AUTOMATING TRANSISTOR RESIZING IN THE DESIGN OF FIELD-PROGRAMMABLE GATE ARRAYS

By

Anthony Bing-Yan Chan

Supervisor: Jonathan Rose

April 2003

AUTOMATING TRANSISTOR RESIZING
IN THE
DESIGN OF
FIELD-PROGRAMMABLE GATE ARRAYS

By
Anthony Bing-Yan Chan

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF BACHELOR OF APPLIED SCIENCE.

DIVISION OF ENGINEERING SCIENCE

FACULTY OF APPLIED SCIENCE AND ENGINEERING
UNIVERSITY OF TORONTO

Supervisor: Jonathan Rose

April 2003

ABSTRACT

Automating Transistor Resizing in the Design of Field-Programmable Gate Arrays

Bachelor of Applied Science and Engineering, April 2003
Anthony Bing-Yan Chan
Division of Engineering Science
Faculty of Applied Science and Engineering
University of Toronto

The manual design and layout of Field Programmable Gate Arrays (FPGAs) is a lengthy process which can be greatly eased through automation.

This research builds upon previous work conducted at the University of Toronto towards the creation of a tool that automatically generates FPGA layouts from an architectural description. Specifically, this research modifies the transistor-level netlist created by that tool by resizing the transistors to improve performance as evaluated by a cost-function.

Several different functions were utilized to investigate the effects of different degrees of trade-off between area and speed. Although the final results varied depending on the function that was implemented, there was a minimum of a 50% improvement in each case.

ACKNOWLEDGEMENTS

Above all else, I would like to thank my supervisor Jonathan Rose whose guidance and support made this research possible. His understanding and advice also helped to ease and overcome problems that presented themselves along the way.

I would also like to thank Ketan Padalia whose work this research is directly based upon. His availability and prompt replies helped to shed light during difficult situations.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1

1.1 MOTIVATION 1

1.2 ORGANISATION OF THESIS 1

CHAPTER 2 BACKGROUND AND PREVIOUS WORK 3

2.1 INTEGRATED CIRCUITS AND CMOS TRANSISTORS 3

2.2 FPGA ARCHITECTURE 4

2.2.1 Routing Architecture 5

2.2.2 Logic Block Architecture 6

2.3 TILE STRUCTURE 7

2.4 VPR AND VPR_LAYOUT 8

2.4.1 Cell-Level Netlist 8

2.4.2 Transistor-Level Netlist 9

CHAPTER 3 VPR_RESIZER TOOL 10

3.1 DESIGN FLOW 10

3.2 PATH EXTRACTION 11

3.2.1 Extracting Paths Components 12

3.2.3 Representative Paths 13

3.3 DELAY CALCULATION 14

3.4 OPTIMIZER 16

3.5 FILE OUTPUT 17

CHAPTER 4 FINAL TOOL AND IMPLEMENTATION 19

4.1 INPUT FILE PROPERTIES 19

4.2 TOOL IMPLEMENTATION 19

4.2.1 Path Extraction 19

4.2.2 Delay Calculation 21

4.2.3 Optimisation Engine 22

4.2.4 Final Design Flow 24

4.3 INVOKING THE TOOL 24

4.4 RESULTS OF TOOL 25

4.4.1 Tool Output for Equally Weighted Paths 25
4.4.2 Tool Output for Unevenly Weighted Paths 28
4.4.3 Transistor Sizes by Cell using Unweighted Paths 31
4.4.4 Transistor Sizes by Cell using Weighted Paths 36

CHAPTER 5 CONCLUSION 38

5.1 FINAL RESULTS 38

5.2 FUTURE WORK 38

APPENDIX A CD CONTENTS 40

REFERENCES 41

LIST OF FIGURES

Figure 2.1 Junction Types of CMOS Transistors in ICs....3
Figure 2.2 Generic FPGA Layout....5
Figure 2.3 MUX vs. LUT....6
Figure 2.4 BLE and Logic Block....7
Figure 3.1 VPR_RESIZER Design Flow....10
Figure 3.2 Path Components....13
Figure 3.3 RC Ladder Approximation of Signal Path....15
Figure 4.1 Routing Structure Defined by Architecture....20
Figure 4.2 Representative Paths chosen for use in Optimiser....21
Figure 4.3 Flow Diagram of Optimiser....22
Figure 4.4 Final Design Flow....23

LIST OF GRAPHS

Graph 4.1 Run Time Progress of Tool without Weighting....26
Graph 4.2 Final Area without Weighting....27
Graph 4.3 Final Delay Sum without Weighting....27
Graph 4.4 Run Time Progress of Tool with Weighting....29
Graph 4.5 Final Area with Weighting....30
Graph 4.6 Final Delay Sum with Weighting....30
Graph 4.7 Percentage Change in Size when using Weighted Paths....30
Graph 4.8 Final Buffer Transistor Size without Weighting....32
Graph 4.9 Final Transistor Size of Non-Buffer Transistors without Weighting....32
Graph 4.10 Run Time Progress of Transistor Sizes for S4T1 without Weighting....34
Graph 4.11 Run Time Progress of Transistor Sizes for S1T1 without Weighting....34
Graph 4.12 Run Time Progress of Transistor Sizes for S1T4 without Weighting....35
Graph 4.13 Final Transistor Size of Non-Buffer Transistors with Weighting....37

LIST OF TABLES

Table 4.1 Paths Listed in File 21
Table 4.2 Command Line Arguments for Tool 24
Table 4.3 Cost-Functions Implemented by Tool 25
Table 4.4 Results of Implementing Tool without Weighting 25
Table 4.5 Final Area and Delay Sum without Weighting 26
Table 4.6 Weights Used for Each Path in File 28
Table 4.7 Results of Implementing Tool with Weighting 28
Table 4.8 Final Area and Delay Sum with Weighting 29
Table 4.9 Final Transistor Sizes by Cell without Weighting 31
Table 4.10 Final Transistor Sizes by Cell with Weighting 36
Table 4.11 Percentage Change in Non-Buffer Transistor Sizes with Weighting 37

Chapter 1 Introduction

1.1 MOTIVATION

Field-programmable gate arrays (FPGAs) are highly versatile devices that can be implemented easy for a large number of applications. They can also greatly reduce time-to-market by facilitating design changes. However, while projects utilizing FPGAs may experience short design periods, the same can not be said of FPGA devices themselves. FPGAs are highly complex integrated circuits that require many person-hours to design.

Such long design cycles have been the motivation behind an ongoing project at the University of Toronto for creating tools capable of automating the design process of an FPGA, thus reducing the design time. Previous work has created a tool that takes an architectural description of an FPGA and creates a transistor level netlist of the device. The size of the transistors described in this netlist are chosen by the previous tool with limited optimization.

The work presented in this paper is an extension of the previous work. The intent is to create a tool that will take the generated transistor-level netlist, analyze the circuit that it describes, and resize the transistors to optimize the described device for both area and speed.

1.2 ORGANISATION OF THESIS

Chapter 2 provides background information and the previous work upon which this paper is based. Chapter 3 describes the theory and intended process behind the transistor resizing tool. This chapter outlines the steps necessary for such a tool to perform its function and the methods by which these steps are executed. Due to problems encountered when working with the input netlists, a few changes were made in the final implementation of the tool as described in Chapter 4. This chapter also describes also describes the specific parameters

chosen for the tool for generating the results that are presented. Chapter 5 presents a summary of the results and describes aspects of this tool that can be improved upon through future work. The source code for the final tool can be found on the companion CD, the contents of which are described in Appendix A.

Chapter 2 Background and Previous Work

2.1 Integrated Circuits and CMOS Transistors

When implemented in integrated circuits (ICs), CMOS transistors generally consist of a polysilicon gate between two heavily doped silicon junctions which are the drain and source for the transistor. In large ICs, transistors often share junctions and these junctions may or may not have contacts. The different junction varieties are shown in Figure 2.1, and applies to both n-channel and p-channel transistors.

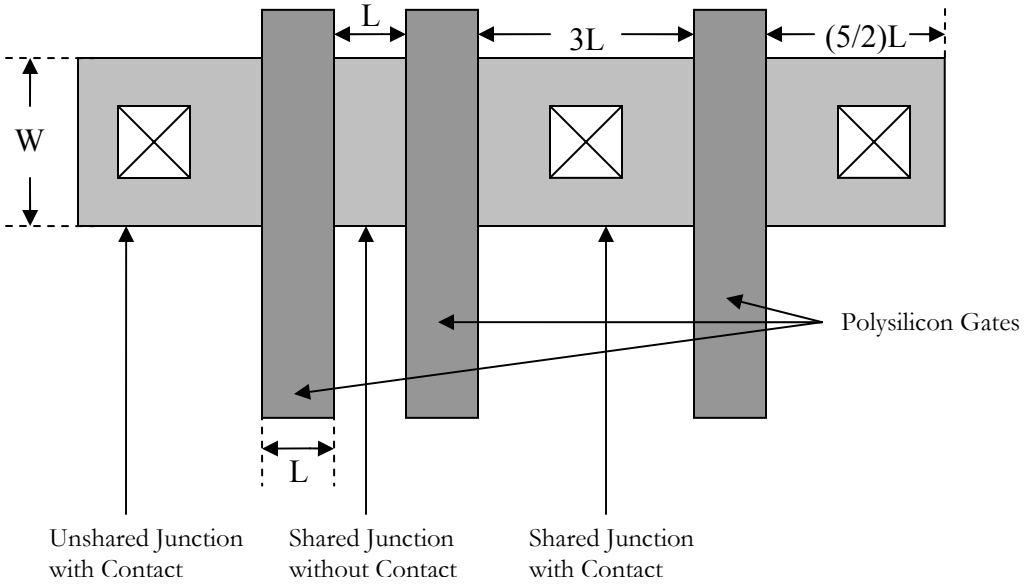


Figure 2.1 Junction Types of CMOS Transistors in ICs

Under static conditions, the drain-to-source resistance of a transistor can be calculated using:

$$R_{DS} = \frac{1}{\mu C_{ox} S(|V_{GS}| - |V_t|)}$$

Where μ, C_{ox} and V_t are technology dependent constants. Furthermore, V_t also varies depending on whether the transistor is p-channel or n-channel. Also, $S = (W/L)$ is the transistor sizing and V_{GS} is the gate-to-source voltage. For obvious reasons, S can not be less than one since L represents the smallest possible dimension.

However, it is also useful to be able to model the transistor as a resistance during signal transitions. Although there is no simple equation to do this, the resistance can be approximated by using the static equation multiplied by a "fudge-factor". A valid constant is 2.5 [4], but a more accurate constant can be determined through simulation and is technology dependent..

In such transistors, there is also significant parasitic capacitances at each junction and gate. Consequently, they can not be neglected when making such calculations. The following equations [4] can be used to calculate these capacitances:

$$C_{gate} = WLC_{ox}$$

$$C_j = 0.63\left(C_j A_j + C_{j-sw} P_j\right)$$

The equation for C_j is used to determine the capacitance for both sources and drains. A_j and P_j are the area and perimeter of the junction and C_j and $C_{j\text{-}sw}$ are technology dependent constants.

2.2 FPGA ARCHITECTURE

FPGAs can be implemented in a variety of ways. The most common method of implementation is through the use of SRAM cells as can be exhibited by products offered by Altera and Xilinx. Alternative programming techniques include antifuses and Flash which are both demonstrated by the products available from Actel.

The remainder of this paper will deal exclusively with FPGAs based on SRAM technology. This is primarily because this is the technology that is the subject of the work upon which this paper is based.

The generic FPGA has a layout similar to that shown in Figure 2.2. Such an FPGA consists of a ring of I/O blocks circling an array of logic blocks connected by routing lines.

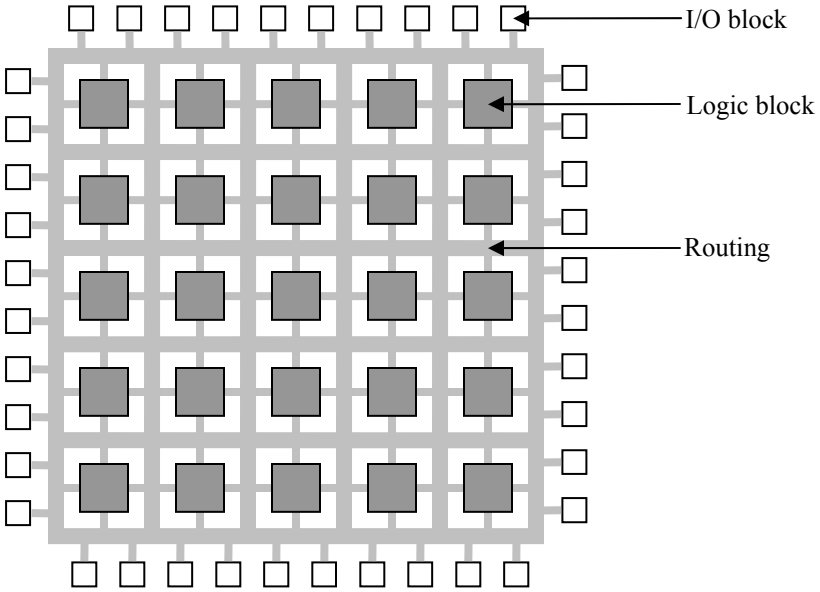


Figure 2.2 Generic FPGA Layout

Although the I/O blocks are also configured by programming SRAM cells, the work contained in this paper focuses on the logic blocks in the array and the connections between them through the routing network. Consequently, only these elements of the device need to be examined in the following subsections.

2.2.1 Routing Architecture

The programmable elements of the routing architecture consist mainly of pass-transistors. One can argue that tri-state buffers are also present, but they generally consist of regular buffers followed by a pass-transistor.

There are several routing styles that can be implemented. The style shown in Figure 2.2 is known as an island-style routing architecture, which features routing surrounding each logic block. Other styles include row-based and hierarchical [3], however island-style FPGAs were the subject of previous work and consequently will be the focus of this paper as well.

In such an architecture, wires run horizontally and vertically, surrounding each of the logic blocks of the array. Pass-transistors are located periodically along these wires to control the propagation of signals through the device. Depending on the periodicity of these pass-transistors, wires can span a number of logic blocks before encountering another transistor. The number of logic blocks that are spanned is used to characterise the length of the wires in a given architecture.

2.2.2 Logic Block Architecture

The programmable elements of a logic block generally consist of multiplexers and look-up-tables (LUTs), which have a similar structure to that of a multiplexer. Both structures consist of a network of pass-transistors. The difference between the two is that the SRAMs are connected to the gates of MUX transistors, whereas in LUTs, they are connected to the source node of the transistors. Figure 2.3 shows the difference between the two.

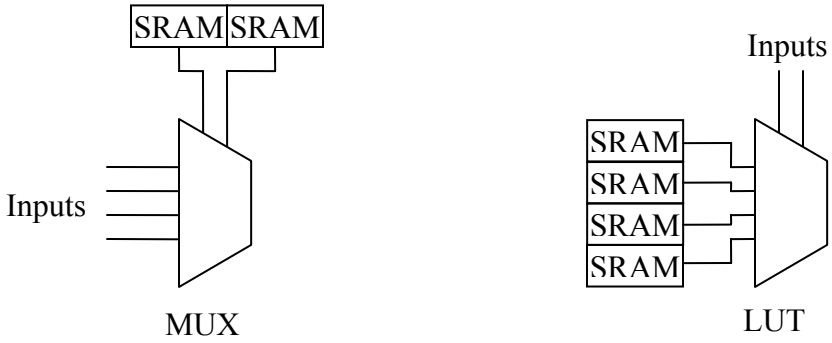


Figure 2.3 MUX vs. LUT

A LUT can be combined with flip-flop to form another structure known as a Basic Logic Element (BLE). The presence of the flip-flop allows for the implementation of both combinatorial and sequential circuits. An entire logic block can consist of one or more BLEs. Both the schematic of a BLE and a logic block are shown in Figure 2.4.

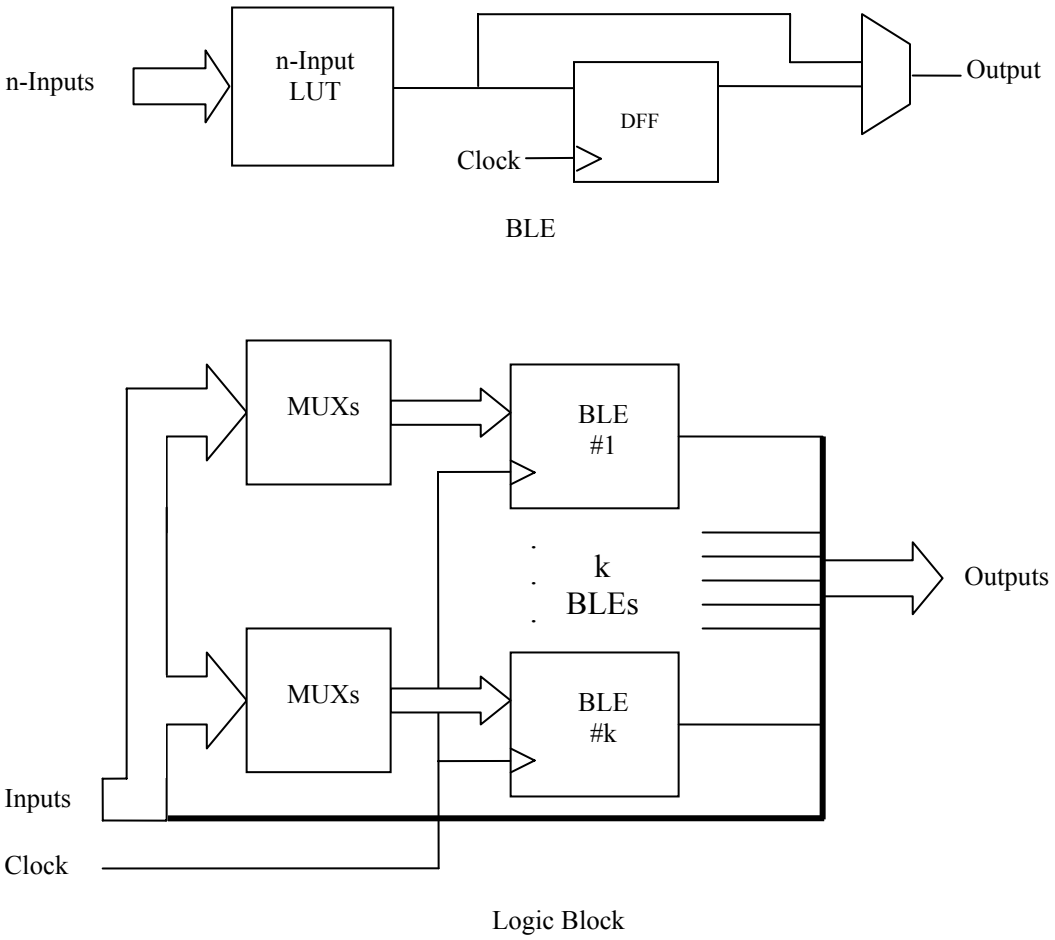


Figure 2.4 BLE and Logic Block

2.3 TILE STRUCTURE

In describing the architecture of an FPGA, one can make use of the repetitive nature of the device. It simply consists of an array of similar logic blocks all surrounded by similar routing lines. Consequently, the entire device can be fully described by one unit, or tile, of the total array.

To describe an FPGA in such a manner, a few additional considerations need to be made. The first is to ensure that the ports on opposite sides of the tile line up so that they are

connected when the tile is arrayed. Another consideration is to ensure that routing lines are offset properly to ensure that they span the desired length.

2.4 VPR AND VPR_LAYOUT

VPR (Versatile Place and Route) is a tool developed though previous work conducted at the University of Toronto. Two inputs which describe the desired FPGA are used, the netlist of the logic blocks and an architecture description file. This tool then invokes a place-and-route engine that creates placement and routing output files and statistics.

VPR_LAYOUT is another tool that builds upon VPR and was also developed at the University of Toronto. In addition to the outputs provided by VPR, this tool also generates cell-level and transistor-level netlists. Each of these netlists describe a tile of the FPGA and are the basis of the work presented in this paper.

2.4.1 Cell-Level Netlist

An FPGA is essentially comprised of buffers, SRAMs, multiplexers, LUTs, flip-flops, and pass-transistor switches. Consequently, an FPGA tile can be defined by describing it in terms of these components, or "cells".

The cell-level netlist accomplishes this by identifying the type of each cell as well as providing a unique identifier to each cell instance. The following is an example of such a netlist produced by VPR_LAYOUT:

```
# FPGA Tile cell-level netlist
# Output by VPR_Layout


# CELL Format: id cell_type "Name" subgroup_type group_type width height num_pins
#              (pin_class node x_offset y_offset) (...) (...) etc for num_pins times

C0 0 "1x_Buffer" 0 0 4 3 4 (5 1 0 0) (0 2 1 0) (1 8 2 1) (6 0 3 2)
C1 0 "4x_Buffer" 0 0 6 5 4 (5 1 0 0) (0 8 0 3) (1 9 3 1) (6 0 3 4)
C2 0 "1x_Buffer" 0 0 4 3 4 (5 1 0 0) (0 3 1 0) (1 10 2 1) (6 0 3 2)
.
.
.
```

In the work to follow, the information that will be used will be the first three parts of each line: cell ID, cell type, and cell name.

2.4.2 Transistor-Level Netlist

The transistor-level netlist begins by stating the port information. It provides the node that each port is connected to and the side that each port is located on. Ports are paired up using port IDs to ensure that during placement, the proper ports are lined up from one side of the tile to the other.

The transistor-level netlist then describes each transistor of the FPGA tile, listing the size of the transistor and each node that it is connected to. The cell information of each transistor is also given, including its cell ID and cell type.

The following is an example of such a netlist:

```
# FPGA Tile transistor-level netlist
# Output by VPR_Layout


# PORT Format: id node constraint_class

# XTOR Format: id drain gate source type size cell_type cell_id subgroup_type group_type

P0 133 0 L
P1 134 48 R
P2 133 48 L
P3 137 24 B
.
.
.
M0 8 2 1 P 2 0 0 0 0
M1 8 2 0 N 1 0 0 0 0
M2 9 8 1 P 8 0 1 0 0
M3 9 8 0 N 4 0 1 0 0
M4 10 3 1 P 2 0 2 0 0
.
.
.
```

Chapter 3 VPR_RESIZER Tool

3.1 DESIGN FLOW

The overall description of the resizing tool can be illustrated in Figure 3.1. The tool takes the netlists generated by VPR_LAYOUT and outputs a new transistor-level netlist with new transistor sizes.

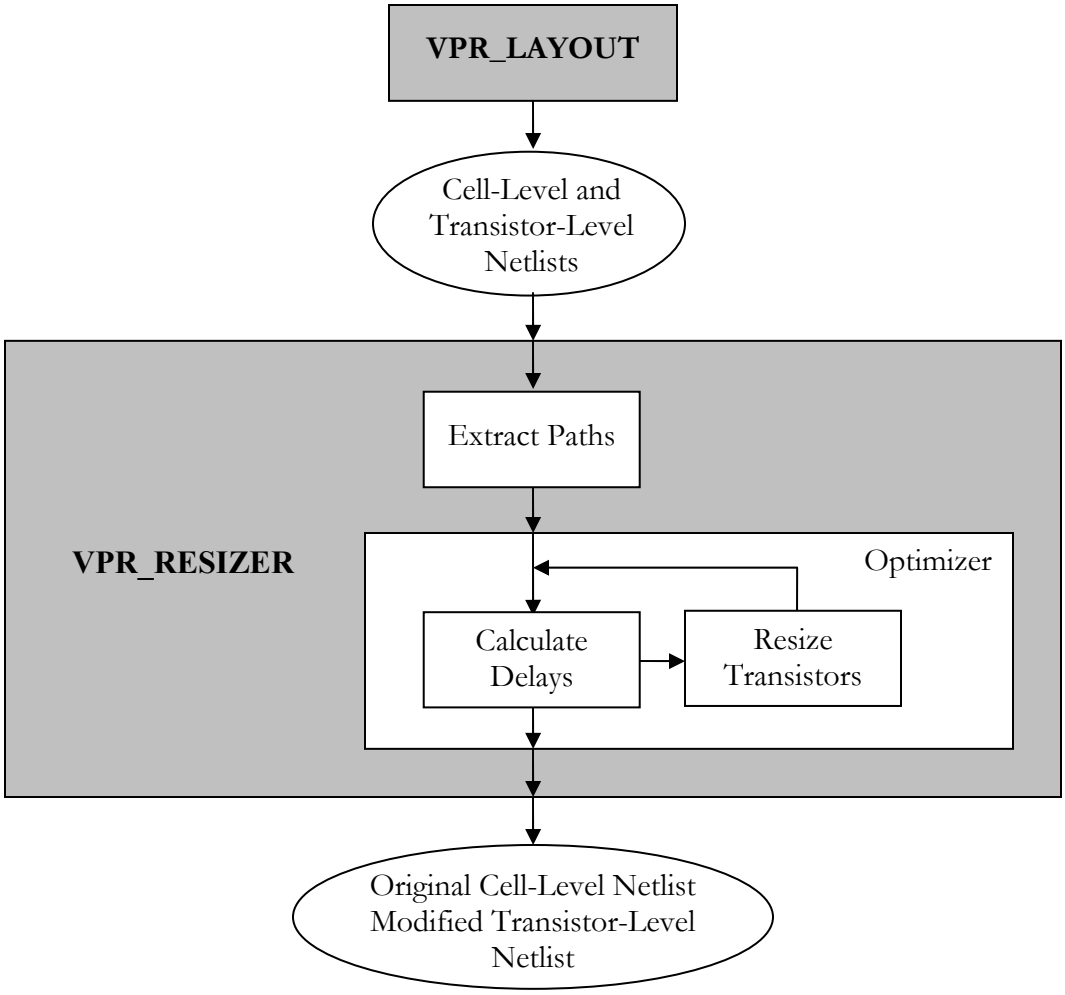


Figure 3.1 VPR_RESIZER Design Flow

The tool begins be parsing the input netlists to extract different signal paths found in the FPGA. Once these paths have been found, their respective delays are calculated. Modifications

are then made on the transistor sizes according to the optimization engine and the delays are recalculated. This process is repeated until the optimizer can not find a better implementation. A new transistor-level netlist is then generated that is simply a copy of the original transistor-level netlist with the updated transistor sizes.

3.2 PATH EXTRACTION

In an FPGA, there are a number of signal paths that exist. Some come from input buffers into a BLE, other output from a BLE to an output buffer. However, the paths used by this tool all involve signals originating at the output of a BLE and ending at the input of another BLE. The motivation behind this is that assuming both BLEs are part of a sequential circuit, these paths would govern the maximum usable clock frequency. By calculating the delay of each of these paths, the critical path could be identified as the one with the longest delay. By reducing the delay of this path, the overall speed of the device could be increased.

As mentioned before, the netlists provide information for a single tile of an FPGA. Thus, on its own, the only path that the given netlists can provide are the feedback paths from the output of a BLE back to one of its inputs (or another BLE for multi-BLE tiles). Although this is useful, most paths of interest involve BLEs from different tiles. Thus, to generate such paths, a "virtual" array of the tile must be created.

Unfortunately, generating such an array results in the problem of having too many paths. Although calculating the delay for each of these paths would not be excessively long, doing so a large number of times would be computationally demanding. The recursive nature of the optimizing engine would make it impractical to utilize the entire set. Consequently, the additional task of reducing the set of all paths down to a representative selection is also necessary.

3.2.1 Extracting Paths Components

Although one method of extracting the paths would be to generate the array of tiles and analyzing that array directly, such a method would require a large amount of memory to execute in a reasonable amount of time. Thus, an alternative approach was taken in which most of the analysis was conducted by looking within a single tile prior to creating the array.

As mentioned before, entire paths can not be extracted by looking at only a single tile (with the exception of the feedback paths). Thus, prior to looking at the "virtual" array, only path components can be extracted. The first is BLE2BLE, which is simply the feedback paths from the BLE output to any of the inputs to the BLE. The second is BLE2OUT which is a path leading from the BLE output to any of the ports of the tile. Conversely, IN2BLE is any path originating at a tile port and ending at any of the BLE inputs. Lastly IN2OUT is any path beginning at one tile port and ending at another. Examples of such paths can be found in Figure 3.2. When finding paths, in addition to nodal and transistor information, if a path begins and/or ends at a tile port, the associated port information is also recorded.

After determining these path components, longer paths can be determined by piecing these components together. Obviously, feedback paths are already complete, but the rest must be connected to form complete paths. This is accomplished by starting with a BLE2OUT path component. When the output tile port is encountered, the port information is accessed to determine the identity of the connecting port of the neighbouring tile. All IN2BLE paths associated with the connecting port are found and added to the previous path component. Together they form a complete path. If there are any IN2OUT paths found on the connecting port, this process is repeated at the next port.

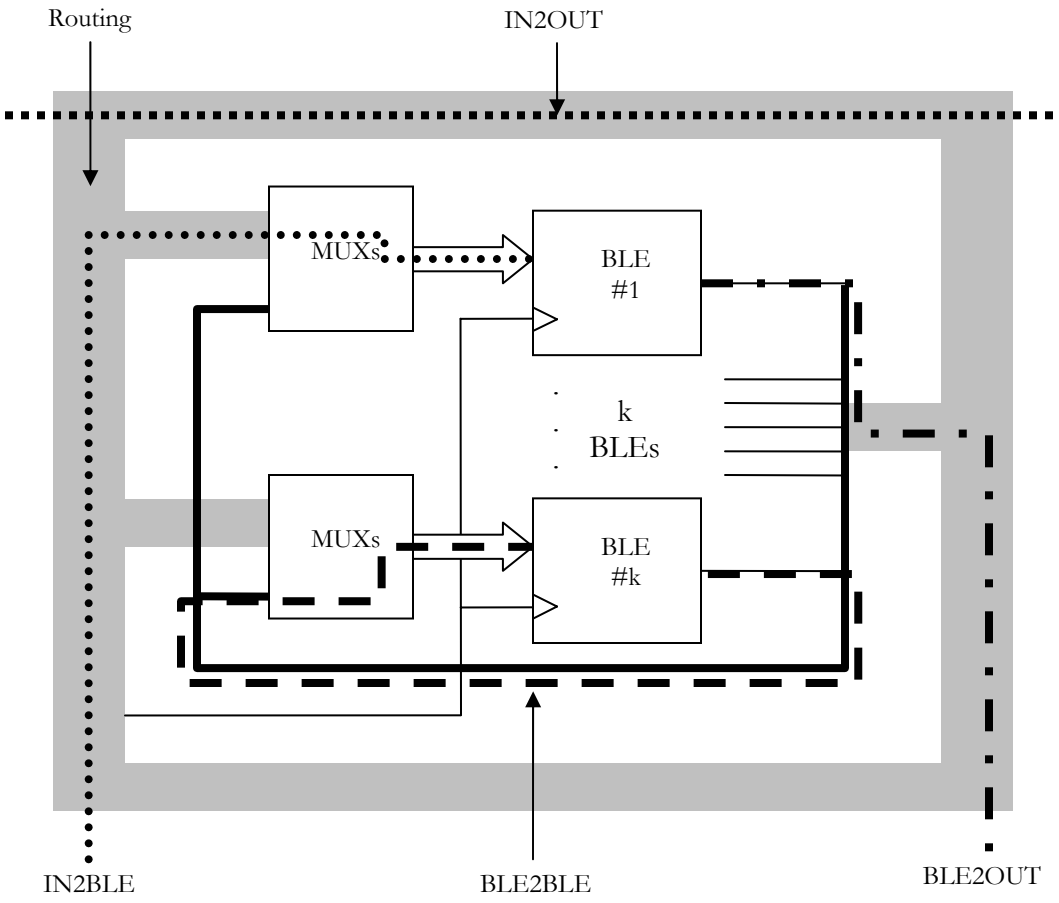


Figure 3.2 Path Components

Using this process, all paths in a large array of tiles can be found in an efficient way. The remaining problem is one of sorting through all the information and reducing the set of paths down to a manageable number.

3.2.3 Representative Paths

For a general architecture, there are a number of paths of interest. As mentioned earlier, one such path is the feedback path found within a single tile. Another common path is that between neighbouring tiles. For designs with multi-length wires, paths involving logic blocks connected to the same wire are also utilized frequently. Paths that travel a longer distance must

travel through pass transistors and/or buffers. The precise distance for this to occur would be dependent on the architecture. Regardless, such paths (and even longer) should be included in any representative set of paths.

3.3 DELAY CALCULATION

One of the most accurate methods of calculating the delays for the extracted paths would be to use SPICE simulations. However, implementing a SPICE engine within this tool is well beyond the scope and time limitations of this research. Rather, preliminary work towards this goal has been completed by converting the transistor-level netlist into a SPICE netlist. This netlist is not currently used, but is output by the tool for possible future use.

Delays were instead calculated using the Elmore delay model. This is not the most accurate method of calculating delays, but it is simple to implement. Also, although it is not very accurate, it is effective in calculating relative delays. Thus it should be sufficient for detecting relative changes as a result of transistor resizing, which is most important when implementing the optimizer.

In creating the model, one should normally also consider resistances that are not directly on the signal path. However, given the complex nature of the entire circuit, this model has also been simplified so that only the resistances of those transistors directly in the signal path are used.

Figure 3.3[1] shows how a path is broken down into a series of RC ladders. Each ladder is terminated and a new ladder begins when the path reaches the gate of a transistor. In this model, each transistor is modelled as a resistor and the capacitance at each node is determined by the contribution of all connecting transistors (even those not in the signal path). Although

[1] The figure does not depict an actual path, it is merely an example

the figure shows R_B, this resistance could be replaced with R_C depending on whether the signal is transitioning from high-to-low or low-to-high.

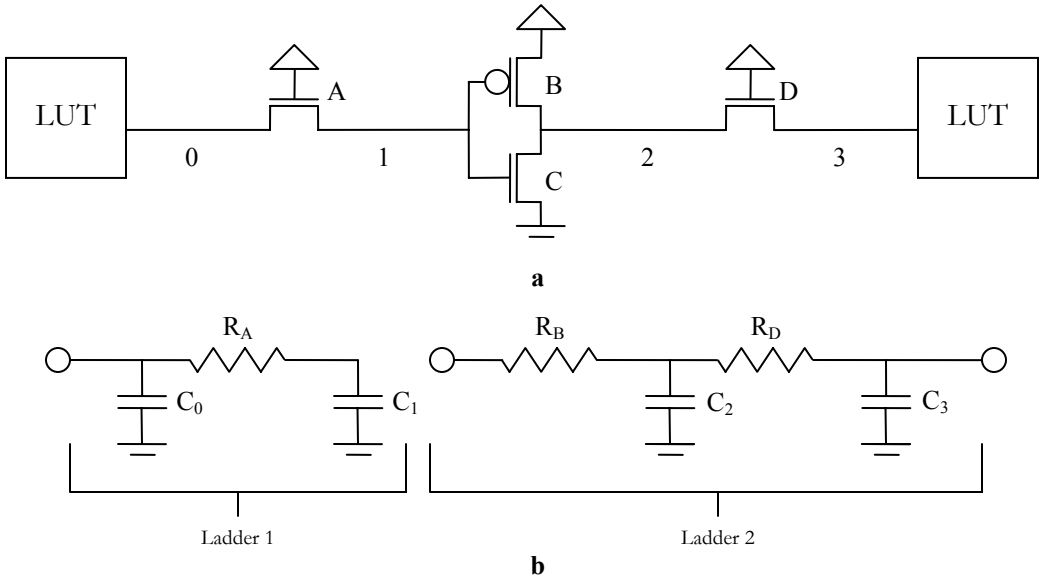


Figure 3.3 RC Ladder Approximation of Signal Path

To create this model, a number of values required calculation: the resistance of each transistor during transitions and the parasitic capacitances of the gate, drain and source. Although transmission line effects should also be modelled for resistance and capacitive effects, they were excluded due to the limitations of this project. However, such contributions are worth including in any future work.

To determine these values, the following equations were used:

$$R_{DS} = \frac{2.5}{\mu C_{ox} S\left(|V_{GS}| - |V_t|\right)}$$

$$C_{gate} = SL^2 C_{ox}$$

$$C_{drain} = C_{source} = 0.63\left(C_j\left(3SL^2\right) + C_{j-sw}(6L)\right)$$

To simplify the analysis, each drain and source were assumed to be a shared junction with contact. Although this is not true, determining the precise junction type of each drain and

source is beyond the scope of this paper. Also, such a distinction would provide a marginal improvement at best. Consequently, despite this approximation, the results should be accurate enough for the purposes of this research.

For an RC ladder with *n* RC stages, the delay of the ladder can be calculated by using the following equation:

$$\tau = \sum_{i=1}^{n} C_i \sum_{k=1}^{i} R_k$$

Which states that the delay can be calculated by first identifying the capacitance at each node along the signal path and the corresponding look-back resistance from that node to the beginning of the RC ladder. Take the product of these two values and the delay is the sum of these products. Note that with this equation, C_0 in Figure 3.3 is unused since the look-back resistance from that node is zero.

Lastly, the total delay of the signal path can be calculated by taking the sum of the delays of the composite RC ladders:

$$\tau_{total} = \sum_{l} \tau_l$$

3.4 OPTIMIZER

As described earlier, an FPGA can be described as the combination of a small handful of basic cells. Thus, if the size of one transistor for one of these cells is changed, the same change must be replicated in the same transistor for each instance of that cell. Consequently, even though thousands of transistors are involved, the limited number of cell types greatly reduces the amount of freedom when adjusting the transistor sizing.

Another point of note is that adjusting the transistor sizes is a trade-off between size and speed. In general, increasing the transistor sizes will increase the speed, but at the cost of silicon

area. Thus, to determine the ideal transistor sizing, a balance must be found that optimizes for both size and speed.

The total size of the device is determined by summing up the sizes of all the transistors. In actuality, this is not the actual area of the device, but rather the area of the transistor gates. Despite its inaccuracy, this method of calculating the device size can be used because it reflects the changes made in the transistor sizes. Since relative changes can be tracked correctly, the same argument for using the Elmore delay model can be made here.

Also associated with each of the paths produced from the path extraction process is a weighting factor. The weights associated with each path can be of any value within (0,1] and are used to optimise for some paths more than other if so desired. These factors are used by multiplying the calculated delays by them to obtain weighted delays.

The optimizer begins by first calculating the weighted delay of each of the paths as well as the total size of the device. These values are then used to evaluate a cost function which is an increasing function of both variables. The optimal transistor sizing is found by minimizing the result of this function which represents a trade-off between size and speed.

A change is then made in the transistor sizing and the weighted delays are recalculated and the cost function is re-evaluated. If an improvement is found in the cost result, the change in the transistor sizing is kept. However, if there was no improvement, the resizing is undone, and an alternative change is attempted. This process is repeated until a state is found in which no improvement can be made by any change in the transistor sizing.

3.5 FILE OUTPUT

Once the resizing is complete, the information is then output in a file that is identical in format to that of the original transistor-level netlist. All node and cell information remains

unchanged and only the transistor sizes are altered. The benefit of maintaining the same format enables compatibility with any ongoing work as well as ease of readability for anybody already familiar with the previous work.

Chapter 4 Final Tool and Implementation

4.1 Input File Properties

Prior to the work presented in this paper, the netlists generated by VPR_LAYOUT have not been used for any other application. Consequently, the accuracy of these netlists have never been verified, which led to a number of problems when attempting to create this tool. All the netlists available for use during the development of this tool featured one or more four-input LUTs and length four wires. Although the following results apply to the single-LUT netlist, the steps presented below can be repeated and applied to any of the other available netlists.

4.2 Tool Implementation

The following subsections describe the specific details required to implement the steps outlined in the previous chapter. Any problems that were encountered are presented along with the solutions to work around them.

4.2.1 Path Extraction

The first problem that was encountered when extracting paths from the original transistor-level netlist was that no BLE2OUT path components could be found. As shown in Figure 2.4, the output of a BLE should be connected to a feedback path as well as a path leading out to the routing. However, the only connection that was described in the source netlist was the feedback path, resulting in the aforementioned problem. The output node of the BLE should also be connected to a drive buffer, the output of which should be separated from routing lines by pass-transistors.

The proper solution to this problem would be to modify VPR_LAYOUT directly to correct the netlist generation subroutine to properly include the missing transistors. However,

for the purposes of this paper, the missing transistors have been manually added to the source netlist. The inclusion of these transistors allow BLE2OUT path components to be found.

As mentioned earlier, the architecture of the FPGA described by the files used by VPR_LAYOUT features length-four wires. Thus, as seen in Figure 4.1, when constructing the circuits described by the transistor-level netlist, one would expect to find continuous wires that span four logic blocks before encountering a pass-transistor, and then a wire spanning another four logic blocks.

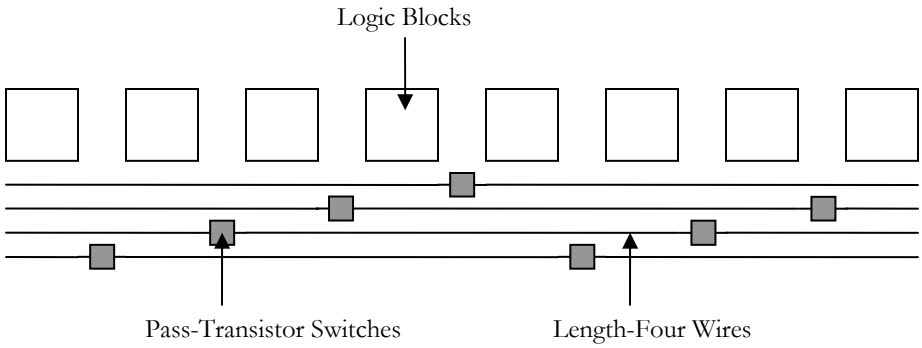


Figure 4.1 Routing Structure Defined by Architecture

Unfortunately, the second problem that was encountered in the input netlist is that one length-four wire was not found to connect to another length-four wire through a pass-transistor. Consequently, paths spanning many wire lengths could not be found. As a result, the list of representative paths was limited.

Due to the problems that were encountered and the time constraints of this project, the process of path selection could not be fully automated. Instead, the path components were output to a file, which were then assembled manually to create the list of representative paths used by the optimizer. This file also indicates the associated weighting to be used by the optimizer for each path. For each entry listed in Table 4.1, four paths were created in the file. Two paths corresponding to the two possible transitions (high-to-low and low-to-high), and two going into either an inverted or non-inverted input to the BLE. The second pair of paths can be

argued as being redundant. However, their inclusion should not result in any negative impact on the optimizer. Of the four types of paths listed in the table, the feedback path has already been illustrated in Figure 3.2, the remaining three types of paths are depicted in Figure 4.2.

Label in File	Description
Feedback	A feedback path from a BLE output to a BLE input within the same tile.
Beside	A path between two BLEs in neighbouring tiles, connected to the same wire.
5Pass	A path between two BLEs separated by five tiles, connected to two different wires which are connected by a pass-transistor.
5Buff	A path between two BLEs separated by five tiles, connected to two different wires which are connected by a drive buffer.

Table 4.1 Paths Listed in File

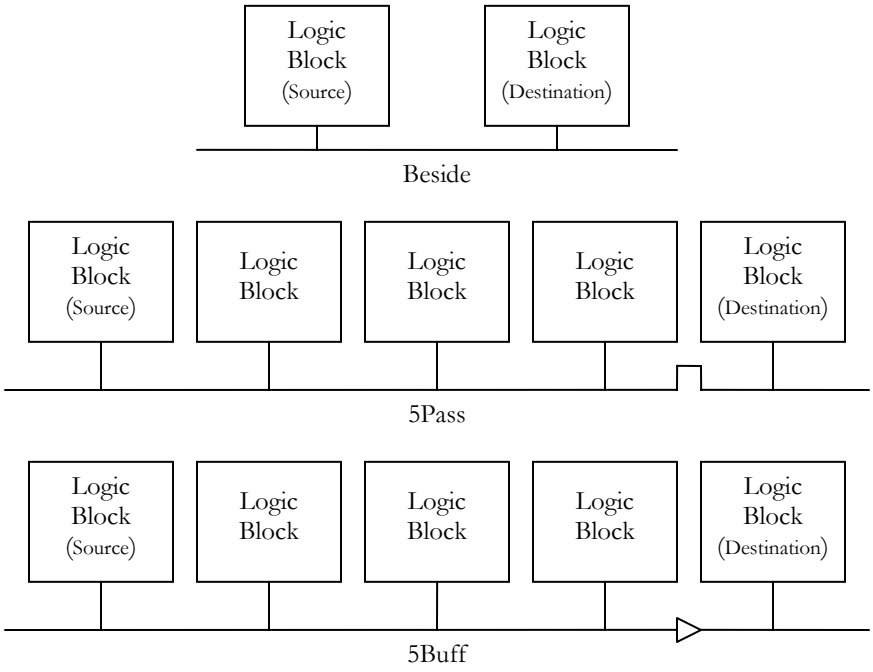


Figure 4.2 Representative Paths chosen for use in Optimiser

4.2.2 Delay Calculation

The transistor models used to create the SPICE netlist as well as to make the delay calculations were based on the 0.25µm process. The values of the parameters were obtained

from [5]. Due to ease of implementation, the SPICE netlist is actually created in the previous stage of the tool while the input netlists are being parsed.

4.2.3 Optimisation Engine

Changes to the transistor sizes were made by either increasing or decreasing the original size by 1%. The order in which the cells were changed were multiplexer, buffer, and then pass transistor switches. Although there are a number of different buffer types defined in the cell-netlist file (differentiated by their original size), all buffers have been given the same cell id. As a result, the tool currently treats all the buffers as equal and modifies them irrespective of their original size. Figure 4.3 shows a detailed flow diagram of the optimiser. To simplify the diagram, what has been excluded is that the tool first attempts to decrease the size of a set of transistors and an increase is attempted after an improvement can not be found. Thus, the final optimiser effectively has double the stages as that shown in the figure.

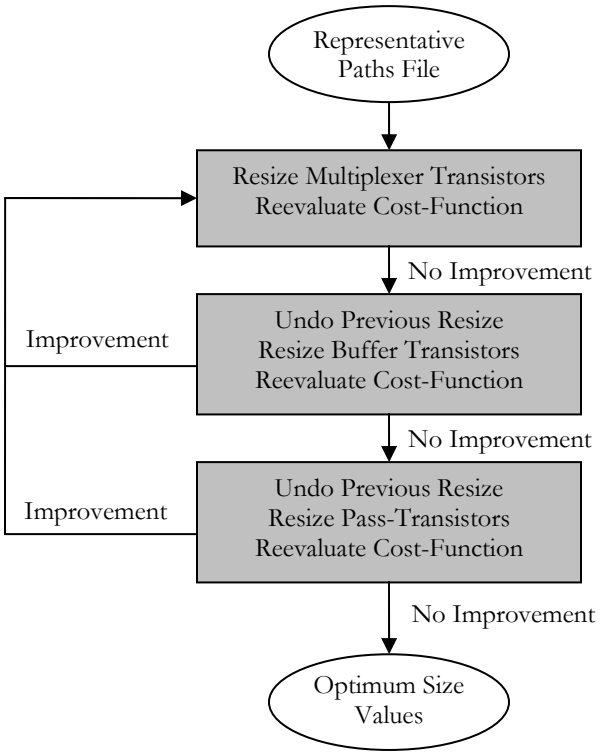


Figure 4.3 Flow Diagram of Optimiser

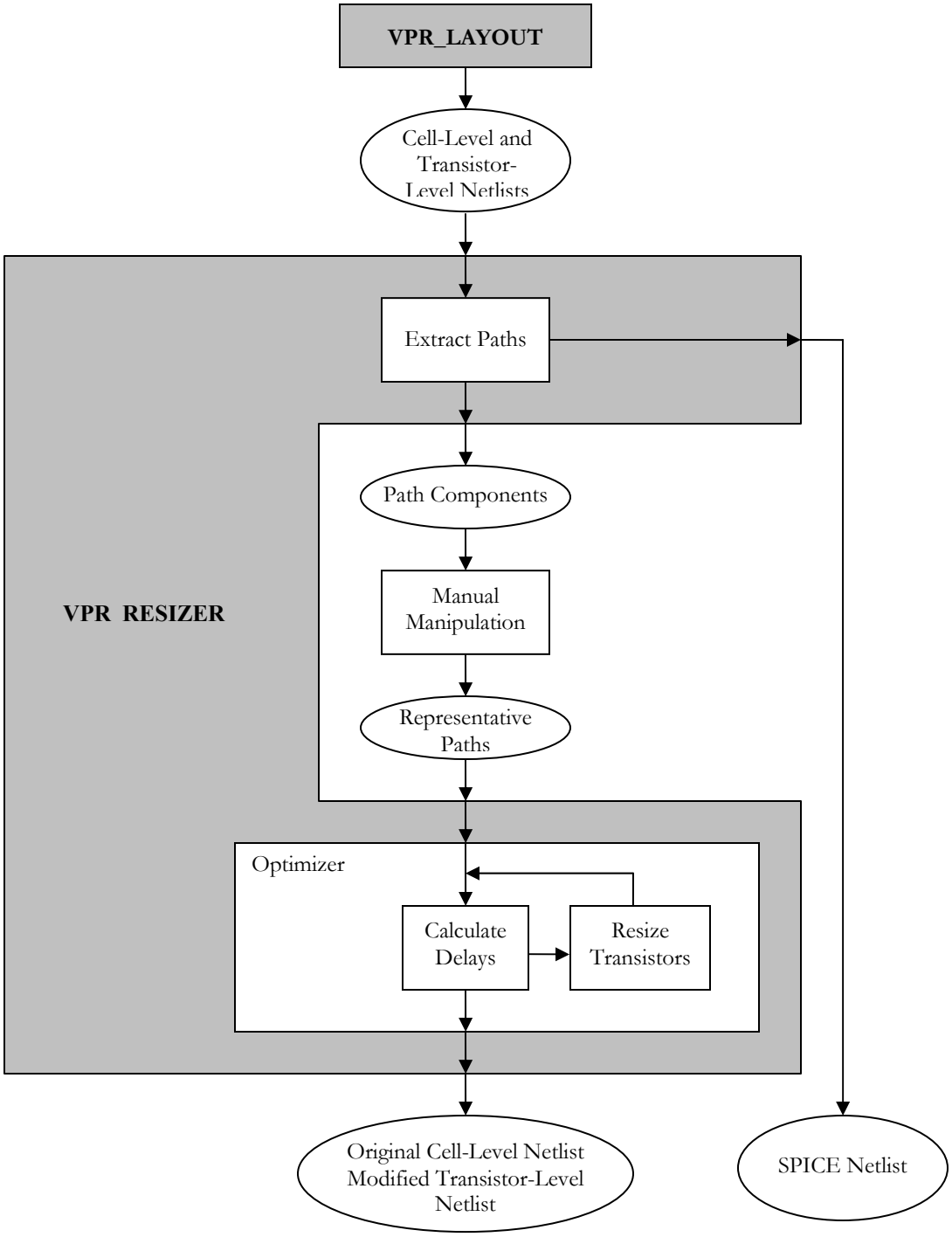


Figure 4.4 Final Design Flow

Several cost-functions were applied during multiple uses of the tool. The purpose of using different functions was to examine the effect of optimizing with varying emphasis on size and speed. For the calculated size (s) and weighted delay sum (t), each cost-function took the form of:

$$f(s,t) = s^x t^y$$

Different cost-functions were realised by varying the values of the exponents x and y.

4.2.4 Final Design Flow

As a result of the problems encountered and implementation choices, the final design flow differs slightly from that shown in the previous chapter. Figure 4.4 shows the final version of the design flow.

4.3 INVOKING THE TOOL

The final tool takes the form of a command line executable. The program requires five arguments (described in Table 4.1) and execution takes the following form:

% vpr_resizer <input_filename> <output_filename> <spice_output_filename> <spice_parameter_filename> <path_output_filename>

Argument	Definition
input_filename	Filename of transistor-level netlist generated by VPR_LAYOUT
output_filename	Filename of transistor-level netlist that will be generated by resizer tool
spice_output_filename	Filename of SPICE netlist that will be generated by tool
spice_parameter_filename	Name of file containing the transistor model parameters for the SPICE netlist
path_output_filename	File name of list of path components to be generated by tool during the path extraction process. The selected paths for the optimizer is generated from the information in this file.

Table 4.2 Command Line Arguments for Tool

4.4 RESULTS OF TOOL

The results of the tool will of course vary depending on the parameters and settings used. The variables include the weighting of the paths and the cost-function that is used. The following subsections provide analyses of the tool for different settings of these parameters. In each case, seven different costs functions will be used. Table 4.3 lists the seven functions and the correspond labels that will be used to refer to them in the rest of this text.

Function	$f = s^4 \cdot t$	$f = s^3 \cdot t$	$f = s^2 \cdot t$	$f = s \cdot t$	$f = s \cdot t^2$	$f = s \cdot t^3$	$f = s \cdot t^4$
Label	S4T1	S3T1	S2T1	S1T1	S1T2	S1T3	S1T4

Table 4.3 Cost-Functions Implemented by Tool

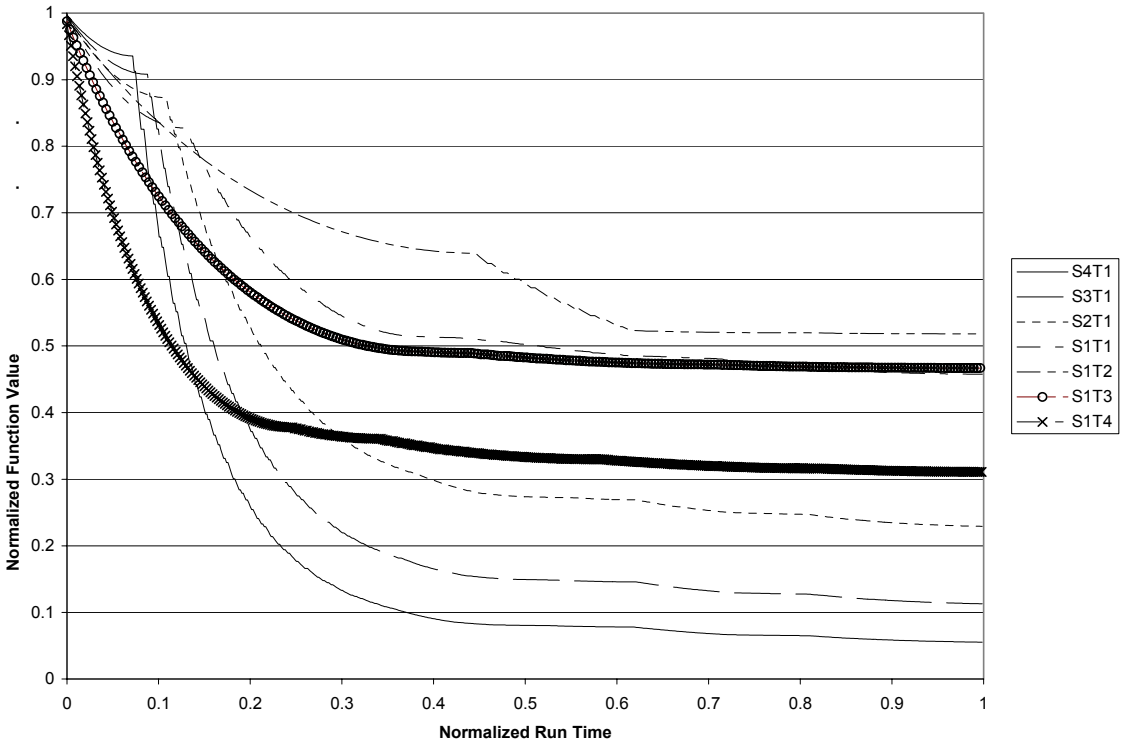
4.4.1 Tool Output for Equally Weighted Paths

Table 4.4 shows the initial results of running the tool with the seven different cost-functions. The final values in the table have been normalized to the initial value of each respective function prior to running the tool. The table also shows the number of times that the loop within the optimizer was iterated[2], as well as the execution time for the optimizer. To further show the effects of the tool, Graph 4.1 shows the progress of the tool. To account for the different runtimes for each function, the x-axis has been normalised.

Function	Final Value	Iterations	Execution Time (s)
S4T1	0.055	655	55.479
S3T1	0.113	672	55.219
S2T1	0.229	673	54.408
S1T1	0.458	711	52.926
S1T2	0.518	231	12.518
S1T3	0.467	279	14.570
S1T4	0.311	447	27.539

Table 4.4 Results of Implementing Tool without Weighting

[2] By this definition, this also corresponds to the number of times a change was made to the transistor sizing.

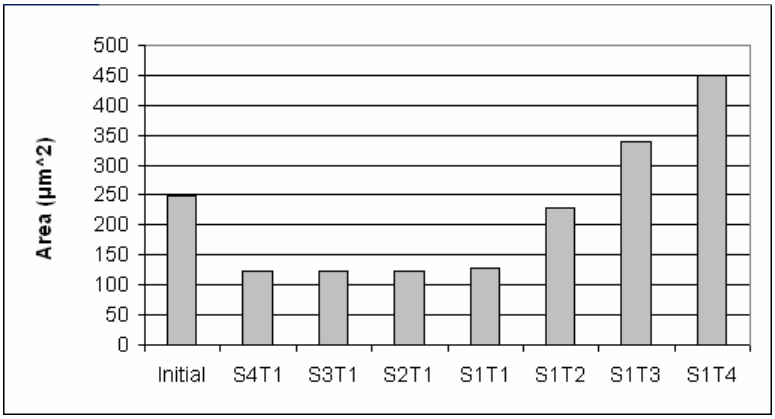


Graph 4.1 Run Time Progress of Tool without Weighting

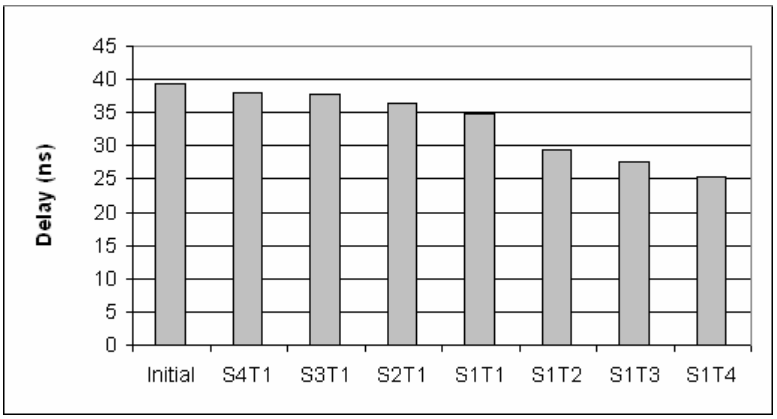
Although there may appear to be an anomaly in the trend of the final values, this is only due to the different functions involved. A better comparison can be made be comparing the inputs to these functions, the final area and delay values, which are presented in Table 4.5. This information is also presented in Graph 4.2 and Graph 4.3.

Function	Initial Area (μm^2)	Initial Delay Sum (ns)	Final Area (μm^2)	Final Delay Sum (ns)
S4T1	249.233	39.292	121.922	37.935
S3T1	249.233	39.292	121.998	37.853
S2T1	249.233	39.292	123.851	36.487
S1T1	249.233	39.292	128.945	34.768
S1T2	249.233	39.292	229.568	29.476
S1T3	249.233	39.292	339.659	27.495
S1T4	249.233	39.292	450.065	25.316

Table 4.5 Final Area and Delay Sum without Weighting



Graph 4.2 Final Area without Weighting



Graph 4.3 Final Delay Sum without Weighting

What is interesting about the results is that varying the exponent when size is the dominant factor does not change the final area and delay values much. This is likely a result of the transistors being optimized to their minimum size. Hence, any increased emphasis on size results in minor differences since the transistors sizes can not be improved upon. This is verified later when the final size of the individual cell transistors is examined later on. However, varying the exponent when delay is the dominant factor results in significant changes in the final values. This is due to the lack of any upper limit to the transistor size, which allows the total area to be increased without bound in order to decrease delay. Also of note is that the optimal result for some of the cost-functions is actually an increase in the total area, whereas there is an improvement in delay in every case.

4.4.2 Tool Output for Unevenly Weighted Paths

To determine the effect of weighting on the results of the tool, the paths were given a decreasing weight with increasing distance. Table 4.6 shows the weights used for each of the paths listed in the file.

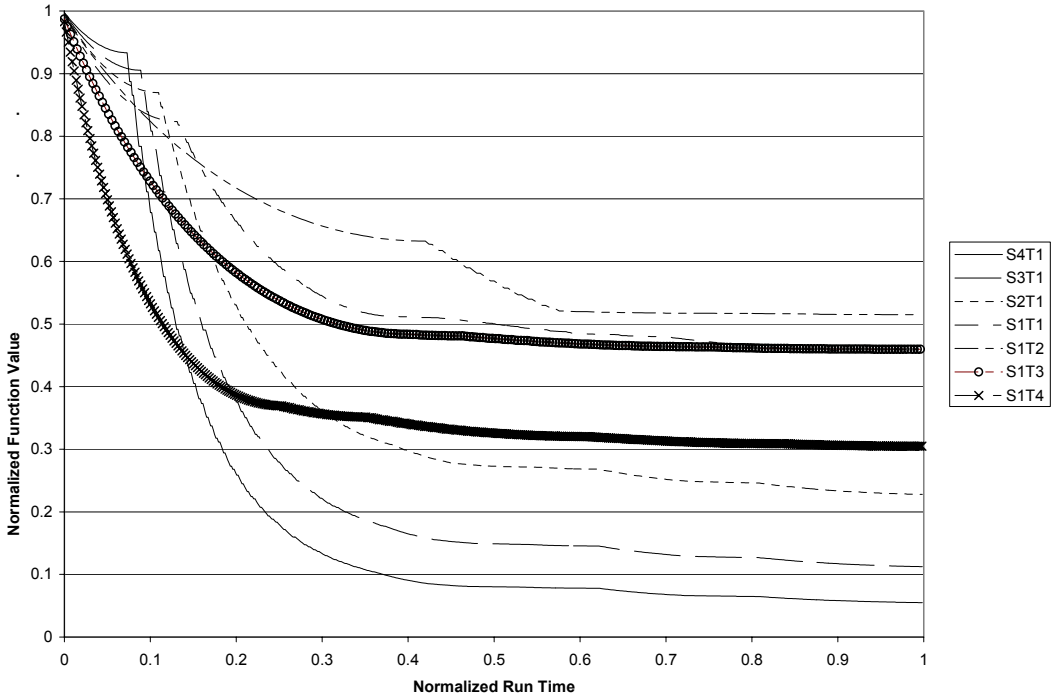
Path	Weight
Feedback	1
Beside	0.8
5Pass	0.6
5Buff	0.6

Table 4.6 Weights Used for Each Path in File

In a fashion similar to that found in the previous subsection, the results of the tool are shown in Table 4.7 and the plot of the tool progress can be seen in Graph 4.4. A comparison with the unweighted results shows that there is not much change in the results for the two sets of data. As the delay component becomes more dominant in the cost-function, a larger (although still small) difference can be seen between the two results. This is not surprising as weighting the paths only has an impact on the delay values.

Function	Final Value	Iterations	Execution Time (s)
S4T1	0.055	658	55.720
S3T1	0.112	675	55.369
S2T1	0.228	675	54.989
S1T1	0.455	709	52.565
S1T2	0.515	250	14.120
S1T3	0.460	271	13.429
S1T4	0.305	439	26.428

Table 4.7 Results of Implementing Tool with Weighting

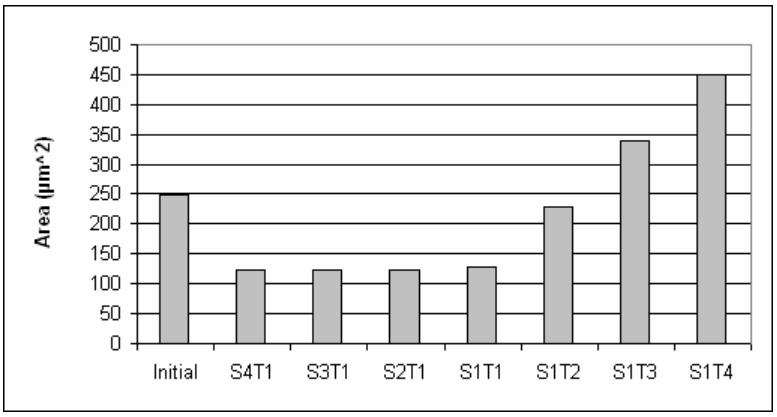


Graph 4.4 Run Time Progress of Tool with Weighting

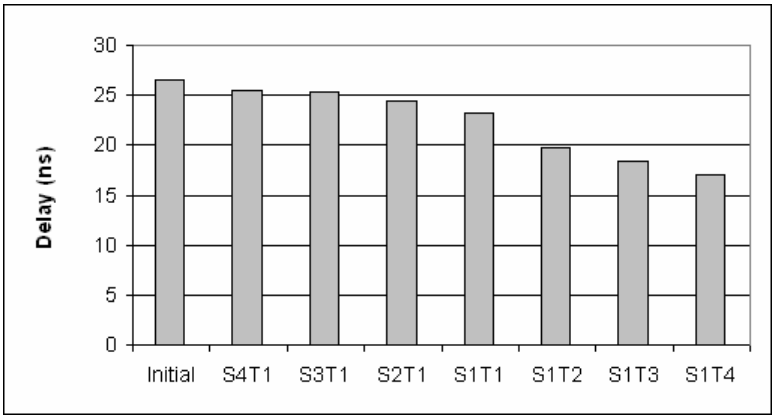
The final area and delay values are shown in Table 4.8 and depicted in Graph 4.5 and Graph 4.6. To properly compare this data with the unweighted results, one can only compare the two sets of final area values since the weighted sum of the delays will always be less than the unweighted sum of the delays. A quick comparison of the two tables shows that there is very little difference. Graph 4.7 explicitly shows the affect of using weighted path delays as a percentage change in the size. As the numbers show, the difference is much less than 1% in each case.

Function	Initial Area (μm^2)	Initial Weighted Delay Sum (ns)	Final Area (μm^2)	Final Weighted Delay Sum (ns)
S4T1	249.233	26.461	121.912	25.432
S3T1	249.233	26.461	121.987	25.376
S2T1	249.233	26.461	123.962	24.390
S1T1	249.233	26.461	129.129	23.230
S1T2	249.233	26.461	228.961	19.815
S1T3	249.233	26.461	338.933	18.433
S1T4	249.233	26.461	448.936	16.973

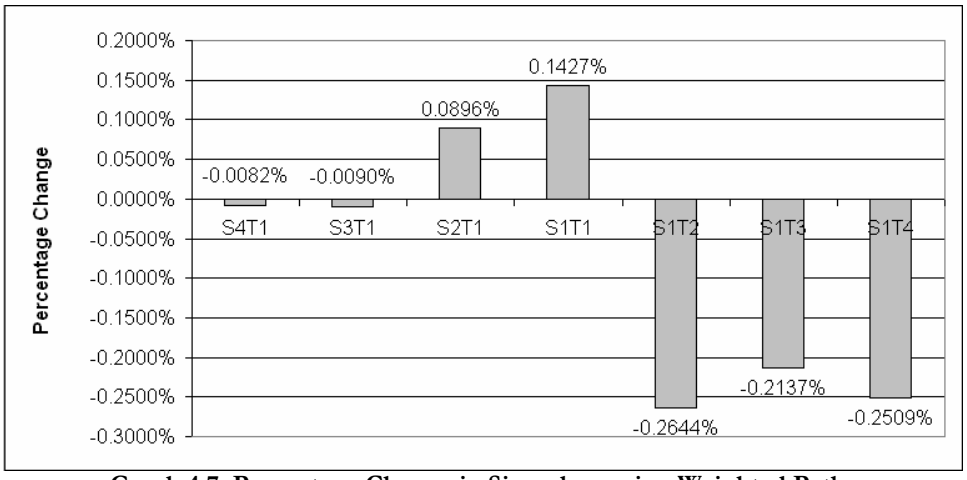
Table 4.8 Final Area and Delay Sum with Weighting



Graph 4.5 Final Area with Weighting



Graph 4.6 Final Delay Sum with Weighting



Graph 4.7 Percentage Change in Size when using Weighted Paths

Although the presented results do not show a significant difference when using weighted paths, this could be a result of the limited number of paths that were constructed for use by the

optimizer. To fully investigate the effect of weighted paths, additional data should be collected and the optimizer should be given a larger representative paths list.

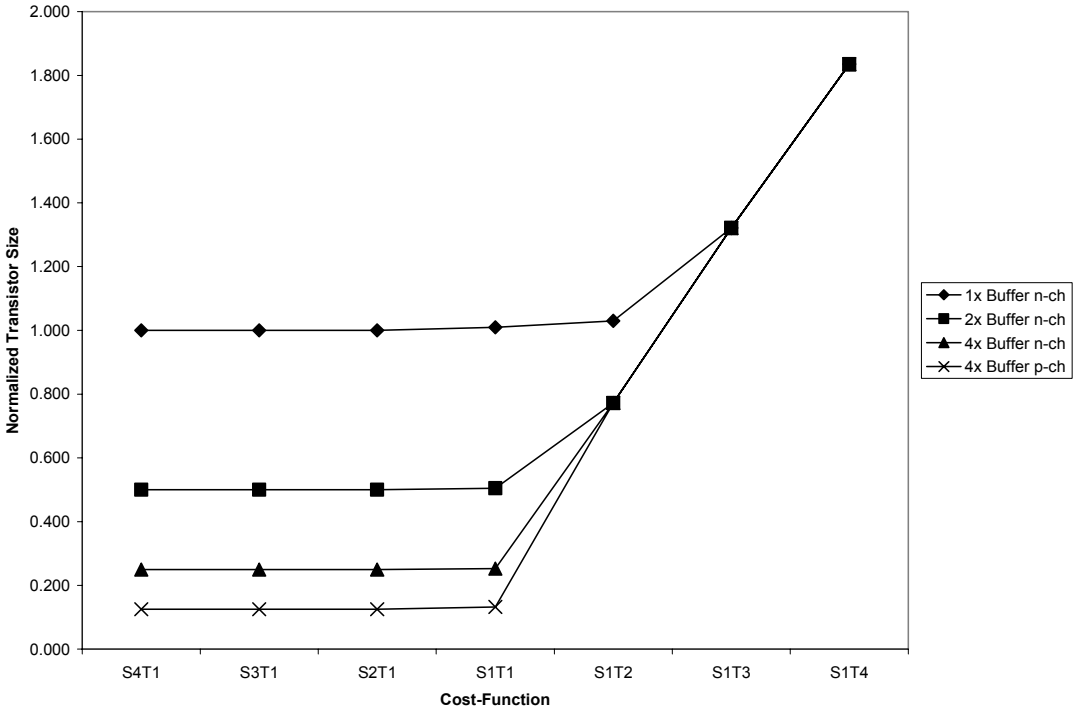
4.4.3 Transistor Sizes by Cell using Unweighted Paths

Further analysis of the tool can be conducted by examining the final transistor sizes for each cell type. Referring to the cell-level netlist file, the different cells that were modified are: three kinds of buffers (1x, 2x, & 4x), three kinds of pass transistors (regular, buffer output, BLE output), and multiplexers. Furthermore, buffers are essentially CMOS inverter stages and as such, two transistor sizes must be specified to fully describe a buffer. A non-inverting buffer is created by putting two buffers in series. Table 4.9 shows the final results for the different implementations of the cost-function.

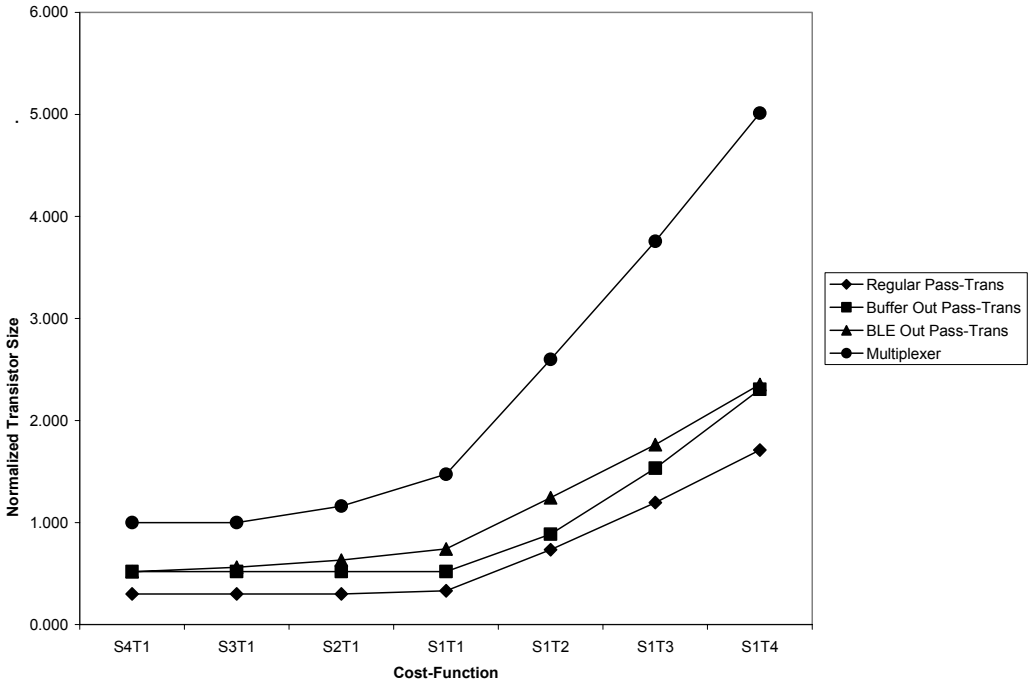
To better visualize the effects of the different cost-functions, the final transistors sizes are plotting against the different functions. Graph 4.7 shows the changes in the buffer transistors and Graph 4.8 shows the multiplexer and pass-transistors. Each curve is normalized to the initial value of its respective cell type. To simplify the plots, the curves for the p-channel transistors of the 1x and 2x buffers have been excluded since they are identical to the n-channel transistor curves for the 2x and 4x buffers respectively.

Cell Type			Initial	S4T1	S3T1	S2T1	S1T1	S1T2	S1T3	S1T4
Buffer	1x	n-ch	1.000	1.000	1.000	1.000	1.010	1.030	1.321	1.835
		p-ch	2.000	1.000	1.000	1.000	1.010	1.544	2.643	3.670
	2x	n-ch	2.000	1.000	1.000	1.000	1.010	1.544	2.643	3.670
		p-ch	4.000	1.000	1.000	1.000	1.010	3.088	5.285	7.339
	4x	n-ch	4.000	1.000	1.000	1.000	1.010	3.088	5.285	7.339
		p-ch	8.000	1.000	1.000	1.000	1.061	6.176	10.570	14.679
Pass-Transistor		Regular	3.339	1.000	1.000	1.000	1.106	2.453	3.994	5.714
		Buffer Out	1.920	1.000	1.000	1.000	1.000	1.703	2.945	4.428
		BLE Out	2.000	1.047	1.123	1.265	1.484	2.489	3.527	4.706
Multiplexer			1.000	1.000	1.000	1.161	1.474	2.599	3.756	5.013

Table 4.9 Final Transistor Sizes by Cell without Weighting



Graph 4.8 Final Buffer Transistor Size without Weighting

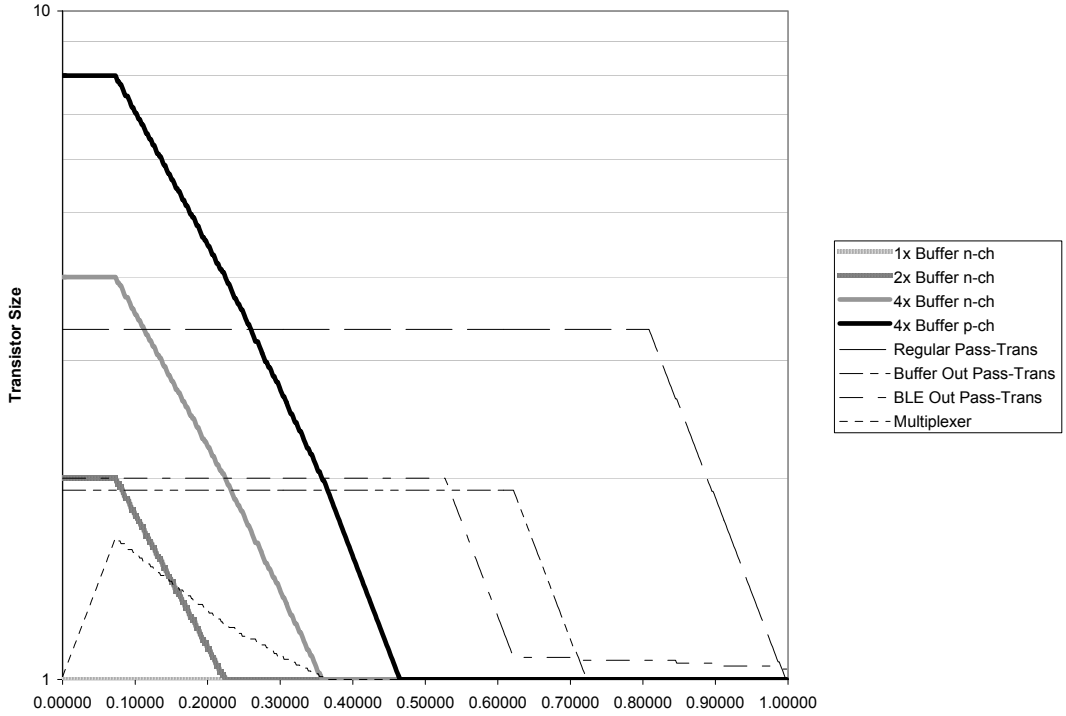


Graph 4.9 Final Transistor Size of Non-Buffer Transistors without Weighting

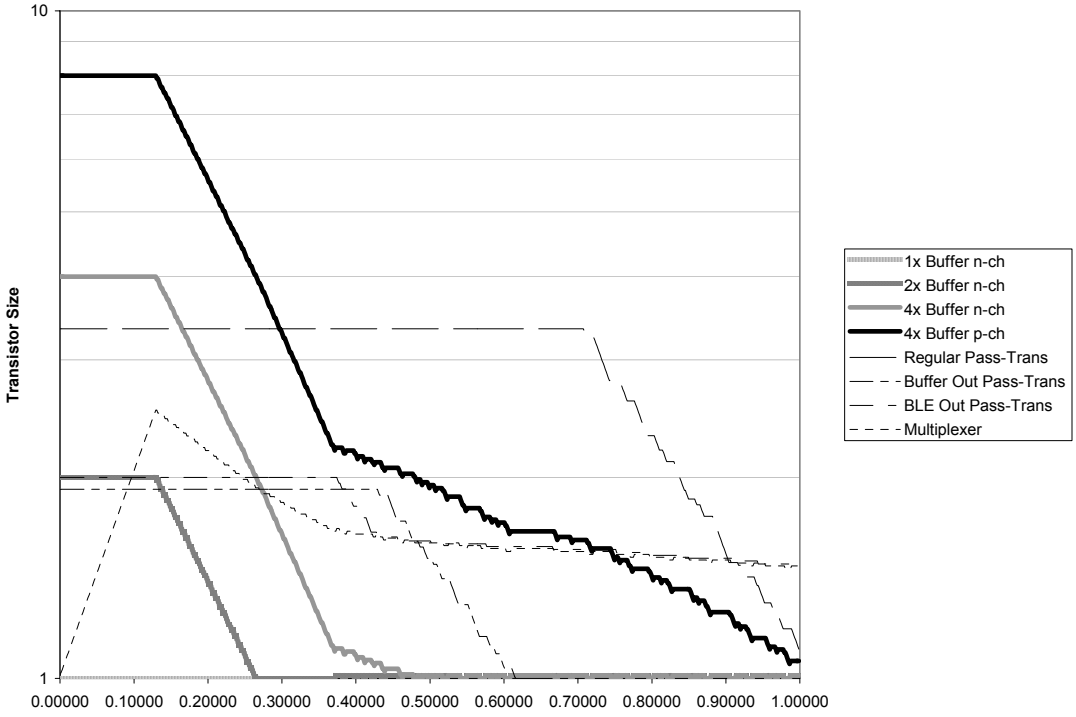
Note that with the graph of the buffer transistors, the curves converge for cost-functions with greater emphasis on speed. The reason for this is that for such cost functions, the transistor sizes are only increased and since all buffer transistors are adjusted together, the curves will be identical in this region. The difference in the curves for cost-functions with greater emphasis on size results from the minimum size limitation and the different initial values. The reason the curves level out is because the transistor sizes can not be reduced further.

Another interesting phenomena is that whereas most transistors are reduced to close to the minimum size when the cost-function emphasizes size, this is not the case for multiplexer and BLE output pass-transistors. This observation is apparent in both the table and the graph. This suggests that these transistors have a large impact on the speed of the device and they are only reduced to minimum size when extreme emphasis is put on size.

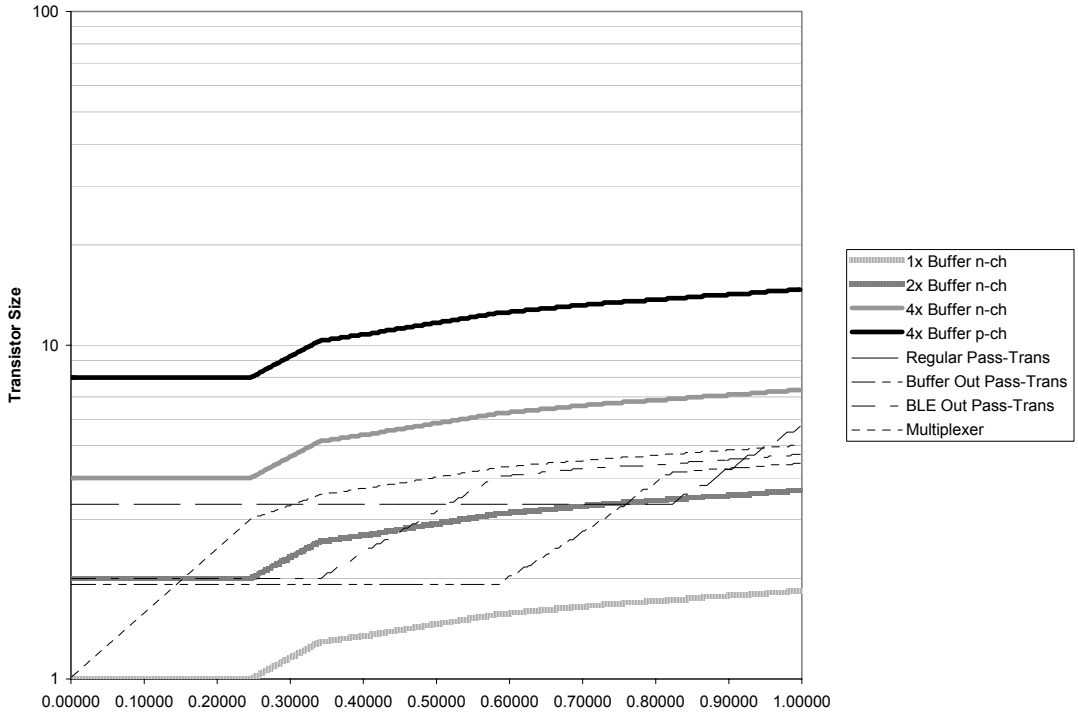
Additional insight can be obtained by looking at the runtime changes to the transistor sizes. The runtime changes of the transistor sizes for each of the cell types for three of the cost-functions can be seen in Graph 4.10, Graph 4.11, and Graph 4.12. The graphs show the results for the two extreme cases (S4T1, S1T4) and the most neutral case (S1T1). To simplify the plots, the curves for the p-channel transistors of the 1x and 2x buffers have been excluded again.



Graph 4.10 Run Time Progress of Transistor Sizes for S4T1 without Weighting



Graph 4.11 Run Time Progress of Transistor Sizes for S1T1 without Weighting



Graph 4.12 Run Time Progress of Transistor Sizes for S1T4 without Weighting

These graphs can be used to verify the order (by cell type) in which the transistors are resized by observing the order in which the curves begin to change. Another observation can be made in the last two graphs which show the interdependence of the transistor sizes. This is exhibited by the continued change in the curve of one transistor type even as the next type is being resized. The reason why this can not be seen in the first graph is that due to the cost-function, the optimal for most of the transistors is the minimum value and there is no further room for movement. Note that the size axis in each graph features a logarithmic scale.

However, the most interesting aspect of these graphs is that in each case, the size of the multiplexer transistors increases at first, even when optimizing primarily for area. This suggests that the minimum size is not the optimal size for the multiplexer transistors when the area is less emphasized. This is further shown in the second graph in which the emphasis on area is

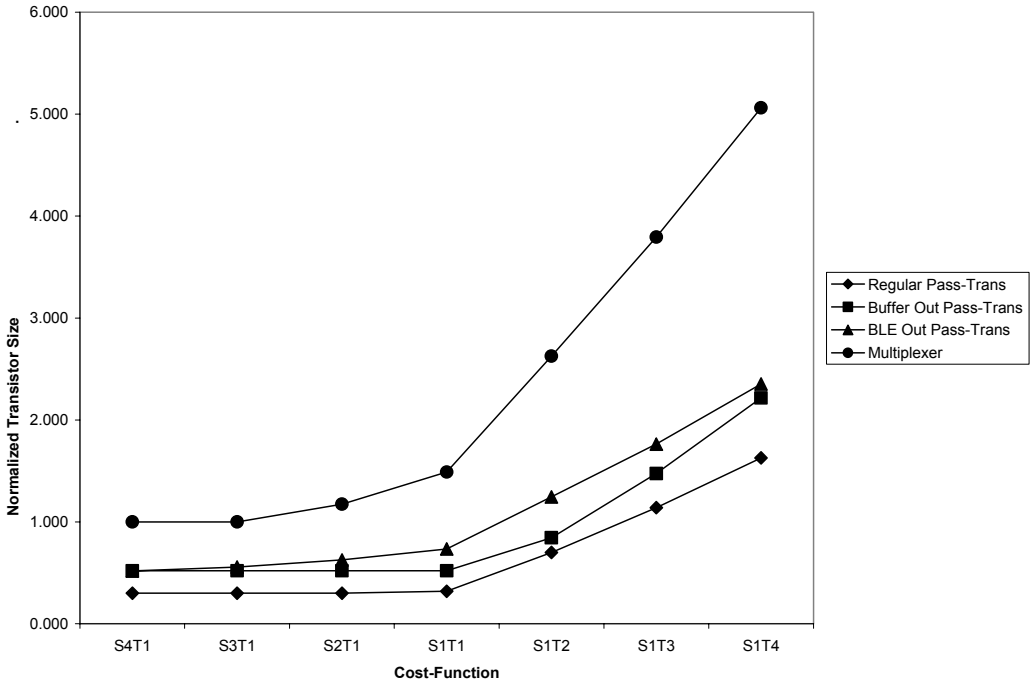
reduced and the size of the multiplexer transistors is one of the few that are significantly above the minimum.

4.4.4 Transistor Sizes by Cell using Weighted Paths

Table 4.10 shows the final transistor sizes when implementing the tool with weighted paths. A direct comparison shows that the differences when weighted paths are used are primarily confined to the non-buffer transistors; there is only one difference among the buffer transistor sizes. Consequently, only the pass-transistor and multiplexer results have been plotted in Graph 4.13. Despite the differences in the values, there is no discernable change in the graph from that of the unweighted version. The effect of using weighted paths is further displayed in Table 4.11, which shows the percentage change in the transistor sizes for non-buffer transistors. Given the similarities between the weighted and unweighted data, the run time plots have not been provided for the weighted data since they would not present any new insight

Cell Type			Initial	S4T1	S3T1	S2T1	S1T1	S1T2	S1T3	S1T4
Buffer	1x	n-ch	1.000	1.000	1.000	1.000	1.010	1.030	1.321	1.835
		p-ch	2.000	1.000	1.000	1.000	1.010	1.544	2.643	3.670
	2x	n-ch	2.000	1.000	1.000	1.000	1.010	1.544	2.643	3.670
		p-ch	4.000	1.000	1.000	1.000	1.010	3.088	5.285	7.339
	4x	n-ch	4.000	1.000	1.000	1.000	1.010	3.088	5.285	7.339
		p-ch	8.000	1.000	1.000	1.000	1.083	6.176	10.570	14.679
Pass-Transistor		Regular	3.339	1.000	1.000	1.000	1.063	2.334	3.800	5.437
		Buffer Out	1.920	1.000	1.000	1.000	1.000	1.621	2.830	4.255
		BLE Out	2.000	1.037	1.112	1.253	1.469	2.489	3.527	4.706
Multiplexer			1.000	1.000	1.000	1.173	1.489	2.625	3.794	5.063

Table 4.10 Final Transistor Sizes by Cell with Weighting



Graph 4.13 Final Transistor Size of Non-Buffer Transistors with Weighting

Cell Type		S4T1	S3T1	S2T1	S1T1	S1T2	S1T3	S1T4
Pass-Transistor	Regular	0.000	0.000	0.000	-3.888	-4.851	-4.857	-4.848
	Buffer Out	0.000	0.000	0.000	0.000	-4.815	-3.905	-3.907
	BLE Out	0.000	-0.980	-0.949	-1.011	0.000	0.000	0.000
Multiplexer		0.000	0.000	1.034	1.018	1.000	1.012	0.997

Table 4.11 Percentage Change in Non-Buffer Transistor Sizes with Weighting

Chapter 5 Conclusion

5.1 Final Results

Although the final values of the tool listed in **Error! Reference source not found.** and Table 4.7 can not be directly compared to evaluate the quality of the cost-functions, the results show that the tool does improve the sizing. Depending on the cost-function used, the results of the function are improved from 50% to 95%, thus showing the effectiveness of the tool. Furthermore, there is an improvement in speed for each function and an improvement in area for all but two of them.

An analysis of the specific sizing of the transistors for each cell type reveals that as the emphasis on size is increased in the cost-function, the multiplexer and BLE output pass-transistors are the last to reach minimum size. This suggests that they contribute significantly to path delays relative to their size.

In the current form, weighting the paths does not appear to have a large effect on the results of the tool. Regardless of the cost-function used, the variance in the numbers between the weighted and unweighted results is well less than 1%. Examining the specific transistor sizing yields a slightly larger difference of just less than 5% for some of the pass-transistors.

5.2 Future Work

One of the obvious areas of improvement is to automate the path selection process. Currently, the work involved in manually creating the paths is not demanding and could be easily automated with an additional function in the program. The current source code actually allows for this and only requires that such a function be written.

Additional work would include improving the delay calculation algorithm. As stated previously, transmission line effects are not current factored in. To implement such an ability would require the results from a placement engine, and optimisation would require iterations of the placement tool before recalculated delays and resizing. Such work is certainly beyond the scope of this project, but could offer interesting results.

The optimizer can also benefit from additional work in two respects. The first would be if an ideal cost-function could be found. As the findings show, different functions lead to different output results. Thus, one must find the most appropriate cost-function to implement in order to obtain relevant results. The second way in which the optimizer may benefit would be the ability to better distinguish between the cells. Currently, all buffers are treated equally, but if buffers of different (original) sizes were resized separately, a better result could possibly be found. In fact this idea could be extended to allow the optimiser to create sub-groups of the existing cells to optimise with more freedom. For example, if the multiplexers could be separated into different groups depending on their functions, these groups could be optimised separately, potentially leading to better end results.

Although the tool can be applied in its current form, the suggestions just mentioned show that additional work can be performed to make this tool even more effective.

Appendix A CD Contents

The CD contains all the source code for the tool. The software used to develop this tool was Microsoft Visual C++ 6.0 and the associated project files are also included.

A compiled version of the tool is included in a separate directory. Examples of the input, path, and output files have also been provided to be run with the tool.

References

[1] Padalia, K.. Automatic Transistor-Level Design and Layout Placement of FPGA Logic and Routing from an Architectural Specification. Bachelor's Thesis. University of Toronto, 2001.

[2] Fung, R.. Optimization of Transistor-Level Floorplans for Field-Programmable Gate Arrays. Bachelor's Thesis. University of Toronto, 2002.

[3] Betz, V., et al. Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers: Boston, 1999.

[4] Martin, K.. Digital Integrated Circuit Design. Oxford University Press: New York, 2000.

[5] Martin, K. "Spice Parameters for a 0.25um Process." Online posting. 13 January 2003 <http://www.eecg.toronto.edu/~martin/nobots/courses/Hspice025.html>