parlour, "Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration," Design Automation Conference (DAC), pp. 343-348, 2002

[27] H. Kalte, G. Lee, M. Porrmann, and U. Ruckert, "REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems," Parallel and Distributed Processing Symposium, Apr. 2005

[28] H. Kalte and M. Porrmann, "REPLICA2Pro: Task Relocation by Bitstream Manipulation in Virtex-II/Pro FPGAs," ACM Computing Frontiers (CF), pp, 403-412, May 2006.

[29] T. Becker, W. Luk, and P. Y.K. Cheung, "Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration," Field-Programmable Custom Computing Machines (FCCM), Apr. 2007

[30] A. Wold, D. Koch, and J. Torresen, "Component-based design using constraint programming for module placement on FPGAs," Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), July 2013

[31] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford. "Invited Paper: Enhanced Architecture, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," Field Programmable Logic and Application (FPL), Aug. 2006

[32] D. Koch, C. Beckhoff , J. Torresen, "Zero Logic Overhead Integration of Partially Reconfigurable Modules, " in SBCCI Symposium on Integrated Circuits and Systems Design, pp. 103-108, 2010.

[33] K. Vipin, S. Shreejith, D. Gunasekera, S.A. Fahmy, N. Kapre. "System-level FPGA Device Driver with High-level Synthesis Support," Field-Programmable Technology (FPT), Dec. 2013

[34] P. Yiannacouras, J.G. Steffan, and J. Rose, (2007). "Exploration and Customization of FPGA-based Soft Processors," Computer-Aided Design of Integrated Circuits and Systems, pp. 266-277, 2007

[35] V. Baumgarte et al., "PACT XPP - A Self-Reconfigurable Data Processing Architecture," The Journal of Supercomputing, vol. 26, no. 2, pp. 167– 184, Sept. 2003