Incorporating Physical Information into Clustering for FPGAs

by

Doris Tzu Lang Chen

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The traditional approach to FPGA clustering and CLB-level placement has been shown to yield significantly worse overall placement quality than approaches which allow BLEs to move during placement. In practice, however, modern FPGA architectures require computationally-expensive Design Rule Checks (DRC) which render BLE-level placement impractical.

This thesis research addresses this problem by proposing a novel clustering framework that produces *better initial clusters* that help to reduce the dependence on BLE-level placement. The work described in this dissertation includes: (1) a comparison of various clustering algorithms used for FPGAs, (2) the introduction of a novel hybridized clustering framework for timing-driven FPGA clustering, (3) the addition of physical information to make better clusters, (4) a comparison of the implemented approaches to known clustering tools, and (5) the implementation and evaluation of cluster improvement heuristics. The proposed techniques are quantified across accepted benchmarks and show that the implemented `DPack` produces results with 16% less wire length, 19% smaller minimum channel widths, and 8% less critical delay, on average, than known academic tools. The hybridized approach, `HDPack`, is found to achieve 21% less wire length, 24% smaller minimum channel widths, and 6% less critical delay, on average.

Acknowledgements

I would like to thank Andrew Kennings for the help, advice, and guidance he has given me over the course of my degree. Much thanks should be directed to Kristofer Vorwerk, who has been extremely helpful and generous with his time in helping me out with any difficulties I encountered during the research process. Without the help of both of you, this thesis would not have come into existence.

Table of Contents

List of Tables

List of Illustrations

Chapter 1

Introduction

The Field Programmable Gate Array (FPGA) has become very popular in the last 25 years, and can be found in a variety of applications. However, the performance of the FPGA is highly dependent on the quality of the Computer-Aided Design (CAD) tool used. As the FPGA becomes more and more powerful due to advances in process technology and architecture research, better tools are needed to take full advantage of its capabilities. Therefore, it is of utmost importance to improve the quality of the design tools used.

1.1 Overview of FPGAs

There are two primary platforms that hardware designs can be implemented upon: ASICs, and FPGAs. The Application-Specific Integrated Circuit (ASIC) is a specially designed, custom manufactured chip. In comparison, the Field Programmable Gate Array (FPGA) has a regular structure, with a standard set of elements that can be programmed to function as any digital circuit. There are several advantages of using FPGAs over ASICs. First and foremost, the FPGA is programmable, whereas the ASIC is not. The programmability of the FPGA allows easy modification of its programmed application. In contrast, the ASIC cannot be modified once manufactured. If a different function is required of the chip, then a new ASIC must be made. This can pose as a significant problem during the developmental stage of hardware designs. If

a bug is found in the design, new chips must be remanufactured with the old ones discarded. However, if the hardware design was based on FPGAs, then the design can be altered easily by simply reconfiguring the FPGA. For this reason, FPGAs are very popular for prototyping designs. FPGAs also have an advantage over traditional ASICs in terms of time-to-market, as the chip manufacturing process can take months, whereas FPGAs are available off the shelf. However, even though there are many advantages to adopting the FPGA for development, FPGAs are not without drawbacks. Because of their programmability, FPGAs are usually much larger in area than an equivalent ASIC, leading to higher silicon costs and power consumption. They also tend to be slower than their ASIC counterpart. Therefore, for applications that require high performance and have stringent power requirements, such as cell phone applications, ASICs are still the preferred choice.

A popular FPGA architecture that is manufactured today is the island-style cluster-based FPGA. An example of this type of architecture is shown in Figure 1.1. The key characteristic of this architecture is the organization of logic blocks and wires. In this architecture, groups of logic, called Configurable Logic Blocks (CLBs) are arranged in a grid-like pattern, separated by routing channels. These channels contain many parallel segments of wires that can be programmed to form connections between CLBs. A more detailed view of the FPGA architecture is shown in Figure 1.2, where the building blocks of the FPGA is labelled. FPGAs interact with off-chip devices through the use of Input/Output (I/O) blocks, located along the periphery of the chip. The square blocks in the interior of the chip are CLBs with routing channels separating them. At the intersection of horizontal and vertical channels, routing switch blocks, such as the one shown in Figure 1.3, control which horizontal and vertical wires are connected. In Figure 1.3, 17 wires can be seen in every horizontal and vertical channel. The routing switch block performs the actual connection of wires, thus allowing horizontal wires to be connected to vertical wires as necessary. The switch shown allows a wire to be connected to one specific wire in every channel to which it is adjacent.

It should be noted that the architecture shown in Figure 1.1 is a very simplified layout of a

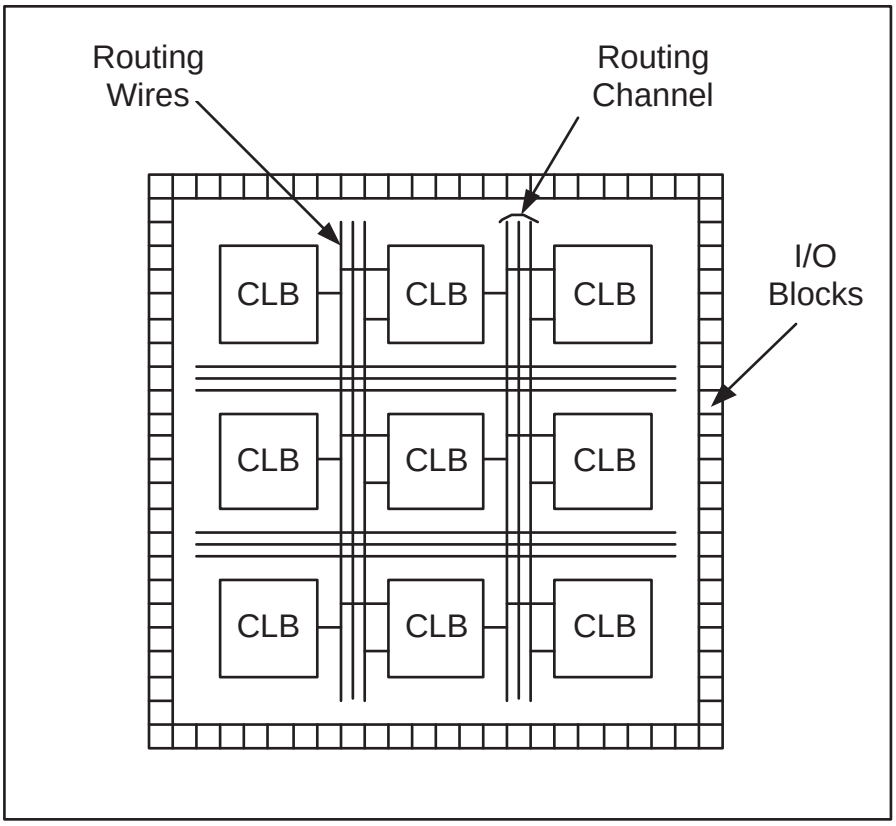

Figure 1.1: Sample Island-Style FPGA Architecture

FPGA. Modern FPGAs, such as the Cyclone II [1] and Stratix III [2] families manufactured by Altera, have many other types of blocks other than logic on the FPGA. These additional hard components further extend the capabilities of the FPGA, by incorporating memory components such as Random Access Memory (RAM) blocks, multiplier blocks such as Digital Signal Processing (DSP) units, and Phase-Locked Loops (PLLs) in the FPGA fabric.

The basic building block of logic in a FPGA is the Basic Logic Element (BLE). A simplified architecture for a BLE is shown in Figure 1.4. The BLE is made up of a Look-Up Table (LUT) for combinational logic, and a register, also referred to as a flip-flop, to store state. A *k*-LUT is essentially a memory component with *k* input pins, and one output pin. Depending on the combination of the input pin values, the row in the *k*-LUT will be addressed and the output set accordingly. There are a few different configurations that the BLE can take. As seen from

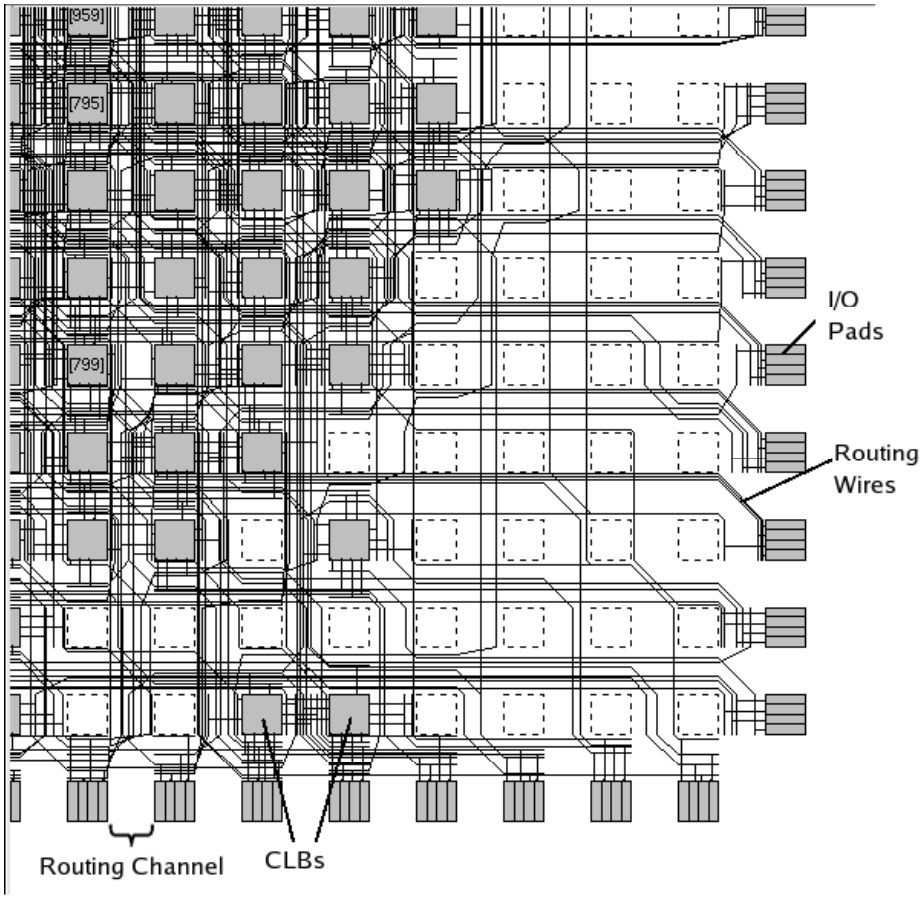

Figure 1.2: Detailed Placed-and-Routed FPGA Design

Figure 1.4, the output of the BLE may either be the output of the LUT, or the output of the register. Again, this diagram is very simplistic, and does not show several control signals (e.g., sets/presets), or other enhancements such as high-speed arithmetic logic (e.g., carry-chains).

The second hierarchical logic structure of the FPGA is the Configurable Logic Block (CLB). A CLB is a collection of BLEs. Several salient features of a CLB are shown in Figure 1.5. It can be seen that the outputs of the BLEs contained in the CLB can be connected to the input of all BLEs within the CLB. These are also referred to as local, or intra-cluster, connections of a CLB. It should be noted that the number of input pins of the CLB is typically less than the sum of the number of input pins of the contained BLEs. Therefore, if the BLEs within the CLB are to be fully utilized, some of the BLEs will need to share inputs. Also, if one of the inputs of a BLE is driven by another

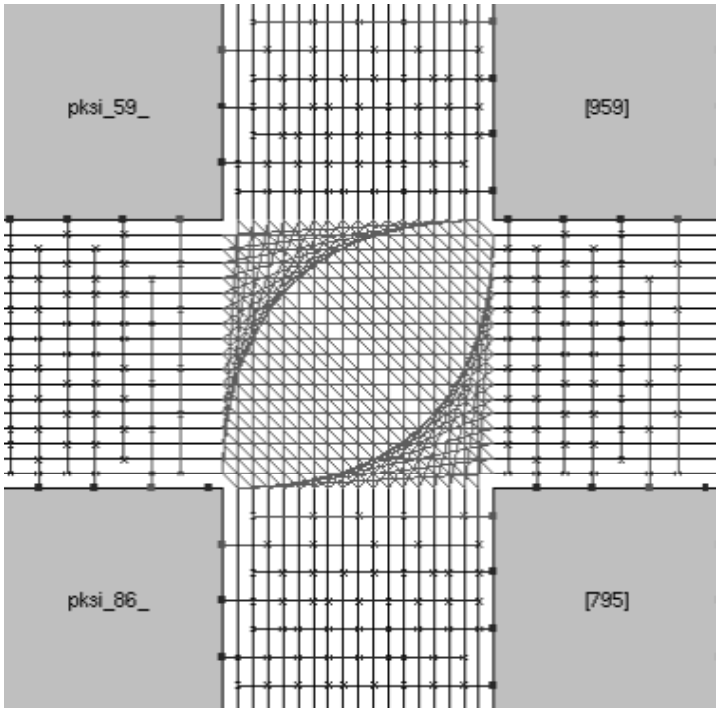

Figure 1.3: FPGA Routing Switch Block

BLE located within the CLB, the output signal can directly feed this BLE without having to be routed outside the CLB. To take advantage of this unique characteristic of the CLB, it is preferable to group together BLEs that have many interconnections. In this FPGA architecture, the wires within the CLB are much shorter than wires between CLBs. Therefore, the delay of intra-cluster connections are much less than inter-cluster connections since it is unnecessary to use routing resources. In Figure 1.5, the number of BLEs contained in the CLB is 3. Although the inputs are depicted to come from the left side of the CLB in this figure, in practice, inputs are generally distributed along the top, left, and bottom sides of the CLB, with the outputs leaving the right side.

There are many parameters and constraints present in the FPGA architecture. Many of these parameters cannot be controlled by the user. For example, parameters such as the number of BLEs per CLB, the number of inputs per CLB, and the number of wires in the routing channel are predetermined by the manufacturer, and consistent throughout the chip. Therefore, it is up to the user of the FPGA to take advantage of the architecture to achieve the best possible performance.

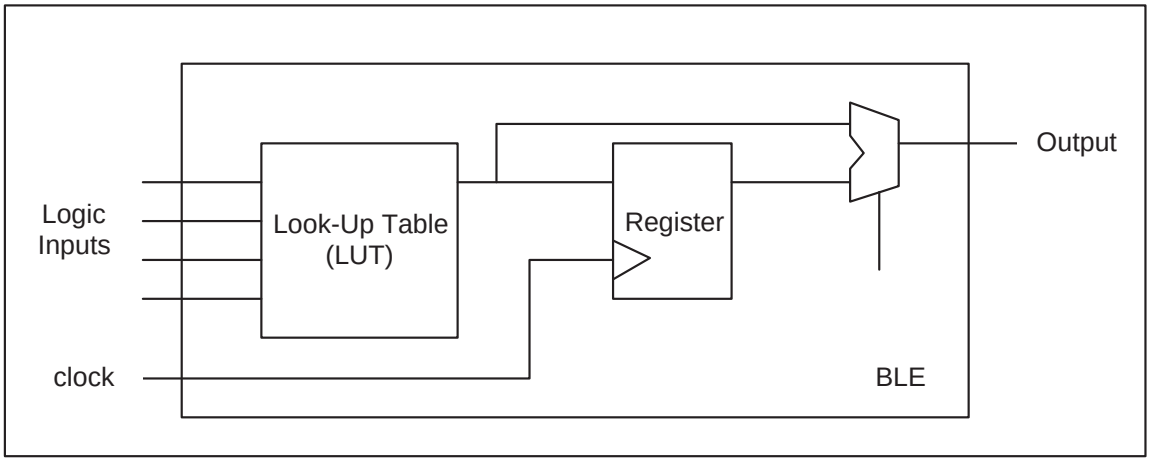

Figure 1.4: A Basic Logic Element (BLE)

However, it is unreasonable to expect all users of the FPGA to fully understand the complete inner workings of the FPGA. This would create a very steep learning curve, and discourage designers from using the FPGA as their primary method of development. Fortunately, tools have been developed to make FPGAs much easier to use, and to help the user maximize the performance of their designs on the FPGA.

1.2 The FPGA CAD Flow

The purpose of the Computer-Aided Design (CAD) flow is to bridge the gap between the hardware designer and the hardware implementation of their design on the FPGA. The CAD tool takes the circuit design, written in Hardware Description Languages (HDL) such as VHDL and Verilog or as schematics, as input. It then executes a number of steps to output a format that can be used directly to configure the circuit onto the FPGA. The overall FPGA CAD flow is shown in Figure 1.6. The main steps of the FPGA CAD flow include *Logic Synthesis*, *Technology Mapping*, *Clustering*, *Placement*, and *Routing*.

The first step is *Logic Synthesis* which, in itself, consists of high-level synthesis and technology-independent logic optimization. High-level synthesis works to convert the HDL of a design into Register Transfer Level (RTL) logic (i.e., registers, logic equations and macro blocks such as

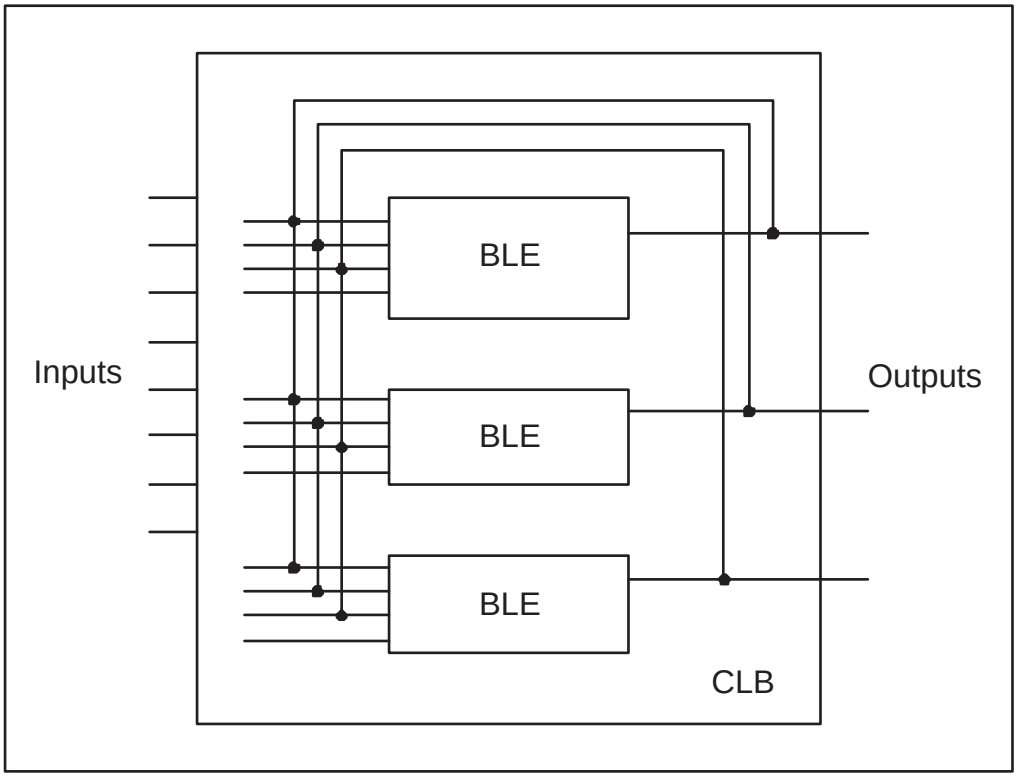

Figure 1.5: A Configurable Logic Block (CLB)

RAM, DSP and arithmetric). Technology independent logic optimization subsequently performs additional optimizations such as the removal of redundant logic, register retiming, and so forth. The most well-known academic synthesis tools include `SIS` [3], `MVSIS` [4] and `ABC` [5]. Then, during *Technology Mapping*, the design is converted into a set of primitive blocks that exist in the FPGA, connected by nets. For the cluster-based FPGA, these primitive blocks are LUTs and registers. Some popular technology mappers include `Chortle` [6], `FlowMap` [7], `CutMap` [8], `DART` [9], `FAST` [10], `IMap` [11] and `DAOmap` [12]. Then, in the *Clustering* stage, these primitive blocks are grouped into larger blocks that exist on the FPGA. In the case of the island-style FPGA described previously, the LUTs and registers are first grouped into BLEs in an intermediate step called register packing. Then, from the resulting set of BLEs, a set of CLBs are made using various clustering algorithms. `VPack`[13], `T-VPack` [13], `iRAC` [14] and `RPack` [15] are examples of academic clustering tools.

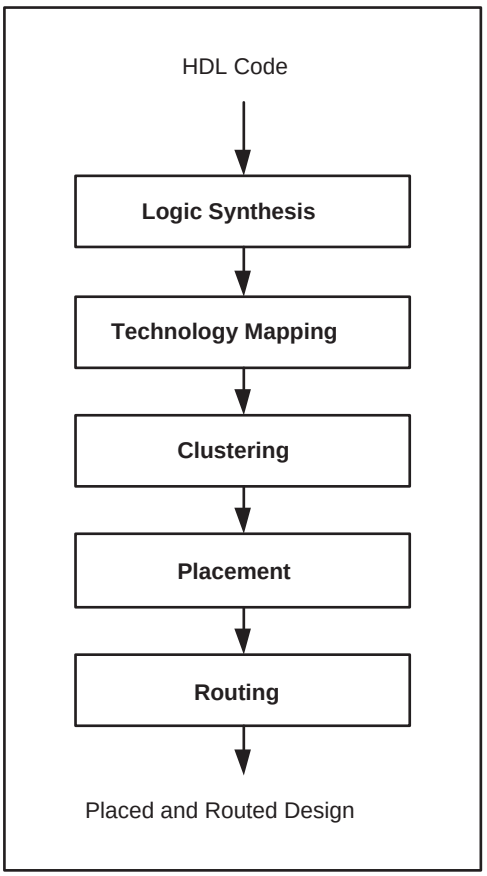

Figure 1.6: FPGA CAD Flow

At this point, a clustered netlist consisting of CLBs and nets that connect the CLBs together is generated. This netlist is then fed into the *Placement* step, where CLBs are moved around on a grid representing the FPGA chip to determine the best location for each CLB in the clustered netlist. Academic placers vary widely in the algorithm used, ranging from algorithms such as simulated annealing in VPR [13, 16], to partitioning algorithms [17–20]. After the placement step has been completed, every CLB is assigned to an x and y coordinate representing its final placement location. This is also referred to as the physical location of a CLB.

Finally, after the physical locations of all CLBs are found and set, the nets that connect CLBs are assigned to specific wires in the routing channels during the *Routing* stage. Routing can be split up into two stages: global, and detailed routing. During global routing, the channel is selected for every net, but the specific wire in the channel is not chosen. Then, during detailed routing,

each net is assigned to a specific wire in the channel. Routers typically perform the two step either sequentially, known as 2-step routing, or simultaneously, where both the channel and wire are chosen at the same time. One notable detailed routing algorithms is SEGA [21, 22]. The most widely used routing algorithm is the Pathfinder [23] algorithm, which is based on the A* search algorithm. There have been extensive studies on routing [24, 25]. VPR [13, 16] also functions as a router in addition to serving as a placement tool.

As the design progresses through each step of the CAD flow, it becomes more and more fixed. Decisions made early in the flow have a dramatically greater impact on subsequent steps of the flow. For example, at the *Placement* stage, the contents of CLBs have been determined, and usually cannot be changed. Therefore, if clustering was performed poorly, the placement problem also becomes more difficult. For example, if a large number of CLBs was made during clustering, the number of blocks that the placer needs to deal with also increases. This can affect the quality of the final placement, as well as increase the runtime of subsequent stages of the flow. Thus, it is important to optimize each step of the CAD flow, and more importantly, steps that occur early on in the flow.

1.3 Definitions of Key Terms

Throughout this thesis, a *cluster* refers to a CLB. *Clustering* is the process of grouping BLEs into CLBs such that they are design rule correct. The *architecture* of an FPGA refers to the maximum number of BLEs that can be put into one CLB. A *netlist* is the description of a hardware circuit, denoted by blocks of logic, or nodes, connected by *edges* or *nets*.

1.4 Statement of Thesis

There are three main objectives in the research documented in this dissertation. The primary objective is to provide a thorough analysis of the *Clustering* step of the FPGA CAD flow, and how it can be enhanced. To achieve this, several clustering algorithms have been implemented within

the same framework to evaluate and compare the performance of each algorithm. The second objective is to further improve upon the performance of the implemented algorithms through the addition of some preliminary physical information. The augmented algorithms have been evaluated to determine whether an improvement can be achieved. These results have been compared to data from other known clustering tools. Then, it is the goal of the thesis to determine how accurate physical information needs to be before a positive impact on clustering can be witnessed. Finally, several cluster improvement strategies have been implemented to study whether post-clustering optimizations can lead to improvements in the final placement.

This dissertation is organized as follows. Chapter 2 provides an overview of clustering algorithms found in the literature, and discusses relevant papers. Chapter 3 describes the implemented clustering algorithms in detail, as well as the enhancements made to these algorithms via the addition of physical information. Two cluster improvement heuristics are described in Chapter 4, which seeks to improve upon any initial set of clusters made by other tools. In Chapter 5, results from the implemented algorithms and heuristics are collected and compared. Finally, the findings are summarized in Chapter 6 with future directions outlined in Chapter 7.

Chapter 2

Background

Clustering serves many crucial functions in the FPGA CAD flow. First, it makes the placement problem smaller. By clustering BLEs into CLBs, the number of blocks that the placement tool needs to deal with decreases substantially. This tends to translate into reduced CPU requirements. The second advantage of performing clustering is that it eliminates Design Rule Checks (DRC) during placement. While making CLBs for a given FPGA architecture, the constraints of CLBs are strictly observed. Therefore, during placement, CLBs can be moved around without the need to worry that the move will result in an infeasible placement. For placement algorithms such as simulated annealing, where thousands of CLB moves are made while placing the circuit, the elimination of DRC checks can significantly speed up the placement process. Lastly, but most importantly, the main objective of clustering is to absorb signals and/or critical connections into CLBs. Critical connections are those connections that are important to the performance of the circuit. The absorption of critical signals into CLBs tends to improve the overall timing performance of the circuit since critical connections do not need to be routed between CLBs. The absorption of signals in general, whether critical or not, tends to reduce the number of signals that require routing between CLBs. This also has a great impact on the overall routability of the circuit.

Clustering can be broken up into two stages: register packing, and clustering. During register packing, the LUTs and registers of the primary netlist are packed into BLEs. In the second step,

the BLEs are clustered to form CLBs. The focus of clustering optimizations is on the second stage, since register packing is fairly straightforward.

The clustering problem is inherently different between FPGAs and ASICs . In ASICs, the main purpose of clustering is to group together standard cells so that the placer will have fewer aggregates to deal with. However, in FPGAs, because of their many architectural features and constraints, the primary objective is to create architecturally legal blocks of logic, rather than to reduce the size of the placement problem. Clustering algorithms can generally be grouped into two categories: seed-based, and depth-optimal methods. Seed-based algorithms work by forming one CLB at a time using an objective function. Depth-optimal methods tend to focus on improving the timing aspect of the circuit, and seek to optimize its performance by duplicating timing-critical logic during clustering.

This chapter seeks to provide an overview of the basic clustering algorithms, and a survey of existing literature. There have been substantial investigations conducted on the optimization of the clustering step. In Section 2.1, the algorithm used during register packing is briefly described. Section 2.2 discusses seed-based clustering algorithms. Depth-optimal techniques, such as logic duplication, are shown in Section 2.3. A brief survey of ASIC clustering algorithms can be found in Section 2.4. There has been some recent work that involves combining the clustering step with the placement step, and these are discussed in Section 2.5. Finally, the connection of the literature discussed to the work presented in this dissertation is shown in Section 2.6.

2.1 Register Packing

The first stage in clustering is the formulation of BLEs from the LUTs and registers in the netlist. A BLE can contain at most a LUT node and a register node, and will have only one output. Therefore, a register can only be packed with a LUT node if one of the outputs is not needed outside the BLE. This can occur in two situations, as illustrated in Figure 2.1. The first situation occurs when the output of the LUT goes only to the input of a register, and is not required by any other node in the netlist. The second situation occurs when the output of a register is only used as an

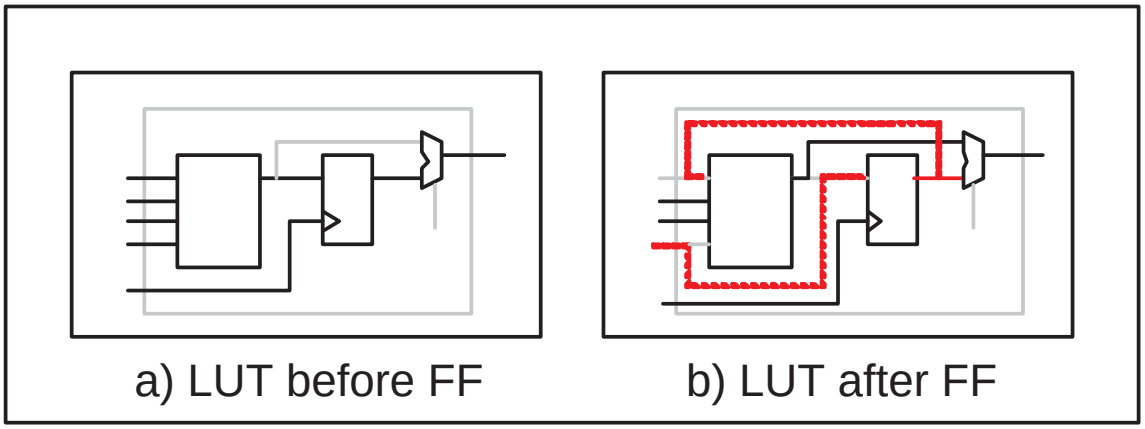

Figure 2.1: Possible BLE Configurations

input to a single LUT node. In this case, the LUT and register can be packed together as long as it does not violate the input constraints of the BLE. In Figure 2.1a, the absorbed net exists between the output of the LUT and the input of the register. However, in Figure 2.1b, the absorbed net is between the output of the register, and the input of the LUT.

The basic register packing algorithm is shown in Figure 2.2. For every register, its input and output nets are examined. If the net only has two terminals, and the terminal is a LUT, then this register is grouped with the LUT to form a BLE. At the end of register packing, all unclustered nodes are placed into separate BLEs.

2.2 Seed-Based Approaches

Seed-based methods are among the most established techniques for clustering BLEs in FPGAs. In such methods, CLBs are made greedily one at a time until every BLE has been clustered into a CLB. Seed-based approaches typically aim to minimize the number of CLBs formed, but can also be modified to take into account other objectives such as timing and power constraints.

One of the most widely known academic seed-based clustering tools is `VPack` and its timing-driven version `T-VPack` [26, 27]. In addition to trying to pack CLBs to capacity, `T-VPack` also accounts for the timing performance of the circuit by attempting to absorb netlist connections that

Procedure: REGISTERPACKING
Inputs: A primary netlist to be packed, N
Returns: A packed BLE-level netlist, N'

```
1  for each register i ∈ N do
2    clus ← new Cluster;
3    clus.add(i);
4    for each edge e ∈ i and i is unclustered do
5      if e is an input edge to i and getNumTerminals(e) == 2 then
6        driverNode ← get the driver of edge e;
7        if driverNode is a LUT then
8          clus.add(driverNode);
9          continue ;
10       fi
11     fi
12     if e is an output edge of i and getNumTerminals(e) == 2 then
13       sinkNode ← get the sink of edge e;
14       if sinkNode is a LUT and sinkNode and i can be added in the same cluster then
15         clus.add(sinkNode);
16         continue ;
17       fi
18     fi
19   od
20 od
21 //At this point, all unclustered blocks go into their own BLE
22 for each unclustered block i ∈ N do
23   clus ← new Cluster;
24   clus.add(i);
25 od
26 return N′;
```

Figure 2.2: Pseudocode for Register Packing.

are deemed to be timing critical. The packing algorithm of VPack and T-VPack starts with the selection of a seed BLE. The BLE with the most fully utilized inputs is usually selected as the seed BLE for a CLB. When timing is of importance, the most timing critical BLE is used as the seed of a CLB. Additional BLEs are added to the CLB until no more BLEs can be added without exceeding CLB constraints, such as number of BLEs per CLB or the number of pins available on the CLB. To choose which BLEs to add to the CLB, a gain value is calculated for every BLE that shares an edge with the current CLB, using a cost function. The gain is a function of the number of shared edges between the BLE and the CLB, and the criticalities of shared edges. T-VPack has been used extensively in academic research as the clustering tool to which all other clustering algorithms are compared against.

Two algorithms of note are present in T-VPack: hill climbing, and unrelated logic clustering. Hill climbing is an addition to the basic flow where BLEs are continually added to the CLB even after the number of inputs has been exceeded. This is done in the hope that an additional BLE will actually reduce the number of inputs needed for the CLB. This can occur when the output of a BLE is needed within the CLB as shown in Figure 2.3 [16]. The third BLE in the diagram generates the signal that is needed as an input in the first group of BLEs. By adding this BLE to the CLB, the input count can actually be reduced by 1, since the signal *c* can be generated locally without needing to route it from external sources. However, in general, this is shown to have limited benefits, with at most $1-2\%$ improvement in logic utilization [16]. The second algorithm allows unrelated logic to be packed together if some CLBs are not full. In this case, BLEs that do not share any inputs or outputs with the current CLB is still added, as long as no CLB constraints are exceeded. This allows T-VPack to pack as tightly as possible.

Although T-VPack can achieve very good results, it does not always give the most optimal answer. It is possible that the unrelated logic packed in this stage is best grouped elsewhere where a greater gain in edge reduction or critical delay is possible. Also, if a group of highly connected BLEs span more than one CLB, it is possible that by rearranging BLEs within the larger group, the edges connecting the CLBs can be reduced.

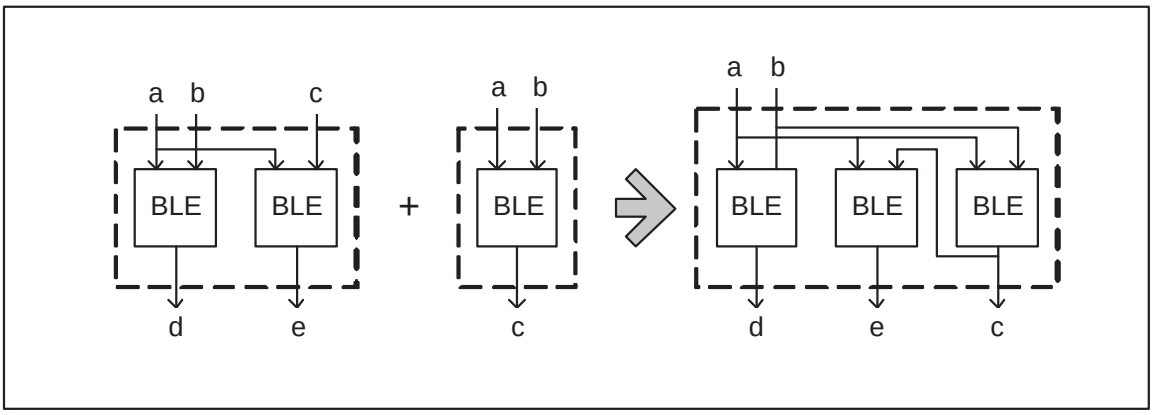

Figure 2.3: Hill Climbing Example from [16]

RPack is proposed in [15]. Like VPack and T-VPack, RPack also packs BLEs one at a time starting with a seed BLE. However, RPack extends VPack by integrating routability into the clustering step to reduce the number of wires required in the routing channel [15]. This is performed by adding a term to the cost function. This extra term accounts for routability by calculating the number of shared input and output pins between an unclustered BLE and the current CLB. This routability term also penalizes BLEs that do not share anything with the current CLB, to deter this algorithm from putting them together. Compared to VPack (non-timing-driven T-VPack), previous research [15] show that RPack can significantly improve circuit routability. However, this research [15] focused only on routability—no performance numbers were presented to indicate the impact of packing for routability on the final quality of the result in terms of timing. Additional research [14] provides numerical results that show that while RPack outperforms VPack, it only produces results that are comparable to T-VPack.

In iRAC [14], another routability-driven packing algorithm is described. This algorithm is also seed-based and packs CLBs one at a time. However, the selection of a seed BLE is different from the method employed by T-VPack. iRAC selects seed BLEs based on its connectivity factor *c*. This is calculated via Equation 2.1, where the separation of a BLE is the sum of the number of terminals on nets connected to the BLE, and the degree is the number of nets directly connected to the BLE.

$$c = \frac{separation}{degree^2} \tag{2.1}$$

This connectivity factor increases the importance of BLEs that have more low-fanout nets. By starting with BLEs that have low-fanout nets, it increases the likelihood that additions to the CLB will result in nets being absorbed in their entirety. Thus, these nets can be removed from the resulting clustered netlist. Since the router will have fewer edges to route, this makes the routing step easier. Another key idea presented in iRAC is the use of Rent's Rule during clustering. iRAC limits the the number of pins that are usable on any CLB to match the Rent parameter of the architecture. By reducing the number of usable pins on CLBs, the demand on the routing channel is also reduced. Numerical results [14] indicate that the improved selection of the seed BLE coupled with the use of the Rent parameter can reduce the number of inter-CLB edges by roughly 30% compared to RPack and T-VPack for the case of 8 BLEs per CLB architecture. However, the number of used CLBs increased substantially by 5% to 6%. This may become a problem in a highly utilized device. Although edge reduction results are encouraging, the effect on performance is unknown since no performance numbers were presented in this paper.

2.3 Depth-Optimal Methods

While capable of achieving very tight packings, seed-based approaches are localized, greedy algorithms that may become trapped in local minima. Another set of methods, called depth-optimal or depth-relaxed methods, seeks to optimize the performance of the circuit by duplicating timing-critical logic during clustering. Through the process of node duplication, a set of CLBs with optimal depth can be obtained through the use of a variety of post-processing, bin-packing methods. TLC [28], MLC [29] and RCP [30] are all examples of depth-optimal clustering tools. These three methods are all multi-level clustering algorithms, with an emphasis on producing timing-optimal designs. The advantage of these methods is that they enable a more global view of

the circuit to be taken. Although most often used for hierarchical FPGAs in which CLBs are futher grouped together (e.g., the Altera APEX20K [31]), they work just as well in architectures with only 1 level of hierarchy. Therefore, these methods still apply to the island-style FPGAs considered in this thesis, where the only hierarchy is that in which BLEs are grouped into CLBs.

In depth-optimal algorithms, three phases are performed: a labeling phase, a clustering phase and a packing phase. In the labeling phase, each node in the circuit is labeled with its depth-optimal delay from the primary inputs of the network. This is performed by traversing the netlist from primary inputs to primary outputs. This generally results in a large number of highly underutilized CLBs. Then, in the clustering phase, the network is traversed from primary outputs backwards to the primary inputs, and a subset of CLBs are selected such that the entire network can be covered. However, there is usually still a large number of CLBs. Therefore, a third phase is needed to pack the CLBs tighter to reduce the number of CLBs needed.

An example of how typical depth optimal methods work is shown in Figure 2.4 and Figure 2.5. These figures, found in [32], describe the process of logic duplication used in conjuction with a depth-optimal algorithm to reduce timing delays. The labelling phase is shown in Figure 2.4. Figure 2.4a depicts a graph that represents a circuit with *a*, *b*, *c*, *d*, and *e* as primary input nodes, and *j* and *k* as primary output nodes. The delay of each node is shown in the graph by a number next to the node, with all inter-cluster delays set to be 3. The architecture of this example is 3 BLEs per CLB. The labels are computed by traversing the graph from primary inputs, to assign the maximum delay encountered at each node. Thus, it can be seen in Figure 2.4b that *f* has a delay of 3. Then, as we propagate the delays forward from *f* to *h*, we consider the entire subgraph based at *h*, shown in Figure 2.4c, and calculate accordingly. If the cluster is performed as circled in Figure 2.4c, then there will not be any inter-cluster delay between *f* and *h*, and its label is only increased by its internal delay.

After labels have been computed for the entire circuit, the clustering phase of the algorithm is executed, propagating backwards from the primary outputs, to form the optimal set of clusters shown in Figure 2.5. As evident from the clusters shown, nodes *b* and *f* have been duplicated.

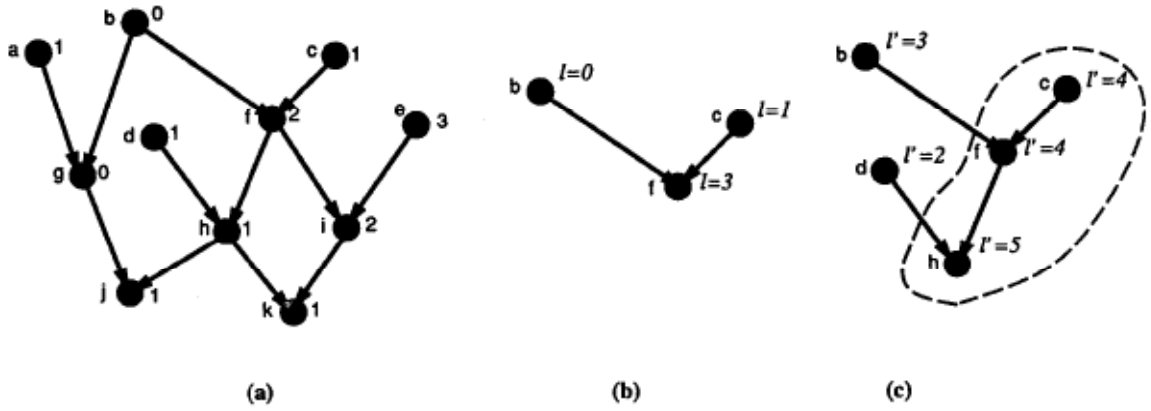

Figure 2.4: Depth-Optimal Example: The Labelling Phase [32]. (a) Circuit graph (b) Labels of nodes *b*, *c*, and *f* (c) Computing the label of *h*

The strength of depth-optimal algorithms is in that they can dramatically shorten timing-critical paths by absorbing them within the CLB. However, in the process of such reductions, logic must be duplicated to provide maximum benefits. Logic duplication can therefore get out of hand very quickly. Although effective at reducing critical path delay, previous experimental results indicate that the process of logic duplication can be hard to control, leading to large increases in area. Also, minimizing logic depth does not mean a reduction in wire length in modern designs. Although the use of timing information during clustering can lead to a better set of clusters, recent research indicates that timing estimates made during clustering may not be accurate when compared to the final placement [33].

2.4 ASIC Clustering Algorithms

It is worth mentioning that a substantial set of literature exists on ASIC clustering techniques [34–37]. The main difference between the ASIC and the FPGA clustering problem is that there are no CLBs in ASICs. Therefore, there are no CLB constraints that need to be taken into account. The main objective of ASIC clustering algorithms is to create larger aggregates of highly connected nodes so as to speed up placement. In contrast to the BLEs and CLBs for FPGAs, the netlist for

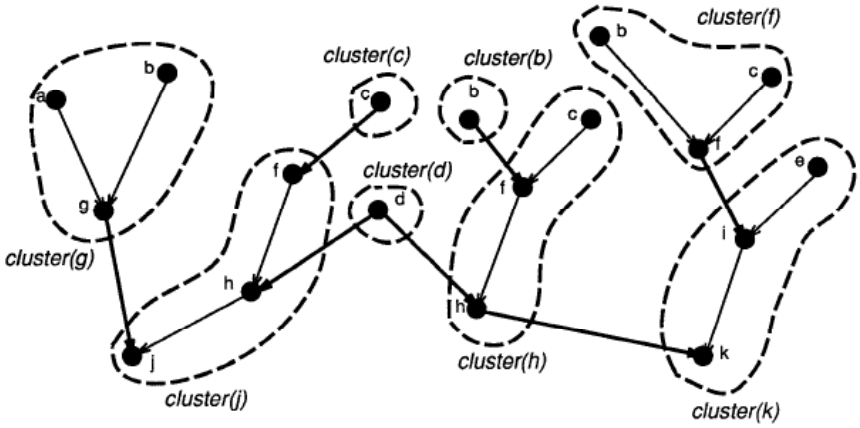

Figure 2.5: Depth-Optimal Example: Optimal Clustering of Figure 2.4a from [32]

ASICs includes macro blocks, and standard cells or nodes.

ASIC clustering methods are usually affinity-based, and work on many clusters simultaneously. Examples of affinity-based algorithms used in the ASIC CAD flow include Best Choice [38], First Choice [39], and Hybrid First Choice (HFCC) [40, 41]. At the beginning of clustering, HFCC [41, 42] computes the affinity of every possible pair of nodes. After sorting the calculated affinities, it starts to make pairings between nodes to form clusters by pairing nodes with the highest affinities to each other. If the nodes in question are already clustered, then the possibility of merging this additional node into the existing cluster is investigated. This process of pairing is continued until no more merging of blocks can be made without violating cluster constraints.

One of the advantages of affinity-based methods is that, unlike seed-based methods, affinity-based methods work on multiple clusters at the same time. Therefore, it is not concerned with the minimization of cluster count. Because of the greedy nature of seed-based algorithms, clusters are packed as tightly as possible. Although this is benefical in terms of area reduction, it is possible that blocks may be added early on that are better off clustered with other, still unclustered, blocks. This problem does not exist in ASIC clustering, since there aree no CLB constraints to take into consideration. This means that the algorithm is always making the best possible decision. Since it is not limited by dense packing, affinity-based methods ensure that good decisions are always

made. However, the side effect of this method is that affinity-based methods produce a much higher number of clusters than seed-based algorithms, which may be problematic if applied to FPGAs. As described later in Chapter 3, an attempt was made to adapt these methods mentioned here to FPGAs with mixed results.

2.5 Simultaneous Clustering and Placement

In the traditional FPGA CAD flow, the clustering step is completed before placement is performed. Normally, placement tools such as VPR perform placement on the netlist of CLBs, and make moves by swapping CLBs between locations on the FPGA grid. However, in recent years, an alternative has been investigated such that BLEs are allowed to move between CLBs during the placement step. This in turn restructures CLBs that were previously formed in the clustering stage. The advantage of performing BLE-level moves during placement is that physical information, as well as more accurate timing information, can be used to make better CLBs. An example of this can be found in SCPlace [43]. SCPlace implements a simulated annealing-based placement method that is capable of moving both CLBs and BLEs. SCPlace uses T-VPack to generate an initial set of CLBs which are feasible for the architecture (i.e., an initial packing must still be performed). Then, during placement, both CLB-level and BLE-level moves are performed. It is fairly easy to make a CLB-level move. However, when BLEs are moved between CLBs, CLB constraints must be observed before the benefit of the move can be evaluated.

SCPlace also implements the net weighting algorithm proposed by Kong [44] to improve its performance. Through experimentation, substantial reductions in wire length of up to 36% and critical path improvement of up to 31% can be found, when compared to VPR (which performs no reclustering and only moves CLBs). It was found that the combination of CLB- and BLE- level moves produce the best results. By performing only 10% of the number of CLB moves that VPR performs, SCPlace was able to compensate for the time it uses to do BLE moves. The significance of SCPlace is that it demonstrates the importance of physical information in correctly predicting wire length and delay information.

However, BLE moves are expensive to make. In modern FPGA architectures, there are many more constraints, such as carry chains, to consider in a CLB. Therefore, each time a BLE move is considered, the legality of the proposed move must be verified. Since there are many such constraints in commercial FPGAs, this DRC check can take a much larger proportion of runtime execution if used in a commercial setting. Hence, if similar results can be achieved by altering the clustering stage with physical information, the need for BLE moves during placement can be eliminated. This should also improve overall runtime of the FPGA CAD flow.

Finally, reclustering can also occur at the end of placement. It is here that physical optimizations can be made, often by exploiting physical information obtainable at this point [45]. Logic replication can also be found at the placement level for FPGAs [46]. Critical paths can be *straightened* whenever possible by means of duplicating logic. An example is shown in Figure 2.6 [46].

In Figure 2.6a, there are 4 paths going through node *c*, between fixed output nodes of *a* and *e*, and input nodes of *b*, and *d*. If no logic duplication is allowed, then node *c* would have to be placed in the center to minimize the maximum path delay of all paths. However, if node *c* can be duplicated to create a copy c', then it may be possible to place nodes *c* and c' in the arrangement shown in Figure 2.6b. In this case, the paths have been straightened, and it can be seen that the length of all paths have effectively been cut in half. By straightening the path, the impact of routing delays on the critical paths can be minimized. However, these physical optimizations do require reclustering and relegalization of the design, and are beyond the scope of this thesis.

2.6 Relation of Past Literature to Current Research

The goal of this work is to develop a new clustering algorithm that can outperform the clustering algorithms mentioned. By extracting the positive characteristics from the existing approaches, it is hoped that a better clustering approach can be found. The investigation carried out in this thesis research is essentially a hybridization of ASIC techniques and seed-based approaches. The key idea is to perform partitioning, such as algorithms normally occurring during placement, to

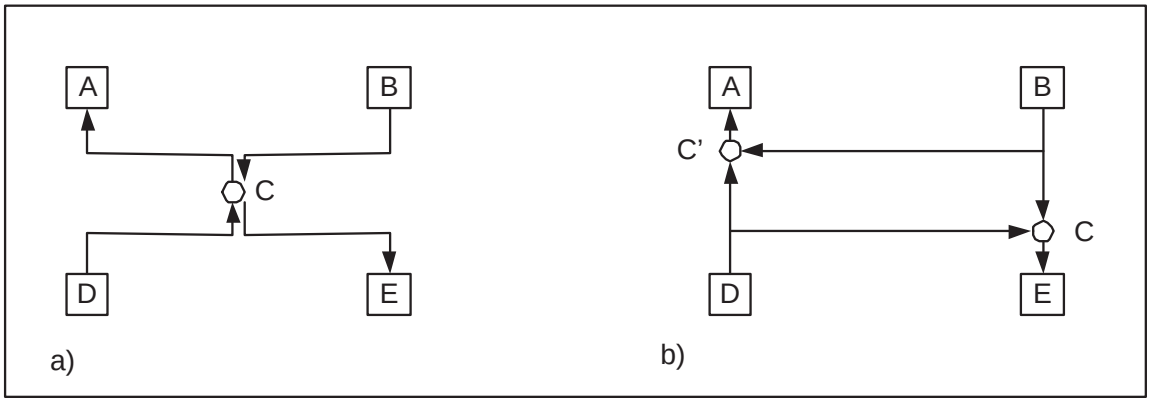

Figure 2.6: Logic Duplication Example [46]

obtain the approximate physical location of BLEs. Then, a seed-based clustering algorithm can be performed while utilizing the additional information. It is hoped that through the use of physical information, the advantages of both approaches can be realized. Therefore, physical information is incorporated into the seed-based clustering algorithm from `T-VPack`. By clustering with some physical information, potentially better clusters can be made, which may lead to better performance of the final placement.

In this thesis research, two packing algorithms—called `DPack` and `HDPack`—are introduced. These three algorithms produce *better initial packings*, which in turn reduce the dependence on computationally-expensive BLE-level placement. `DPack` and `HDPack` incorporate the concept of "physical clustering" [41] within a novel hybrid framework for timing-driven FPGA packing. These techniques employ a quick min-cut, partitioning-based global placer to determine *approximate* BLE locations. By using this information, these tools are capable of making more informed decisions which, in turn, can lead to reduced wire lengths and critical path delays.

Chapter 3

Clustering Algorithms

From Chapter 2, it can be seen that seed-based and depth-optimal algorithms have been widely used and adapted for FPGAs. Both types of clustering algorithms generally have very fast runtimes and provide good solutions. However, the depth-optimal methods generally provide better performance than seed-based algorithms. Since depth-optimal methods are allowed to duplicate timing-critical nodes, the critical path delay of circuits can generally be shortened. Although these methods give good performance gains, an area increase is inevitable and can sometimes be quite substantial. Simultaneous clustering and placement methods also show promise by allowing the contents of clusters to change during placement. However, since each change in cluster content must be preceded by a DRC check, the runtime of such algorithms is greater than clustering algorithms alone. Therefore, the focus of the research here is to improve upon the most widely used seed-based clustering algorithms, and attempt to achieve equivalent, or better, results found in the existing literature without the use of node duplication or BLE-level moves in placement. Although BLE-level placement algorithms will likely remain a necessity in commercial FPGA placement, it is the premise of this work to *reduce* the reliance on this step by producing better CLBs in the first place. The idea is to create a better set of clusters, and a better final placement, without incurring area or runtime penalties. To this end, several known algorithms have been implemented and modified to see whether the results can be improved in ways other than node duplication and

BLE-level placement algorithms.

Two clustering algorithms are implemented as part of this thesis research. First, the primary algorithm, called `DPack`, is discussed in Section 3.1. Section 3.2 describes a second algorithm, called `HDPack`, which is an extension to `DPack` by hybridizing it with an affinity-based algorithm. The process of augmenting both algorithms with physical information is presented in Section 3.3.

3.1 Greedy Packing (`DPack`)

In pursuit of better clusters, a seed-based packing algorithm, similar to `T-VPack`, was developed. The pseudocode for this clusterer, `DPack`, is shown in Figure 3.1. Like `T-VPack`, a seed BLE is selected as the most critical, unpacked block. Kong's path counting algorithm [44] was implemented as a tie-breaking mechanism during seed selection, with the block that has the highest path count selected as the seed. It should be noted that logic depth is also used as a secondary mechanism to break ties [16]. After the seed BLE has been chosen, a cost function is computed for all blocks that are connected to this BLE. This cost function is given by

$$\text{Cost}_{ij} = \lambda \times E_{ij} + (1-\lambda) \times \text{Crit}_{ij} \tag{3.1}$$

where

$$E_{ij} = \sum_{e \in E_h \,|\, i,j \in e} \frac{1}{|e|-1} \text{ and } \text{Crit}_{ij} = \sum_{e \in E_h \,|\, i,j \in e} \text{Criticality}(e).$$

Here, E_h represents all nets in the netlist, E_{ij} models connectivity, and $\text{Criticality}(e)$ is the estimated timing criticality of net e. From this equation, it can be seen that each net is weighted by the number of terminals on it, similar to [47]. This increases the importance of nets that have fewer fanouts, and increases the likelihood that they will be absorbed. In Equation 3.1, λ varies between 0 and 1 and controls the preference between edge absorption and timing criticality. The BLE with the highest computed cost is added to the CLB. This is continued until either the CLB is

full, or other constraints, such as the number of pins available on the CLB, are exceeded. Then, a new seed BLE is chosen to start a new CLB, and the process is repeated until the circuit has been packed.

DPack also incorporates the hill-climbing and unrelated logic packing algorithms from [16]. When the pin constraints of a CLB have been reached, but the CLB is not full, the clusterer enters a hill-climbing phase; BLEs are continuously added to the CLB even if the number of pins on the resulting CLB exceed what is feasible. This is done in the hopes that, by adding more BLEs to the CLB, the number of pins can be reduced as more edges are absorbed. If, after reaching the maximum number of BLEs per CLB, the pin constraints are still violated, the last feasible arrangement is restored. If a CLB is not full, then additional BLEs that have no direct connection (i.e., unrelated logic) with the BLEs in the CLB may be added provided that the CLB constraints are not violated.

3.2 Hybridized Packing (HDPack)

The second clustering algorithm is built on DPack. For this approach, an affinity-based algorithm was incorporated into the clustering flow. This affinity-based algorithm, called Hybrid First Choice clustering (HFCC), has been successfully used in ASIC clustering. Since HFCC has been applied successfully to large-scale placement, it seems worthwhile to explore the usefulness of this algorithm in the context of FPGA placement. First, Section 3.2.1 describes the HFCC algorithm, and Section 3.2.2 provides the details of the hybridization of HFCC and DPack.

3.2.1 Affinity-Based Packing (HFCC)

In HFCC, objects are initially placed onto a "free" list which contains the set of objects which have not been paired. The *affinity* for pairing any two objects is calculated using Equation 3.1 [41]. Then, starting with the highest affinity value, pairings are made between the specified blocks, which may be unclustered BLEs or clusters, as long as no CLB constraints are violated. The

Procedure: DPACK
Inputs: A netlist to be packed, N
Returns: A packed netlist, N'

```
1  Perform timing analysis on the circuit;
2  Compute block criticalities via Kong path counting;
3  Sort block criticality from highest to lowest;
4  seedBLE ← most critical unclustered node;
5  while seedBLE > 0 do
6    clus ← new Cluster;
7    clus.add(seedBLE);
8    while clus.getNumBLEs() < maxNumBLEsPerCLB do
9      for each BLE that shares an edge with clus do
10       Calculate the cost according the equation;
11       if BLE can be added (passes DRC) then
12         costVector.add(BLE, cost);
13       fi
14     od
15     BLEtoAdd ← getHighestCostBLE(costVector);
16     if BLEtoAdd is not valid then
17       BLEtoAdd ← get best unrelated BLE to add;
18     fi
19     if BLEtoAdd is valid then
20       clus.add(BLEtoAdd);
21     else
22       break ;
23     fi
24   od
25   Add clus into N′;
26   seedBLE ← most critical unclustered node;
27 od
28 return N′;
```

Figure 3.1: Pseudocode for DPack.

algorithm repeatedly removes the object with the highest affinity from the free list, and pairs it with the object that (originally) yielded this high affinity, even if that object had already been paired. Once an object has been paired, it is said to have formed a "cluster". Pairings are made continuously, until no more pairings can be made without exceeding CLB constraints. The basic pseudocode for the affinity clustering algorithm is found in Figure 3.2.

HFCC has the advantage of making CLBs simultaneously, without the worry of packing clusters to the fullest. However, the algorithm can terminate with a large number of CLBs. This is due to the fact that ASIC clusters do not have constraints such as in FPGAs. ASIC clusters are not required to pack for a minimum number of clusters, and hence there is no need to pack unrelated logic. This is not a deficiency in ASIC clustering algorithms; rather, this is only seen as a deficiency when these algorithms are applied to FPGAs. However, this makes it difficult to compare results fairly to other clusterers that pack for minimum area. Also, this artificial bloat in the number of clusters wastes FPGA area, and affects the performance of the circuit design. This large set of clusters can be difficult to pack together in later stages of the algorithm due to pin constraints and a lack of a hill-climbing phase. Consequently, HFCC packings typically contain several percent *more* CLBs than DPack or T-VPack; for highly-utilized devices, this can be a significant drawback.

In practice, this algorithm is followed by several post-processing steps to reduce the number of clusters. These steps include the merging of single BLEs into CLBs when possible, and the merging of half-filled blocks. However, these post-processing routines are inherently greedy, and the only optimization goal during this phase is to minimize the number of clusters. This is similar to depth-optimal methods (without duplication) in which the bin-packing applied after the initial clustering cannot effectively group clusters together to reduce the CLB count. This may have a detrimental effect on the quality of clusters, both in terms of wire length and critical path delay.

3.2.2 Formulation of HDPack

Both DPack and HFCC have numerous associated advantages. HFCC is noted to be very effective at making good pairwise packings and in minimizing the number of external nets in the clustered

```
Procedure: HFCC
Inputs: A netlist to be packed, N, StoppingCost
Returns: A packed netlist, N'

1  // Do affinity clustering ...
2  for each edge e ∈ N do
3    for each cell i, j ∈ e do
4      Cost_ij ← compute affinity cost for pairing i, j;
5    od
6  od
7  Sort all affinity Cost_ij from largest cost to lowest;
8  StoppingCost ← predetermined Cost_ij value at which to stop;
9  for each Cost_ij do
10   Attempt to pack cell i and j together;
11   if DRC was not successful then
12     continue ;
13   fi
14   if Cost_ij < StoppingCost then
15     break ;
16   fi
17 od
18
19 return N';
```

Figure 3.2: Pseudocode for HFCC.

netlist. However, it often creates a large number of CLBs, and the employed post-processing routines have a negative impact on the quality of clusters made. In contrast, DPack is known to achieve good critical delay reduction while being able to pack for a minimum number of CLBs. Thus, it is proposed to combine the two approaches in order to take advantage of the benefits of both DPack and HFCC. It is hoped that this hybridized flow, HDPack, can yield the same high net absorption offered by HFCC, while still preserving the critical delay reduction from DPack. The pseudocode for HDPack is shown in Figure 3.3.

In this combined approach, HFCC is used as a *pre-packing* step before DPack is called to perform clustering. First, HFCC is used to make initial pairings; when the affinity values of the pairings in the HFCC packer fall below a certain threshold, HFCC packing is stopped. At this point, a large number of CLBs is generally required. However, unlike HFCC, none of the original post-processing routines are used. Instead, this list of "intermediate" clusters is fed to DPack to complete the clustering process. In this stage, DPack looks at this set of clusters, and computes costs using Equation 3.1. Then, it starts to fill CLBs, starting with the CLB that has the highest number of contained BLEs, highest number of used pins, and highest criticality. The BLE with the highest computed cost is then added to the CLB until the CLB becomes full. However, unlike DPack, it is possible that the BLE has already been packed into another CLB by HFCC. In this case, the BLE will be removed from its current cluster and added to this CLB only if its original CLB was *not* full.

The effectiveness of HDPack has a strong dependency on the handoff point between HFCC and DPack. If HFCC performs too few affinity-based matches, the true benefit of the hybridization may not be visible. On the other hand, if HFCC almost finishes off all the possible pairings, DPack may not have enough room to achieve a minimum set of clusters without significantly messing up the decisions HFCC made. Therefore, this handoff point, or threshold, must be clearly analyzed, parameterized, and the possible values swept to determine the best configuration. This point is calculated according to Equation 3.2, where Aff_{max} and Aff_{min} are the maximum and minimum affinities found, respectively. The hybrid cutoff value, β, is a parameter that is then swept from 0 to 1 to see which value provides the best wire length and critical delay improvement. If the affinity

values fall below the calculated threshold, HFCC terminates, letting DPack finish packing the rest of the blocks. In practice, it was found that this hybrid flow produces very good improvements in wire length and critical path delays over traditional methods.

$$Aff_{thres} = (Aff_{max} - Aff_{min}) \times \beta + Aff_{min} \tag{3.2}$$

3.3 Incorporating Physical Information

The concept of "physical clustering" has been employed successfully in ASIC placement for some time [41]. In these approaches, an initial placement for cells in the unclustered netlist is determined via a quick global placement operation. During this global placement, cells are allowed to overlap, since circuit legality is not a concern. The clustering method leverages the inter-cell distances from this approximate placement to make more informed "tie-breaking" decisions, and to make better clustering decisions when packing unrelated logic. The core of this research is to determine whether the same approach can be used in FPGAs with positive outcomes by incorporating physical information into the clustering algorithms described previously.

Before physical information was incorporated into DPack and HDPack, a simplistic, top-down, min-cut partitioning-based global placer was first developed. This placer uses hMetis [40, 48] to recursively bi-partition and place the primitive netlist. A sample figure illustrating how min-cut partitioning is typically performed is shown in Figure 3.4. First, all the nodes are placed onto the chip, with positions set in the centre as shown in Figure 3.4a. Then, these nodes are divided into two groups, or partitions, as shown in Figure 3.4b. This division between the nodes is called a cut. The objective of this cut selection is to minimize the number of nets that connect between the two sections. Thus, the algorithm encourages highly-connected nodes to remain within a common partition. After the nodes have been divided, each resulting region is then partitioned further, independently of the other partition. The nodes are then further distributed as shown in Figure 3.4c.

Procedure: HDPACK
Inputs: A netlist to be packed, N
Returns: A packed netlist, N'

```
1  Call HFCC to perform affinity-based clustering(N, StoppingCost);
2  // Finish off using greedy ...
3  Put each unclustered BLE into its own cluster;
4  Collect statistics (num pins, etc.) for each cluster;
5  Sort the list of clusters first (num BLEs contained, num pins, criticality);
6  seedClus ← cluster with highest # BLEs, highest # pins, highest criticality;
7  while seedClus is valid do
8    for each BLE B that shares an edge with clus (not in full cluster) do
9      Calculate the cost of putting B in seedClus;
10     if B can be added to seedClus without violating DRC then
11       costVector.add(B, cost);
12     fi
13   od
14   BLEtoAdd ← getHighestCostBLE(costVector);
15   if BLEtoAdd is not valid then
16     BLEtoAdd ← get best unrelated BLE to add;
17   fi
18   if BLEtoAdd is in another cluster already then
19     Remove BLEtoAdd from its original cluster;
20   fi
21   if BLEtoAdd is valid then
22     seedClus.add(BLEtoAdd);
23   else
24     break ;
25   fi
26   if seedClus.getNumBLEs() = numBLEsPerCLB then
27     Mark seedClus as full, and therefore cannot be modified anymore;
28   fi
29   seedClus ← most fully used, yet still incomplete cluster;
30 od
31 return N';
```

Figure 3.3: Pseudocode for HDPack.

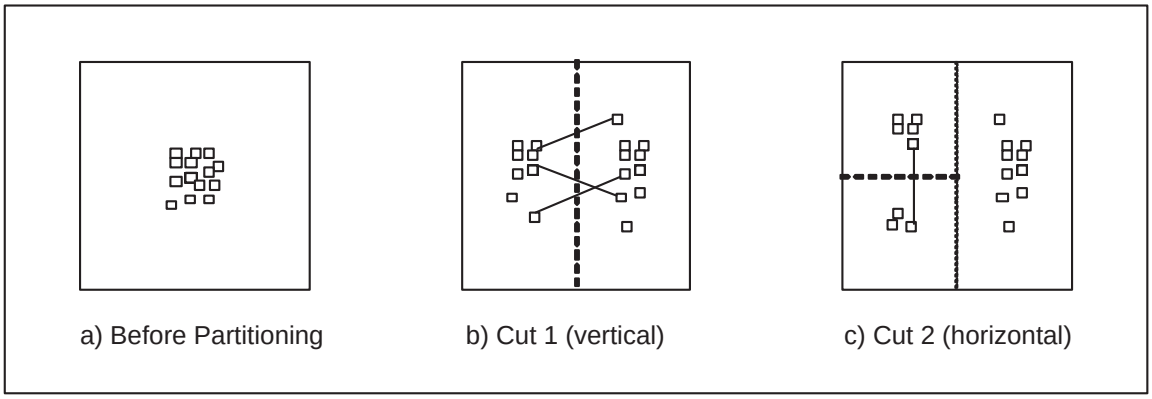

Figure 3.4: Min-Cut Partitioning Algorithm

This is performed recursively until some stopping criterion has been reached. During placement, this may occur when all nodes are suitably spread throughout the chip area, and physical locations can be assigned.

It should be noted that the technique used in this research does not employ placement feedback, or branch-and-bound partitioning, as in [49]. These methods are used to fine-tune the accuracy of a partitioner, and are not used since only a basic partitioner is required for our purposes. Placement feedback [50] is a method to make accurate terminal propagation during partitioning by using the concept of feedback from control system applications. Branch-and-bound partitioning [51] essentially enumerates partitioning solutions, and uses bounds to discourage unnecessary exploration.

Since accurate information is not necessary, the partitioning algorithm does not need to run to the point where every block has a unique location. Instead, this min-cut partitioning placer is augmented with a stopping criterion that depends on the number of nodes in the current partition. `hMetis` will then be recursively called until one of two conditions is reached. Once the number of nodes in a partition is either (a) less than a predetermined amount *or* (b) the depth of the partitioning tree has exceeded a threshold, the partitioning algorithm stops. The BLEs within the partition are then assigned the same *x* and *y* grid locations. It should be noted that this is perfectly acceptable since it is not the intention of this fast partitioning to generate legal placements, but rather, to

provide a rough idea of which BLEs may end up close together. Then, a clustering algorithm such as DPack or HDPack is executed.

To account for physical information, the cost function in Equation 3.1 is augmented with an additional cost term; the new cost function is given by Equation 3.3

$$\text{Cost}_{ij} = \lambda \times E_{ij} + \gamma \times \text{Crit}_{ij} - (1 - \lambda - \gamma) \times \text{Dist}_{ij} \tag{3.3}$$

where

$$\text{Dist}_{ij} = \frac{|x_i - x_j|}{\text{GridSize}_x} + \frac{|y_i - y_j|}{\text{GridSize}_y}$$

where E_{ij} and Crit_{ij} are the same as in Equation 3.1. In this formulation, λ and γ control the preference between edge absorption and timing, respectively. The Dist_{ij} term is a calculation of the Manhattan distance between the current CLB and the potential BLE, normalized by the grid size. As a consequence of this formulation, this cost *penalizes* objects that are far apart. Although several other formulations of the cost function were also considered and tested, this formulation was found to yield the best performance.

Another modification to DPack and HDPack was made in the way unrelated logic clustering is performed. In the original algorithm, any BLE that could fully utilize the remaining available inputs of a CLB was added. In practice, there can be many blocks with the same number of inputs. To break ties, we use the physical distance between the potential BLEs and the current CLB. Consequently, the BLE that is closest to the current CLB is added to the CLB.

Chapter 4

Cluster Improvement Algorithms

Since clustering is an important algorithm in the overall FPGA CAD flow, additional time spent on improving the quality of the resulting CLBs should be reflected in the final overall quality of placements. In addition to the clustering algorithms proposed in Chapter 3, two cluster improvement algorithms were implemented and investigated. These improvement algorithms are complementary to any of the aforementioned clustering algorithms. That is, the algorithms introduced in this chapter are not intended to produce an initial set of CLBs, but rather, to *further improve* on an existing set of CLBs. When used with a placement method such as `SCPlace`, these algorithms are still valid. If these cluster improvement techniques can be performed quickly, a better initial packing may be made. This better initial set of clusters would likely translate into both a better initial placement and a potential reduction in runtime due to the need to perform fewer placement perturbations to obtain a high quality final result.

Given an initial clustering of BLEs into CLBs, two heuristics are implemented to further improve upon the clusters. Both of the described heuristics work as follows. A number of improvement attempts are performed. In each attempt, a pair of CLBs is selected. This selection can be random or require that the selected pair of CLBs share some common connections. This requirement that CLBs share common connections makes sense; it is likely that to improve the absorption of edges — either for routability or for timing — a selected pair of CLBs must have some common edges

which can potentially be absorbed.

This chapter starts with the general outline of the improvement algorithms in Section 4.1. Then, in Section 4.2, a greedy improvement scheme is described that performs swaps and moves to improve the quality of clusters. Section 4.3 shows a branch-and-bound scheme that takes a different approach to improve clusters.

4.1 Overall Flow

Both cluster improvement algorithms take place *after* an initial set of CLBs has been created and works as follows; for a given set of CLBs, the algorithm selects pairs of CLBs — either random pairs of CLBs or pairs of CLBs that share common edges — and attempt to rearrange these two BLEs through either of the improvement heuristics described below. This rearrangement is then evaluated to see if an "improved" pair of CLBs can be obtained. The pseudocode for this overall flow is presented in Figure 4.1.

The two proposed heuristics differ in how they select BLEs to move. In Swaps and Moves, random BLEs are selected from each CLB and are either swapped (when two BLEs are selected to be switched) or moved (when one BLE is selected to move to an empty spot in the other CLB). The two CLBs are then evaluated to see if the quality, in terms of timing or wire length, has improved. The second heuristic is based on branch-and-bound, and enumerates all possible packings of BLEs into the pair of selected CLBs to find an improved packing of BLEs into CLBs.

4.2 Swaps and Moves

The first proposed heuristic involves the simple greedy swapping of BLEs between CLBs. This heuristic is similar to that originally proposed for ASIC clustering [47] in which the objective was to absorb as many edges as possible into clusters. Pairs of connected clusters were randomly selected and a cell (pair of cells) was moved (swapped) between the two chosen clusters. In

Procedure: IMPROVECLUSTERPAIRS
Inputs: A packed netlist, N
Returns: An improved packed netlist, N'

```
1 pass ← 1;
2 while pass < max_pass do
3   Select two connected CLBs i and j;
4   ImproveCLBs(i, j);
5 od
6 return N';
```

Figure 4.1: Outer Loop of Improvement Heuristics.

performing the swap, edges were weighted according to Equation 4.1

$$w_i = \frac{1}{p_i - 1} \tag{4.1}$$

where w_i and p_i represent the weight and number of pins on edge i, respectively. This weighting scheme tends to give priority to low fan-out edges which are easier to absorb completely into a cluster. Upon performing either a move or swap of cells between clusters, the total absorption of edges into the pair of clusters is computed. If the absorption of edges is improved, then the move (swap) is retained, otherwise it is discarded. In [47], the moves and swaps are performed using annealing such that it is likely some worsening swaps are chosen during the improvement heuristic. However, since only improving moves are allowed here, this implementation is greedy.

Several additions to the algorithm in [47] were necessary to adapt it for the FPGA. The above algorithm is purely driven by edge absorption. This usually has the effect of reducing wire length in the final routed circuit. However, for FPGAs, the critical path delay is also an important parameter to optimize for. To account for timing, a "unit-delay" timing analysis is performed and a slack for each connection i in the circuit is computed. The slack is used to compute a criticality for each connection given by Equation 4.2

$$Criticality(i) = 1 - \frac{slack(i)}{MaxSlack} \tag{4.2}$$

where *MaxSlack* is equal to the largest slack found in all the connections in the circuit. This criticality is then squared so that critical connections appear more critical, while relatively non-critical connections are ignored. The cost of a CLB is calculated by summing up the number of edges absorbed (as in [47]) *and* by summing up the absorption of critical circuit connections. If both edge absorption and criticality absorption are improved after either a move or swap, then the new arrangement of BLEs to CLBs is accepted. Otherwise, the original assignment of BLEs to CLBs is restored.

4.3 Branch and Bound

Although random greedy moves and swaps do improve clusters, this scheme may be somewhat limited in its ability to improve CLBs. For instance, it might be the situation that many attempts are made that do not meet the architectural constraints of the CLB. Furthermore, it might be necessary that *more than two* BLEs should change CLBs in order to obtain an improvement. To overcome the potential limitations of simple moves and swaps, an *enumerative* heuristic, which is based on branch-and-bound, is proposed to improve pairs of CLBs. Since the number of BLEs per CLB is limited to a fairly small number in modern architectures, branch-and-bound is practical. Furthermore, because of the enumerated nature of branch-and-bound, if a better packing of BLEs into CLBs exists, it will be found. Complex packing constraints such as limited inputs, outputs and control signal constraints are handled seamlessly by branch-and-bound.

This technique essentially performs a constraint-aware bin packing between two CLBs. The algorithm initially begins with no assignment of BLEs to either CLB. It then attempts to assign each BLE to the first CLB and then to the second CLB in a subsequent pass. When assigning a BLE to a particular CLB, the architectural constraints need to be obeyed. It should be noted that some computations must be performed carefully. For instance, to ensure that the input limits on a CLB are not violated, input counts cannot be simply be “incremented“ when a BLE is assigned to a CLB. This assignment needs to be *deferred* until the location of the source BLE of an edge is known. If the source BLE of an edge is in the same CLB as this BLE, the edge is absorbed, and thus will not

add to the input count of the CLB.

The general outline of the algorithm is very similar to the end-case partitioner introduced in [51], and is provided in Figure 4.2. The method begins by assigning every BLE in the 2 CLBs to either one or the other, and checks for feasibility. If it is a feasible solution, then its cost is found and the method checks to see if the solution can be bound. Appropriate information is kept during the enumeration to improve both the absorption of edges as well as the absorption of connections deemed to be timing critical; various stacks are kept in order to be able to track the current edge absorption and criticality absorption.

Procedure: BRANCHANDBOUND
Inputs: Two CLBs $clus_i$ and $clus_j$
Returns: Two potentially better CLBs $clus'_i$ and $clus'_j$

```
1  Compute the current cost of clusters i and j
2  bestSoln = empty;
3  assignmentStack.add(clus_i.nodes() and clus_j.nodes());
4  for each k of 0 and 1 do
5    currentNode ← last node in assignmentStack;
6    if k = 0 then
7      assign currentNode to clus'_i
8    else
9      assign currentNode to clus'_j
10   fi
11   if arrangement is feasible then
12     check if it is boundable
13     if boundable then
14       pop currentNode from assignmentStack
15       if assignmentStack.empty() and k = 0 then
16         break
17       fi
18     else if all nodes have been assigned then
19         a complete solution that is the best one found so far
20         bestSoln ← current assignment
21       fi
22     else
23       the arrangement is infeasible, and is bound
24     fi
25   od
26   return clb'_i and clb'_j;
```

Figure 4.2: Pseudocode for Branch and Bound.

Chapter 5

Numerical Results

To measure the effectiveness of the algorithms implemented as per Chapter 3 and 4, several experiments were conducted. This chapter documents the many investigations carried out using the implemented clusterers, and is organized as follows. In Section 5.1, the method through which the algorithms are evaluated is described. This section also defines several key metrics that are used during comparison. Section 5.2 compares `DPack` and `HDPack` both with and without the use of physical information. In Section 5.3, the impact of the accuracy of physical information has on the quality of the routed designs is investigated. Then, `DPack` and `HDPack` are compared with other existing tools in Section 5.4. In Section 5.5, several concepts from other tools were integrated into these algorithms to see if any additional improvement can be achieved. Section 5.6 presents the results from the use of cluster improvement heuristics from Chapter 4.

5.1 Experimental Setup

To make a fair comparison between the implemented tools and other existing clustering algorithms, the twenty largest designs from the MCNC benchmark set were used [52]. The circuits in this set vary in size and circuit structure. The benchmark set has been used widely in academic research for FPGAs, and facilitates direct comparisons to be made to other known clustering tools.

5.1.1 General Experimental Flow

Before any comparisons can be made, a baseline must first be established. In the comparisons that follow, the baseline flow uses T-VPack, followed by VPR for placement and routing. Then, to compare the clustering algorithms outlined in Chapter 3, DPack and HDPack were employed to perform packing. The resultant netlists are then placed and routed by VPR. The three constructed flows can be summarized as follows: (1) the baseline flow of T-VPack+ VPR, (2) DPack + VPR, and (3) HDPack + VPR.

A good clustering algorithm should be able to perform well under a variety of situations and constraints. Therefore, the implemented clustering algorithms were tested on a set of FPGA architectures. The range of architectural sizes used corresponds to $N = \{2, 4, 8, 12\}$ BLEs per CLB. By using both large and small CLB sizes, it becomes possible to determine whether a particular clustering algorithm can perform well in all cases. For each architecture, the number of CLB inputs is calculated as $I = 2N + 2$. This was shown to yield good area efficiency by achieving an average of 98% logic utilization [26]. The grid size is set to the smallest square grid that can accommodate a particular design.

The results obtained from placement tools often vary from one run to another. In particular, the placement generated from VPR can change drastically in quality depending on the random number, or seed, used in the particular compilation. Sometimes, a 20% variance in the critical path delay can be observed. Therefore, to reduce this wide variance in results, each design is run 5 times using the same architecture and tool configurations with randomly generated values as the seeds used for placement and routing. The results obtained from these 5 runs are then averaged before they are compared to other flows.

One of the advantages of VPR is its flexibility, achieved by a long list of variable parameter settings. Of note, VPR's "timing tradeoff" was set to 0.5 for all tests. This indicates that while placing the design, the impact of changes on both wire length and timing are both considered as equally important. All wire length and critical path delays reported are obtained after routing. Segment 1 routing architecture was used for all designs; this indicates that in the routing channel,

all wires have lengths of 1 and can only form connections between neighboring CLBs.

In the rest of the chapter, the quality of placements are quantified and compared through the measure of several metrics. The *wire length* of a design refers to the total number of wire segments that the final routed design requires. Generally, the lower the wire length, the better, since less resources are needed. *External nets* refers to the number of nets that exist in the clustered netlist. Fewer external nets is preferred, since less inter-CLB wires are needed for the design. The area efficiency of a clustering algorithm is often measured by the number of CLBs in the clustered netlist. In this case, the fewer the number of CLBs, the better the algorithm. However, a reduction in one metric can sometimes mean an increase in another. Therefore, relevant metrics must be compared side-by-side to obtain the entire picture of whether or not an algorithm is beneficial.

5.1.2 Low-Stress Routing Setup

In FPGA research, there are generally 2 levels of routing test setups: low-stress, and high-stress. The purpose of low-stress routing experiments is to mimic a fixed architecture, and to evaluate the performance of benchmark designs in terms of wire length and timing.

To form a fair comparison, the minimum channel width for each design is found for a given grid size. Since the minimum channel width found for a particular design can vary depending on the random seed used, the search for the minimum channel width is performed 5 times and then averaged. The design is then routed again, with a channel width that is 20% greater than the minimum channel width found. Since the number of wires in the routing channel is much greater, the router needs to do less work to route the design. The router is then free to choose different wires and channels to optimize for wire length and critical delay. Thus, a fair comparison can be made by comparing the performance of designs routed at the same channel width and same grid size.

5.1.3 High-Stress Routing Setup

High-stress routing tests typically are used for architecture evaluation and they are particularly important during the process of designing an FPGA. During these tests, in addition to optimizing for timing and wire length, the tool also aims to optimize area. To minimize area, high-stress routing tests route for the smallest channel width possible. Since the routing resources on an FPGA take up a substantial portion of the chip, high-stress routing conditions involve trying to reduce the number of wires required per routing channel. Therefore, minimum channel widths can often be used as a metric of comparison between clustering tools.

5.2 Results for Implemented Algorithms

This section presents data comparisons between `DPack` and `HDPack` and the baseline of `T-VPack` in both low-stress and high-stress routing conditions.

5.2.1 Low-Stress Routing

The first set of experiments compares `DPack` and `HDPack` to `T-VPack` in low-stress routing conditions [16]. In the first experiment, physical information is not used in the cost function formulation. Since only edge absorption and timing information are employed, there is only one trade-off parameter in the cost functions of the packing tools. For `DPack`, a λ of 0.8 was found to yield the best results in terms of wire length and critical path delay. For `HDPack`, the best results were obtained using a λ of 0.9. The number of external nets after packing, and the final routed wire lengths and critical delays are shown in Table 5.1. The presented data is calculated by first normalizing the collected statistic against the baseline flow (`T-VPack`), and then averaged geometrically across all designs for the given architecture.

As shown in Table 5.1, both algorithms result in significantly better net absorption and wire length reduction than the baseline flow. A consistent reduction can be seen in the final wire lengths with a greater variability in critical delay reduction. For `DPack`, the wire length reduction ranges

from 6.9% to 10.8%. However, the critical delay improvement does not seem to be very substantial, ranging from 0% to 3.7% reduction. It also appears that implementation of the Kong path counting algorithm during clustering did not improve critical delay significantly. For HDPack, the wire length reduction varies from 12.6% to 17.5%, with an average of 15% across all architectures. There is also no significant reduction in critical delay.

A clear difference between DPack and HDPack can be seen in terms of the wire length improvement. DPack achieves an average of 9% improvement in wire length, whereas HDPack is capable of 15% improvement. This can be directly attributed to the HFCC algorithm present in HDPack. Since the HFCC method employed in HDPack pairs BLEs that share the highest affinities, it is able to make the best decisions early on, and is "unconcerned" with packing CLBs fully. In contrast, DPack is limited in the sense that it must complete one CLB before moving on to another. It is possible that in this process, some BLEs that are packed may have been better off packed with other, still unpacked, BLEs.

Another important observation that can be made from Table 5.1 is the trend in wire length reduction. There is no visible trend in DPack. This is expected, since DPack is very similar to T-VPack. However, HDPack show an increase in wire length reduction as the architecture size increases. This illustrates the key benefit of the affinity-based HFCC algorithm. As the architecture size increases, DPack may be forced to pack BLEs that have a much lower gain with the current cluster while packing for minimum area. Therefore, the gap in wire length reduction between the HFCC and the other flows is expected to increase as the architecture size is increased.

Table 5.1: Packing without physical information.

	DPack				**HDPack**			
N	**# CLB**	**Ext Nets**	**WL**	**Crit**	**# CLB**	**Ext Nets**	**WL**	**Crit**
2	1.003	0.966	0.902	0.963	1.020	0.948	0.873	0.937
4	0.997	0.928	0.892	0.985	1.000	0.858	0.874	1.007
8	0.998	0.911	0.900	0.999	0.998	0.832	0.847	0.984
12	1.002	0.937	0.931	0.986	0.998	0.844	0.825	1.013
Geomean	**1.00**	**0.94**	**0.91**	**0.98**	**1.00**	**0.87**	**0.85**	**0.98**

In the second experiment, physical information was used during packing. As described in Chapter 3, an additional weighting factor in the cost function was added to control the importance of physical information during packing. Since there are now two independent weighting factors (c.f., Section 3.3), a two-dimensional sweep was performed to find the best configuration. For DPack, the best results were obtained using $\lambda = 0.2$ and $\gamma = 0.4$, leaving the physical information weight to be 0.4. For HDPack, the best configuration was found with $\lambda = 0.2$, $\gamma = 0.2$, and a physical weight of 0.6.

Results using physical information are shown in Table 5.2. With physical information, DPack was able to achieve significant reductions in wire length and critical path delay, with an average improvement of 16% and 8%, respectively. This represents an improvement of up to 10% in wire length and up to 8% in critical path delay compared to DPack without physical information. Significant improvements for HDPack are also evident, with an average improvement of 21% and 6% for wire length and critical delay, respectively. Contrasted with Table 5.1, improvements up to 8% in wire length and up to 5% in critical delay can be seen when compared to HDPack without physical information.

The average run-times were computed for all five runs of T-VPack-based clustering, placement, and routing for all twenty design runs for each architecture, and similarly for DPack and HDPack. The run-time ratios of the DPack-based and HDPack-based flows were computed and compared to T-VPack. These results, both with and without physical information, are summarized in Table 5.3. Generally, the use of physical information incurred negligible run-time penalties in the context of

Table 5.2: Packing with physical information.

	DPack				**HDPack**			
N	**# CLB**	**Ext Nets**	**WL**	**Crit**	**# CLB**	**Ext Nets**	**WL**	**Crit**
2	1.012	0.962	0.862	0.900	1.034	0.966	0.846	0.915
4	1.006	0.937	0.834	0.920	1.017	0.900	0.804	0.960
8	1.007	0.908	0.823	0.922	1.012	0.873	0.768	0.939
12	1.012	0.942	0.834	0.937	1.022	0.864	0.763	0.963
Geomean	**1.01**	**0.94**	**0.84**	**0.92**	**1.02**	**0.90**	**0.79**	**0.94**

the entire place-and-route run-time for most architectures. However, for the case of $N = 12$, the MCNC benchmarks that were considered were clustered into such small netlists that placement and routing time approached that of the packing time. Consequently, these results tend to show more variability, which may not be indicative of performance on much larger, real-world designs.

5.2.2 High-Stress Routing

A high-stress routing test was conducted to determine the minimum channel widths required for each design. The search for minimum channel width was performed 5 times for each design for all architectures under consideration. The average channel width was computed for each case and then normalized to the minimum channel width found by the baseline flow. Physical information was enabled during these tests. The channel width improvement relative to `T-VPack` is shown in Table 5.4. Both `DPack` and `HDPack` were extremely successful in reducing minimum channel widths, with 19% and 24% improvement on averaged, when compared against `T-VPack`.

5.3 How Much Physical Information is Enough?

Even though partitioning algorithms are fast, they still incur some penalty in terms of run-time. However, it is possible that after some point in the initial partitioning, further partitioning would not give much wire length and critical delay reductions. If this point can be quantified and found, there would be no need to incur the additional run-time penalty. Since the physical information obtained

Table 5.3: Run-time comparison vs. baseline.

	DPack		**HDPack**	
N	**No Physical**	**Physical**	**No Physical**	**Physical**
2	0.959	0.965	0.991	0.985
4	0.974	0.985	0.990	0.985
8	1.070	1.074	1.027	1.024
12	1.288	1.255	1.150	1.130
Geomean	**1.065**	**1.064**	**1.038**	**1.029**

Table 5.4: Improvement in minimum channel widths.

N	DPack	HDPack
2	0.805	0.764
4	0.808	0.776
8	0.760	0.755
12	0.854	0.756
Geomean	**0.81**	**0.76**

in the clustering algorithms in this work is obtained through performing recursive partitioning, it is easy to control how far to partition the design. Therefore, a test was set up to determine the optimal tree depth of recursive partitioning that leads to the best wire length and delay trade-offs.

The initial partitioning algorithm was set as follows; the algorithm terminates when either (a) all end partitions were of a specified partition depth *or* (b) when partitions contained less than a set number of cells in the primitive netlist. The partition depth was varied from 0 (where no partitioning is performed at all) to 14, for each of the 20 designs in the benchmark suite. This test was conducted across four architecture sizes of $N = 2, 4, 8, 12$. Wire length improvement results are shown in Figure 5.1 and critical delay reductions are shown in Figure 5.2.

From the two graphs, a dramatic initial reduction in both circuit metrics as partition depth is increased can be seen. Wire length improvement is greatest at a partition depth of 5, beyond which the average wire length reduction increases only slightly before flattening out. The trend for critical delay reduction is less apparent. The best critical delay improvement occurs with partition depth of 2 or 5. Even though the result for the partition depth of 2 is slightly greater, the wire length reduction at this point is not ideal. For almost all architectures, a partition depth of 5 yields the best overall wire length and critical delay reduction. Additional partitioning is unnecessary, and may even be detrimental to the quality of results.

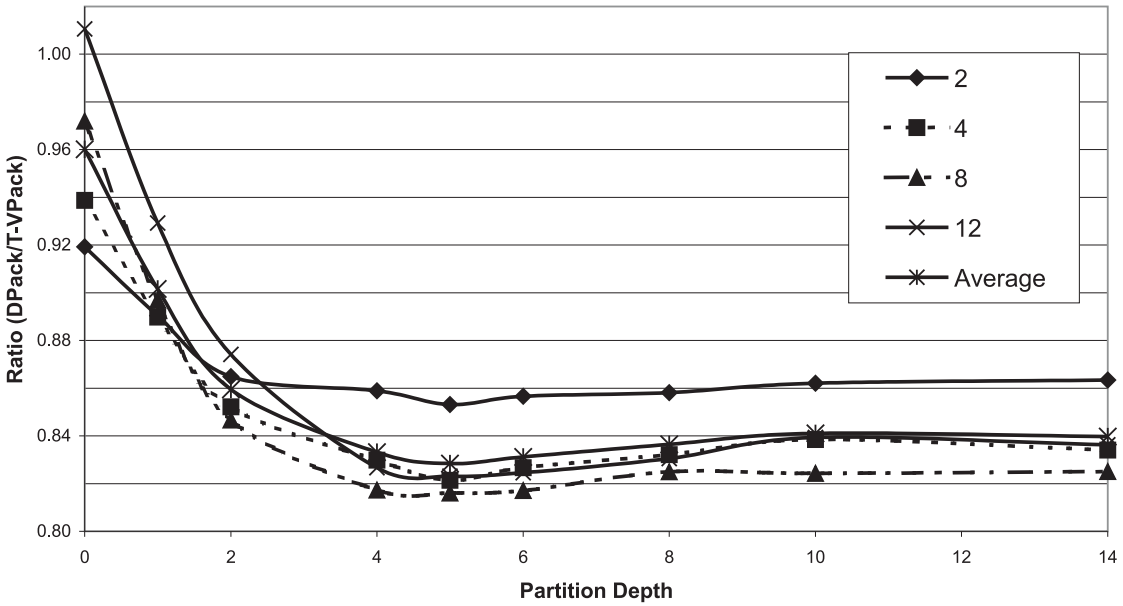

Figure 5.1: Wire Length Reduction vs. Partition Depth.

5.4 Comparison to Other Methods

To see how the implemented algorithms compare to other existing tools, cluster statistics were compared against `T-VPack`, `RPack`, and `iRAC`. It should be noted that `RPack` and `iRAC` were primarily geared toward addressing routability; neither of these tools dealt with timing (as the implemented algorithms do), which skews results against the algorithms outlined in this thesis.

5.4.1 Low-Stress Routing Tests

To compare the performance of all algorithms under low-stress routing conditions, the number of CLBs, number of nets in the CLB-level netlist, and the average number of pins used per CLB for the $N = 8$ case were obtained for each tool, and are shown in Table 5.5. It should be noted that comparisons with other architecture sizes are omitted since results are only available for comparison with `iRAC` and `RPack` at $N = 8$. All results are normalized with respect to `T-VPack`. Since a lower number of nets and lower pin usage are properties usually associated with less wire length used and better routability of the clustered design, `iRAC` was found to give the best

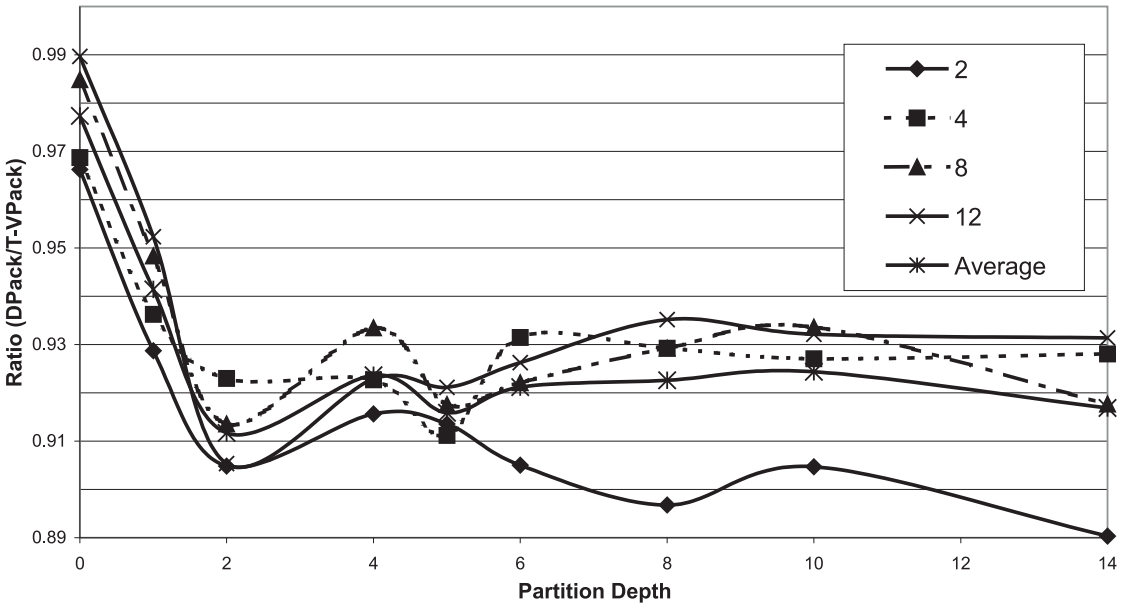

Figure 5.2: Critical Delay Reduction vs. Partition Depth.

packing results, with the lowest number of nets and average used-pins-per-cluster. However, this was achieved at the cost of significantly more CLBs. The next best clustering results were found to be from HDPack.

It is important to make the comparison since from [53], it can be seen that depopulating clusters help routing at the expense of increase in area. Therefore, it is critical to make sure that the improvement seen from the use of physical information is not a manifestation of depopulation of clusters. From the results presented in Table 5.5, the improvement seen from the use of physical information is *not* a manifestation of depopulated CLBs. Table 5.5 indicates that there is *very little* increase in the number of clusters made by DPack or HDPack. Although iRAC was able to achieve 24% reduction in the number of nets, it came at a cost of a 8% increase in cluster count. In contrast, DPack achieved 9% reduction in nets without impacting the number of clusters made. Although an increase of 2.5% was observed in the average number of pins used per CLB in DPack, which may lead to more difficulty in routing [54], this was not found to be a significant issue during testing.

Results from Table 5.5 are also plotted in Figure 5.3 to compare the 5 clustering tools in a

graphical manner. For each of the 5 tools compared, the average pin usage is plotted against the external net improvement. A lower number of nets and lower pin usage are characteristics usually associated with less wire length used and better routability of the clustered design. Therefore, the clustering tool that is closest to the lower left corner of the graph is expected to have the best performance. From Figure 5.3, iRAC is found to give the best clustering results, with the lowest number of nets and average used pins per cluster. However, this is achieved with the consequence of an increase in the number of clusters made. The next best clustering results are found with HDPack.

5.4.2 High-Stress Routing Tests

For high-stress routing tests, the minimum channel width improvements of DPack and HDPack are compared to several known clustering tools. The improvement in minimum channel widths were compared to RPack [15] as follows. For the $N = 8$ architecture, RPack cites a 16.5% improvement in minimum channel width versus VPack (c.f., [15], Table 3). In [14], however, it is shown that RPack does not provide any improvement versus T-VPack (c.f., [14], Table 2), where it is also pointed out that T-VPack provides better results than its non-timing-driven counterpart VPack. Given that, for $N = 8$, DPack and HDPack yield improvements of 24% and 24.5%, respectively, compared to T-VPack as seen from Table 5.4. It was concluded that these results here outperform RPack even though minimum channel widths were not considered as an objective in the clustering algorithm.

Table 5.5: Comparison between known tools, $N = 8$, $I = 18$.

Packer	# CLB	Ext Nets	Pins Used
T-VPack	1.000	1.000	1.000
R-Pack	1.009	1.071	0.954
iRAC	1.078	0.757	0.870
DPack	1.007	0.908	1.025
HDPack	1.014	0.870	0.961

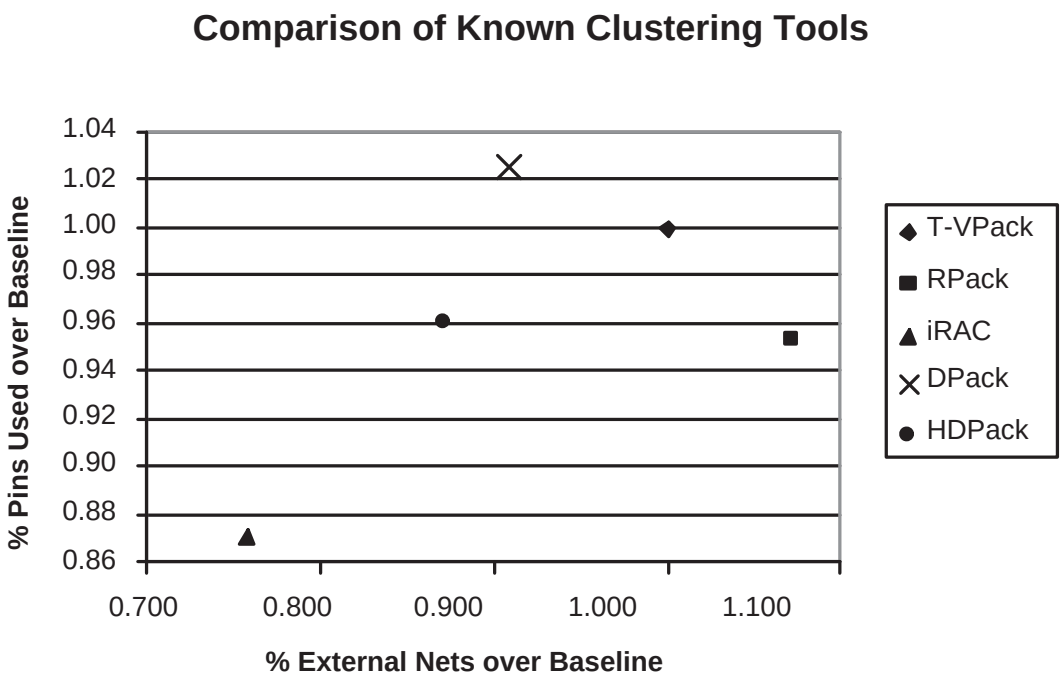

Figure 5.3: Effect of Depopulation: N=8

Both RPack and iRAC report results for architectures of size $N = 8$. The grid sizes used in the results presented in Table 5.4 are the same as those reported by RPack [15]; therefore, the minimum channel widths found for $N = 8$ can be compared directly to those in Table 3 of [15], where a 16.5% improvement (over T-VPack) is reported, compared to HDPack's 24.5% improvement. It should be noted that HDPack's improvement was obtained using the timing-driven flow without specifically attempting to optimize for wire length, congestion, or minimum channel widths.

The performance of DPack and HDPack are also compared to other tools such as iRAC [14]. However, a comparison with iRAC [14] is harder to make than the comparison to T-VPack. Since iRAC produces more CLBs compared to other packing methods, the results in [14] use a different grid size and VPR "IO_RAT" value, a parameter that dictates the number of IO blocks that fit in the width of a CLB. Although an attempt was made to reproduce the T-VPack results presented in [14], it was unsuccessful. Furthermore, the results presented in [14] for minimum channel width

experiments were also obtained in combination with a modified version of VPR—called iRAP—that includes a congestion term in the placement algorithm's objective function. It is reasonable to expect that the modified placement algorithm *also* served to reduce channel widths. Nevertheless, the results of 24% and 24.5% reduction in channel widths for DPack and HDPack, respectively, compare favorably to the 38% reduction obtained by iRAC+iRAP algorithm. It is possible that if this test was performed using a congestion-driven placer, the gap may be reduced even further.

5.5 Integration of RPack and iRAC

Although numerical results thus far indicate that DPack and HDPack outperform the baseline, it is desirable to find additional concepts that would improve upon the results. Therefore, several concepts from RPack and iRAC were integrated into DPack in an effort to assess their potential benefits (similar results were found for HDPack). The incorporation of RPack was straightforward since it consisted of adding a new term to the cost function. To determine the optimal balance between the new RPack term, the RPack term was multiplied by ζ, with the cost calculated as per Equation 3.3 multiplied by $1 - \zeta$. The wire length and critical delay improvements as ζ is varied are shown in Figure 5.4. The best value for ζ is approximately 0.20. At this point, the inclusion of RPack improved the critical delay and wire length by 1% and 2%, respectively. However, this may not be statistically meaningful. As ζ tends to 1.0 and RPack dominates the packing objective, both wire length and critical path improvements suffer. In addition, a high-stress test was performed with the inclusion of the RPack term, but the minimum channel width did not benefit, as the improvement over T-VPack decreased by 1% to 2%.

The incorporation of iRAC algorithms into the tools was less successful, and did not improve upon the current best results. iRAC depopulates by limiting the number of pins used per CLB, as well as by trying to absorb low-fanout nets. A test was conducted to establish the effect that a decrease in the number of edges and in pin utilization has on the resulting packing statistics. As shown in Figure 5.5, as the number of pins used per CLB is decreased, the number of CLBs generated increases. At the same time, the number of external nets also decreases. However, the

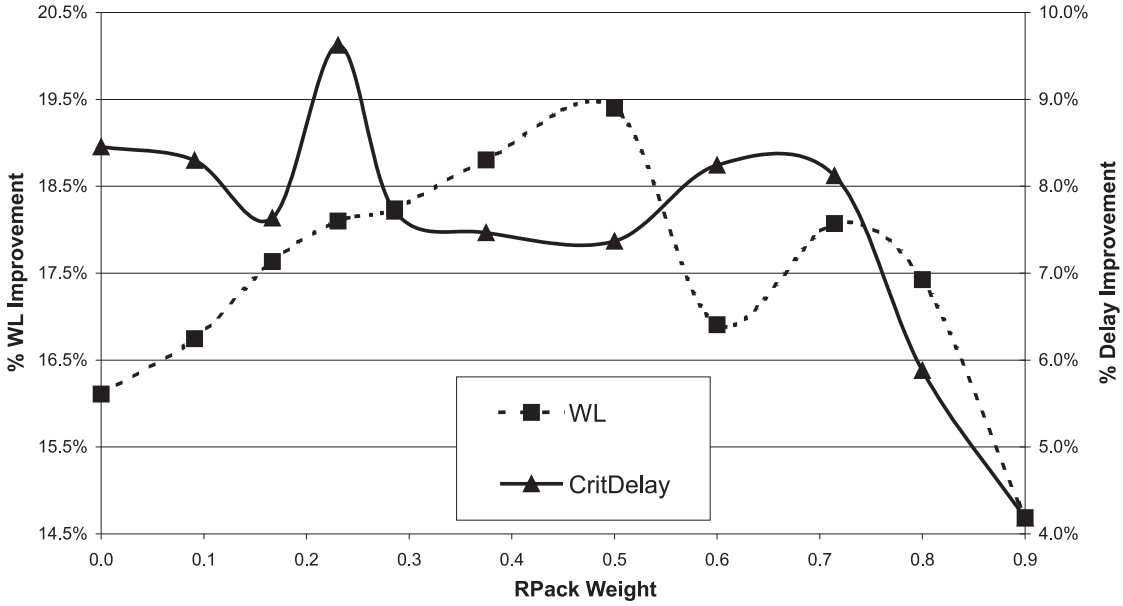

Figure 5.4: Incorporation of RPack Sweep: N=8

wire length and critical delay do not exhibit a similar trend in that they remain fairly consistent (even increasing slightly as the pin counts are reduced). This indicates that, even though a packing algorithm can show good results from a *packing* point of view (i.e., good external net and pin count reduction), the impact on the final wire length and critical delay may not show the same trend. Since performance of the placed design is the ultimate objective, it may not be sufficient to merely compare packing statistics.

5.6 Effectiveness of Improvement Algorithms

This section seeks to determine the effectiveness of the improvement heuristics proposed in Chapter 4. To do so, `T-VPack`, `DPack` and `HDPack` are executed in conjunction with the proposed improvement algorithms. Four different flows were constructed for each clusterer: (1) the baseline flow of `T-VPack`+`VPR`, (2) `T-VPack`+greedy swapping+`VPR`, (3) `T-VPack`+branch-and-bound+`VPR`, and (4) a combined flow in which both greedy swapping and branch-and-bound are used; i.e., `T-VPack`+greedy swapping+branch-and-bound+`VPR`. The same setup is then used by substituting

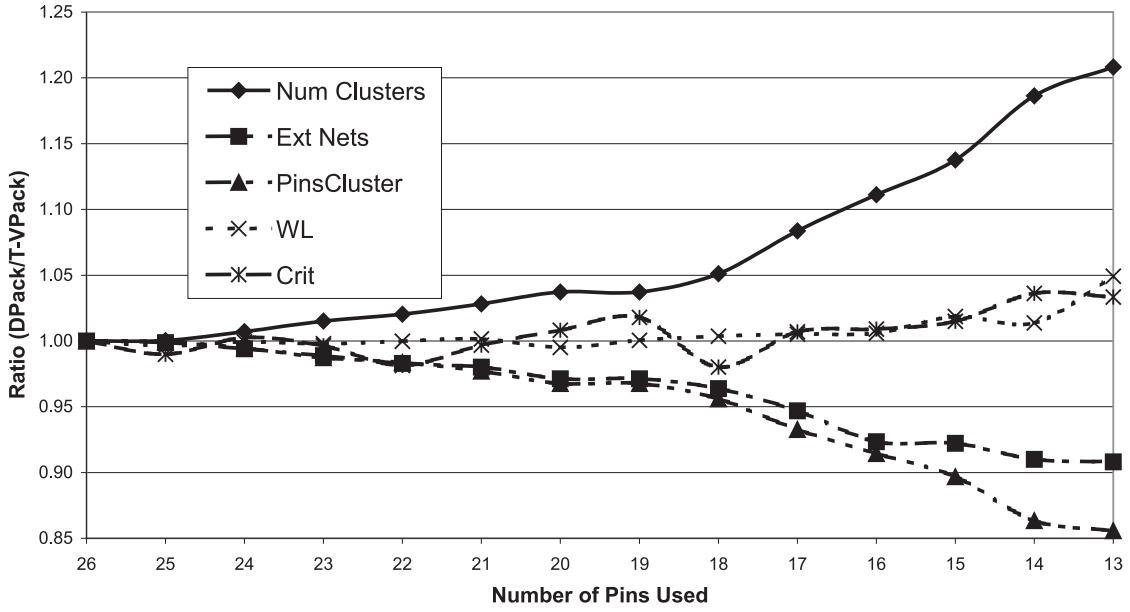

Figure 5.5: Effect of Reducing Pin Count: N=8

DPack or HDPack for T-VPack. The results for T-VPack are collected from all architectures and are tabulated in Table 5.6. For each architecture and design, the collected data was normalized with respect to the baseline. The geometric average was then computed across all designs for the given architecture. Results for DPack and HDPack are shown in Tables 5.7 and 5.8, respectively. These improvement heuristics did not appear to affect the number of CLBs significantly, and therefore this metric is left out of these tables for the sake of clarity.

From Table 5.6, both the Greedy and B&B algorithms were successful in reducing the number of external nets and wire length for T-VPack. Using both algorithms appears to be slightly more successful. This reduction ranged from 1.5% for the Greedy flow, 1.6% for the B&B flow, and 2.5% on average in the combined flow. It can be seen that the B&B flow is slightly more effective at reducing the number of edges than the Greedy flow, but the inclusion of the greedy swapping algorithm is still beneficial. In terms of wire length, the B&B flow provides greater reduction than the greedy heuristic. It should be noted that the improvements from the two heuristics are not cumulative.

When both heuristics are performed, up to a 4.9% reduction in wire length can be seen. However, this reduction seems to decrease as the cluster size is increased. It was found that although there was no significant reduction in the number of clusters, there is a consistent reduction across most circuits when packed for a cluster size of 2. Intuitively this makes sense, since it is much easier to empty out a cluster of 2, than it is to be able to relocate all used BLEs inside a much bigger cluster. Due to this reduction in number of clusters, a significant reduction in wire length was seen. However, although the algorithms were able to reduce the number of inter-CLB edges and wire length (to a small extent), critical paths were not significantly affected (i.e., it was not significantly reduced or increased) on average.

Table 5.7 shows the results obtained from using the improvement heuristics after DPack. From the data presented, the number of external nets shows a significant drop of 3% when both heuristics are used. Although this is good news, the final routed wire length did *not* decrease, as one might have expected. Generally speaking, fewer nets in the circuit netlist usually translate into a reduction in wire length. However, this is not witnessed in this case. While the number of external nets dropped by 3%, the overall wire length actually *increased* by an average of 5.3%, with the critical delay worsening by 3%. This same observation can be made in the case of HDPack, from Table 5.8. In the case of HDPack, external nets decreased by an average of 3.7% when both improvement heuristics are used. However, wire lengths also increased in this case by 2.5%. There may be a few explanations for this phenomenon. First, as proved from previous sections, DPack and HDPack perform better clustering than T-VPack. Therefore, it can be argued that the set of initial clusters

Table 5.6: Effect of Improvement Algorithms on T-VPack.

Arch	Greedy			B&B			Both		
N	Ext Net	WL	Crit	Ext Net	WL	Crit	Ext Net	WL	Crit
2	0.980	0.971	0.985	0.986	0.979	0.956	0.973	0.951	0.987
4	0.983	0.996	0.987	0.978	0.977	0.997	0.966	0.975	0.977
8	0.989	1.007	1.017	0.982	0.985	0.997	0.975	0.996	1.009
12	0.988	0.993	1.003	0.990	0.990	0.998	0.980	0.982	1.031
Geomean	**0.985**	**0.992**	**0.998**	**0.984**	**0.982**	**0.987**	**0.974**	**0.975**	**1.000**

provided by DPack and HDPack are already very good, and further manipulation of the cluster set would likely lead to a worse set of clusters. Secondly, the improvement heuristic operates only on connectivity and criticality, whereas both DPack and HDPack uses physical information while clustering. Since the improvement heuristics are unaware of physical information, it may make some decisions that is counter to the actions made during initial clustering. The data presented here is a perfect example of why examination and comparison of cluster statistics is not enough; to make a fair judgement, the final routed wire lengths and critical delays must be compared. In this case, if only cluster statistics are compared, it would seem that the improvement heuristics provide significant benefit since up to 5% reduction in the number of external nets can be witnessed. However, the comparison of the final wire lengths lead to the conclusion that it is best not to use the cluster improvement heuristics in conjunction with DPack and HDPack; that is, improvement heuristics are most likely going to worsen the quality of the final placement.

There is a number of possible explanations why the proposed cluster improvement heuristics did not have as significant a benefit. First, the MCNC benchmark designs are very small compared to actual industrial designs. Therefore, it is possible that a more significant improvement may be visible on a benchmark set of larger designs. Also, some of the comparisons differ in only a few percentage points, and may be attributed to noise. Thus, further investigation should be performed to see if these algorithms may be enhanced further.

Table 5.7: Effect of Improvement Algorithms on DPack

Arch	Greedy			B&B			Both		
N	Ext Net	WL	Crit	Ext Net	WL	Crit	Ext Net	WL	Crit
2	0.986	1.046	1.032	0.992	1.010	1.026	0.982	1.034	1.031
4	0.979	1.066	1.035	0.974	1.042	1.025	0.962	1.065	1.046
8	0.983	1.050	1.004	0.974	1.020	1.003	0.964	1.061	1.022
12	0.984	1.037	1.019	0.980	1.025	1.024	0.971	1.050	1.021
Geomean	**0.983**	**1.050**	**1.023**	**0.980**	**1.024**	**1.019**	**0.970**	**1.053**	**1.030**

Table 5.8: Effect of Improvement Algorithms on HDPack

Arch	Greedy			B&B			Both		
N	Ext Net	WL	Crit	Ext Net	WL	Crit	Ext Net	WL	Crit
2	0.976	1.025	1.019	0.984	1.010	0.996	0.969	1.018	1.033
4	0.970	1.039	0.991	0.966	1.016	0.990	0.950	1.037	1.003
8	0.981	1.028	0.998	0.971	1.014	1.005	0.962	1.036	1.004
12	0.985	1.009	0.987	0.983	0.999	0.999	0.973	1.009	0.999
Geomean	**0.978**	**1.025**	**0.999**	**0.976**	**1.010**	**0.998**	**0.963**	**1.025**	**1.009**

Chapter 6

Conclusions

In this thesis, the clustering phase of the FPGA CAD flow was explored in detail. First, several clustering algorithms were implemented within a common framework to facilitate comparisons. Then, physical information was added to these clustering algorithms to see if better clusters could be made. To obtain physical information prior to clustering, a min-cut partitioning algorithm was performed such that approximate physical locations could be generated for BLEs. Then, the obtained physical information was incorporated into two types of clustering algorithms to evaluate the advantage(s) of the additional circuit information. The focus was not to obtain architecturally correct placements, but rather to obtain reasonable physical information with little effort.

The flow described employs top-down min-cut partitioning-based placement prior to clustering to generate rough physical locations for BLEs. This physical information was then incorporated into two clustering algorithms of `DPack` and `HDPack`. `DPack` is a seed-based algorithm that is similar to `T-VPack`. It was implemented with a few modifications made to it from previous work. `HDPack` was then created by using ASIC methods along with `DPack`. Both tools were then enhanced with physical information. By using the approximate physical locations and relative positions of BLEs, it is hoped that better clusters can be made, and a better final placement can be achieved.

From the results presented in Chapter 5, `DPack` yielded an average reduction of 16% in wire length, and 8% in critical path delay compared to `T-VPack`. Similarly, `HDPack` resulted in an

average improvement of 21% in wire length and 6% in critical delay compared to T-VPack. Under high-stress conditions, significant improvements were seen in the minimum channel widths required by the benchmark set, ranging from 19% to 24% reduction. Neither DPack nor HDPack required a significant increase in the number of CLBs. Although an increase in the average number of used pins per CLB was observed, it did not appear to impact routability in the tests. Therefore, it was concluded that the use of physical information during FPGA packing can improve the final quality of results and reduce the need for BLE-level placement. Although a slight increase in the average number of used pins per CLB was observed, it did not impact routability in the testing process.

As a part of the study, several experiments were conducted in order to determine how accurate the generated physical information needs to be before it has a positive impact on the quality of routed designs. It was found that the wire length and critical delay of routed designs decrease very quickly as the number of partitions increases, but levels off, and slightly increases as the circuits are partitioned more finely. A good partition depth to stop at is found to be 5. By partitioning to a depth of 5, the greatest improvements in wire length and critical delay can be achieved. Therefore, although physical information does aid in better clusters, it does not need to be very accurate. Hence, only approximate locations are necessary for producing better CLBs. Furthermore, more accurate locations may be detrimental to the quality of results.

Several comparisons were made between the implemented algorithms and existing tools such as T-VPack, RPack, and iRAC. It was found that DPack and HDPack outperform T-VPack and RPack, but do not improve upon the results obtained from iRAC. However, it should be noted that iRAC was able to achieve better external net absorption at the expense of a higher number of clusters. Therefore, it is less area-efficient when compared to DPack and HDPack. Some key ideas in RPack and iRAC were added to DPack to determine whether they can improve the current best results. It was found that the addition of a routability term, found in RPack, did result in $1-2\%$ improvement in the key metrics; however, no such improvement was found when iRAC concepts were added.

In addition to the proposed clustering algorithms, two cluster improvement heuristics were explored in this thesis. The Swaps and Moves algorithm randomly swaps BLEs between two CLBs and accepts the rearrangement if both absorption and criticality costs improve. The Branch-and-Bound algorithm uses enumeration with the hope that more complicated rearrangements (i.e., more than just moves or swaps of BLEs) can lead to a greater improvement in CLBs.

From the results presented in Chapter 5, the improvement heuristics demonstrated a reasonable reduction (on the order of a few percent) in the number of inter-CLB edges. This reduction is visible despite the fact that no depopulation was used and the number of CLBs remained mostly the same. This might be useful in highly utilized designs in which increasing the CLB count is not possible. The decrease in inter-CLB edges results in a small decrease in wire length in the case of `T-Pack`. Unfortunately, the critical paths remain largely unaffected on average. However, although the number of external nets did indeed reduce, the final routed wire lengths and critical delay were found to be *worse* for both `DPack` and `HDPack`. Hence, it is concluded that the heuristics employed were able to improve upon clusters made by `T-VPack`, but not on the cluster set generated by `DPack` and `HDPack`.

Chapter 7

Future Work

From the obtained results, substantial gains can be found in terms of reduction in wire length, but the reduction in critical delay is less significant. Future work may be to integrate concepts from depth-optimal methods into `DPack` and `HDPack` to see if these methods can further improve the quality of results. It is also possible that in conjunction with a congestion placer, the results can be further improved upon.

In each clustering algorithm presented, a clustered netlist is generated after clustering is performed with physical information. Each CLB in the clustered netlist has an x and y location computed from averaging the locations for all BLEs it contains. It is possible to generate a placement based on this information, using legalization techniques [55]. This initial placement may in turn help the placement process, and may be a better start to the placement step of the FPGA flow than a random placement. This has the potential of leading to a better quality of final placements, or the ability to achieve the same quality in a shorter timeframe.

Bibliography

[1] Altera Corporation, San Jose, CA, *Cyclone II Device Handbook, Volume 1*, June 2006. http://www.altera.com.

[2] Altera Corporation, San Jose, CA, *Stratix III Device Handbook, Volume 1*, November 2006. http://www.altera.com.

[3] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Laboratory, University of California, Berkeley, 1992.

[4] M. Gao, J. Jiang, Y. Jiang, Y. Li, S. Singha, and R. K. Brayton, "MVSIS," in *Proceedings of the International Workshop of Logic Synthesis*, 2001.

[5] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*, Release 61225. http://www.eecs.berkeley.edu/~alanmi/abc.

[6] R. J. Francis, J. Rose, and K. Chung, "Chortle: a Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays," *Proceedings of the Design Automation Conference*, pp. 613–619, 1990.

[7] J. Cong and Y. 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*, vol. 13, no. 1, pp. 1–12, January 1994.

[8] J. Cong and Y.-Y. Hwang, "Simultaneous Depth and Area Minimization in LUT-based FPGA Mapping," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 68–74, 1995.

[9] A. Lu, E. Dagless, and J. Saul, "DART: Delay and Routability Driven Technology Mapping for LUT based FPGAs," *Proceedings of the International Conference on Computer Design*, pp. 409–414, 1995.

[10] A. R. Naseer, M. Balakrishnan, and A. Kumar, "Direct Mapping of RTL Structures Onto LUT-based FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 7, pp. 624–631, July 1998.

[11] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, no. 11, pp. 2331–2340, 2006.

[12] D. Chen and J. Cong, "DAOmap: a Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pp. 752–759, 2004.

[13] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *Field-Programmable Logic and Applications* (W. Luk, P. Y. Cheung, and M. Glesner, eds.), pp. 213–222, 1997.

[14] A. Singh and M. Marek-Sadowska, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 59–66, 2002.

[15] E. Bozorgzadeh, S. Ogrenci-Memik, and M. Sarrafzadeh, "RPack: Routability-Driven Packing for Cluster-based FPGAs," in *Proceedings of the Asia South Pacific Design Automation Conference*, pp. 629–634, 2001.

[16] V. Betz, J. Rose, and A. Marquardt, eds., *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[17] P. Maidee, C. Ababei, and K. Bazargan, "Fast Timing-Driven Partitioning-Based Placement for Island Style FPGAs," in *Proceedings of the Design Automation Conference*, pp. 598–603, 2003.

[18] P. Banerjee, S. Bhattacharjee, S. Sur-Kolay, S. Das, and S. C. Nandy, "Fast FPGA Placement Using Space-Filling Curve," in *Proceedings of the International Conference on Field Programmable Logic and its Applications*, pp. 415–420, 2005.

[19] M. Hutton, K. Adibsamii, and A. Leaver, "Timing-Driven Placement for Hierarchical Programmable Logic Devices," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 3–11, 2001.

[20] N. Selvakkumaran, A. Ranjan, S. Raje, and G. Karypis, "Multi-Resource Aware Partitioning Algorithms for FPGAs with Heterogeneous Resources," in *Proceedings of the Design Automation Conference*, pp. 741–746, 2004.

[21] G. Lemieux and S. Brown, "A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays," in *Proceedings of the ACM Physical Design Workshop*, pp. 215–226, 1993.

[22] S. Lee, Y. Cheon, and M. Wong, "A Min-Cost Flow Based Detailed Router for FPGAs," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pp. 388–393, 2003.

[23] L. McMurchie and C. Ebeling, "PathFinder: a Negotiation-Based Performance-Driven Router for FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 111–117, 1995.

[24] J. S. Swartz, V. Betz, and J. Rose, "A Fast Routability-Driven Router for FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 140–149, 1998.

[25] S. Wilton, "A Crosstalk-Aware Timing-Driven Router for FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 21–28, 2001.

[26] V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 551–554, 1997.

[27] A. Marquardt, V. Betz, and J. Rose, "Speed and Area Tradeoffs in Cluster-based FPGA Architectures," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 84–93, 2000.

[28] J. Cong and M. Romesis, "Performance-Driven Multi-Level Clustering with Application to Hierarchical FPGA Mapping," in *Proceedings of the Design Automation Conference*, pp. 389–394, 2001.

[29] C. Sze, T.-C. Wang, and L.-C. Wang, "Multilevel Circuit Clustering for Delay Minimization," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1073–1085, 2004.

[30] M. Dehkordi and S. Brown, "Performance-Driven Recursive Multi-Level Clustering," in *Proceedings of the IEEE International Conference on Field-Programmable Technology*, pp. 262–269, 2003.

[31] Altera Corporation, San Jose, CA, *APEX 20K Programmable Logic Device Family Data Sheet*, Mar 2004. `http://www.altera.com`.

[32] R. Rajaraman and D. F. Wong, "Optimal Clustering for Delay Minimization," in *Proceedings of the Design Automation Conference*, pp. 309–314, 1993.

[33] V. Manohararajah, G. R. Chiu, D. P. Singh, and S. D. Brown, "Difficulty of Predicting Interconnect Delay in a Timing Driven FPGA CAD Flow," in *Proceedings of the Workshop on System Level Interconnect Prediction*, pp. 3–8, 2006.

[34] Y. Wei and C. Cheng, "Ratio Cut Partitioning for Hierarchical Designs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 911–921, 1991.

[35] J. Li, J. Lillis, L.-T. Liu, and C.-K. Cheng, "New Spectral Linear Placement and Clustering Approach," in *Proceedings of the Design Automation Conference*, pp. 88–93, 1996.

[36] J. Cong, H. Li, and C. Wu, "Simultaneous Circuit Partitioning/Clustering with Retiming for Performance Optimization," in *Proceedings of the Design Automation Conference*, pp. 460–465, 1999.

[37] J. Cong and S. Lim, "Edge Separability Based Circuit Clustering with Application to Circuit Partitioning," in *Proceedings of the Asia South Pacific Design Automation Conference*, pp. 429–434, 2000.

[38] C. Alpert, A. Kahng, G.-J. Nam, S. Reda, and P. Villarrubia, "A Semi-Persistent Clustering Technique for VLSI Circuit Placement," in *Proceedings of the ACM International Symposium on Physical Design*, pp. 200–207, 2005.

[39] G. Karypis and V. Kumar, "Multilevel k-Way Hypergraph Partitioning," in *Proceedings of the Design Automation Conference*, pp. 343–348, 1999.

[40] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, March 1999.

[41] K. Vorwerk and A. Kennings, "An Improved Multi-Level Framework for Force-Directed Placement," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 902–907, 2005.

[42] G. Karypis, *Multilevel Optimization and VLSI CAD*, ch. 3. Boston: Kluwer Academic Publishers, 2002.

[43] G. Chen and J. Cong, "Simultaneous Timing Driven Clustering and Placement for FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and its Applications*, pp. 158–167, 2004.

[44] T. Kong, "A Novel Net Weighting Algorithm for Timing-Driven Placement," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pp. 172–176, 2002.

[45] V. Manohararajah, D. P. Singh, S. D. Brown, and Z. G. Vranesic, "Post-Placement Functional Decomposition for FPGAs," *Proceedings of the International Workshop of Logic Synthesis*, pp. 114–118, 2004.

[46] G. Beraudo and J. Lillis, "Timing Optimization of FPGA Placements by Logic Replication," in *Proceedings of the Design Automation Conference*, pp. 196–201, 2003.

[47] W.-J. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 3, pp. 349–359, March 1995.

[48] G. Karypis and V. Kumar, "hMeTiS: A Hypergraph Partitioning Package," tech. rep., Department of Computer Science and Engineering, University of Minnesota, Minneapolis, 1998.

[49] J. A. Roy *et al.*, "Capo: Robust and Scalable Open-Source Min-Cut Floorplacer," in *Proceedings of the ACM International Symposium on Physical Design*, pp. 224–226, 2005.

[50] A. B. Kahng and S. Reda, "Placement Feedback: A Concept and Method for Better Min-Cut Placements," in *Proceedings of the Design Automation Conference*, pp. 357–362, 2004.

[51] A. Caldwell, A. B. Kahng, and I. L. Markov, "Optimal partitioners and end-case placers for standard-cell layout," in *Proceedings of the ACM International Symposium on Physical Design*, pp. 90–96, 1999.

[52] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," tech. rep., Microelectronics Center of North Carolina, 1991.

[53] M. Tom and G. Lemieux, "Logic Block Clustering of Large Designs for Channel-Width Constrained FPGAs," in *Proceedings of the Design Automation Conference*, pp. 726–731, 2005.

[54] R. Tessier and H. Giza, "Balancing Logic Utilization and Area Efficiency in FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and its Applications*, pp. 535–544, 2000.

[55] U. Brenner and J. Vygen, "Legalizing a Placement with Minimum Total Movement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 12, no. 12, pp. 1597–1613, 2004.

Glossary of Terms

ASIC Applicaton-Specific Integrated Circuit.

BLE Basic Logic Element.

Channel Width The number of wires in the routing channel between CLBs.

CLB Configurable Logic Block.

Critical Path The longest path in a circuit, which determines the maximum operating frequency.

FPGA Field Programmable Gate Array.

IO Input/Output.

VPR Versatile Place and Route, a placement and routing tool for research in FPGAs, and can be obtained at `http://www.eecg.toronto.edu/vaughn/vpr/vpr.html`.

Wire Length The sum of wire segments needed to route a circuit.