Direct Synthesis of Netlists Into Pre-Routed FPGAs

by

Daniel Di Matteo

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Direct Synthesis of Netlists Into Pre-Routed FPGAs

Daniel Di Matteo

Master of Applied Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2013

This thesis introduces a new approach to compilation for FPGAs, which we call direct synthesis. We take a technology-mapped circuit netlist and directly map it into a pre-placed and routed FPGA overlay. Solving this problem may help to address the increasing portion of compile time that is attributed to placement and routing, and the tremendous amount of area and energy consumed by the highly flexible FPGA routing network. This thesis presents a direct synthesis algorithm and an algorithm for generating the pre-placed and routed FPGA overlays. Using the direct synthesis flow which we have designed, we can successfully map circuits less than 100 BLEs in size, after modest modifications to the architecture of the FPGA overlay circuit. While we show that direct synthesis problem is challenging, further architectural modifications are proposed which can allow the direct synthesis of larger circuits to succeed.

Acknowledgements

I would like to thank my supervisor, Jonathan Rose, for his guidance, knowledge, and support. His feedback and constructive criticism have helped my growth as an engineer and a scientist. Thanks to everyone in Pratt 392 for providing a fun workplace. Thanks to Alex Rodionov for help with all the software development, thanks to Jason Luu for help with everything FPGA-related, thanks to Henry Wong for the insightful observations, and thanks to Braiden Brousseau for all the interesting discussions. Finally, I'd like to thank my parents: this work wouldn't be possible without your love and support.

Contents

1 Introduction **1**

1.1 Research Goals 3

1.2 Thesis Overview 4

2 Background **5**

2.1 FPGA Architecture 5

2.2 FPGA CAD 7

2.2.1 Common Algorithms used in FPGA CAD 10

2.3 Related Work 13

2.3.1 CAD Using Pre-Placed and Routed Logic 14

2.3.2 Generation of Target Networks 15

2.4 Summary 17

3 The Design of a Direct Synthesis Algorithm **18**

3.1 Approach 19

3.2 LUT and Register Pre-Packing 20

3.3 Initial Assignment 21

3.4 Assignment Optimization 21

3.4.1 Move Generation 22

3.4.2 Cost Function 22

3.4.3 Cooling Schedule 24

3.5 Route-Through Augmentation 26
3.5.1 Increasing Route-Through Opportunities via BLE Depopulation . 28
3.6 Tuning the Direct Synthesis Algorithm 30
3.6.1 Cost Function of the Optimizer 31
3.6.2 Temperature Updating in Cooling Schedule 32
3.6.3 Moves per Temperature of the Optimizer 33
3.6.4 Termination Condition of Optimizer 34
3.6.5 Route-Throughs 34
3.6.6 Effect of Enforcing Depopulation Level 35
3.7 Summary 38

4 Target Network Generation **40**
4.1 Topology Characterization 41
4.1.1 The Fanout and Wire Length Distributions 42
4.2 Generation 43
4.2.1 Target Instantiation and Placement 43
4.2.2 Inter-Cluster Route Generation 45
4.3 Effective Topologies for Target Networks 48
4.3.1 Training the Distributions to Training Data 48
4.3.2 Sweeping the Topological Parameters 50
4.3.3 Comparison 52
4.4 Summary 53

5 Results **54**
5.1 Methodology 54
5.2 Quality vs. Subject Circuit Size 55
5.3 Quality vs. I of the Target Networks 58
5.4 Assessing the Performance of the Direct Synthesis Algorithm 61

5.5 Routability of the Target Networks 64
5.6 Conclusion 65

6 Conclusion **66**

6.1 Future Work 67
6.1.1 Adding Flexibility to the Pre-Routed Target Networks 67
6.1.2 Multiple Target Networks 67

Bibliography **69**

List of Tables

3.1 Temperature scaling parameter as function of move acceptance ratio . . . 25

3.2 Training set of subject circuits 30

3.3 Default Direct Synthesis settings used in tuning 31

3.4 gamma values tested 33

3.5 Temperature scaling parameter as function of move acceptance ratio . . . 33

3.6 Improvements to PSC from Performing Route-throughs 37

4.1 Parameters for the fanout and wire length distributions derived from training 49

4.2 Direct Synthesis settings used in Chapter 4 50

4.3 Results of direct synthesis for a target network topology trained to training circuits 50

4.4 Distribution parameters used to create experimental target networks . . . 51

4.5 Parameters for the best fanout and wire length distributions derived from experimentation 51

4.6 Results of direct synthesis for a target network topology derived from experimentation 52

4.7 Comparison of distribution parameters derived from training and experimentation 53

5.1 Direct Synthesis settings used throughout Chapter 5 55

5.2 Parameters used to generate Target Networks throughout Chapter 5 . . . 55

5.3 Validation set of subject circuits 60
5.4 Minimum I for a successful mapping of each benchmark circuit 62
5.5 Minimum channel width required to route subject circuits and target networks 65

List of Figures

1.1 Example Direct Synthesis Problem 2

2.1 FPGA architecture 6

2.2 Cluster and BLE architecture 6

2.3 FPGA routing architecture 7

2.4 FPGA CAD flow 8

2.5 Resource-routing graph for a single 2-LUT logic block [5] 13

3.1 Example Direct Synthesis Problem 19

3.2 A sample mapping consisting of one BLE and its fanout 23

3.3 A mapping that can be legalized through the use of a route-through . . . 27

3.4 Routing-resource graph representing a target network 28

3.5 Routing-resource subgraph for a single logic cluster 29

3.6 PSC vs c of the optimizer's cost function 32

3.7 PSC and Runtime vs. Move Factor 34

3.8 Percent Failed Connections vs. Runtime for various values of Move Factor 35

3.9 PSC vs. Runtime for various values of ε 36

3.10 PSC vs. Depopulation with constant cluster size 38

3.11 PSC vs. Depopulation with compensated cluster size 39

4.1 Wire length and fanout distributions for a packed and placed circuit . . . 42

4.2 A placed target network consisting of unrouted IOs and logic clusters . . 44

5.1 Quality of direct synthesis for subject logic circuits of increasing size. . . 56
5.2 Connectedness versus cluster size, for an $N = 10$, $I = 22$ architecture. . . 57
5.3 Quality of direct synthesis for subject logic circuits of increasing size. . . 59
5.4 Quality of direct synthesis for target networks of increasing I. 61
5.5 PSC of validation circuits mapped to cloned and standard target networks 63
5.6 PSC of validation circuits mapped to cloned and standard target networks 64

Chapter 1

Introduction

Two of the key issues in Field-Programmable Gate Arrays (FPGAs) today are the growing compile times and the high cost of the programmable routing network flexibility. The first issue arises because the size of FPGAs increases at a pace faster than single-threaded CPU performance. The second issue occurs because FPGA routing architectures, essential for their core programmability, consume a large fraction of the silicon area and significant portions of the power budget - the former being a significant barrier to the adoption of FPGA-like programmable cores in large SoCs. These issues motivate us to explore the subject of this thesis: the *direct synthesis* of netlists into pre-placed and pre-routed FPGAs.

Direct synthesis, illustrated in Figure 1.1, begins with a *subject circuit* that is a technology-mapped netlist of basic logic elements (BLEs) as shown in Figure 1.1(a). The second input is a fully packed, placed and routed circuit on an FPGA, called the *target network*, as illustrated in Figure 1.1(b). The goal of direct synthesis is to assign the BLEs of the subject circuit to specific BLEs in the target network, such that all of the connections required in the subject circuit can be made by the correct programming of the LUTs and the internal crossbar of the clusters. Figure 1.1(c) shows the successful mapping of the small example circuit of Figure 1.1(a) into the network of Figure 1.1(b).

You can observe how, for example, the connection of `BLE 1` to `BLE 3` in the subject circuit led to the packing/placement shown in the solution. Once the solution is determined, the target network needs to be modified in two simple ways: the specific programming of the logic function of the BLEs, and the programming of the internal crossbar (assumed to be fully-connected for now) within the logic cluster.

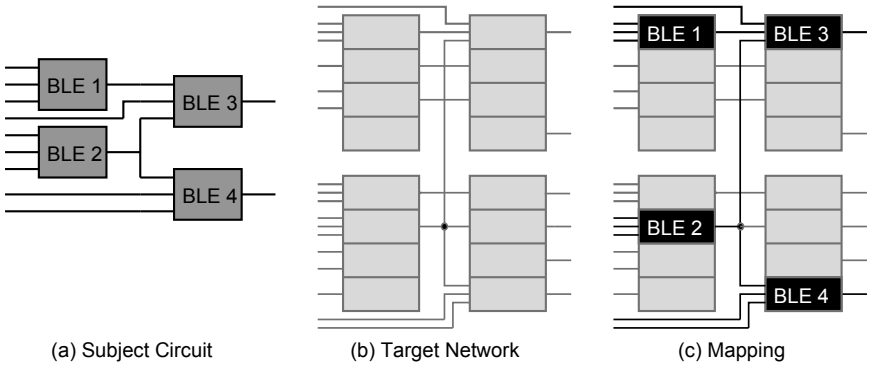

Figure 1.1: Example Direct Synthesis Problem

Direct synthesis has the promise of addressing fast compile times, by avoiding slow place and route. It may be possible to create an algorithm which is faster than the traditional CAD flow by packing, placing and routing the subject circuit in a single, combined step. A library of target networks could be created and categorized in an off-line fashion, and a viable candidate selected at compile time by a sufficiently intelligent direct synthesis algorithm. Since each target network represents an almost completely pre-compiled FPGA, it could be viewed as a partial solution to the traditional CAD flow which simply needs to be modified and verified as legal. Computer science tells us that, for many classes of problems, verifying a solution is much faster than creating a solution from nothing (especially if P != NP). This might be exploited to produce a faster FPGA CAD flow. Although the compile time issue motivates this work, the focus of this thesis is less ambitious: it is to look for ways to simply make direct synthesis *successful* - as will be discussed later, the problem itself is quite difficult.

In addition, exploring direct synthesis allows us to study, in a new way, the true

requirement for flexibility in FPGAs. Pre-routing an FPGA such that all connections between all logic blocks and all IO pads are fixed (by programming all switches in the routing fabric) would be the most extreme case of reduced routing flexibility one could explore. We would like to gain a broad understanding of how the removal of the routing flexibility influences the compilation process. Firstly, it is not clear that successful compilation will be possible any longer for the same broad range of application circuits. Secondly, if it does prove to be possible, this may have implications for how routing architectures are designed. Removing flexibility from the routing architecture would serve to create smaller, faster, more power-efficient FPGAs.

Our work on this problem takes the first steps towards both of these motivations by simply exploring whether direct synthesis is possible, and if so, to determine under which conditions. We have found that successful direct synthesis is very difficult to achieve. This is because the choice of the target network immediately removes a significant amount of the flexibility of the native FPGA. However, we believe this is an interesting problem to attempt to solve as it will shed insight into the fundamental need for flexibility in an FPGA, and perhaps help with the compile time problem as well.

1.1 Research Goals

The goal of this research is to explore the possibility of direct synthesis by:

1. Designing an algorithm that can perform direct synthesis of a subject circuit onto a target network.

2. Creating a flexible tool that can generate target networks with parameterizable architectures.

3. Experimenting with the direct synthesis algorithm and target network architectures to determine under what conditions direct synthesis is possible.

1.2 Thesis Overview

This thesis is organized as follows: Chapter 2 provides relevant background and describes related prior work on pre-routed FPGAs. We will present the two key parts of the problem - the direct synthesis algorithm (in Chapter 3), and the generation of the target network in Chapter 4. Chapter 5 gives a series of experiments attempting to synthesize circuits on several target architectures, and Chapter 6 concludes and gives suggestions for future directions.

Chapter 2

Background

This chapter provides the reader with the necessary background on FPGA architecture and CAD used in later discussions on direct synthesis. It also summarizes past research relevant to direct synthesis.

2.1 FPGA Architecture

FPGAs are digital integrated circuits whose functionality can be programmed (and re-programmed) by a hardware designer. FPGAs contain programmable logic blocks (LBs) and input/output pads (I/Os) that are interconnected with programmable routing. While commercial FPGAs contain additional, hardened complex blocks, including on-chip memories, this work focuses on a simple, homogeneous FPGA architecture as shown in Figure 2.1 [5].

The FPGA logic blocks that we will consider are clusters of Basic Logic Elements (BLEs), as illustrated in Figure 2.2a. Each logic block contains N BLEs, I inputs, and N outputs. The BLEs are *fully-connected* (with a full cross-bar), such that each BLE input can be driven by any of the I logic block inputs or any BLE output. A BLE is a pairing of a K-input Look-Up Table (LUT) and a flip-flop, together with a multiplexer that allows the BLE output to be driven by the registered or unregistered output of the

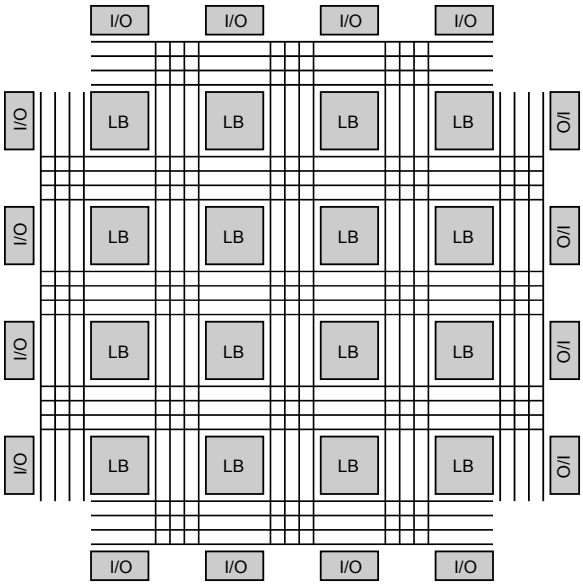

Figure 2.1: FPGA architecture

LUT. Figure 2.2b illustrates the internals of a BLE. The logic block inputs and outputs are connected to the programmable routing of the FPGA to provide connections between I/Os and other logic blocks.

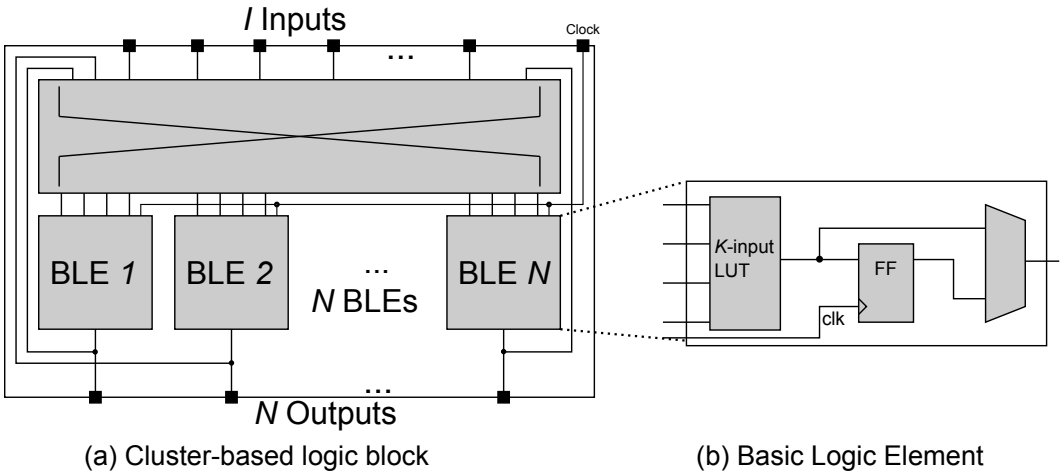

Figure 2.2: Cluster and BLE architecture

The FPGA routing architecture [26] consists of channels of wires that run between logic blocks, spanning the vertical and horizontal distance of the chip, as shown in Figure 2.3. The number of wires (or *tracks*) within each channel is denoted by W. Signals

that originate or terminate at logic blocks gain access to the routing network through the programmable switches that compose the *connection block*. The fraction of wires that a logic block input pin can connect to is referred to as the input connection block flexibility, or $F_{c_{in}}$. Similarly, the fraction of wires that a logic block output pin can connect to is referred to as the output connection block flexibility, or $F_{c_{out}}$. Connections between intersecting tracks are formed by groupings of programmable switches called a *switch block*. Wires that are incident on a switch box have the option of driving one of a number of several other wires. This number of wires is referred to as the switch block flexibility, or F_s. The routing architecture in Figure 2.3 can be characterized as having $W = 4$, $F_{c_{in}} = F_{c_{out}} = 0.5$, and $F_s = 3$.

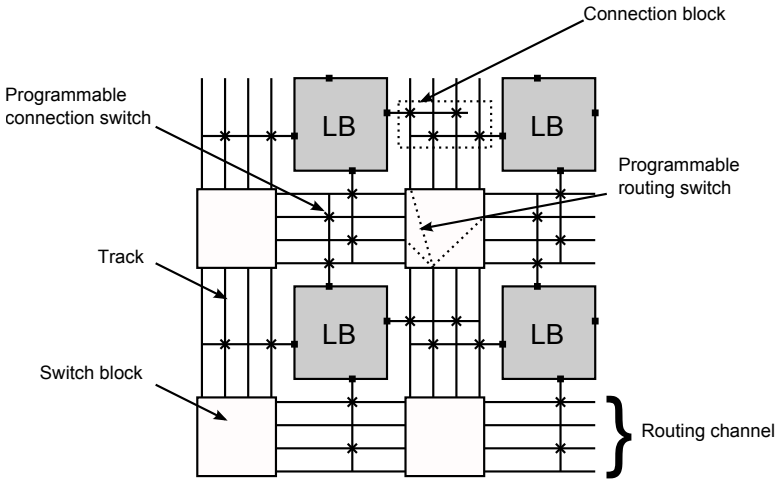

Figure 2.3: FPGA routing architecture

2.2 FPGA CAD

The use of FPGAs requires Computer-Aided Design (CAD) tools to convert a description of the designer's circuit to the set of programming bits necessary to configure the FPGA (i.e., to configure the programmable logic and routing). The steps in FPGA CAD flow are depicted in Figure 2.4.

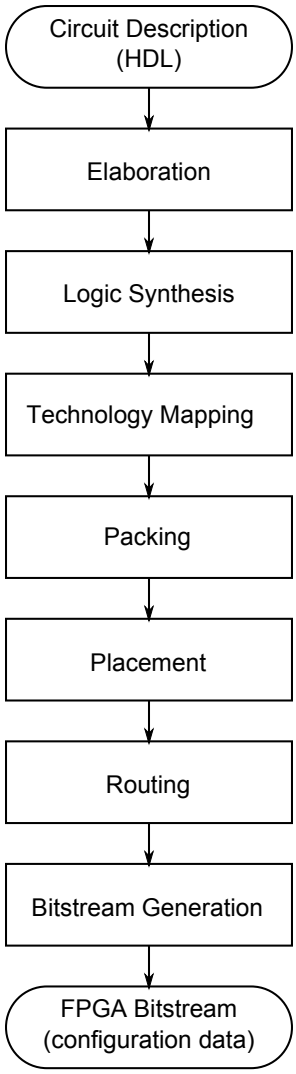

Figure 2.4: FPGA CAD flow

The input to the flow is a circuit description, usually written in a Hardware Description Language (HDL) such as Verilog [1], SystemVerilog [2], or VHDL [3]. The first step in the flow, elaboration [13], parses this human-readable description and constructs a circuit representation consisting of generic logic gates, registers (flip-flops), and larger primitives, such as memories. Logic synthesis performs technology-independent optimization [7] of this circuit representation by, for example, removing redundant logic in the design. Technology mapping [24] replaces the generic logic gates in the circuit with the logic gates contained in the logic block (typically k-input Look Tables, or LUTs) with

specific functionality, such that the functionality of the circuit is preserved, but it is now represented using circuit elements which exist on the FPGA. Technology-dependent logic optimizations may also be performed during the technology mapping step.

The packing step [20] in the flow performs two functions. First, it combines the appropriate circuit elements together in the same structure as the BLEs in the FPGA. Second, it groups BLEs of the circuit representation into clusters that can be mapped to the logic blocks on the FPGA. This clustering step is timing-driven, optimizing the operating frequency by ensuring that circuit elements on the critical path are placed in as few clusters as possible, minimizing the use of the slower inter-cluster routing to transmit signals between these elements. At the end of the packing stage, the designer's desired circuit is wholly represented by circuit inputs and outputs, logic blocks, and fixed-function blocks like multipliers and/or memories (the use of these blocks is not covered in direct synthesis).

The placement step [10] is an optimization problem that maps I/Os and logic blocks of the designer's circuit to physical I/Os and logic blocks that exist in specific locations on the FPGA, with minimal wire length between these components. The most common optimization goals of an FPGA placement algorithm are to maximum the operating frequency of the circuit and the routability of the circuit. Minimizing the total wire length of the design works to address the second goal. The assumption that a circuit will be routable after finding a good placement is one of the key reasons that the back-end steps of the FPGA CAD flow (i.e., packing, placement, and routing) can be performed separately, and not as one larger optimization problem. This does not hold true for the direct synthesis problem presented in this research, however - simply minimizing wire length is not enough to guarantee routability once the routing is fixed.

The final step, routing [34], forms connections between signal sources and sinks in the circuit by configuring the internal-to-the-cluster connections, connection blocks, and switch blocks in the FPGA's routing network. There must be a one-to-one mapping

of signals to a set of connected tracks in the network for the routing to be legal. In addition to achieving a legal routing configuration, most routers will also seek to optimize the circuit's maximum achievable clock frequency by giving signals on the critical path sufficiently fast routes in the routing network. At the end of the routing phase, the state of all the programmable logic and routing elements are known. This FPGA state is aggregated into a file called a bitstream, which can finally be used to program the FPGA to act as the designer's desired circuit.

2.2.1 Common Algorithms used in FPGA CAD

Simulated Annealing

Simulated annealing is a general-purpose heuristic optimization algorithm that seeks to find a minimum-cost state of a system [15]. It is most commonly used in placement, to optimize an initial placement with respect to metrics such as wire length and critical path delay [5][19]. The paragraphs below summarize the general simulated annealing algorithm. The reader may wish to view Algorithm 1 for an outline of the algorithm first.

The cost of the system is measured by a *cost function*, which assigns a scalar value (cost) to a given system state. The closer that a state is to the optimal state, the lower the cost. Given an initial state, a series of random modifications to the state (known as *moves*) are proposed by the *move generator*. A number of moves are proposed for each *temperature*, where the temperature is a changing parameter of the optimizer that controls how the optimization proceeds. The initial temperature is generally set high (as described in more detail below), and is then lowered as the optimization proceeds. All the work done at a single temperature is referred to the as the 'inner loop' of the algorithm.

The inner loop of the algorithm proceeds as follows. Moves are proposed with the

chance of being accepted or rejected. Moves which decrease the cost of the system (i.e., improve quality) are accepted outright and the system takes this new state. Moves which increase cost have a chance to be accepted, where the probability is a function of the temperature of the system and how much the move increases cost (i.e., degrades quality). The probability decreases exponentially with the quotient of increase in cost over temperature - poor moves (which increase cost) are more likely to be accepted at high temperatures. After a certain number of moves have been proposed, the inner loop is exited and the temperature is updated. The outer loop then either terminates the optimization (if the termination condition is met), or performs another iteration of the inner loop.

When annealing begins, the temperature of the system is generally set high, allowing 'poor' moves in order to avoid getting caught at a local minimum of the solution space. As the optimization continues, the temperature is lowered. This prevents the quality from degrading once a reasonable locale in the solution space has been reached. As the temperature becomes very low, essentially only moves which decrease cost are accepted.

The initial temperature, the method by which the temperature is updated, the number of moves to perform per temperature, and the termination condition are all parameters of the optimization generally referred to as the *cooling schedule*. The cooling schedule often takes different forms depending on what problem simulated annealing is being applied to.

In FPGA placement, a cost function is designed which is function of the total wire-length of the circuit being placed, and in timing-driven placers, an estimate of the critical path delay of the circuit. Moves consist of swapping logic blocks (or, more commonly, entire logic clusters) between positions in the chip, in the hopes that these moves reduce cost. Adaptive cooling schedules have been designed, which lower the temperature of the annealing at different rates depending on how well the annealing process is performing. These adaptive cooling schedules also seek to maximize the time that the annealer spends

Algorithm 1: General simulated annealing algorithm

Input: initial state S, initial temperature T

Output: optimized state S_{opt}

while *termination condition not met* **do**

    // begin "inner loop"

    for *move := 1 to movesPerTemperature* **do**

        S_{new} = generateMove()

        ΔC = Cost(S_{new}) - Cost(S)

        if $\Delta C < 0$ **then**

            $S = S_{new}$

        else

            r = uniformRandomNumber(0,1)

            if $r < e^{-\frac{\Delta C}{T}}$ **then**

                $S = S_{new}$

    // end "inner loop"

    T = updateTemperature()

$S_{opt} = S$

at temperatures which produce the most constructive moves [5].

Maze Routing

Many sophisticated FPGA routing algorithms exist, and maze routing often serves as the foundation and/or 'inner loop' of these algorithms. The Lee algorithm for maze routing [18] performs a breadth-first search (through a data structure that is representative of the routing architecture) from the source of the route towards the sink. As the wavefront of the search expands, available components of the routing architecture are annotated with a cost. Once the sink terminal is reached, the algorithm then backtracks, choosing

the lowest-cost path to perform the route.

A common choice for a data structure to represent the routing architecture of an FPGA is a routing-resource graph [5]. The main benefit of such a structure is that it is highly flexible and able to represent a wide variety of routing architectures. Nodes in this graph represent wires and pins, where edges represent programmable switches in the FPGA. Two special nodes in the graph, the source and sink nodes, are used to model logical equivalence in certain structures in the FPGA. All routes begin and terminate at a sink and source node, respectively. When routing a signal to an element with logically equivalent input pins (e.g., a K-input BLE), any one of the available input pins can be used to transmit the signal to the sink node. The capacity (c) of a node is equal to the maximum number of distinct signals that may drive a node. It follows, then, that the signal sink for a K-input BLE must have a capacity of K. Figure 2.5 depicts part of an FPGA routing-resource graph.

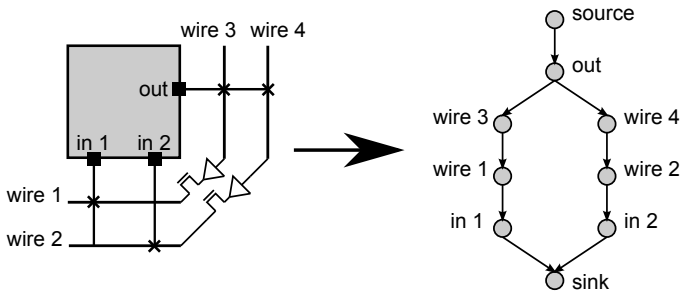

Figure 2.5: Resource-routing graph for a single 2-LUT logic block [5]

2.3 Related Work

As outlined in Chapter 1, this exploration of direct synthesis requires development in two main areas: the design of a direct synthesis algorithm, and the creation of a tool which can generate target networks. Although, to our knowledge, direct synthesis has never been attempted, there do exist CAD approaches for using pre-placed and routed logic.

Similarly, there exist techniques related to the problem of generating target network architectures. This section will review existing work in both of these areas.

2.3.1 CAD Using Pre-Placed and Routed Logic

The problem of mapping circuits to pre-routed logic is similar to traditional library-based technology mapping [28], where subject logic is mapped to different, but logically equivalent, structures of pre-connected logic elements existing in a library. Technology mapping algorithms fall in two main categories: topological algorithms that map logic by matching structural or graph-theoretical properties of the subject logic and library logic elements [8], and Boolean matching techniques that transform the Boolean logic of the subject into functionally equivalent representations that can be implemented by the library logic elements [29]. We imagine these techniques would be unable to efficiently perform direct synthesis, since a target network must be treated as a single library element which is orders of magnitude larger in size than the library elements used in traditional technology mapping. Other CAD techniques which perform compilation using post-technology mapped, pre-placed and pre-routed logic will be summarized below.

Tessier designed an FPGA CAD flow in [31] which seeks to speed up compile time by composing designs out of pre-placed macroblocks. In that work a pre-processing step is performed which first creates clusters of inter-connected macroblocks. An initial floor-plan is created by performing simulated annealing-based placement of these macroblock clusters to placement bins (consisting of multiple logic blocks) on the device. Following that, an analytical placement constrained to each placement bin is performed to place the individual logic blocks within each macroblock. Finally, the placement is refined using a low temperature global anneal to improve routability. This flow produced a 4× speedup in combined place and route time with respect to a commercial tool, while producing designs that exhibited, on average, 28% lower operating frequencies.

Lavin et.al [17] takes this a step further, where they experiment with macros that

contain both pre-placed and pre-routed logic. Here designers must describe their circuit using a hard macro design entry tool - circuits are composed of coarse-grained functional blocks, and each functional block has an associated hard macro. Hard macros are created (well before the designer begins, offline) by placing and routing each functional block once (through the traditional flow), and then saving the resulting placement and routing. The tool then performs the 1-to-1 mapping of functional blocks to hard macros, and the final design is created through the place and route of the larger hard macros. This technique leverages the re-use of the placement and routing of functional blocks at a far more coarse grained level than the approach in this thesis. Their technique provides runtime speedups of 30× over industrial tools, while producing circuits with operating frequencies that are 75% as fast as those produced by industrial tools.

Hung and Wilton [11] describe a system for rapidly routing connections to allow for debugging without going through a full placement and routing step. Rather, they create a network of multiplexers from within the routing fabric itself (on top of the original design being debugged), and create a clever and very fast routing algorithm that can select the settings of a smaller subset of the multiplexers. This 'virtual overlay network' approach suggests that the regular routing may be similarly abstracted. Although we do not make use of this approach in the work presented in this thesis, we believe that it could be a powerful addition to this work.

2.3.2 Generation of Target Networks

In the construction of a target network, we wish to create a pre-routed FPGA overlay that is amenable to direct synthesis. The interconnection between logic blocks, although inflexible (due to pre-routing), is a form of routing network. The traditional method of creating and evaluating FPGA routing networks is largely empirical. Designers modify existing networks to propose novel hypothetical architecture which are then evaluated by compiling benchmark circuits to that architecture, using a CAD flow such as VTR [27].

The results of that CAD flow - estimates of area, speed and power, inform the architect as to the quality of the architecture. Such a methodology is heavily influenced by the CAD algorithms used during the compilation, however. The quality of a highly unorthodox architecture could be difficult to measure if the CAD algorithms would require tuning or modification to effectively target it. The removal of routing flexibility in the target networks makes using such an approach difficult.

In order to create target networks that can be used for successful direct synthesis, these target networks must have structures similar to subject circuits themselves, since there exists no flexibility to 'shape' or configure the routing network appropriately. That task is similar to the one that in the past was called 'synthetic benchmark creation' whose purpose was to create larger benchmark circuits when there were few real ones available. Those works, for example [9] [12] and [25] analyzed real circuits to determine a number of characteristic parameters and then attempted to generate synthetic circuits with those same parameters.

Darnauer and Dai [9] characterized circuits by their number of LUTs, IOs, average fanin, and Rent parameter [16]. LUTs were then instantiated and placed, and a top-down approach was used to create nets by recursively partitioning the circuit and connecting both partitions such that the measured Rent parameter would be consistent with that of the characterization.

Instead of a Rent parameter based-construction, Hutton et al. [12] focus on characterizing circuits by measuring the distribution of LUTs at each combinational delay level, the fanout distribution of all net drivers, and the wire length distribution of connections in a circuit. Combinational logic is created to have properties matching these distributions, and is interconnect together with flip flops and other combinational logic to create larger sequential circuits. Our method of generating target networks makes use of Hutton's general approach, particularly his characterization methodology.

Pistorius et al. [25] introduced a method of generating larger benchmarks with hi-

erarchical structure that better matches real circuits. Their characterization of circuits includes a measure of the hierarchy of netlists created through recursive bipartitioning. Sub-circuits are created in a fashion similar to [12], and a interconnection is created to connect these sub-circuits into a hierarchical structure consistent with the earlier characterization of existing circuits.

2.4 Summary

This chapter has summarized past work that is related to the problems of direct synthesis and target network generation. An algorithm for performing direct synthesis is presented in Chapter 3. Chapter 4 describes the methodology by which target networks are created.

Chapter 3

The Design of a Direct Synthesis Algorithm

The goal of this research is to explore how it may be possible to map circuits to pre-routed FPGAs, a problem which we call direct synthesis. The inputs to direct synthesis are the *subject circuit* and the *target network*. The subject circuit is a technology-mapped netlist consisting of only K-input look-up tables (LUTs) and registers. The target network is a set of pre-placed logic clusters (that contain K-input LUTs and registers as internal BLEs) and I/O pads, with fixed inter-cluster routing (ICR), and (un-programmed) intra-cluster routing. Each logic cluster in the target network has I inputs and N BLEs that we assume to be fully connected through the intra-cluster routing (although we are aware that commercial FPGAs do not use fully-connected clusters, we leave addressing that issue to future work).

The definition of the direct synthesis problem is as follows: determine an implementation of the subject circuit in the target network by changing only the target network's configuration SRAM bits controlling the LUT functions and the routing of the cluster crossbars. This requires the assignment of the subject circuit's LUTs, registers and I/Os to specific LUTs, registers and I/Os in the target network. The resulting modified tar-

get network is called a *mapping*. Figure 3.1 gives an illustration of a small version of the problem, reprised from Chapter 1. The quality of a mapping will be measured as the Percentage of Successful Connections (PSC) - that is, the percentage of source-sink connections (where a fanout N net consists of N source-sink connections) in the subject circuit that are successfully connected after all LUTs, registers, and IOs from the subject circuit are implemented in the target network. A subject circuit that is successfully directly synthesized to a target network will therefore have a PSC equal to 100.

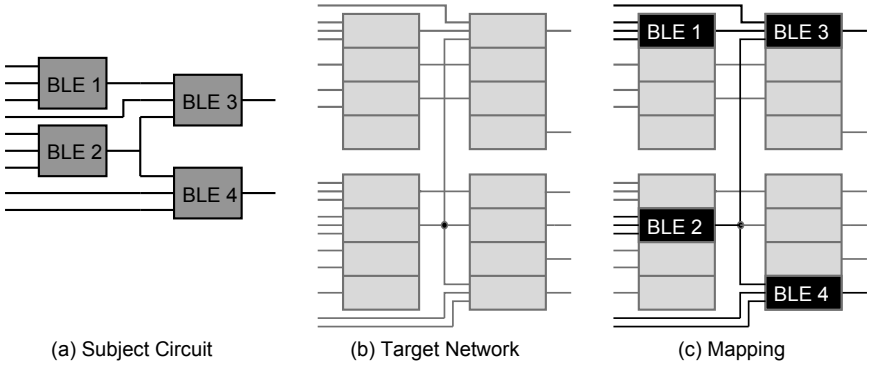

Figure 3.1: Example Direct Synthesis Problem

3.1 Approach

The reader will observe that the direct synthesis problem as posed is very similar to the classical physical synthesis problem for FPGAs, requiring packing, placement and routing of the subject circuit. The result, however, is constrained by the placement and routing of the target network, which is a highly constrained problem that has proven to be difficult to solve. The act of assigning a specific LUT, register or I/O to one of those in the target network can result in immediate failure if there aren't routes to and from that location in the target that provide the required connectivity.

For that reason, it seems clear that the packing step, which makes such an assignment, must consider the matching of the connectivity between the subject and the target. This

means that the packing, placement and routing must essentially be done at the same time. This differs from the traditional flow which separates packing, placement and routing into separate phases. The traditional flow is based on the assumption that a packing and placement with sufficiently low total required wirelength will succeed in routing.

To this end, we have developed an algorithm which performs simultaneous packing and placement in a routing-aware fashion. The overall algorithm has four phases:

1. Pre-packing of LUTs and registers into basic logic elements (BLEs).

2. Initial packing/placement that assigns BLEs in the subject circuit to the target network.

3. Optimization of the packing and placement that seeks solutions for which the routing is correct.

4. Routing augmentation that makes use of 'route-throughs' by configuring unused LUTs into wires.

In the following sections each of these phases will be described, with design elements of the algorithm being validated with experimental results in the final section.

3.2 LUT and Register Pre-Packing

The subject circuit arrives with separate LUTs and registers, and so the first step is to pair these together (where possible) into BLEs. A complete BLE may consist of both a LUT and a register, or single LUT or single register. To pack to BLEs the algorithm from [5] is used. The algorithm scans the input netlist and selects LUTs that only drive a single register (and have no other fanout) and packs these LUTs and registers together into single BLEs. All remaining LUTs and registers are each given their own BLE. From

this point on we will refer to BLEs and I/Os in both the subject circuit and target network as *nodes*.

3.3 Initial Assignment

To create an initial assignment that we will attempt to legalize in the future, a random assignment of subject nodes to target nodes is performed, as constrained by the types of the nodes. Primary Input nodes, BLE nodes, and Primary Output nodes in the subject are only assigned to the same type of node in the target. A more sophisticated, fanout-based initial assignment was used during an earlier phase of this research, which assigned high-fanout nodes in the subject to sufficiently high-fanout nodes in the target. This was motivated by the observation that insufficient fanout was a common source of routing failure. However, subsequent changes to the cost function of the assignment optimizer eliminated the need for specific attention to this issue during the initial assignment phase.

3.4 Assignment Optimization

Once the initial assignments of subject nodes are made to the target nodes, the next step is to modify this assignment so that the implementation moves as close as possible towards a correct solution - one in which all of the subject circuit's connections are made correctly in the target. We do not know, in general, if a solution exists, but in Chapter 5 we will explore cases in which we do know that there is at least one valid solution.

For generality of optimization, we chose to use Simulated Annealing as the core optimizer. Our principal reason for using this approach (as opposed to, say, an analytical approach [6]) is its ability to easily modify the cost function used, to help guide the optimization process. We also chose this approach realizing that, if direct synthesis were to succeed, and our efforts turned towards compile time reduction, that we would likely seek a faster basic approach. In our application to direct synthesis, simulated annealing is

used to optimize the assignment of nodes from the subject circuit to the target network. The specific move generator, cost function, and cooling schedule that have been designed for use in direct synthesis will be described below.

3.4.1 Move Generation

The move generator selects random swaps of like-type nodes (BLE to BLE or I/O to I/O) including swapping with empty locations, which amount to single node moves. An additional constraint was explored, which proposed moves such that no subject node lands in a target node with insufficient fanout, as one of the main causes of routing failure is that a node in the target simply does not have sufficient fanout for the subject node. The use of this constraint did not prove to increase the number of successful connections, and as such it is not used in the direct synthesis algorithm. We hypothesize that this is so because the cost function, to be described next, will heavily penalize assignments in which subject nodes are placed in locations with insufficient fanout.

3.4.2 Cost Function

In direct synthesis, the goal is to make sure that every source-sink connection in the subject circuit is implemented with a unique (and non-shorted) source-sink connection in the target. To ensure that this occurs, the cost function of the optimizer must accurately measure how well connections in the subject circuit are being mapped to wires in the target network. A cost is accumulated for each node in the subject netlist, and depends on whether there exist routes in the target network connecting the assigned locations of the node and each of its fanout nodes. This is dependent upon where the driving node and each of its fanout nodes have been assigned in the target.

We first define some notation. Given a node A, from the subject circuit, the other subject circuit nodes that are directly driven by A will be referred to as the fanout nodes of A, $N_{FO,A}$. The cluster in the target network to which a fanout node of A is mapped to

will be referred to as a *fanout cluster* of A. The set of all the fanout clusters of a subject node A is referred to as $C_{FO,A}$ The cluster in the target network to which the node A itself has been mapped to will be referred to as C_A. A cluster in the target network that is driven by a pre-routed connection originating at the location to which the subject node A has been mapped to is referred to as a *reachable cluster* of A. Let the set of all the reachable clusters of A be referred to as $C_{R,A}$. Similarly, let the set of all clusters that are not driven by a pre-routed connection originating at the location to which the subject node A has been mapped to be referred to as the set of *unreachable clusters* of A, or $C_{U,A}$. Finally, we define an operator, *nodes*(), which, given a set of target clusters, returns the set of subject nodes that have been mapped into to those clusters

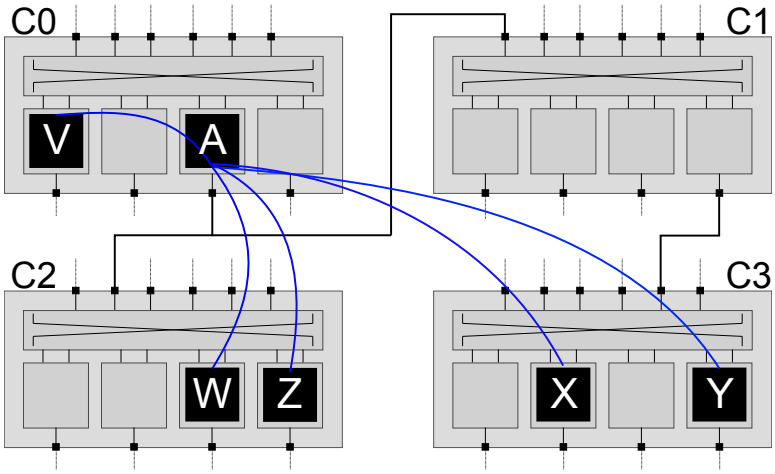

Figure 3.2: A sample mapping consisting of one BLE and its fanout

To provide some intuitive understanding of the cost function as we define it, the mapping in Figure 3.2 will be analyzed. Figure 3.2 shows a partial mapping of six BLEs from a subject circuit (shown as black squares) to a target network consisting of four logic clusters, with pre-routed connections existing between these clusters. That is, with respect to BLE A, $C_A = \{C0\}$, $C_{R,A} = \{C1, C2\}$, and $C_{U,A} = \{C3\}$. BLE A drives five sinks, BLEs V through Z, as shown by the blue arcs connecting the BLEs, therefore $N_{FO,A} = \{V, W, X, Y, Z\}$ and $C_{FO,A} = \{C0, C2, C3\}$. The expression $nodes(C2, C3)$

would evaluate to $\{W, Z, X, Y\}$.

Analyzing the connections that exist in Figure 3.2, we see that the connection to BLE V is legal because it will be implemented using local routing (which we model as fully connected). The connections to BLEs W and Z are legal, since they reside in a reachable cluster. It would be better, however, if these BLEs were moved into cluster C0 as it would free up the inter-cluster wire driving cluster C2 and improve packing density. The cost function should therefore lightly penalize the use of this intercluster wire, motivating the optimizer to eliminate it if possible. Finally, connections to BLEs X and Y are unsuccessful, since they both exist in an unreachable cluster. The cost function must heavily penalize the existence of such unsuccessful connections.

We therefore price each node in the subject circuit in the following way:

$$Cost(NodeA) = |(C_{FO,A} - C_A) \cap C_{R,A}| + c|N_{FO,A} \cap nodes(C_{U,A})| \tag{3.1}$$

where c is a tunable parameter that penalizes unsuccessful connections. The first term in the cost function exists to encourage BLEs to move into the same cluster and make use of the abundant intra-cluster connections where possible, by attributing a cost to the number of used intercluster wires. In the context of Figure 3.2, $(C_{FO,A} - C_A) \cap C_{R,A}$ is C2, the set of reachable clusters that contain a fanout node of A (but not including node A's local cluster). The second term heavily penalizes unroutability, by attributing a cost for each unroutable sink connection. Again, in Figure 3.2, $N_{FO,A} \cap nodes(C_{U,A})$ is the BLEs X and Y, each of which are unsuccessfully connected to A. The mapping in Figure 3.2 would therefore be assigned a cost of $1 + 2c$. Suitable values for c will be discussed in Subsection 3.6.1.

3.4.3 Cooling Schedule

The cooling schedule of the optimizer, that is, the initial temperature, the rate of temperature reduction, the number of moves per temperature, and the termination condition

are all adapted from the one described in [5], with minor modifications to better tune it to our unique optimization problem.

As in [5], the initial temperature is set high enough such that, initially, almost every move is accepted. Prior to starting the annealing process, every subject node is moved once (randomly), and the resulting cost from every move is recorded. The initial temperature is then set to twice the standard deviation of these recorded costs. Setting the initial temperature in this manner results in roughly 98% of moves being accepted, initially. This differs from [5], where it is required that the initial temperature be set to $20\times$ the standard deviation of the recorded costs to achieve such a high move acceptance.

The inner loop of the optimizer proposes a certain number of moves, after which the temperature is updated. The temperature is updated in the same manner as in [5]:

$$T_{new} = \gamma T_{old}$$

where γ is dependent upon the number of moves that were accepted at T_{old}, α. The relationship between α and γ is shown in Table 3.1. The use of four distinct values of γ allows the cooling rate of the optimizer to vary during the different optimization phases corresponding with each range of α. Larger values of γ can be used to slow how quickly the optimizer lowers the temperature. The values of γ used in the optimizer will be presented in Subsection 3.6.2.

Table 3.1: Temperature scaling parameter as function of move acceptance ratio

α	γ
$\alpha > 0.96$	$\{\gamma_1 \in \mathbb{R} \mid 0 < \gamma_1 < 1\}$
$0.8 < \alpha \leq 0.96$	$\{\gamma_2 \in \mathbb{R} \mid 0 < \gamma_2 < 1\}$
$0.15 < \alpha \leq 0.8$	$\{\gamma_3 \in \mathbb{R} \mid 0 < \gamma_3 < 1\}$
$\alpha < 0.15$	$\{\gamma_4 \in \mathbb{R} \mid 0 < \gamma_4 < 1\}$

The number of moves proposed at each temperature (i.e., the number of moves proposed within the inner loop of the optimizer) is governed by the following relation:

$$movesPerTemperature = MoveFactor \times numSubjectNodes$$

where Move Factor is a tunable parameter which allows us to trade quality of results for increased runtime. This differs from [5], where the number of moves is proportional to the number of movable blocks to the power of $\frac{4}{3}$. The decision to use a linear relationship was motivated by the desire to produce an algorithm with linear runtime complexity. Larger values of Move Factor produce better results (with diminishing improvements) at the cost of longer runtime. Reasonable values for Move Factor are discussed in Subsection 3.6.3.

Finally, the optimization is terminated when the temperature is sufficiently low, according to the following relation proposed in [5]:

$$T < \varepsilon \frac{Total\ Mapping\ Cost}{Number\ Of\ Subject\ Circuit\ Nodes},$$

where ε is typically much less than one, to ensure that optimization is stopped only when temperature is much lower than the average cost attributed to a single node in the subject circuit. A smaller value of ε can be used to increase quality at the expense of increased runtime (see Subsection 3.6.4).

3.5 Route-Through Augmentation

One way to enhance the routability, after the assignment optimization above, is to make use of unused LUTs to act as route-throughs - since a lookup table can be configured as a wire, an unused connection from a source to a sink cluster in the target can be passed on to a downstream cluster by configuring the LUT as a route-through. We do this as a post-optimization step (rather than doing it simultaneously with the assignment step) because it would be counter-productive to use up the empty LUTs too soon.

As an example, consider the mapping in Figure 3.3. Cluster C1 contains a BLE (labeled 'RT') that has a pre-routed connection to cluster C3. Configuring this BLE as a wire would allow BLE A's output to successfully route to BLEs X and Y, making the entire configuration legal. We employ a basic maze router [18] that traverses and annotates a routing-resource graph to perform route-throughs. The maze router performs

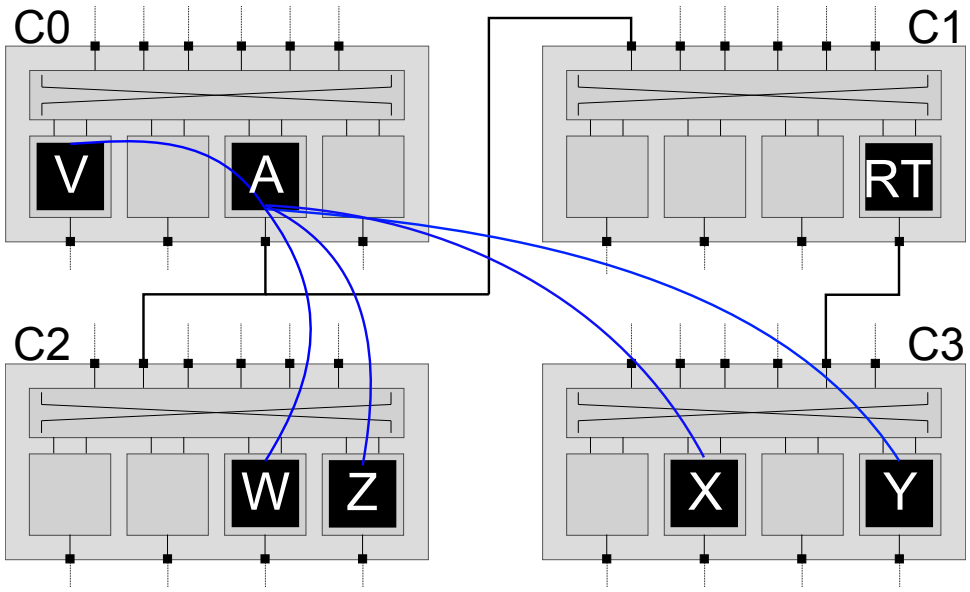

Figure 3.3: A mapping that can be legalized through the use of a route-through

routes in a greedy fashion (i.e., shortest path available), with no congestion avoidance or re-routing [23]. Since routing flexibility has been entirely removed from the target networks, each track has a pre-defined driver and sink(s), and thus the ability to perform congestion avoidance is much lower.

The routing-resource graph used to represent the target network is shown in Figure 3.4, where the missing subgraphs representing the routing within a cluster are illustrated in figure 3.5. Representing all the primary outputs as a single sink node with capacity equal to the number of primary outputs allows the router to decide the mapping of subject circuit primary outputs to target network primary outputs. The routing resource-graph was built in an extensible fashion such that structures like multiplexers could easily be added to the target networks and modeled appropriately by the router. In Chapter 5, it will be proposed that the use multiplexers in the target networks may be one method of improving the success of direct synthesis. While this is not something that has been done, the extensible nature of the routing resource graph facilitates this future work.

A critical component of the routing-resource graph is the presence of edges from the

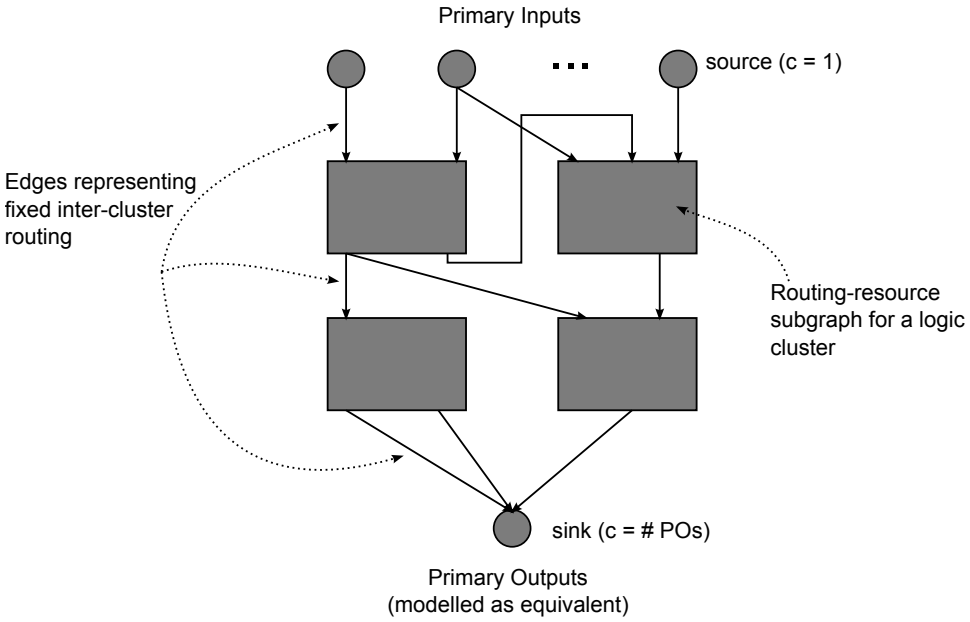

Figure 3.4: Routing-resource graph representing a target network

crossbar node to each of the BLE output pins. These edges, referred to as *route-through paths*, allow the router to perform route-throughs by modelling an unoccupied BLE as a wire. In the case that a BLE is unused, the output pin will not be carrying a signal (i.e., it is 'free'). The maze router is then able to expand the search wavefront for any signal that can reach an input of a cluster through the output pin of an unused BLE into the inter-cluster routing that it drives. In the event that a BLE is used, the router will be unable to use it as a route-through since its output pin will be carrying a signal. (and only has capacity sufficient for one signal).

3.5.1 Increasing Route-Through Opportunities via BLE Depopulation

Since the ability to perform route-throughs relies on the existence of unused BLEs, increasing the number of unused BLEs in a target network may improve routability, and therefore the quality of direct synthesis we are able to achieve. In order to test this hypothesis, we have added the ability to synthesize to target networks with an additional

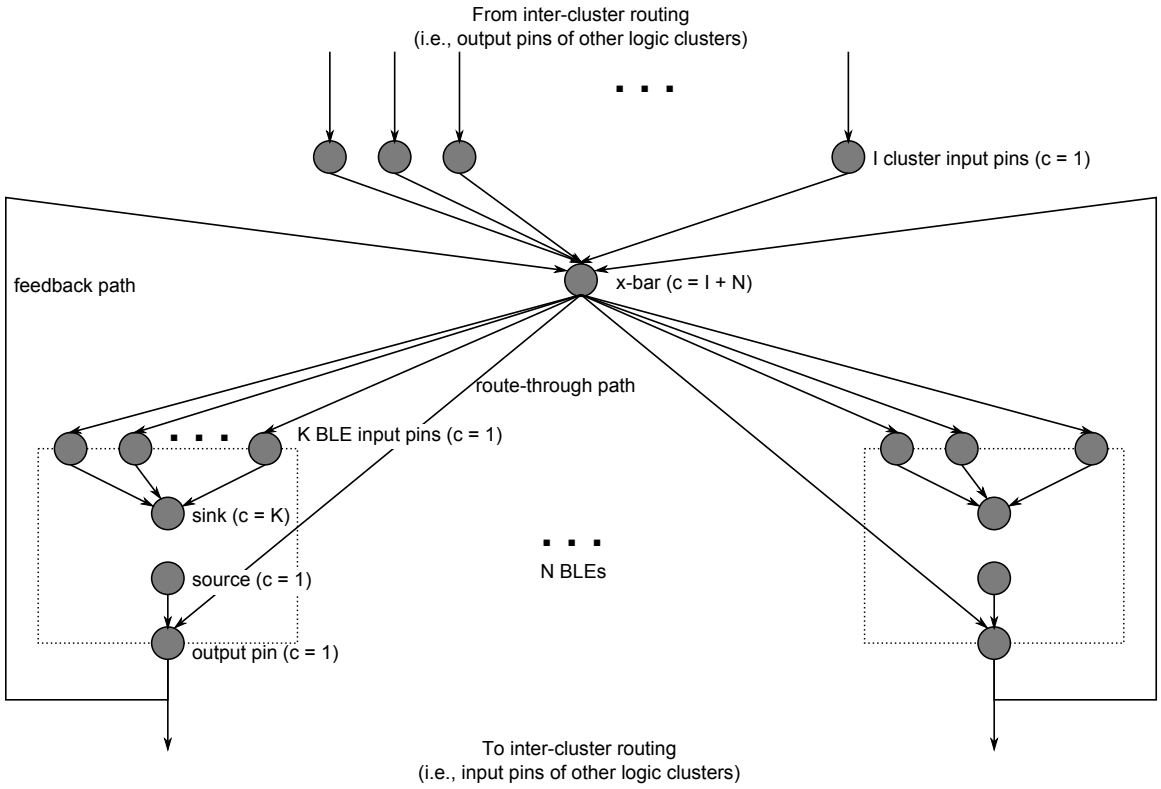

Figure 3.5: Routing-resource subgraph for a single logic cluster

constraint that limits how many BLEs in each cluster can be used to implement subject circuit BLEs. This constraint is referred to as BLE depopulation. For example, setting a BLE depopulation of two would ensure that, before route-throughs are performed, every logic cluster in the target network would have at least two unused BLEs available to perform route-throughs.

Two main algorithmic changes are required to support this feature. The first is a modification to the initial assignment of nodes from the subject circuit to the target network. When depopulation is enforced, the initial assignment must not violate the new cluster population limit. Once initial assignment is complete, each cluster must have the minimum number of free BLEs. The second modification is to the move generator of the assignment optimizer. Single-node moves must be constrained such that they do not increase the population of the destination cluster beyond the limit. Note that node swaps do not need to be constrained since they do not modify the population of either cluster

involved in the swap. This constraint on single-node moves ensures that each cluster continues to respect the population limit that was set in the initial assignment.

3.6 Tuning the Direct Synthesis Algorithm

The final two phases of the direct synthesis algorithm, assignment optimization and route augmentation, both contain tunable parameters that influence how well the algorithm performs. In this section we show how these parameters influence the performance of the algorithm and suggest a set of values for these parameters that produces good results.

This will involve running the direct synthesis algorithm using a set of benchmark subject circuits and target networks, where the quality of results will be measured as the Percentage of Successful Connections (PSC). PSC will be averaged across all subject circuits. Each circuit will run through the direct synthesis flow three times, where each run will be initialized with a different random seed, and the average PSC of the three runs will be reported as the PSC achieved for that one subject circuit. This is done to balance the measurements against the variance of results that occurs from the random nature of the assignment optimizer. The set of subject circuits used in these experiments (referred to as the *training set*) is from the MCNC benchmark suite [33] and are given in Table 3.2, along with their size and number of I/Os.

Table 3.2: Training set of subject circuits

Circuit	**Number of 4-LUTs**	**Number of Primary Inputs**	**Number of Primary Outputs**
apex3	869	54	50
apex4	1262	9	19
C6288	527	32	32
dalu	500	75	16
diffeq	1494	64	39
misex3	1397	14	4
seq	1750	41	35
tseng	1046	52	122

Table 3.3: Default Direct Synthesis settings used in tuning

Direct Synthesis Parameter	**Value**
c	4
γ	{0.5, 0.9, 0.95, 0.8}
Move Factor	6
ε	0.005
Route-throughs	ON
depopulation	0

In each experiment, the settings used in the direct synthesis algorithm are shown in Table 3.3 (unless otherwise stated in the specific experiment). These default settings for γ and ϵ were taken from [5], with the remaining settings found via initial experimentation performed to broadly search the space of parameter values.

The target networks that each subject circuit will be synthesized to contain logic clusters with 40 inputs and 10 BLEs, each BLE consisting of a 4-input LUT and a flip flop. The number of clusters is set equal to the smallest perfect square large enough to fit each subject circuit (e.g., 196 clusters when mapping subject circuit 'seq'). The number of I/Os per target network is equal to $16\sqrt{N_c}$, where N_c is the number of clusters. The topology of the target networks is set according to best configuration found, to be explained in greater detail in Chapter 4, Subsection 4.3.2.

3.6.1 Cost Function of the Optimizer

As described earlier, the cost function of the optimizer contains a parameter, c, which determines how much cost is attributed to unsuccessful connections to nodes. To determine what value of c is appropriate, the training set of subject circuits was synthesized using different values of c. For each value of c, the average PSC (arithmetic average over all three seed sweeps of all 8 subject circuits) was recorded. Figure 3.6 shows how PSC varies with c. A value of c equal to 14 provided the highest average PSC, but any value in the range [4,16] provides good results (average PSC varies less than 2% within this

range).

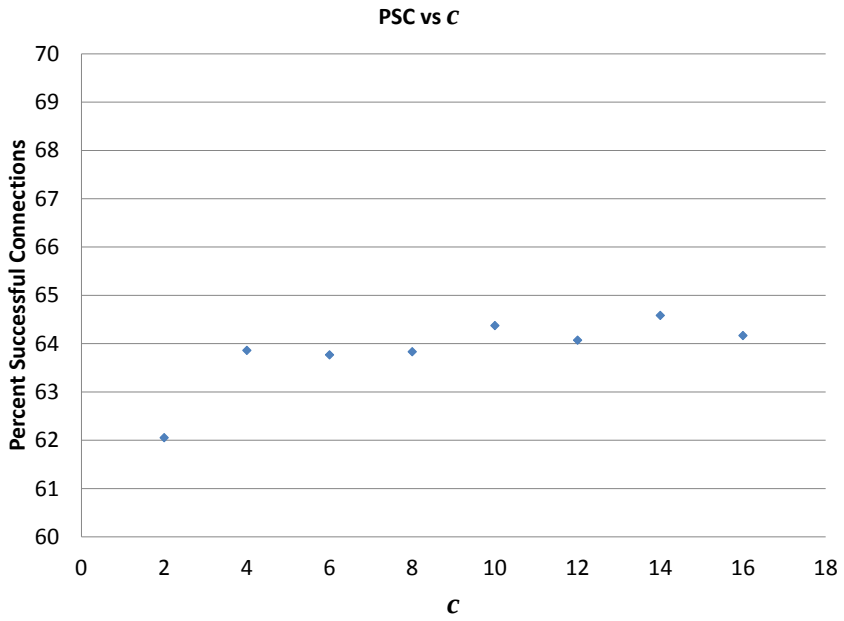

Figure 3.6: PSC vs c of the optimizer's cost function

3.6.2 Temperature Updating in Cooling Schedule

The temperature of the optimizer is scaled down by the factor γ at the end of every iteration of the inner loop. Using the values from [5] as a starting point (see Table 3.5), other values within a window of these were explored. The search for values was restricted to a window around these values, and not left unconstrained, since it is clear that the largest values of γ possible will always produce better solutions. Instead, it was desired to maintain roughly the same 'shape' of a set of γ values, so as not to let runtime explode. Table 3.4 shows the set of all γ values explored. In total, 81 different sets of γ were possible, through choosing one of the three values for γ_1 through to γ_4.

For each set of γ values, direct synthesis was performed using the training set of subject circuits. The average PSC (arithmetic average over all three seed sweeps of all 8 subject circuits) was recorded. The best set of values found, listed in Table 3.1, yielded

Table 3.4: gamma values tested

γ **Parameter**	**Values Tested**
γ_1	0.4, 0.5, 0.6
γ_2	0.7, 0.8, 0.9
γ_3	0.85, 0.90, 0.95
γ_4	0.75, 0.80, 0.85

a 2.1 % increase in QoR with a 3.4 % decrease in runtime with respect to the values used in [5].

Table 3.5: Temperature scaling parameter as function of move acceptance ratio

α	γ **- Betz et al. [5]**	γ **- This work**
$\alpha > 0.96$	0.5	0.6
$0.8 < \alpha \leq 0.96$	0.9	0.7
$0.15 < \alpha \leq 0.8$	0.95	0.95
$\alpha < 0.15$	0.8	0.85

3.6.3 Moves per Temperature of the Optimizer

The move factor controls how many moves are proposed by the optimizer at each temperature. To determine the impact of increasing the move factor on the direct synthesis algorithm, a number of different values are used to perform direct synthesis. For each value of the move factor, direct synthesis was performed using the training set of subject circuits. The average PSC (arithmetic average over all three seed sweeps of all 8 subject circuits) was recorded. Figure 3.7 shows how both average PSC and average runtime change as a function of the move factor. It is clear that increasing the move factor increases PSC, but with diminishing improvements, all the while with runtime increasing at the same rate.

To determine the value of move factor that best balances PSC and runtime, the data in Figure 3.7 is re-plotted with Percent Failed Connections (i.e., 100 - PSC) on the y-axis and runtime on the x axis. The value of move factor that provides the best PSC/runtime

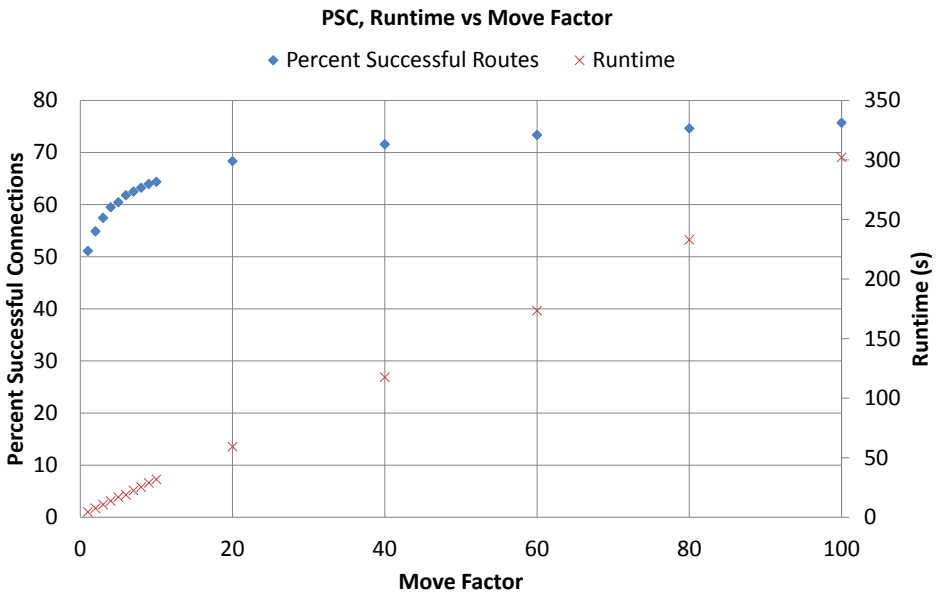

Figure 3.7: PSC and Runtime vs. Move Factor

tradeoff must therefore be located closest to the origin. A move factor of 6 provides the best PSC/runtime tradeoff.

3.6.4 Termination Condition of Optimizer

The parameter ε controls when the optimization is terminated. Figure 3.9 shows how the average PSC and average runtime (arithmetic average over all three seed sweeps of all 8 subject circuits) are both affected by a range of ε values. Decreasing ε extends the runtime of the optimizer and improves PSC, where an ε value of 0.0005 provides the best PSC/runtime trade and an ε value of 0.0001 hits the upper bound on PSC improvement that can be gleaned from decreasing ε.

3.6.5 Route-Throughs

Prior to discussing how the depopulation parameter impacts the results of the direct synthesis algorithm, we wish to investigate how performing route-throughs (with no depopulation enforced) influence PSC. On average, the use of route-throughs increases the

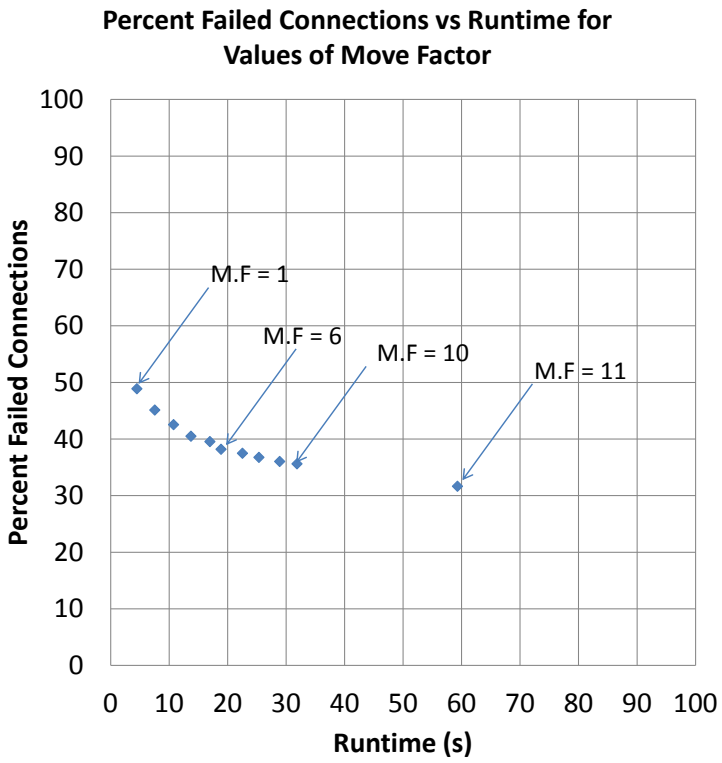

Figure 3.8: Percent Failed Connections vs. Runtime for various values of Move Factor

average PSC (arithmetic average over all three seed sweeps of all 8 subject circuits) by 9%. Table 3.6 shows the breakdown of improvements on a per-circuit basis.

3.6.6 Effect of Enforcing Depopulation Level

To measure the impact of cluster depopulation, we perform two experiments. The first experiment performs direct synthesis with varying amounts of BLE depopulation. For all values of depopulation attempted, the same target network is used. In order for this to be feasible with a depopulation of 5 (and a cluster size of 10 BLEs), the target network contains twice the number of logic clusters than would otherwise be required. The use of the same target network allows us to eliminate the effect of a different sized target network from impacting results as depopulation is varied. Figure 3.10 reports the average PSC (arithmetic average over all three seed sweeps of all 8 subject circuits) for each value

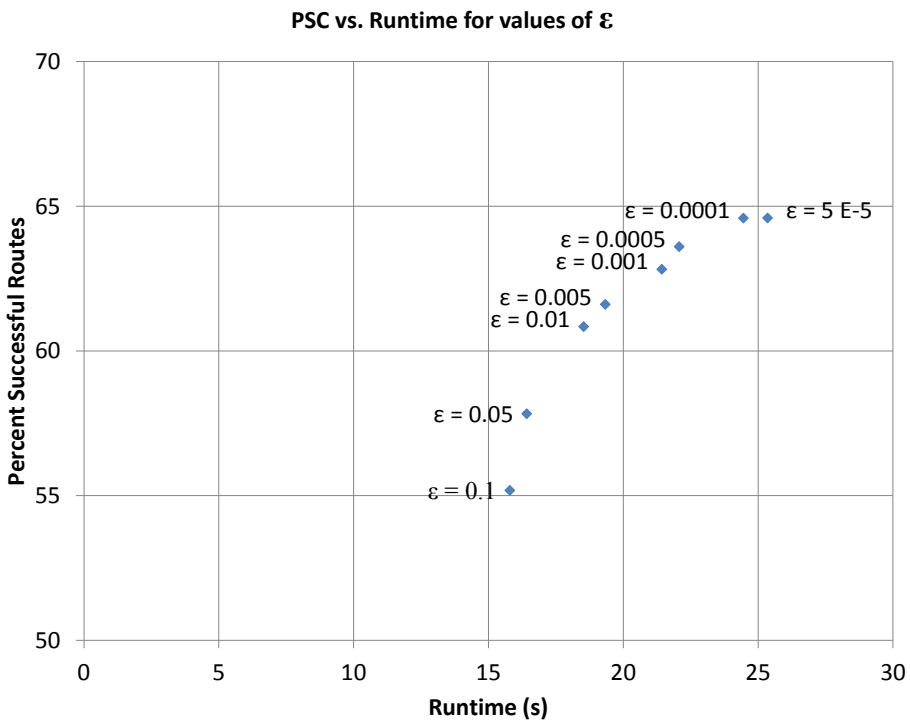

Figure 3.9: PSC vs. Runtime for various values of ε

of BLE depopulation.

The results show that the average PSC with respect to the baseline performance (depopulation = 0) does not improve significantly. Despite the fact that depopulation increases the ability to perform route-throughs (which should be a positive influence on quality), depopulation also has the potential to negatively impact quality. This negative impact is from the fact that depopulation constrains the ability of the direct synthesis algorithm to fully pack to logic clusters. Consider, for example, the case when 5 out of 10 BLEs in a cluster are depopulated - the packing density is reduced greatly, and now nets that may otherwise have been routed legally through the clusters' local interconnect are now forced to find legal inter-cluster routing.

To try to remove the effect of reduced packing density, a second experiment was performed. In this experiment we again attempt direct synthesis with varying levels of BLE depopulation, but we increase the cluster size by 1 BLE for every BLE depopulated,

Table 3.6: Improvements to PSC from Performing Route-throughs

Circuit	**PSC without Route-throughs**	**PSC with Route-throughs**	**Improvement**
apex3	59	61	4%
apex4	51	56	10%
C6288	54	66	23%
dalu	71	80	12%
diffeq	54	56	5%
misex3	53	57	8%
seq	52	53	3%
tseng	59	64	9%
Average	**56**	**62**	**9%**

such that the available number of BLEs is always consistent with the baseline (i.e., 10 free BLEs). Figure 3.11 reports the average PSC (arithmetic average over all three seed sweeps of all 8 subject circuits) for each value of BLE depopulation with cluster size compensated for depopulation.

This second BLE depopulation experiment shows a small but consistent improvement in PSC with increasing depopulation. We are able to observe the positive impact of greater routability through BLE depopulation after compensating for the negative impact of decreased packing density. Note that the percent successful routes for the baseline (depopulation = 0) differs from the first experiment (Figure 3.10). This is due to the fact that the first experiment involved a target network with double the number of clusters. Increasing the number of clusters in this experiment is unnecessary since we grow the cluster size to accommodate the depopulation (what we are concerned with here is not the absolute quality of results, but how it changes relative to the baseline).

Despite the fact that BLE depopulation with compensated cluster sizes can increase PSC, it is not used. The additional area cost of the extra BLEs required to maintain the same effective cluster size makes this approach less attractive than others as a means of improving direct synthesis. For example, in Chapter 5, it will be shown that increasing the number of cluster inputs is another method of improving the success of direct synthesis.

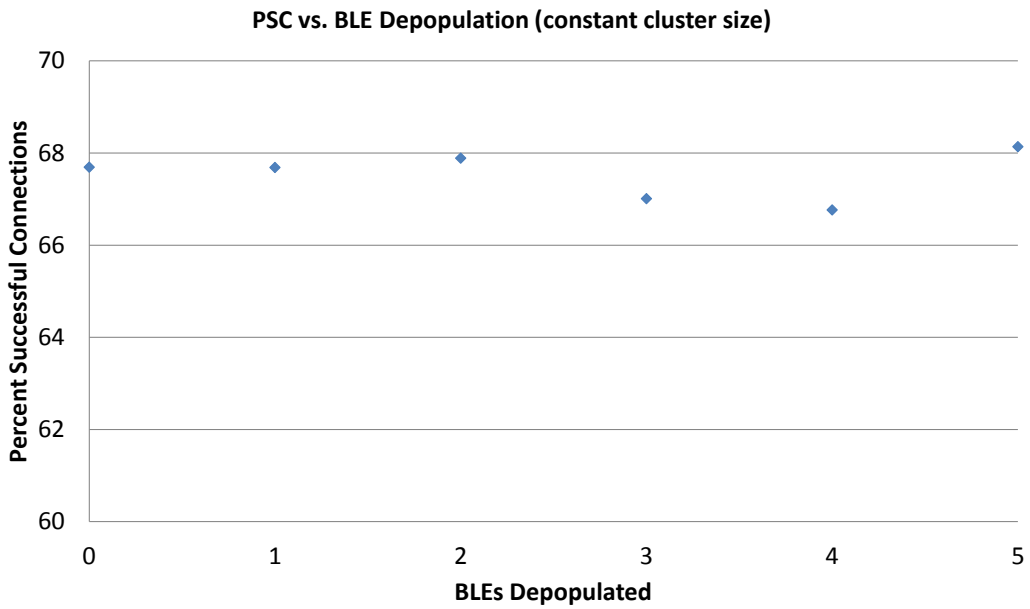

Figure 3.10: PSC vs. Depopulation with constant cluster size

The cost in silicon area of adding an additional input pin to the cluster is lower than adding an additional BLE [1], and the improvement to PSC gained from adding cluster inputs is greater than that gained from BLE depopulation with compensated cluster sizes.

3.7 Summary

This chapter described an algorithm for solving the direct synthesis problem. Given a technology-mapped netlist (the subject circuit) and a set of pre-placed and routed logic clusters and IOs (the target network), we describe how to implement the subject circuit in the target network without modifying the placement or inter-cluster routing of the target network. The following chapter describes the methodology we have designed to create target networks that are feasible for direct synthesis.

[1] Adding an input pin adds less switch points to the cluster input crossbar than adding another 4-input BLE. Adding an input does not require additional silicon area for logic, while adding a BLE obviously does.

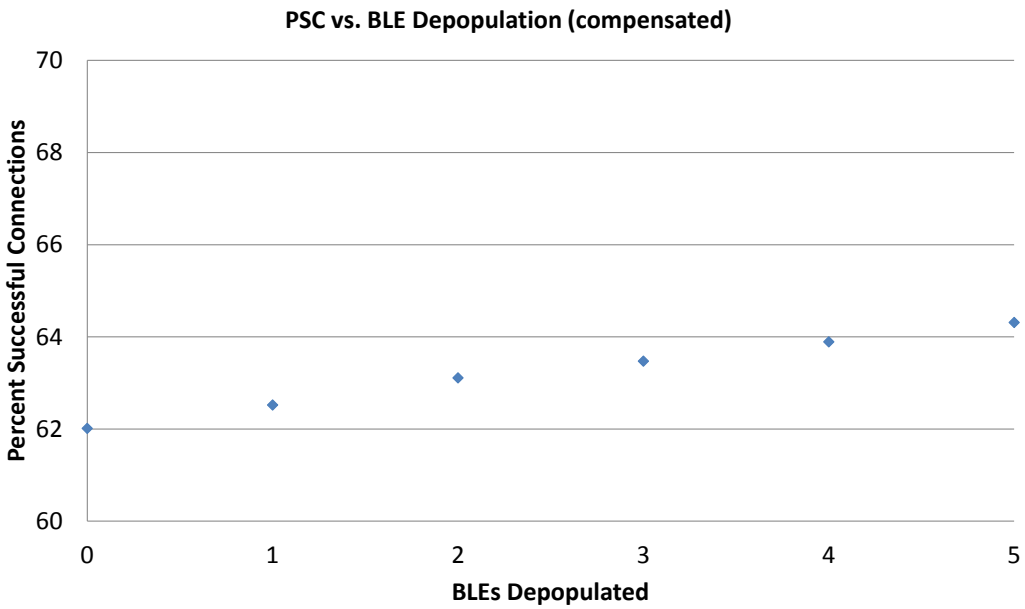

Figure 3.11: PSC vs. Depopulation with compensated cluster size

Chapter 4

Target Network Generation

The previous chapter presented an algorithm for directly synthesizing a subject circuit to a target network. The creation of the target network – the pre-placed and routed FPGA circuit – is the second key component of direct synthesis. The closer the target network is to the subject circuit being mapped, the greater the likelihood of success. In this chapter we describe a methodology for creating target networks with architecture and interconnect topology that are suitable for direct synthesis.

There are two groups of inputs to the network generation process: the parameters of the architecture itself, and characteristics that describe the topology of the target network. The first, the architectural parameters of the underlying FPGA, are the number of inputs to the logic clusters (I), the number of BLEs in a cluster (N), and K, the size of the LUTs. The second are parameters relating to the subject circuits being mapped, that is, their size (the number of clusters (N_c), and a characterization of their interconnect topology (the wire length distribution and fanout distribution).

The output of the network generation process is a netlist consisting of Primary Inputs, Primary Outputs, the required number of clusters in a square aspect ratio, and connections between cluster-level outputs or primary input drivers and the cluster inputs or primary outputs.

In the following sections we proceed to describe the characterization methodology used to capture the interconnect topologies of the circuits we wish to directly synthesize, the algorithm used to generate target networks with similar topologies, and a study of the impact of different topological characteristics on the success of direct synthesis.

4.1 Topology Characterization

Circuits are characterized on two levels: firstly, the number of gates (i.e., logic blocks) and IOs, and, secondly, the topology of the wires that exist between these blocks. While the first form of characterization is trivial, characterizing topology effectively is more challenging.

The most precise characterization of a circuit's topology is the hypergraph representing the circuit itself, but this is infeasible for two reasons: this cannot be used to generate a suitable target network since it requires us to know the subject circuit a-priori, and this characterization can only be used to create target networks of the same size as the characterized circuit. This highlights two general requirements for an effective set of characterization metrics:

- The metrics should be general enough to capture the topology of a large number of circuits by sampling a subset of them.
- The metrics should capture topology characteristics independent of circuit size.

Two metrics have been used to perform this characterization: the fanout distribution [12] and the wire length distribution [30]. The fanout distribution measures, for each value of inter-cluster fanout existing in a packed circuit, the number of inter-cluster nets with that much fanout. An inter-cluster net is a connection between a single source (either the output of a primary input or BLE) and multiple sinks (inputs to clusters and/or a primary output). The fanout of an inter-cluster net is equal to the number of sinks,

excluding any sink BLEs which are local to a source BLE's cluster (only inter-cluster fanout is counted). The wire length distribution measures, for each value of wire length existing in a packed and placed circuit, the number of wires (i.e., source to single sink connection) with that given length. Length is defined as the Manhattan distance from placement location of the source to the placement location of the sink.

Figure 4.1 illustrates the fanout and wire length distributions for a sample circuit that has been packed and placed. These metrics capture the topological characteristics of a circuit by describing both how many sinks a pin drives (via the fanout distribution), and roughly where these sinks are in relation to the source (via the wire length distribution).

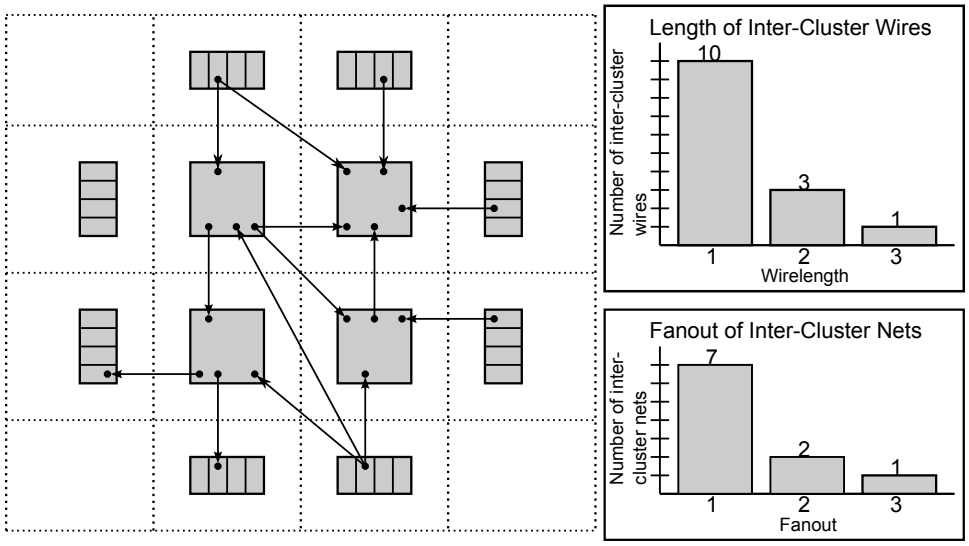

Figure 4.1: Wire length and fanout distributions for a packed and placed circuit

4.1.1 The Fanout and Wire Length Distributions

In order to use these distributions to generate topologies for target networks of different sizes (i.e., different numbers of logic clusters and IOs), the distributions should contain a normalized (or relative) number of fanout/wires, instead of the absolute number (which is what is shown in Figure 4.1). To do so, the fanout and wire length distributions are represented as probability density functions (PDFs), where the random variable repre-

sents either the fanout of a net or length of a wire (in the case of the fanout distribution and wire length distribution, respectively). The probability of observing a net or wire with the given fanout or wire length value is therefore proportional to how frequent such a value would be observed in the circuit (or circuits) that were characterized.

The log-normal distribution was chosen to model both fanout and wire length distributions. This was motivated by the fact that the fanout and wire length distributions of our subject circuits (and, likely, most real circuits) are heavily skewed to the right. Previous works have used the Weibull distribution [30] or derived new distributions based on Rent's rule [35].

Section 4.2 presents the target network generation algorithm, in which sampling these distributions allows the construction of target network routing topologies with fanout and wire length distributions consistent with these characterizations. In Section 4.3 the methodology used to arrive at suitable values for the distributions will be discussed.

4.2 Generation

There are two steps to the target network generation algorithm: instantiation of the I/Os and clusters, and then the generation of all of the inter-cluster source-sink connections.

4.2.1 Target Instantiation and Placement

The first step of instantiation creates N_c logic clusters, each with I inputs and N BLEs of input size K. The number of inputs and outputs is derived from the size and architectural parameters of the target, as follows. IOs are created in groups of virtual IO pads, where the number of IO pads created is equal to $4\sqrt{N_c}$ (the minimum number of IO pads sufficient to surround the periphery of the logic blocks, as seen in Figure 4.2). It is typical to design FPGAs so that the number of pins leaving an IO pad is equal to the number of pins on one side of a logic cluster (so that their demand for routing wires is

similar [14]), so the number of IOs per pad is equal to $\frac{1}{4}(3N + 2)$ (a logic cluster has N outputs and a nominal value of $2N + 2$ inputs). In the event that the value if I is not equal to $2N + 2$ (which is allowed by the algorithm), the number of pins per side of a logic block is still modelled as being equal to $\frac{1}{4}(3N + 2)$. This is done to avoid creating an unreasonably large number of IOs in experiments when we increase I to large values (see Chapter 5). Half of the IOs in each virtual pad are designated as inputs (the other half are outputs). Therefore the total number of inputs and the total number of outputs are both equal to $\frac{1}{2}\sqrt{N_c}(3N + 2)$.

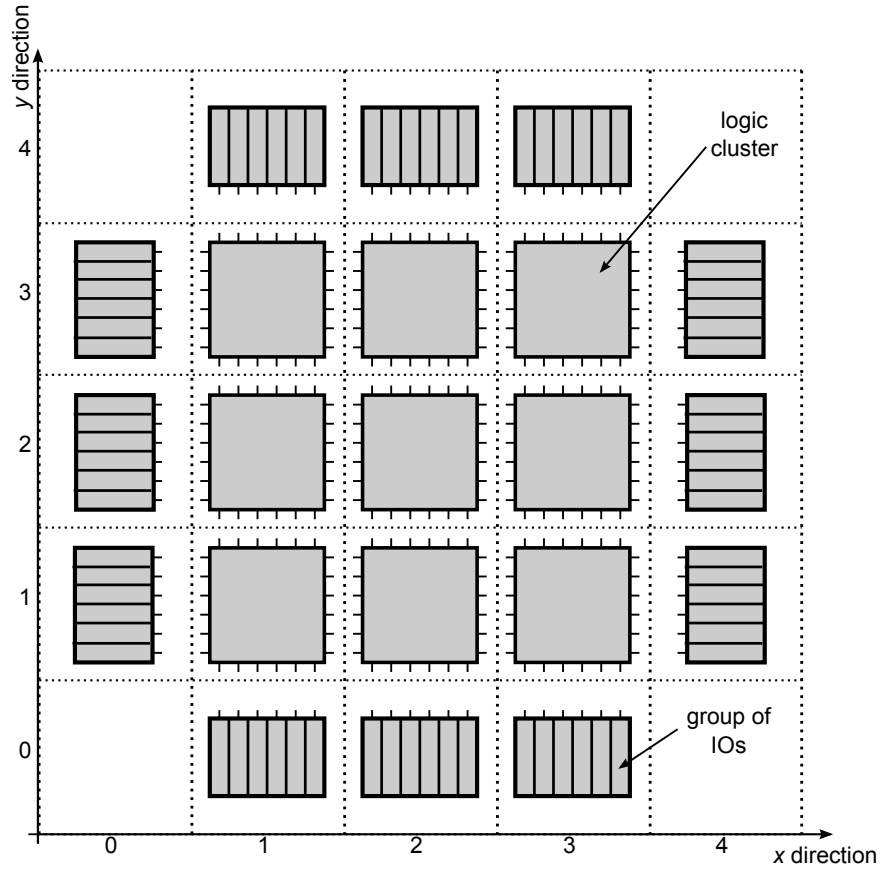

Figure 4.2: A placed target network consisting of unrouted IOs and logic clusters

After instantiating the logic clusters and IO pads, a placement is created, in a grid-like fashion, with logic clusters tiled in the center of the grid with IO pads tiled along the periphery of the grid. Each logic cluster and IO pad is assigned a unique (x, y) location

in this grid, as this will be necessary to generate proper wire length distributions in the next step, below. We refer to the logic cluster or IO pad at an (x, y) location as a *tile*. Note that, at this point, there exist no connections between any logic clusters or IOs. Figure 4.2 illustrates the state of the target network prior to inter-cluster route generation.

4.2.2 Inter-Cluster Route Generation

Using the fanout and wire length distributions, routes are added to the pre-placed target network such that the resulting pre-placed and routed design has a similar fanout and wire length distribution. Connections are added between PIs and cluster inputs, cluster outputs and cluster inputs, and cluster outputs and POs. Note that all these routes are inter-cluster routes; no local cluster routes are made, as these are implemented quickly and easily in the intra-cluster crossbars. Algorithm 2 summarizes the entire process used to generate the fixed inter-cluster routing in the target network, and will be described in detail below.

The first step in route generation is a pre-processing step involving the fanout and wire length distributions. In this step, the necessary number of samples from the fanout and wire length distributions are collected and stored. The number of samples collected from each is equal to the number of signal sources in the target network (i.e., PIs and cluster outputs): $\frac{1}{2}\sqrt{N_c}(3N + 2) + N \times N_c$. When collecting samples from the fanout distribution, any sample which is greater than N_c is rejected, and replaced by a new sample (an inter-cluster net should not have fanout greater than the number of clusters). After all fanout samples have been collected, the entire process is repeated if the sum of samples is greater than the total number of sinks in the target network (i.e., POs and cluster inputs): $\frac{1}{2}\sqrt{N_c}(3N+2)+I \times N_c$. The process does not need to be repeated often, as the mean of the fanout distribution should not be set greater than the average number of sinks per source in the target network. Wire length samples are collected with no such

Algorithm 2: Target network routing generation algorithm

Input: pre-placed target network, fanoutDistribution, wireLengthDistribution

Output: pre-placed and routed target network

```
// Pre-processing step:  get samples from distributions
```

$fanoutValues$ = getFanoutSamples($fanoutDistribution$)

$wireLengthValues$ = getWireLengthSamples($wireLengthDistribution$)

```
// Routing step:
```

foreach *tile in target network,* $tile_{source}$ **do**

  foreach *output pin of* $tile_{source}$, pin_{source} **do**

    fo = getValue($fanoutValues$)

    for $i = 1$ **to** fo **do**

      wl = getValue($wireLengthValues$)

      $tile_{sink}$ = getTile(wl units away from $tile_{source}$)

      pin_{sink} = getRandomInputPin($tile_{sink}$)

      makeRoute(pin_{source}, pin_{sink})

```
// Post-processing step:  drive any un-driven pins
```

foreach *tile in target network with un-driven input pins,* $tile_{sink}'$ **do**

  foreach *un-driven input pin of* $tile_{sink}'$, pin_{sink}' **do**

    $tile_{source}'$ = getNeighbour($tile_{sink}'$)

    pin_{source}' = getOutputPin($tile_{source}'$)

    makeRoute(pin_{source}', pin_{sink}')

constraints, as any outlier wire length values will not be detrimental to the algorithm, nor can the sum total of wire lengths prevent the algorithm from creating a legal target network.

The second step of the algorithm uses these samples to guide the generation of routes.

Routing is added to the target network by routing the output of each signal source in the target network one at a time. To route the output from an output pin of a generic tile (either an IO group or a cluster), first a fanout value is taken from the collection of fanout samples generated in the pre-processing step. This fanout value dictates how many sinks will be routed for this one source. For each sink to be routed, a wire length value is acquired from the collection of wire length samples generated in the pre-processing step. This wire length value is used to determine which tile in the target network will act as the sink. Of all the tiles in the target network that are located at this distance away from the source tile, one is selected randomly with the two constraints the the sink tile must have a free (i.e., un-driven) input pin, and must not have any input pins already driven by the specific source tile output pin. If none exists at that distance, then the search for sink tiles is continually expanded to include tiles at wire length + 1 and wire length - 1, until either a sink tile is found, or it is determined that no suitable tile exists anywhere in the target network. If a sink tile is found, the connection is created from the output pin of the source tile to any one of the free input pins on the sink tile.

The main routing step of the algorithm, just described, is not guaranteed to create routing in which every input pin in the target network is driven by a route. The final step in route generation is a post-processing step designed to rectify this. For each tile with free input pins, referred to as the sink tile, a route will be added to drive each of the free input pins, referred to as sink pins. For each sink pin, a tile neighbouring the sink tile is randomly selected, with the constraint that at least one of its output pins does not already drive the sink tile. Fanout is then added to one of these output pins (randomly selected from amongst those that do not drive the sink tile) by adding a connection from that output pin to the sink pin. This process is repeated for every free input pin. While this post-processing step also is not guaranteed to produce target networks where every input pin is driven, the set of circumstances necessary for this to occur have been very rare in practice. Generally, this process results in the creation of target networks where

all input pins are driven.

4.3 Effective Topologies for Target Networks

Given that the topology of the target network's routing is fixed once created, it is necessary for the success of direct synthesis that it is similar to that of the subject circuit being mapped to it. As such, the shape of the fanout distribution and wire length distribution must be carefully selected.

Two different methodologies will be used to arrive at an effective topology, and their efficacy compared. In the first, the fanout and wire length distributions will be constructed such that they fit the fanout and wire length distributions measured from a training set of subject circuits. The training circuits are the same used in Chapter 3, Table 3.2. In the second, a range of parameter values (close to what was found in the initial fitting) will be tested to construct many pairs of fanout and wire length distributions. Each pair will then be used to construct a target network to be used in the synthesis of the training circuits. The pair of distributions that produces target networks that yield the highest average PSC when mapping the same training circuits will be then be used in this research.

4.3.1 Training the Distributions to Training Data

A log-normal distribution is defined by two parameters, μ and σ, which are, respectively, the mean and standard deviation of the underlying normal distribution [1]. The task of fitting a log-normal distribution to a set of data amounts to finding estimates for μ and σ. There are two sets of data to be fit: the set of all inter-cluster fanout values, measured over all the training circuits, and the set of all the wire length values, also measured over all the

[1] μ and σ are, respectively, the mean and standard deviation of the natural logarithm of the random variable (e.g., wire length). If $wirelength$ is log-normally distributed, $\ln(wirelength)$ is normally distributed with mean μ_{wl} and standard deviation σ_{wl}

training circuits. These sets of data were measured after first packing each training circuit with T-VPack (to a N=10, K=4, I=22 architecture), and then placed using VPR5 [21]. Given the set of n samples (each denoted by x) the maximum likelihood estimators for μ and σ of the associated log-normal distributions were computed as follows [32]:

$$\widehat{\mu} = \frac{\sum_{k=1}^{n} \ln x_k}{n}, \widehat{\sigma} = \sqrt{\frac{\sum_{k=1}^{n} (\ln x_k - \widehat{\mu})^2}{n}} \tag{4.1}$$

Table 4.1 presents the parameter values computed for the fanout and wire length distributions, where the subscripts fo and wl refer to parameters for the fanout and wire length distributions, respectively.

Table 4.1: Parameters for the fanout and wire length distributions derived from training

Fanout Distribution	**Wire Length Distribution**
$\widehat{\mu}_{fo} = 0.347$	$\widehat{\mu}_{wl} = 1.417$ ($mean_{wl} = 5.36$)
$\widehat{\sigma}_{fo} = 0.661$	$\widehat{\sigma}_{wl} = 0.725$

The parameters in Table 4.1, except $\widehat{\mu}_{fo}$, were then used to construct target networks. Instead of using the value of $\widehat{\mu}_{fo}$, a value of μ_{fo} was used such that the mean of the fanout distribution would be equal to the average fanout to be observed in the target network such that all cluster-level sources drive a signal and all cluster-level sinks in the target network were driven. Since the total number of sources (PIs and cluster outputs) and the total number of sinks (cluster inputs and POs) for a target network are known, the mean of the fanout distribution should scaled such that the average fanout observed in the target network is equal to $\frac{numSinks}{numSources}$. Given a value for σ_{fo} and a mean fanout value, the μ_{fo} parameter is computed as follows:

$$\mu_{fo} = \ln\left(\frac{\#POs + N_c \times I}{\#PIs + N_c \times N}\right) - \frac{\sigma_{fo}^2}{2} \tag{4.2}$$

Target networks were created with an $N = 10$, $K = 4$, $I = 40$ architecture (where N_c was set to the smallest value sufficiently large to fit each training circuit). Each

subject circuit from the training set was then synthesized to an appropriately sized target network. Three synthesis attempts were performed for each subject circuit, each with three different random seeds. The settings used in the direct synthesis are summarized in Table 4.2. This topology produced an average PSC (arithmetic average over all runs of every circuit) of 60.4. Table 4.3 shows how average PSC varies across each of the 8 subject circuits from the training set.

Table 4.2: Direct Synthesis settings used in Chapter 4

Direct Synthesis Parameter	**Value**
c	4
γ	{0.5, 0.7, 0.95, 0.85}
Move Factor	6
ε	0.0005
Route-throughs	ON
depopulation	0

Table 4.3: Results of direct synthesis for a target network topology trained to training circuits

Subject Circuit	**Average PSC**
apex3	58.9
apex4	53.6
C6288	76.2
dalu	82.2
diffeq	48.7
misex3	57.2
seq	51.6
tseng	54.3
Average	**60.4**

4.3.2 Sweeping the Topological Parameters

In the hopes of producing a better target network topology than that which was derived from training, fanout and wire length distributions were created by sweeping the respec-

tive μ and σ parameters through a range of values. The values used for each parameter are presented in Table 4.4. Note, firstly, that μ_{fo} is always set to the appropriate value such that the mean fanout is always sufficient to drive all sink pins in the target network (computed using Equation 4.2). Secondly, that instead of presenting μ_{wl}, instead the associated *mean* of the wire length distribution is presented. This is simply to provide better context for the reader, as the mean describes the average wire length that will be observed in the target network, where μ_{wl} instead is the mean of the underlying normal distribution.

Table 4.4: Distribution parameters used to create experimental target networks

Distribution Parameter	**Values**
μ_{fo}	Computed using Equation 4.2
σ_{fo}	$\{1.2, 1.4, 1.6, 1.8, 2.0\}$
$mean_{wl}$	$\{3.0, 3.5, 4.0, 4.5, 5.0\}$
σ_{wl}	$\{0.8, 1.0, 1.2\}$

A target network was created for each of the 75 unique combinations of σ_{fo}, $mean_{wl}$, and σ_{wl} in Table 4.4. All target networks were created with an $N = 10$, $K = 4$, $I = 40$ architecture (where N_c was set to the smallest value sufficiently large to fit each training circuit). Each subject circuit from the training set was then synthesized to an appropriately sized target network. Three synthesis attempts were performed for each subject circuit, each with three different random seeds. The settings used in the direct synthesis are summarized in Table 4.2.

Of the 75 target networks created, the target network that produced the highest average PSC was generated with the set of distribution parameters summarized in Table 4.5.

Table 4.5: Parameters for the best fanout and wire length distributions derived from experimentation

Fanout Distribution	**Wire Length Distribution**
μ_{fo} Derived via Equation 4.2	$\mu_{wl} = 1.18$ ($mean_{wl} = 4.5$)
$\sigma_{fo} = 1.6$	$\widehat{\sigma}_{wl} = 0.8$

The best topology produced an average PSC (arithmetic average over all runs of every circuit) of 65.5. Table 4.6 shows how average PSC varies across each of the 8 subject circuits from the training set.

Table 4.6: Results of direct synthesis for a target network topology derived from experimentation

Subject Circuit	**Average PSC**
apex3	65.3
apex4	59.5
C6288	73.3
dalu	82.9
diffeq	59.4
misex3	60.4
seq	58.2
tseng	65.4
Average	**65.5**

4.3.3 Comparison

The set of parameters found through experimentation produced target networks that yielded an average PSC (when synthesizing the training circuits) which was 8.4% higher than those derived from training. That experimentation yielded a better result than training is likely due to the fact that the training procedure is based on the premise that producing target networks with similar fanout and wirelength as the subject circuits produces target networks that result in higher PSC, where in the parameter sweeping procedure the distribution parameters are being varied in order to directly optimize for PSC.

Referring to Table 4.7, one can see that the set of parameters derived from experimentation produces target networks with, on average, shorter wires, but with more variance in wire length. The mean fanout was equal, by construction, but the variance in fanout is the major difference between the two sets. This suggests that a greater variance in

Table 4.7: Comparison of distribution parameters derived from training and experimentation

Parameter	Training	Experimentation
μ_{fo}	-	-
σ_{fo}	0.661	1.6
μ_{wl}	1.417	1.18
σ_{wl}	0.725	0.8

inter-cluster fanout produces better target networks. This could be due to the fact that not all cluster outputs drive inter-cluster fanout; it is more efficient to have the some cluster outputs be assigned a high fanout at the cost of creating cluster outputs with little (and even some with zero) fanout. This then allows the assignment optimizer to map subject nodes with completely local fanout (i.e., intra-cluster) to locations in the target with little (or no) inter-cluster fanout without penalty, while mapping high-fanout subject nodes to high-fanout locations in target network. This extra fanout ('extra' with respect to the fanout observed in the traditionally-packed subject circuit) increases 'connectedness' in the target network, which is shown to improve the quality of direct synthesis in Chapter 5.

4.4 Summary

This chapter presented the method used to capture the topology of subject circuits and an algorithm for creating target networks with inter-cluster routing of a similar topology, to maximize the success of direct synthesis. The following chapter will show under what conditions direct synthesis can be made to succeed.

Chapter 5

Results

The previous two chapters presented the two constituent tools that were designed to solve the direct synthesis problem: an algorithm for mapping a subject circuit to a target network (Chapter 3), and an algorithm for creating target networks suitable for direct synthesis (Chapter 4). In this chapter a set of experiments are performed to illustrate the capabilities of the direct synthesis algorithm and target network generation. These capabilities are being assessed in order to answer the following questions:

1. Under what conditions can direct synthesis be performed successfully?

2. How can direct synthesis be made more successful?

5.1 Methodology

In the following sections, direct synthesis will be performed on a variety of subject circuits and target networks. The settings used in all invocations of the direct synthesis algorithm in this chapter are listed in Table 5.1. All of these settings, save for the Move Factor, are the suggested settings presented in Chapter 3, Section 3.6. The move factor is instead set to 10× the value which provides the best quality/runtime trade-off, in order to improve the chances of successful direct synthesis (while still keeping runtimes feasible).

Table 5.1: Direct Synthesis settings used throughout Chapter 5

Direct Synthesis Parameter	**Value**
c	4
γ	{0.5, 0.7, 0.95, 0.85}
Move Factor	6
ε	0.0005
Route-throughs	ON
depopulation	0

The target networks used during direct synthesis have been created with an $N = 10$ BLEs per cluster, $K = 4$-input LUT architecture. The number of inputs per cluster, I, is varied within the experiments as described in the upcoming sections, and will be reported along with the presentation of each experiment. The number of clusters created for each target network is always the minimal number sufficient to fit the subject circuit which is being synthesized (in a square aspect ratio). All target networks are created with the distribution parameters listed in Table 5.2. Note that μ_{fo} always varies with the architectural parameters of the target network, as discussed in Chapter 4, Section 4.3.

Table 5.2: Parameters used to generate Target Networks throughout Chapter 5

Distribution Parameters	**Architecture Parameters**
μ_{fo} = -	N_c = [varies]
$\sigma_{fo} = 1.5$	$N = 10$
$\mu_{wl} = 1.109$	$K = 4$
$\sigma_{wl} = 1.0$	I = [varies]

The following sections will now investigate the performance of direct synthesis.

5.2 Quality vs. Subject Circuit Size

We expect that successful direct synthesis is more likely with smaller circuits, so in a first experiment we directly synthesize subject circuits of different sizes (beginning with small ones) to target networks created using our flow. Each subject is synthesized

to a minimally-sized target network, each constructed with 22 inputs per cluster, as recommended by [4]. The subject circuits are the 157 circuits from the MCNC benchmark suite [33] that are under 500 BLEs in size when packed to a $N = 10$, $K = 4$, $I = 22$ architecture using T-VPack from VPR 5.0 [22].

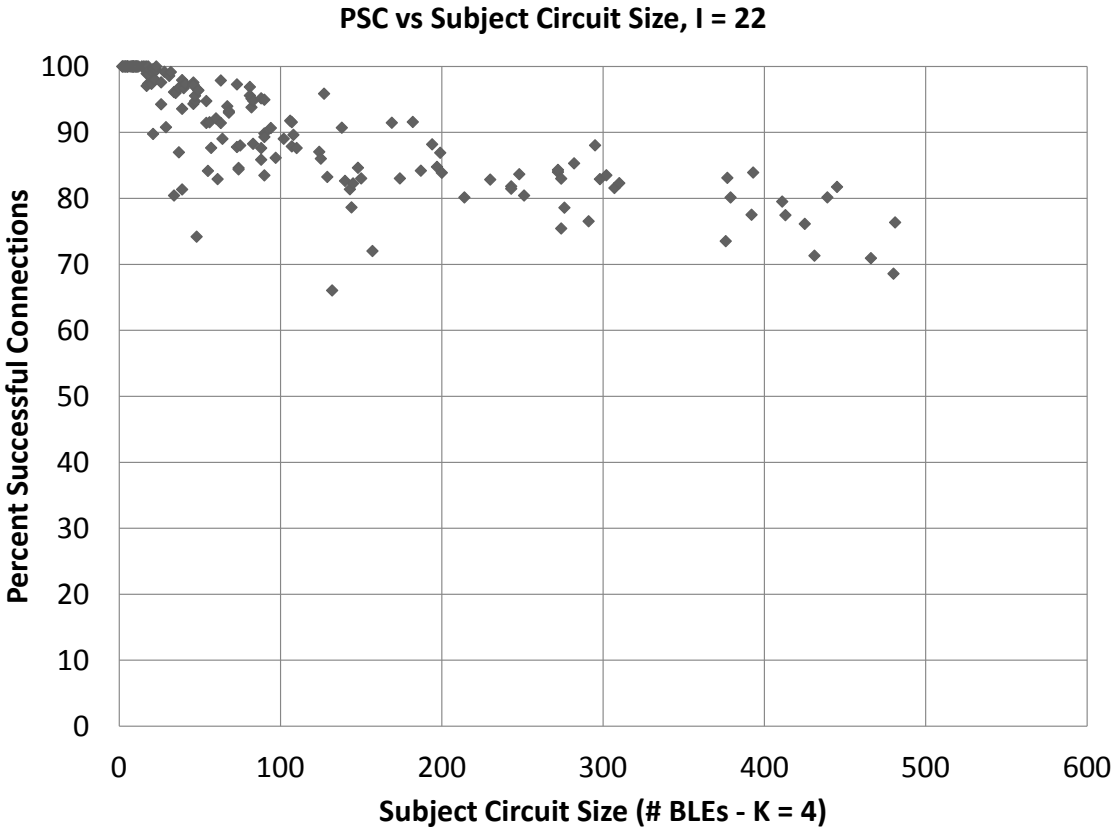

Figure 5.1: Quality of direct synthesis for subject logic circuits of increasing size.

Figure 5.1 is a plot of PSC versus subject circuit size, where each data point represents the resulting average PSC (arithmetic average of three seeds sweeps) for a single subject circuit after direct synthesis to a single target network. The direct synthesis is successful (i.e., PSC = 100) for all subject circuits up to (and including) 17 BLEs in size, where the largest single subject that was successfully synthesized was 23 BLEs in size.

The general trend is that PSC decreases with increasing subject circuit size. One reason for this is that the 'connectedness' of the target network, the average ratio of

clusters in the target network that a single cluster output has pre-routed connections to, drops with increasing size of the target network. To show this, recall from Chapter 4 that the average inter-cluster fanout of a cluster output is equal to

$$\frac{\frac{1}{2}\sqrt{N_c}(3N+2)+N_c\times I}{\frac{1}{2}\sqrt{N_c}(3N+2)+N_c\times N} \tag{5.1}$$

where N_c is the total number of clusters in the target network. The connectedness of a target network is therefore equal to the term in 5.2 divided by N_c, which simplifies to

$$\frac{2\sqrt{N_c}I+3N+2}{N_c\left(\left(2\sqrt{N_c}+3\right)N+2\right)} \tag{5.2}$$

This product will decrease in value as N_c increases in value, illustrating that connectedness drops with size of the target network, as shown in Figure 5.2.

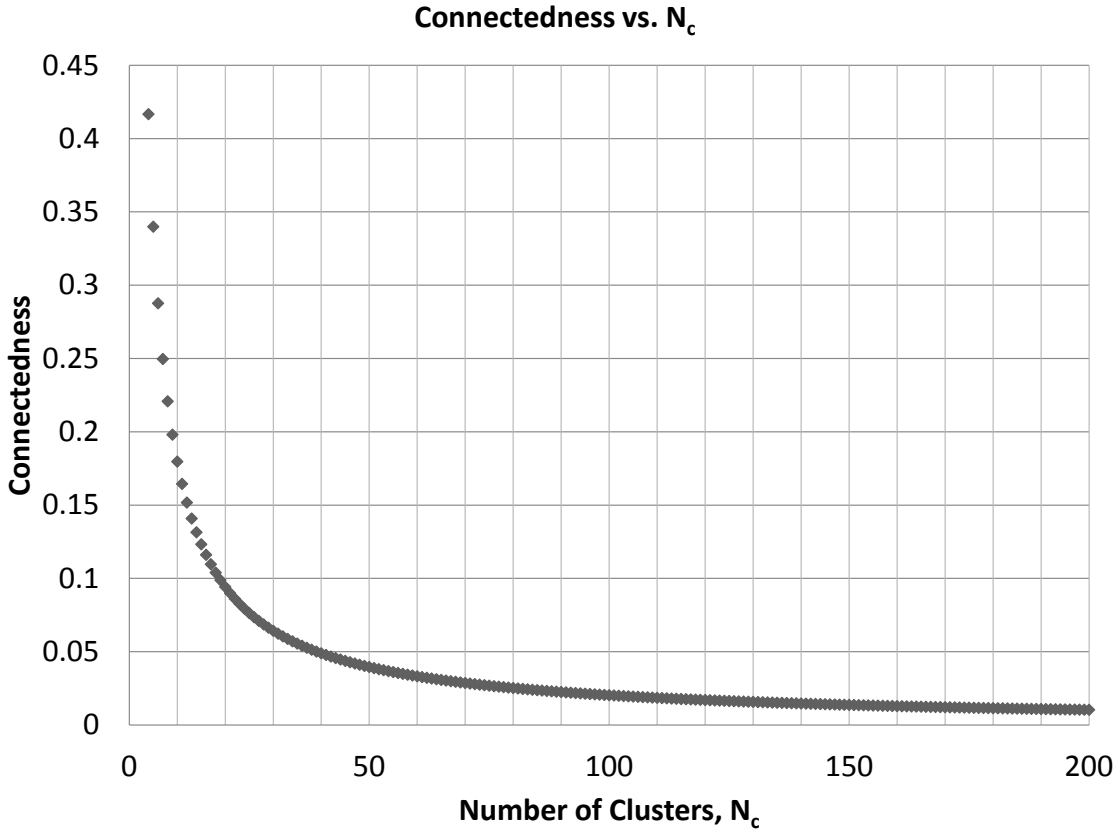

Figure 5.2: Connectedness versus cluster size, for an $N = 10$, $I = 22$ architecture.

Connectedness influences the success of direct synthesis because connectedness is essentially a measure of the fraction of the target network area in which a subject circuit node must be mapped in order to form a legal connection to one its driving nodes. Decreased connectedness results in a situation in which there are less legal configurations for a given subject circuit in the target network, thereby decreasing the probability that the optimizer can find a legal solution to the direct synthesis problem.

Connectedness decreases with the size of the target network, but it does increase with the number of inputs per cluster in the target network. Increasing I should therefore results in more successful direct synthesis. To test this, we repeat the experiment, using the same subject circuits, but synthesize them to target networks constructed with $I = 40$ inputs per cluster. This value of I is the largest number of inputs for a cluster containing 10 4-input LUTs that we consider reasonable (a value of I greater than $N \times K$ is definitely not reasonable).

Figure 5.3 shows the results of this experiment. Performing direct synthesis using target networks with $I = 40$ improved the quality of direct synthesis, as evident by the larger subject circuits which are successfully synthesized. All subject circuits up to 47 BLEs were successfully mapped (vs. 17 BLEs with $I = 22$), where the largest single subject that was successfully synthesized was 97 BLEs in size (vs. 23 BLEs with $I = 22$).

To understand how the quality of direct synthesis improves with I, a more comprehensive exploration of I values will be performed in the next section.

5.3 Quality vs. I of the Target Networks

The previous experiment indicated that mapping success improves with number of inputs to the cluster (I) in the target network. In this section, we 'relax' the architectural parameter I to become large, in order to investigate what it might take to achieve success for larger circuits. While we don't view larger values of I as realistic, architecturally, we

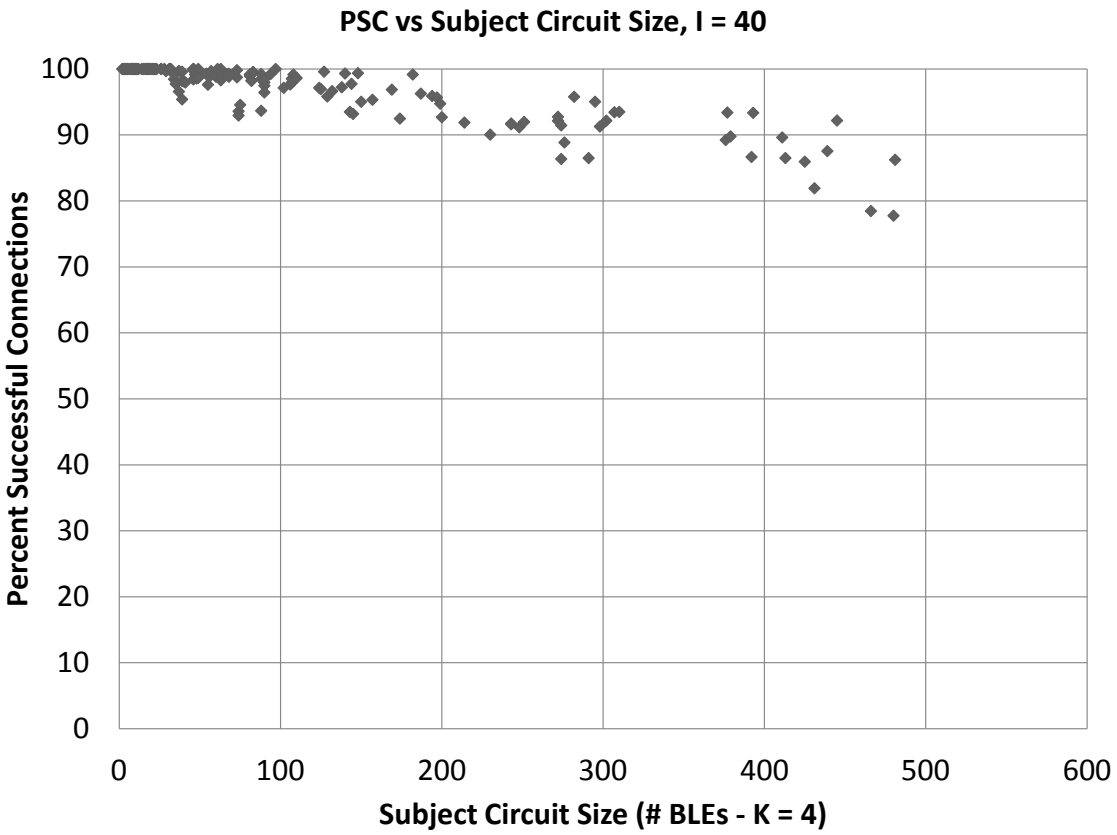

Figure 5.3: Quality of direct synthesis for subject logic circuits of increasing size.

feel it will give an indication of the required amount of connectedness required for success. This connectedness may be obtained through more realistic methods, such as exposing some of the routing multiplexers to the direct synthesis process. In this experiment we sweep I of the target network from a value of 20 to 300, in increments of 10. In these experiments we use a set of larger subject circuits with similar sizes to the original training set that was used to tune both the direct synthesis and target network generation algorithms. Table 5.3 describes the 9 MCNC circuits used as subjects circuits for this experiment, which we refer to as the validation set of subject circuits.

Figure 5.4 gives the result of the experiment (with the PSC averaged across a single seed sweep for each of the 9 validation circuits). PSC improves as the number of cluster inputs increases, as expected. The minimum value of I required to successfully synthesize

Table 5.3: Validation set of subject circuits

Ciruit	Number of 4-LUTs	Number of Primary Inputs	Number of Primary Outputs
alu4	1522	14	8
apex1	700	45	45
apex2	1878	38	3
apex5	535	117	88
C5315	620	178	123
C7552	739	207	107
cps	757	24	109
ex5p	1064	8	63
s298	1930	4	6

each subject circuit is listed in Table 5.4, where the circuits are listed in increasing order of size. For the three largest circuits which did not achieve success within the range of 20-300 cluster inputs, the highest value of PSC attained is listed. This supports, again, the conclusion drawn from the first experiment that direct synthesis is more difficult to achieve for larger circuits.

Another interesting observation can be made: despite the fact that an unrealistic number of cluster inputs are required for a successful mapping, the average number of *used* cluster inputs is constant across various values of I (roughly 13 inputs, on average, are ever used). This leads us to believe that instead of increasing target network connectivity by adding input pins, one could simply add multiplexers to the cluster input pins to achieve the same state of high connectivity. Since only a small number of inputs are actually used, the large number of incoming signals could negotiate for each cluster input pin. Modifications to the router (i.e., negotiated congestion-based routing) used in the direct synthesis algorithm could make this possible. This remains something we wish to explore in future work. The insights we will be looking for will be to add in the minimum amount of flexibility to achieve routability, which might well be less than the full flexibility of the FPGA.

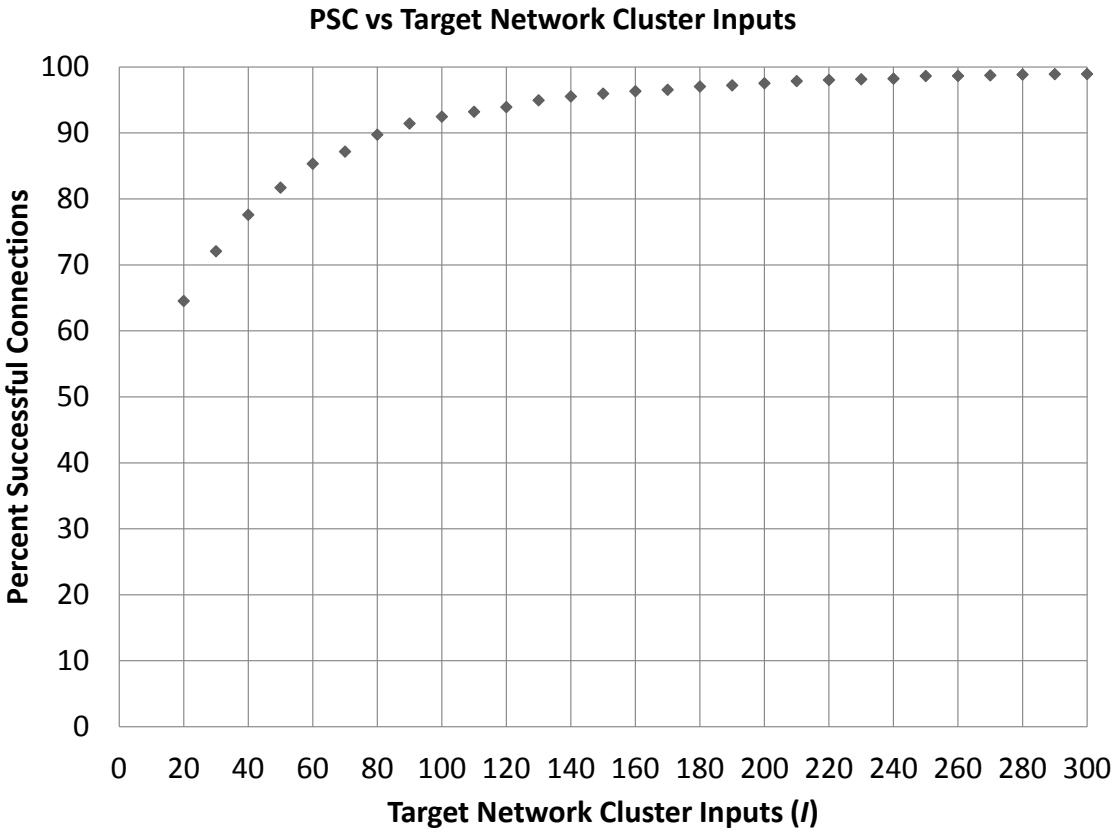

Figure 5.4: Quality of direct synthesis for target networks of increasing I.

5.4 Assessing the Performance of the Direct Synthesis Algorithm

The quality of the mapping produced by the direct synthesis flow is dependent upon the algorithm used to perform the mapping and the target network provided. In this experiment we attempt mappings with target networks for which we know there exists at least one legal configuration for a given subject circuit. With such a target network we can gain some understanding of how well the direct synthesis algorithm can effectively find solutions.

In this experiment we use the same 9 MCNC benchmark circuits listed in Table 5.3. For each subject circuit, we create a target network that the subject circuit is guaran-

Table 5.4: Minimum I for a successful mapping of each benchmark circuit

Circuit	I_{min}
apex5	160
C5315	220
apex1	230
C7552	200
cps	230
ex5p	300
alu4	(PSC = 99 at $I = 300$)
apex2	(PSC = 93 at $I = 300$)
s298	(PSC = 97 at $I = 300$)

teed to legally map to by directly determining that target through the regular synthesis process. That is, we pack each subject circuit to an $N = 10$, $K = 4$, $I = 22$ architecture using T-VPack from VPR 5.0 [21]. We then create a target network, called a *clone*, that has the same number of inputs, outputs and clusters as this packed circuit, with identical inter-cluster routing. Therefore the packing of the subject circuit produced by T-VPack is one existing configuration that is guaranteed to be a legal mapping to the clone. We should note, though, that there is less connectivity in general in the clones, compared to the target networks generated, as the target generation sought to make use of all available cluster input pins. Each subject circuit is then mapped to its clone using the direct synthesis flow. We also map each subject circuit to a target network created using our automated flow. The target networks were generated with $I = 22$.

Figure 5.5 shows the resulting PSC of the mappings for this experiment, which shows the direct synthesis algorithm is unable to find the solution. This implies that we should seek improvements in the algorithm, although it may indeed be challenging to find the single point solution. We also note that the generated target networks, which in general have more connectivity, achieve a lower PSC for every subject circuit except apex5, C5315, and C7552. Since these 3 subject circuits have been mapped to standard target networks with many extra clusters, and thus BLEs, (as a result of being IO-bound

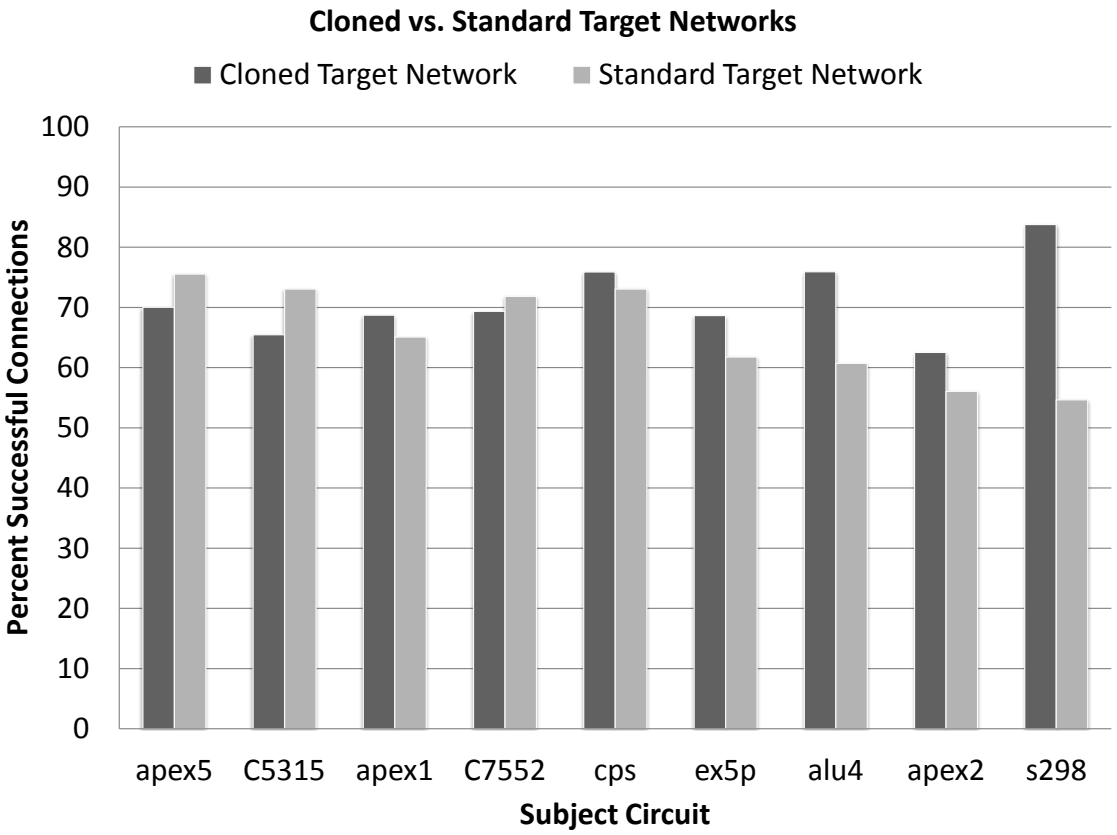

Figure 5.5: PSC of validation circuits mapped to cloned and standard target networks

circuits), we suspect that the better synthesis results when mapping to the standard networks is due to the use of route- throughs. The cloned target networks, being a mirror image of the packed circuit, do not contain any additional clusters beyond what are necessary.

To test this hypothesis, this experiment was repeated with the same set of inputs and parameters, but with the use of route-throughs disabled. Figure 5.6 shows the results of the experiment. Without the use of route-throughs, none of the 9 subject circuits achieve better direct synthesis when mapped to the standard target networks.

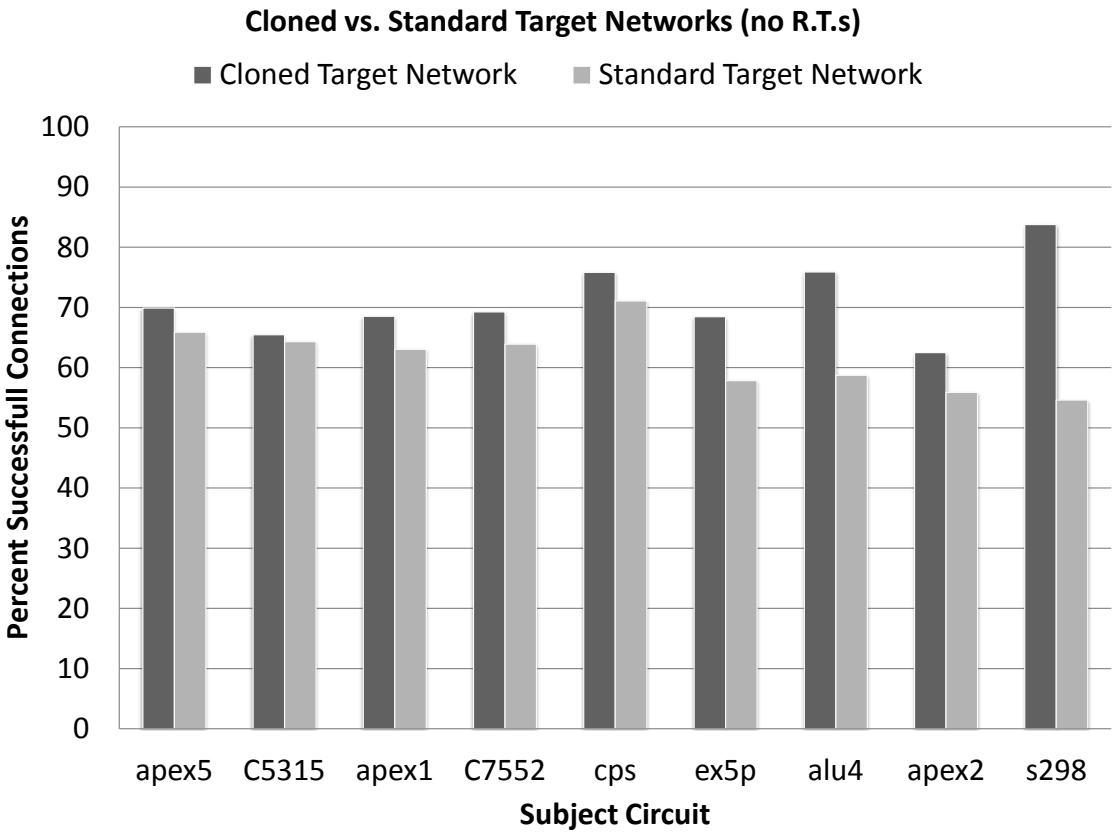

Figure 5.6: PSC of validation circuits mapped to cloned and standard target networks

5.5 Routability of the Target Networks

For the direct synthesis flow to be feasible, the target networks that are generated must be able to be compiled onto an FPGA as an overlay circuit. We know, at generation time, the logic overhead of a target network because we generate it with a known number of clusters and IOs. What is unknown, however, is the routing demand placed upon the FPGA compared to that of the subject circuit if it were compiled directly to the FPGA (without the overlay). To study this we have placed and routed the 9 benchmark circuits in Table 5.3 using VPR 5.0 [21] with the sample architecture set as $N = 10$, $K = 4$, $I = 22$ (identical to the target networks). The minimum-size target networks generated to map each subject circuit were also were also packed and placed using the same tool and architecture. The minimum channel width required in a minimally-sized FPGA for the

successful routing of all circuits and overlays is presented in Table 5.5.

Circuit	W_{min} **of circuit**	W_{min} **of associated target**
alu4	36	60
apex1	44	50
apex2	50	60
apex5	28	60
C5315	30	60
C7552	28	60
cps	34	50
ex5p	52	60
s298	34	60

Table 5.5: Minimum channel width required to route subject circuits and target networks

All of the subject circuits required a lower channel width than their associated target network. The fact that the target networks exhibit a higher routing demand is likely due to the fact the every single cluster input in the target network is driven by a net (by design). This is not true for the subject circuits, where the average number of used cluster inputs is lower than 22.

5.6 Conclusion

The direct synthesis problem proves to be a difficult challenge for any subject circuit of non-trivial size. Relaxing the architecture to the highest point we consider feasible (i.e., $I = 40$ for a $N = 10$, $K = 4$ architecture), we are able to map subject circuits under 100 BLEs in size. Relaxing the architectural constraints further, by adding more logic cluster inputs, is necessary for successful mapping of larger subject circuits. Since adding cluster inputs effectively raises average fanout, one key to success appears to be connectivity of the clusters in the target network.

Chapter 6

Conclusion

In this thesis we have posed a new kind of synthesis problem, *direct synthesis*, which seeks to synthesize directly from a technology-mapped netlist, called a *subject circuit*, into a placed and routed FPGA overlay, referred to as a *target network*. We are motivated to do this because it could shed light on a new way to attack the rising compile time issue and also provide insight into the needs for flexibility in FPGAs. In order to explore the possibility of direct synthesis, two tools were designed. The first was an algorithm for performing the direct synthesis of a subject circuit to a target network, and the second was a tool for the automated construction of target networks with parameterizable architectures.

It is shown the direct synthesis problem proves to be a difficult challenge for any subject circuit of non-trivial size. Relaxing the architectures of the target networks to the greatest extent we consider feasible (i.e., $I = 40$ for a $N = 10$, $K = 4$ architecture), we were able to successfully map subject circuits under 100 BLEs in size. Through the derivation of a property of target networks that we term *connectedness*, we present an argument suggesting that the difficulty of the direct synthesis is related to connectedness. This was then supported by showing that as connectedness is increased, by adding additional cluster inputs to the target networks, successful mapping of larger circuits is possible.

6.1 Future Work

We see two main directions of future work seeking to address ways to improve the success of direct synthesis: increasing connectedness of the target networks by incorporating multiplexers back into the routing, and a system for providing multiple target networks to choose from at synthesis time

6.1.1 Adding Flexibility to the Pre-Routed Target Networks

It appears that the removal of all routing flexibility makes the direct synthesis problem too difficult for all but the smallest circuits. Incorporating multiplexers, carefully, back into the routing of the target network such that we can increase connectedness without scaling I to infeasible values is one method of improving the success of direct synthesis. Of course, the decision of how many routing multiplexers to add is a deep one; adding many multiplexers may result in the target networks appearing like an FPGA with a traditional routing architecture.

6.1.2 Multiple Target Networks

We believe that by providing a multitude of target networks (which are 'free' to compile, offline), target networks could be made which are better matched to the topology of the subject circuits, and thus would perform better in direct synthesis. In our experimentation, entire sets of circuits were mapped to target networks with the same topology, despite the fact that the circuits did not have the same topologies. This forced the target network topologies to be designed such that the direct synthesis results of the entire group would be maximized, potentially at the cost of subject circuits with characteristics which deviate from the group.

Consider, for example, the comparison of an I/O bound subject circuit and a logic bound subject circuit. The two circuits have a different number of I/Os relative to logic,

and a different demand for routing dedicated to I/Os vs. cluster-to-cluster connections. Each subject circuit would benefit from a more customized target network being available for direct synthesis, instead of a single target network which trades off on these two competing characteristics.

Bibliography

[1] IEEE Standard Verilog Hardware Description Language. *IEEE Std. 1364-2001*, 2001.

[2] IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. *IEEE STD 1800-2009*, pages 1–1285, 2009.

[3] IEEE Standard VHDL Language Reference Manual. *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pages c1–626, 2009.

[4] Vaughn Betz and J. Rose. How much logic should go in an FPGA logic block. *Design Test of Computers, IEEE*, 15(1):10–15, 1998.

[5] Vaughn Betz, Jonathan Rose, and Alexander Marquardt, editors. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[6] Huimin Bian, Andrew C. Ling, Alexander Choong, and Jianwen Zhu. Towards scalable placement for FPGAs. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '10, pages 147–156, New York, NY, USA, 2010. ACM.

[7] Robert Brayton and Alan Mishchenko. ABC: An Academic Industrial-Strength Verification Tool. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer*

Aided Verification, volume 6174 of *Lecture Notes in Computer Science*, pages 24–40. Springer Berlin Heidelberg, 2010.

[8] J. Cong and Y. Ding. On area/depth trade-off in lut-based fpga technology mapping. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(2):137–148, 1994.

[9] J. Darnauer and Wayne Wei-Ming Dai. A Method for Generation Random Circuits and Its Application to Routability Measurement. In *Field-Programmable Gate Arrays, 1996. FPGA '96. Proceedings of the 1996 ACM Fourth International Symposium on*, pages 66–72, 1996.

[10] M. Gort and J.H. Anderson. Analytical placement for heterogeneous FPGAs. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 143–150, 2012.

[11] Eddie Hung and Steven J.E. Wilton. Towards simulator-like observability for FPGAs: a virtual overlay network for trace-buffers. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '13, pages 19–28, New York, NY, USA, 2013. ACM.

[12] M. Hutton, J. P. Grossman, J. Rose, and D. Carneil. Characterization and parameterized random generation of digital circuits. In *Design Automation Conference Proceedings 1996, 33rd*, pages 94–99, 1996.

[13] P. Jamieson, K.B. Kent, F. Gharibian, and L. Shannon. Odin II - An Open-Source Verilog HDL Synthesis Tool for CAD Research. In *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, pages 149–156, 2010.

[14] Peter Andrew Jamieson. *Improving the Area Efficiency of Heterogeneous FPGAs with Shadow Clusters*. PhD thesis, University of Toronto, 2007.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[16] B.S. Landman and Roy L. Russo. On a Pin Versus Block Relationship For Partitions of Logic Graphs. *Computers, IEEE Transactions on*, C-20(12):1469–1479, 1971.

[17] C. Lavin, B. Nelson, and B. Hutchings. Improving Clock-Rate of Hard-Macro Designs. In *Proceedings of the 2013 International Conference on Field-Programmable Technology (FPT)*, pages 246–253, 2013.

[18] C. Y. Lee. An Algorithm for Path Connections and Its Applications. *Electronic Computers, IRE Transactions on*, EC-10(3):346–365, 1961.

[19] Adrian Ludwin and Vaughn Betz. Efficient and Deterministic Parallel Placement for FPGAs. *ACM Trans. Des. Autom. Electron. Syst.*, 16(3):22:1–22:23, June 2011.

[20] Jason Luu, Jason Helge Anderson, and Jonathan Scott Rose. Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and Complex Interconnect. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, pages 227–236, New York, NY, USA, 2011. ACM.

[21] Jason Luu, Ian Kuon, Peter Jamieson, Ted Campbell, Andy Ye, Wei Mark Fang, and Jonathan Rose. VPR 5.0: FPGA CAD and Architecture Exploration Tools with Single-Driver Routing, Heterogeneity and Process Scaling. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '09, pages 133–142, New York, NY, USA, 2009. ACM.

[22] Alexander (Sandy) Marquardt, Vaughn Betz, and Jonathan Rose. Using cluster-based logic blocks and timing-driven packing to improve fpga speed and density. In *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field*

Programmable Gate Arrays, FPGA '99, pages 37–46, New York, NY, USA, 1999. ACM.

[23] L. McMurchie and C. Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Field-Programmable Gate Arrays, 1995. FPGA '95. Proceedings of the Third International ACM Symposium on*, pages 111–117, 1995.

[24] A. Mishchenko, S. Chatterjee, and R.K. Brayton. Improvements to Technology Mapping for LUT-Based FPGAs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):240–253, 2007.

[25] J. Pistorius, E. Legai, and M. Minoux. PartGen: a generator of very large circuits to benchmark the partitioning of FPGAs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(11):1314–1321, 2000.

[26] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli. Architecture of Field-Programmable Gate Arrays. *Proceedings of the IEEE*, 81(7):1013–1029, 1993.

[27] Jonathan Rose, Jason Luu, Chi Wai Yu, Opal Densmore, Jeffrey Goeders, Andrew Somerville, Kenneth B. Kent, Peter Jamieson, and Jason Anderson. The VTR project: architecture and CAD for FPGAs from verilog to routing. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, FPGA '12, pages 77–86, New York, NY, USA, 2012. ACM.

[28] Richard L. Rudell. *Logic Synthesis for VLSI Design*. PhD thesis, EECS Department, University of California, Berkeley, 1989.

[29] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan. Efficient sat-based boolean matching for fpga technology mapping. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 466–471, 2006.

[30] Sarma Sastry and Alice Parker. On the Relation Between Wire Length Distributions and Placement of Logic on Master Slice ICs. In *Proceedings of the 21st Design Automation Conference*, DAC '84, pages 710–711, Piscataway, NJ, USA, 1984. IEEE Press.

[31] Russell Tessier. Fast placement approaches for FPGAs. *ACM Trans. Des. Autom. Electron. Syst.*, 7(2):284–305, April 2002.

[32] R.E. Walpole, R.H. Myers, S.L. Myers, and K. Ye. *Probability and Statistics for Engineers and Scientists*. Pearson Education, 8th edition, 2007.

[33] S. Yang. *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.

[34] Chi Wai Yu, W. Luk, S. J E Wilton, and P.H.-W. Leong. Routing optimization for hybrid fpgas. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 419–422, 2009.

[35] Payman Zarkesh-Ha, Jeffrey A. Davis, William Loh, and James D. Meindl. Prediction of Interconnect Fan-out Distribution Using Rent's Rule. In *Proceedings of the 2000 International Workshop on System-level Interconnect Prediction*, SLIP '00, pages 107–112, New York, NY, USA, 2000. ACM.