UNIVERSITY OF SOUTHAMPTON

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

High Level Synthesis With Interconnect Prediction

Bleddyn I. Lawrence

July, 2005

A thesis submitted for the title of
Doctor of Philosophy.

UNIVERSITY OF SOUTHAMPTON

High Level Synthesis with Interconnect Prediction

by

Bleddyn Idris Lawrence

A thesis submitted for the degree of
Doctor of Philosophy.

School of Electronics and Computer Science,
University of Southampton

July, 2005

Abstract

High Level Synthesis (HLS) is used to aid the implementation of a design into hardware. HLS achieves this by testing lots of different design architectures with metrics e.g. delay, power, area, etc. to guide decision-making. Hence the metrics need to be obtained quickly while maintaining a high degree of accuracy. The estimators are fundamental to the acceptance of HLS as a design methodology. By enhancing the understanding of how a design will be implemented in hardware, the accuracy of the estimators can be improved. Using the knowledge of the hardware implementation, an a priori estimate of the routing layout of a design can be established and then used to improve the accuracy of the pre-existing estimators, which will in turn give a better representation and improve decision making. A HLS system is presented called MOODS (Multiple Objective Optimisation in Data and control path Synthesis), which has been developed by Southampton University and LME Design Automation Ltd. In order to improve the level of estimation of the behaviour of design at the physical level, i.e. after the design has been implemented in hardware, interconnect predictors will be introduced into MOODS. Circuit partitioning will form the foundations of all interconnect predictors by forming a relative placement. This relative placement can then be used to estimate the average interconnect length of a circuit and quickly obtain a floorplan. This thesis shows that the average interconnect delay of a design on FPGA correlates to a high degree with average interconnect length predicted by MOODS, and when used to guide optimisation design optimality improves significantly. It is also shown that when an APR tool is given hierarchical information obtained during circuit partitioning to guide placement and routing of an FPGA, the overall design optimality in terms of area and delay is improved. The floorplan can then be used to find individual Wire Lengths, as the approximate position of every module on the chip will now be known. These individual Wire Lengths will then be shown to be highly desirable when deciding on whether to perform certain transformations to the design architecture, during optimisation. Finally the future development of interconnect predictors with their applications will be shown.

Table of Contents

Chapter 1 Introduction

1.1 Problem Realisation

Designs that are now implemented on a chip are composed of millions of logic elements or memory cells. This produces a mammoth task when designing a circuit that will perform a function, as the design considerations are so large, i.e. how all the components of the chip interact to produce an optimal design in terms of metrics (Definition 1) such as the size of the chip needed to implement the design, the Clock Period (Definition 2), etc. A list of useful definitions can be found in Appendix 1. The larger the number of components that are needed to realise a design, the more possible combinations of components a design architecture can have, which dramatically increases the complexity of the design process, this in turn dramatically increases the time taken to design an architecture that carries out the desired functionality satisfying all the designer's requirements for the proposed design. Hence a method is needed to handle large designs and speed up the design process. One approach is to raise the level of abstraction of the starting point and to use automatic behavioural synthesis tools.

Definitions:

(1) **Metrics** – A Metric is a value (technology specific) that represents a physical property, which can then be used to compare with their respective design constraint. I.e. the metric for the size of a Xilinx Virtex chip (FPGA) would be the number of slices needed for all the components that are needed for the design to be implemented on the FPGA. The constraint would then be the number of slices available on a particular Vertex chip.

(2) **Clock Period (CP)** - The Clock Period is the minimal period of the clock wavelength, which has to be greater than the delay of the critical path. If the clock period is less than the critical path the functionality of the design is destroyed.

When optimising the design before the design has been implemented on the targeted architecture, estimates of the design's physical characteristics when implemented in hardware need to be made. These estimates are then used to guide optimisation so that a design obtained through optimisation satisfies the design's objectives. A more accurate estimation of the physical characteristics means a better decision can be made as to

which design architecture should be chosen during optimisation. This leads on to the timing closure problem, which is the motivation for this thesis. Timing closure is the process of reducing the number of times a design is run between the optimisation stage and the implementation of a design on hardware. Timing closure can become a problem when a design is implemented on its desired platform and does not meet the designer's objectives. The design architecture will then have to be re-optimised or re-designed, being careful that the design is changed so that the properties of the design satisfy the constraints that caused the failure in the first place. But when trying to satisfy the constraint that caused the failure of the design objective, a different constraint might be unsatisfied, meaning the design has to be re-run in order to satisfy the last constraint failure. Or when optimising the design in order to pass the first failed constraint it does not accurately depict the design once implemented on the desired platform, so the design might fail the same constraint again.

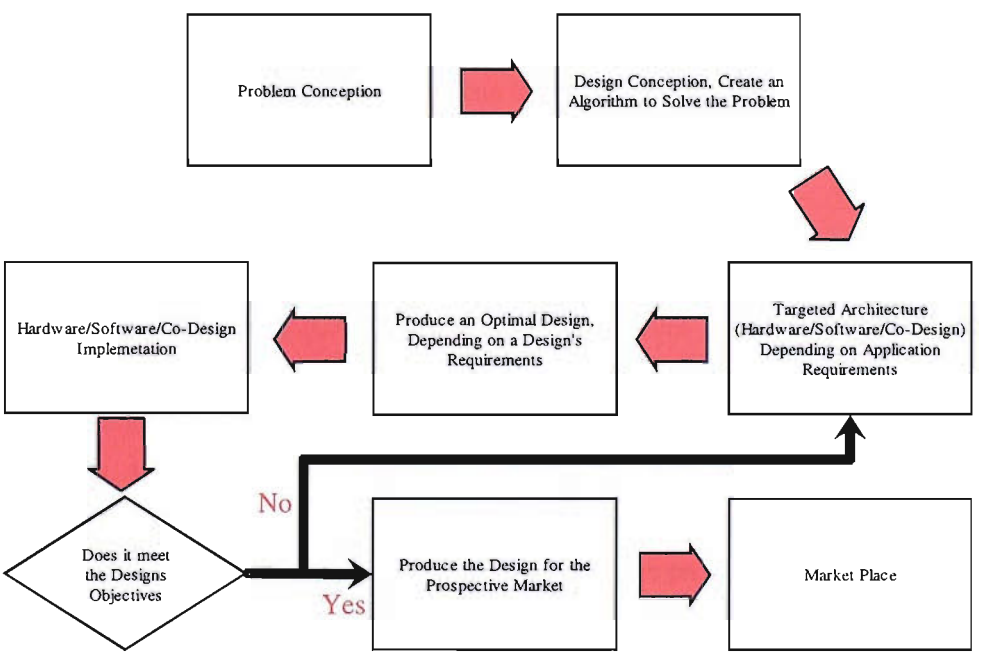


Figure 1: Design Process for a Product to Enter the Market Place

In order to produce timing closure, hence reduce the number of times the design has to re-enter the design optimising stage, an accurate estimation of all the design properties that affect a design once implemented on the desired platform can be applied to the optimisation stage. If the design properties are accurate enough, problems can be

foreseen later on in the design cycle (not meeting design objective) and measures taken to solve problems before the design has been implemented on the desired platform. The design will then satisfy the design objectives without having to enter a costly loop, which can cost time, and time is money.

There are two ways of increasing the accuracy of the estimates of a design's physical characteristics once implemented in hardware. Firstly pre-existing characteristics can be improved by considering individual design architectures and then accurately characterising them into a library, so designs can use characteristics of pre-existing similar designs in the library as estimates. Secondly extra metrics can be added to further understand the behaviour of a design implemented in hardware. Both these methods are employed in this thesis, but the latter is the underlying principle. Interconnect Prediction will be used to further estimate the accuracy of how a design will perform once implemented in hardware. Interconnect prediction on a chip heavily influences the CP, hence the overall delay (Number of CPs * CP) of a circuit, hence the main focus of using interconnect prediction during optimisation is to reduce the detrimental effect that timing properties can have on a circuit. But first the platform on which interconnect prediction will be used to aid the optimisation of the delay of a design will be discussed.

1.1.1 Role of High Level Synthesis

Computer Aided Design (CAD) tools are used to reduce the complexity of VLSI design, by automating steps in the design process. This should make steps in the design process easier for a user to implement and be free from errors (in terms of what the step produces). Another benefit is that the solution should be guaranteed optimal while also reducing the time to complete the step. If a design were implemented in hardware and every minute detail was carried out by the designer, the process would be very time consuming and expensive. So CAD tools are developed to aid a designer. There are three general areas of CAD:

1. Verification
2. Synthesis
3. Design Management

Verification is the process whereby once a design is implemented in Hardware the implementation is tested to see whether it satisfies the design objectives set by the designer. Design Management is the overseeing of the design process, ensuring that

research, theory validation, limiting factors (such as cost, yield, etc.), time to market are satisfactorily managed. In this thesis only optimising a design depending on a designer's objective shall be considered. There are various ways that a design can be optimised with many different considerations. In our case we shall presume the design's functionality remains the same, only different design architectures that carry out the same task shall be considered in the process of optimising a design. This process of deriving the (optimised) design circuit architecture from design specifications is called Synthesis. Synthesis can be broken down into different levels of abstraction. When the synthesis is at a stage when the higher level is a behavioural description and the lower level is a structural description, the synthesis process from higher level to the lower level is known as High Level Synthesis (HLS). HLS uses an objective function, with design constraints such as area, delay, etc. to produce an optimal design with respect to the design constraints. HLS takes the behavioural description and characterises it in terms of estimated physical characteristics of a design, as implemented in hardware. These estimates known as metrics are then used in the objective function, which is then minimised according to design constraints (such as CP, designated chip area, etc.) The objective function is minimised with respect to the design constraints, by altering the design structure, while keeping the original functionality of the design. The architecture of the design can change many times; hence the estimates need to be obtained very quickly while remaining accurate.

With the current rate of progress in technology, designs need to be in the market place as soon as possible or competitors will realise their version of the design first. Hence a methodology is needed that produces an optimum design satisfying a designer's demands, but in as short amount of time as possible. High level Synthesis (HLS) is such a methodology; it is used to aid the implementation of a design into hardware. HLS achieves this by testing lots of different design architectures which all carry out the same task. The different architectures are compared using metrics, e.g. delay, power, area, etc. to guide decision-making. Hence the metrics need to be obtained quickly while maintaining a high degree of accuracy. This high accuracy allows intelligent decisions concerning the design's architecture to be made that in turn increase the quality of the eventual design. The estimators are fundamental to the acceptance of HLS as a design methodology. Multiple Objective Optimisation in Datapath Synthesis (MOODS) is a HLS tool [20], developed in the Electronic Systems Design Group (ESDG) at the University of Southampton. This tool will form the foundations for the

work covered in this thesis. Once the HLS tool has converted the behavioural description of a design into an optimised structural description of the design, this description then needs to be implemented on a chip. The process of implementing a design on a chip is called Physical Synthesis, when both the higher and lower levels are structural representations.

1.1.2 Automated Placement and Routing

Another CAD tool is used for the implementation of a design onto hardware. The tool that converts a design from a circuit netlist to a description of a design that can be implemented on a chip is called Automated Placement and Routing (APR) tool. This tool does not alter the netlist of a design, but decides on where all the elements that make up the hardware get placed on the chip, as the name would suggest. Placing (assigning cells to locations on a chip) and Routing (assigning nets in the netlist to tracks in the routing channels on the chip) a chip takes a long time to complete, with the majority of the time spent routing as it is the most complex task. This is why it is important to minimise the number of times the APR tool is visited. Ideally the APR tool should only be visited once through a correct characterisation of the design during HLS, hence producing timing closure (Definition 3).

Definition:

(3) **Timing Closure** – Timing closure is the problem where multiple design iterations are caused due to unrealistic design representations, increasing the time to develop a product.

1.1.3 Producing Timing Closure with HLS

A HLS tool can also be used to tackle the problem of timing closure by enumerating all the physical properties accurately early in the design cycle (in order to satisfy design constraints before the costly (in time) process of implementing a design in hardware). If the design's physical properties once implemented in hardware are accurately depicted in HLS, the design objective should be fulfilled when implemented in hardware, without the need to re-synthesise a design due to failing the design objective. For example the HLS tool passes on a design to the APR tool, but if when processed the design's clock period is too large then the design re-enters the HLS tool, with a condition that the clock period needs to be reduced. To reduce the clock period the area is increased, but once the design has been processed the design is found to be too large

for the chip. Hence another iteration of the design loop needs to be undertaken, and by removing these multiple iterations the timing closure problem will be solved.

There are two strategies for producing timing closure through increasing the accuracy of estimating the physical characteristics of a design once implemented in hardware. The first strategy is when a design is run from HLS then run through the APR tool, and then the physical characteristics of the design (area, delay, etc.) can be passed back to the HLS tool. The HLS tool can then be re-run with accurate physical characteristics of the design in hardware. The synthesised design is then passed back to the APR tool and re-run. If the new design architecture still fails to meet the design objectives then the process is repeated and the design is passed back to the HLS tool, where the process repeats until the design objectives are met or the design cannot be optimised anymore. The problem with this methodology is that even though the accuracy of the metrics used during synthesis would be very high, every time the design is passed back to HLS, the new physical characteristics can become redundant very quickly once the design architecture has been altered. A slight change in the design architecture during synthesis can lead to a dramatic change in the hardware implementation. To combat this change in the architecture, the APR tool could be run after small changes to the design architecture during HLS. But this would be extremely slow as an APR tool performs a very complex task. Due to the complexity, the APR tool takes a long time to complete placement and routing. Hence the slowness of running the APR tool frequently goes against the underlying principle that HLS compares many different design architectures so that it can extensively search the design space, while still remaining feasible in run time. Visiting the APR tool many times negates the objective of using Physical Estimation to improve design optimality, while producing timing closure. The next section will introduce those properties of a physical implementation that will be estimated with the intention to satisfy the design objective. The second methodology is to use highly accurate estimates of the physical characteristics of a design if placed on hardware, while also being quick to quantify. This is where interconnect prediction will be used as the routing characteristics of a design heavily influence the delay of a circuit and can also affect the area of a chip. So by taking the interconnect characteristics into account, augmentations of the design architecture during HLS can be accurately evaluated by how the changes in design architecture will affect the delay of a chip once placed on hardware.

1.2 Representation of a Design

When creating a design in hardware there are several possible levels of description, which include:

- Behavioural Level, this is the highest level of abstraction, similar to C programming, as it does not describe the hardware implementation.
- Register Transfer Level (RTL), concentrates on design at the register level.
- Gate Level, this describes the gates, flip-flops and their interconnections.
- Switch Level, this is the most detailed where the layout of all the wires, transistors and resistors on an Integrated Circuit are described.

The two levels that are used and produced by HLS are the Behavioural Level and the RTL Level. A language needs to be used that describes both these levels so that optimisation can be performed. The description needs to model the eventual hardware that would be produced if the design were implemented on a chip. This model of the hardware can then be simulated, allowing design faults to be identified and corrected before the design is implemented in hardware. Hence Hardware Description Languages (HDLs) have been produced. HDLs can be used to describe the behavioural level and the RTL and the more detailed the description that is used is dependent on the design's designated chip.

Behavioural HDL (BHDL) does not consider the structure of a design, as implied by the name. This means the functions that are performed in a design are not considered in terms of delay or area, only that the tasks are carried out in the design are undertaken in the correct order so as to make the design functionally correct. As delay of functions is not considered, the CP of a design is not used, which is essential beyond the behavioural stage.

Register Transfer Level (RTL) uses black boxes to represent functions in a design. These black boxes are then used within the optimisation process of the HLS tool, as they contain all the physical estimates of the functional units (when implemented on a chip) needed for a design to be successfully optimised. During optimisation different architectures are formed which all represent the same design, the estimates of area, delay, etc. are then used to decide whether one architecture will lead to a higher reduction in the cost function (Definition 4) compared to another design architecture.

Definitions:

(4) **Cost Function** – A cost function (also known as an objective function) represents all the design criteria on which the optimisation process will base its decisions.

(5) **Critical Path** – The Critical Path is the largest Register to Register delay of a circuit.

(6) **Signals Nets / Nets** – Signal Nets are signal nets are defined as sets of points that are to be electrically connected together.

These black boxes contain all the information needed to represent the structure of a design. In this thesis these black boxes will be called Macros. A Macro is a collection of pre-characterised logic functions that perform a particular task such as addition. The values in the black boxes are area of a functional unit, delay to carry out the function, power consumption. At the RTL stage these values are estimates of the actual values that would be found at the physical level. The clock period is now considered so as to ensure the design is legal. By legal we mean the critical path (Definition 5) is smaller then the desired CP. At the behavioural level there is high level of abstraction from the eventual hardware implementation, which allows more flexibility in the design. Hence estimation of a design's performance is achieved faster than if a design is implemented in hardware and its physical properties such as area and CP are recorded.

1.3 Design Flow from Software Representation to Hardware Implementation

Figure 2 shows the design process from a design's conception to its implementation in hardware. The first step of the software to hardware refinement process is to write the proposed design in behavioural VHDL. To verify that the design has been correctly written according to its functionality, the design needs to be compiled and simulated in an HDL simulator. The HDL Simulator uses a testbench to simulate a design written in HDL, to test whether a design works prior to physical implementation. A testbench is written to provide stimuli for simulation. These signals are input (Definition 6) signals that will be found in the final physical implementation.

These signals can represent values that are bit vectors, integers, hexadecimal, etc. When input into the HDL simulator, the design will simulate the actual working of the design as if those same inputs were applied to the final physical implementation of the design. But only timing issues are considered, no physical considerations such as cross talk are

modelled. When the design is simulated observation of all the signal values that are produced (as if the design was running in hardware) can be undertaken. Obviously prior (correct) knowledge of how the design is meant to behave is needed. If the behaviour of the design simulation matches the desired behaviour, then the design is deemed to be functionally correct. If the design does not simulate correctly the design is altered to carry out its task correctly. Once the correct behaviour is achieved the design is loaded into MOODS. MOODS synthesises the Behavioural VHDL (at this level there does no need for the consideration of the clock or registers within the code), where the synthesis process converts the Behavioural VHDL to RTL VHDL. The RTL VHDL is an optimised structural description of the Design that was input into MOODS. This optimisation is dependent on design objectives set by the user/designer. This conversion will be discussed in much greater detail in section 2.7.

A low-level synthesis program is used to convert the VHDL of the design optimised in MOODS into Electronic Design Interchange Format (EDIF). An EDIF file contains the netlist of the design, which shows what components are used and the topological layout. During the conversion from RTL VHDL to EDIF, the design is further optimised at the Gate Level without changing the functionality of the netlist. The optimisation process can remove redundant logic decides the most optimal mapping in terms of criterion such as area, delay, etc.

The EDIF then can be transferred to an Automated Placement and Routing (APR) tool. This starts with the netlist contained in the EDIF file and then places the components which are needed for the design on a chip and routes the design together. This final stage is at the Physical Level, where the circuit is designed with the components that would be used in the actual manufacturing of the circuit.

1.4 High level synthesis with Interconnect Prediction

As stated earlier, when design exploration is performed during HLS, metrics are used in order to decide which design architectures are more optimal than others (depending on the priorities chosen by the designer). The more comprehensive the information provided by these metrics, the better the final design architecture will be. Using information both from a High (Abstract) Level and Physical Level perspective, decisions can be made at a higher level where the most design flexibility exists, but with the knowledge of the decision's consequences at the physical level [2]. Figure 2

shows in a more detail when timing closure is performed; the dashed line represents the loop, which occurs if the implementation of a design does not meet certain conditions, and hence needs to be re-optimised (to enable the design to be implemented in hardware).

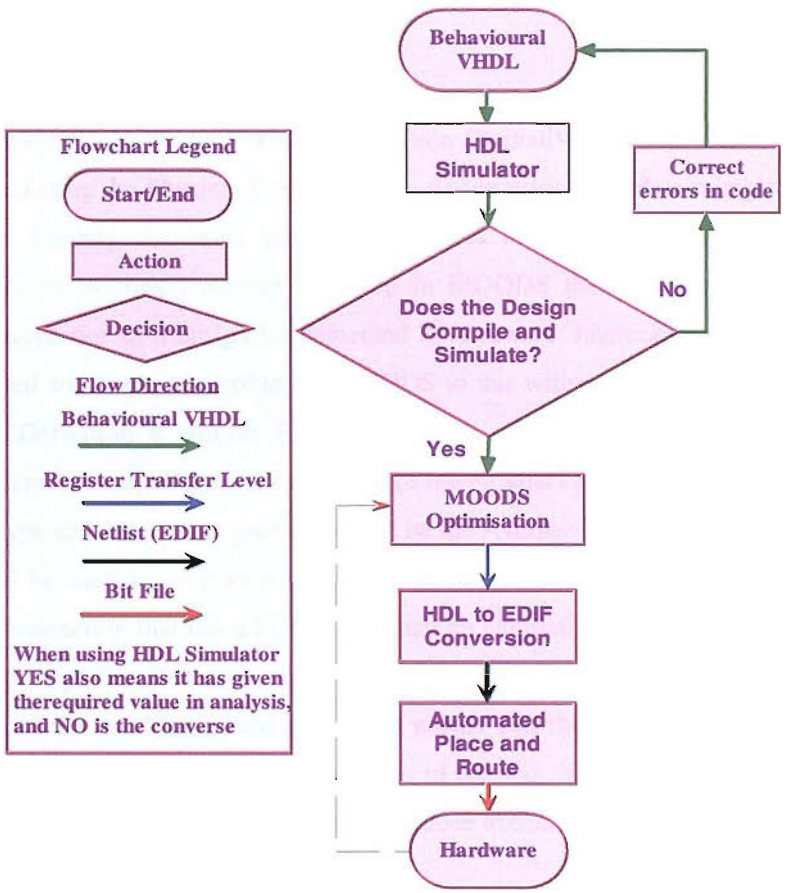


Figure 2. Design Process from a Design Represented in Software to a Design Implemented in Hardware

In this thesis only delay (including interconnect prediction) and area will be considered when optimising a design. At the moment MOODS, when minimising the objective function of a design, does not use any interconnect properties as a constraint in order to minimise the CP (total delay) of a circuit. This causes a discrepancy between what MOODS thinks is an optimal design architecture in terms of delay and area, but really is sub-optimal through a poor routing layout, which can in turn increase the CP if nets on the critical path are elongated through not being able to find an optimal path. The

delay caused by interconnects between gates in ASICs is becoming an important factor, as the delay caused by gates is decreasing. But in this thesis the main focus will be on interconnect prediction for FPGAs. To allow for a range of different designs to be implemented on an FPGA, the interconnect resources dominate the area and delay of an FPGA, making interconnect a greater influence when choosing a design architecture, and that is why in this thesis the target chip for the design will be an FPGA.

1.5 Interconnect Properties

To allow MOODS to measure tradeoffs between alternative design architectures, some way of predicting the Physical Properties of a design prior to hardware implementation is needed. Finding the exact physical properties of a design would be too time consuming, so estimated metrics are used in MOODS that represent the predicted physical properties of a design implemented in hardware. Interconnect properties will be estimated and made accessible for MOODS to use within optimisation (in the Cost Function (Definition 8 section 7.1)), where area and delay of functional units are already characterised). There are two types of interconnect prediction that will be used. The first type of interconnect prediction will be the Average Interconnect Length (AIL), which will be used to give an estimate of the level of congestion. If a design has a possible architecture that has a higher AIL than an alternative architecture for the same design, this signifies that the second architecture's nets will generally take up less routing resources of the targeted chip. This means that the nets have more chance to attain optimal paths, hence reducing the delay of the nets. If interconnect prediction can be performed accurately then designs that produce optimal routing plans (Definition 7) should be chosen, this will then reduce the CP.

Definition:

(7) **Routing Plan** – Routing plan is the layout of the routing on a chip.

The reason for CP reduction is that if routing plans are optimal, then interconnects between macros should be optimal; hence the delay between macros will be optimal which should reduce the CP if the macros being considered lie on the critical path. The next factor to consider is how long nets are between macros; hence knowledge of the approximate location of the macros is needed during HLS. This approximation of the final placement of the macros in hardware will be known as a Floorplan. The floorplan will provide the approximate distances that macros lie from each other. This will allow

greater accuracy on how the structural changes to a design will affect the interconnect layout of the final physical implementation. For example the longer interconnects are between macros, the larger the delay, hence the larger the detrimental effect on the design, especially if these macros lie on the critical path.

1.6 Structure of Thesis

In this chapter the focus has been on what type of physical information is required and why the information is desirable to be included in MOODS to aid design exploration during the optimisation loop. Chapters 2, 3, 4 will show how this physical information will be obtained prior to the design entering the APR stage and in what way the information can be applied. Chapter 2 will discuss circuit partitioning in detail, which will form the foundations of all the physical estimates. An estimate of the interconnect complexity using only hierarchical information obtained from circuit partitioning will be pursued in chapter 3. This estimate will then be used in MOODS cost function, so as to influence decision-making during HLS. Also the hierarchical information will be shown to improve a design's optimality in terms of area and delay when used to guide place and routing within an APR tool. To obtain more in-depth information, a floorplan will be constructed. This floorplan will be used to only gain knowledge of the rough proximity all the shapes are to each other. This new information will then be made available to MOODS as discussed in chapter 4. The Heuristic will base transformation decisions on a design's architecture depending on where they are placed on the circuit and what the circuit conditions are. Results presented in chapters 3 and 4 will prove that Physical Level estimation is a viable resource in HLS. Chapter 5 will discuss what enhancements could be made and what future developments concerning MOODS are possible using physical estimates to improve the final design architecture after design exploration.

Chapter 2 Literature Survey

2.1 High Level Synthesis

As explained in chapter 1, a process in which a representation goes from behavioural level to structural level is called High Level Synthesis (HLS). A HLS CAD tool is used to provide an optimal RTL representation in a relatively short amount of time satisfying the designer's requirements. A CAD tool is used in order to automate processes that are carried repeatedly; if a process was just carried out once there would be no point in automating it. Once a process is automated it frees up time that can be spent carrying other tasks. Automation also reduces the risk of human error.

Before a design is synthesised in a HLS tool, first a design language needs to be used to describe the function. This description can then be used to model the hardware. Faults in the design can then be identified and corrected before synthesis has even been started. Behavioural VHDL (B VHDL) is used to describe the design as if it were implemented in hardware, but in an abstract manner. The description is abstract as it only contains the functionality without any timing specifications, allowing the user to concentrate on the algorithms contained in the design rather than any timing issues. When the HLS tool receives a functionally correct design described in BVHDL, the HLS tool transforms the description into a format for efficient optimisation. This format is known as a data structure and it is designed so that information is easily available, easily alterable and requires minimal memory requirements and computational power. All these factors (except for minimal memory requirements) reduce the period of time that the HLS tool takes to complete its task, as the same data structure is used throughout optimisation. The data structure within the HLS tool is now at the structural level, as it includes structural (include area, delay, power, etc.) information about the target components obtained from libraries. These library components can be thought of as black boxes that can be used to represent functional units in a design such as Registers, ALUs, and MUXes. From this point we shall refer to these black boxes as Macros. As defined in Chapter 1, a Macro is a collection of pre-characterised logic functions that perform a particular task such as addition. The metrics (delay, power, area, etc.) values are all estimates of what the actual values would be if the respective macro was implemented within a design on a chip. These values need to be estimated, as other than actually physically implementing a

design there is no way of knowing the actual design metrics for all possible configurations that a large set of macros can take within the design space. The more accurate the metrics used for HLS, the more likely the design will meet its objectives, whether in terms of meeting a clock period or fitting on a chip. This leads onto the next question, why cannot the design simply be implemented in hardware and find out the real values, then augment the design accordingly to meet the designs objectives? This is where timing closure shall be discussed, but first an outline of HLS must be completed.

The data structure is augmented through the design space. Design space means all possible structural configurations that a design can take while remaining legal i.e. structures that do not change the functionality of the design. This can be done through a random process (randomly picking different design architectures, while remaining functionally correct) or through an intelligent process (using knowledge of the optimality of previous designs to guide the optimisation of future designs). For every design synthesised within a HLS tool, exploration of the entire design space could be pursued, but would take a long time when designs are very large. For HLS to produce an optimal design an Objective Function is used. The objective function within a HLS is used to compare design criteria such as delay and area of different architectures. This objective function is then minimised to find the most optimal design, i.e. the design architecture that minimises the objective function will be the most optimal design, and this design will then be passed onto the APR tool.

Some formal definitions follow:
Given a function $f: A \rightarrow \boldsymbol{R}$ from some set A to the real numbers ($\boldsymbol{R}$). This function becomes a cost function (also known as an objective function) when the following property is sought: an element x_0 in A, such that $f(x_0) < f(x)$ for all x in A (minimisation) or such that $f(x_0) > f(x)$ for all x in A (maximisation). The challenge of finding the minimum/maximum of a cost function is called an Optimisation Problem. The domain A of f is called the search space, while the elements of A are called feasible solutions or candidate solutions.

So to explore the design (search) space, algorithms are used that find global or local minima. A local minimum is a point where the objective function takes on its smallest value among all points in the immediate vicinity. Global and local minima are shown in Figure 3.

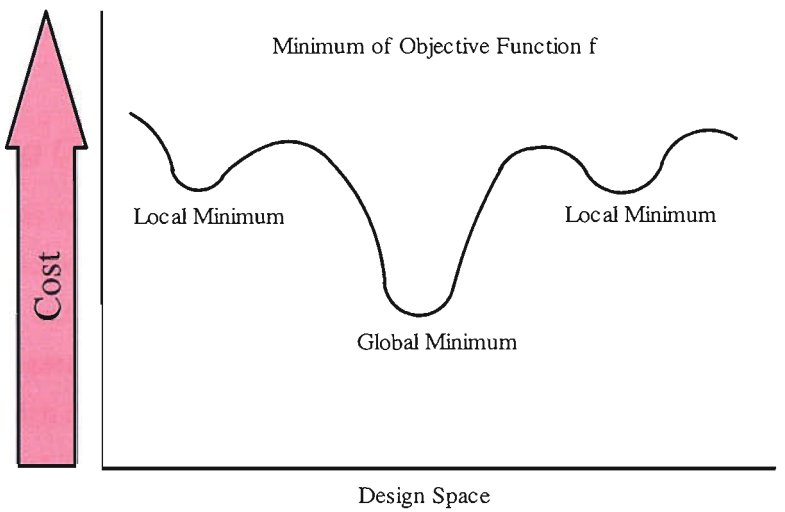


Figure 3. Graph to Show the Local and Global Minima of an Objective Function *f*

Optimisation algorithms that attempt to find the global minimum during HLS in MOODS will be discussed in section 2.7.3. The depth of their search for the global minimum greatly depends on the time given in order to search the design space. Once an optimal design has been found, the final step in HLS is to convert the data structure into RTL VHDL, which can then be passed on to an APR tool. This conversion process is out of the scope of this thesis but details can be found in [20].

2.2 Automated Placement and Routing (APR)

The optimisation constraints for routing on an FPGA are used to minimise interconnect lengths. These constraints depend on two major factors, which are Routing Demand on the routing channels (Definition 8), which is how many routes wish to be placed within the routing channel.

Definition:

(8) **Routing Channel** - The routing channel runs between the cells on a chip and this is where nets are placed.

The other factor is routing supply, which is how many routes can be placed on tracks within the routing channel. The higher the routing demand on a routing channel, the more likely that the routing channel will become full, which can force some paths to be placed in alternative routing channels making the path non-optimal. This sub-optimality will then increase the net length; hence increase delay between cells on the FPGAS that are connected by the nets having to find sub-optimal paths. Hence the APR tool will try to

minimise the demand on each routing channel while decreasing the distance between cells on the layout.

2.3 Timing Closure Problem

Figure 4 gives an overview of the design flow from design conception right through to physical implementation. A problem occurs when a design fails its design objectives. After being placed and routed in the APR tool, there are two options available to the designer. Firstly a designer may wish to tweak the design depending on their competency, and how close the design is to meeting its objectives. Secondly the designer may go back to the HLS tool and repeat the synthesis step, but this time with different constraints in order to meet the design's objective, post APR.

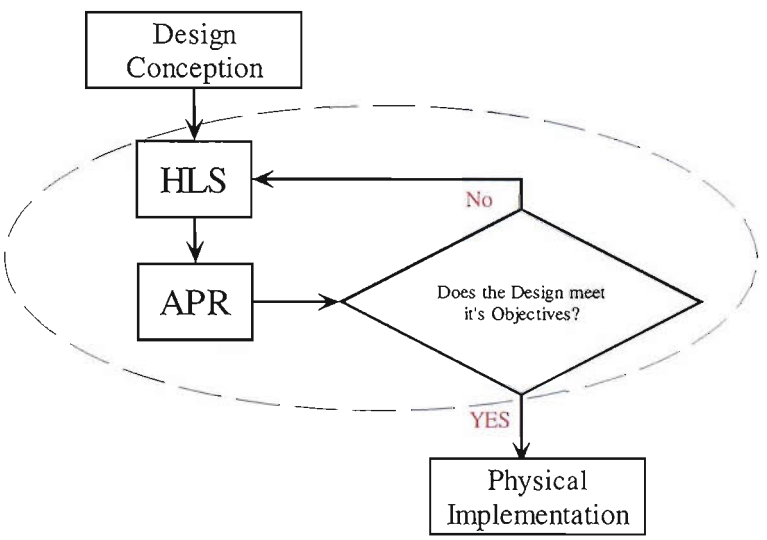


Figure 4. An Overview of the Design Flow from Design Conception to Physical Implementation

The second option is a very time consuming loop, especially if it is repeated many times. This is where reducing the number of loops can produce timing closure. Timing closure is the motivation for this thesis, as by providing a more highly optimal design through increasing the accuracy of the estimation of the physical characterisation of a design, there is less chance that the design will fail to meet the design objectives when implemented in hardware. An increase in the level of characterisation of a design (Interconnect Prediction), allows the HLS tool when optimising the design, all possible problems (with respect to area and delay) can be realised, and thus the problems can then

be avoided. Figure 4 shows the opportunity where timing closure can be performed. Closure can be obtained by reducing the number of times a design has to be passed back to HLS, due to the design failing one of its design objectives, such as it might be too big for a chip, or it the critical path delay exceeds the desired CP.

The more accurate a design is depicted (in terms of its physical implementation) during HLS, the more optimal the final design will be when placed on the chip. This will now be shown in section 2.6 through some examples. But first the general architecture of an FPGA will be described; as it is important to understand the platform on which the design will be eventually placed (a detailed description will be given in Chapter 3). This is important, as accurately depicting the properties of the architecture during HLS will enable accurate optimisation.

2.4 General FPGA Architecture

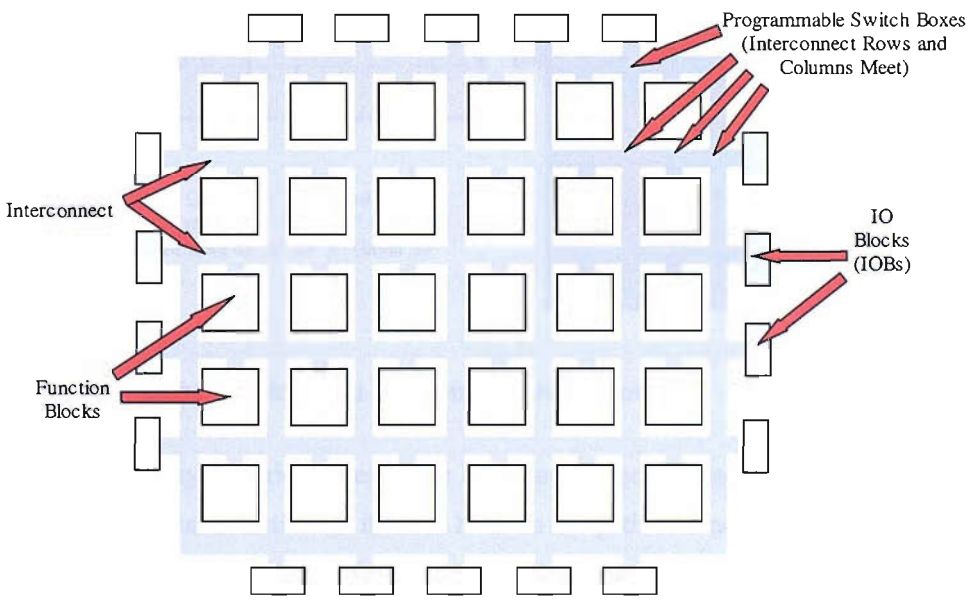


Figure 5. General Architecture of an FPGA

An FPGA architecture generally consists of an array of functional blocks, interconnects and IO connections. All these elements are reconfigurable, allowing the FPGA to be re-used to represent different design architectures. The general FPGA structure (known as Island Style) can be seen in Figure 5.

Each functional block can be reconfigured to provide a combinational or sequential logic function. The interconnect structure is comprised of horizontal and vertical routing

channels. These routing channels run in between the functional blocks. The horizontal and vertical routing channels are connected using programmable switches. How horizontal and vertical tracks combine to form a route is shown in Figure 6. Figure 6 shows this Horizontal Vertical routing on a Manhattan Grid, which can be used to model the chip layout. A Manhattan grid consists of a rectangular space between two parallel rows of pins (terminals). The locations of these pins are fixed and aligned with vertical grid lines. A Manhattan grid is often used to represent the layout of a chip.

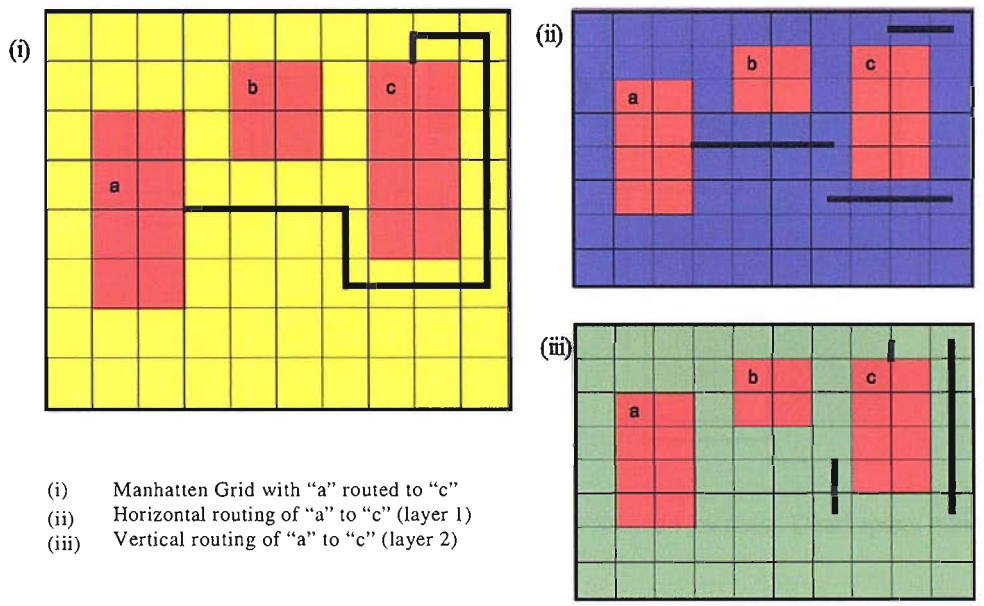


Figure 6. Manhattan Grid with Horizontal-Vertical Routing.

This means one layer carries wires in the horizontal direction and the other layer carries wires in the vertical direction. If there is bend in the path of a net, then the net uses the programmable switch to change from a vertical interconnect to a horizontal interconnect or vice versa as shown in Figure 6 (ii & iii). Programmable switches are also used to connect the routes leading from the functional blocks and IOBs.

2.5 Overview of the Optimisation Considerations during Automated Place and Route

Now that the general architecture of an FPGA has been described, how an Automated Place and Route (APR) tool optimises a design architecture needs to be considered, as the

final design architecture will be very dependent on how the design is optimised during Place and Route. Hence by understanding the way an APR tool optimises a design, a better estimation can be made during HLS of the final design architecture post Place and Route. An APR tool consists of three main stages:

Placement → Global Routing → Local (Detailed) Routing

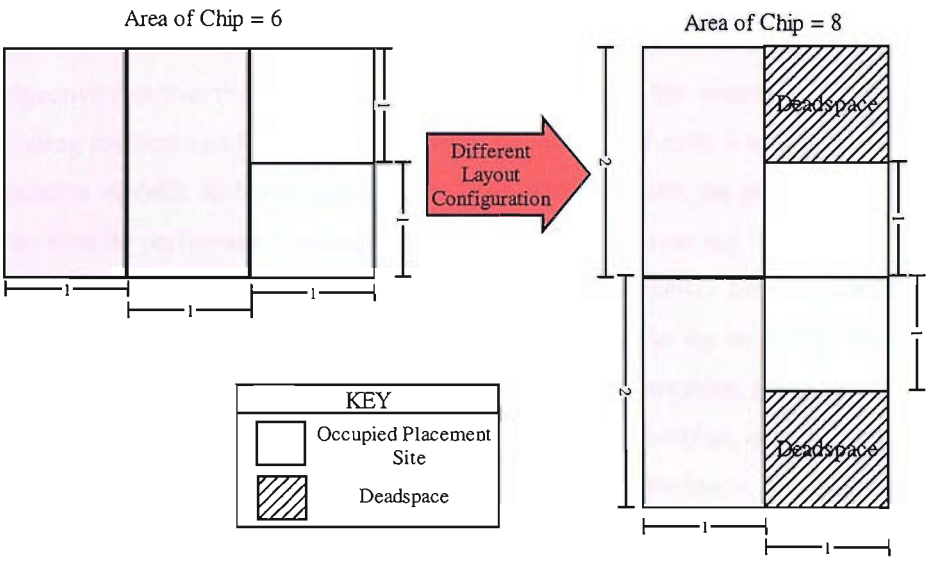


Figure 7. Diagram Showing How Different Layouts can Increase Dead space which in turn Increases the Size of a Chip.

An APR tool, when placing a circuit, can consider many constraints but in this thesis only clock period (hence total delay) and area are targeted during optimisation. The first reason an APR is a slow process is that macros are not being dealt with, the logic blocks or gates (cells) that make up these macros are. A cell is a small circuit macro such as a two input NAND gate or a CLB. This increases the complexity of the Place and Route problem, as now there are many more elements to deal with, while also having to contend with the routing that joins all these cells together. The placement phase attempts to minimise the Dead space (Definition 9), while providing a good foundation for delay minimisation.

Definition:

(9) **Dead Space** – A placement site that is not occupied is known as Dead Space.

An example of how dead space can be minimised can be seen in Figure 7. The benefit of dead space minimisation is that with less dead space the smaller the chip that can be used, while also allowing macros to be placed closer together. The placement phase is relatively quick compared to the routing stage, as there are fewer constraints and estimates of the

routing are used to aid the optimality of the placement in terms of delay (CP). Before placement a floorplan is generally constructed which is a generalised (more abstract) placement, where macros/cells are grouped into regions. These regions are then arranged into an optimal placement and then the macros/cells within the regions are optimally placed. This reduces the complexity of the process while keeping the placement optimal, as macros are highly connected; hence they should be placed together.

Routing Layout is much more complex as the routing placement has a highly complex objective function that has many constraints; this makes the routing stage very slow. The routing problem can be defined as follows: Given a set of cells, a set of signal nets and the location of cells on the layout that has been obtained from the placement stage, routing can now be performed. Routing consists of finding paths on the layout surface, in which wires can be placed in order to connect pins together to satisfy the design's functionality. To ensure the delay is minimised these paths are kept to the minimum possible length with respect to constraints. The complexity of the placement problem is reduced by grouping cells/macros together into regions to form a floorplan, optimising the floorplan with respect to area, delay, then finding an optimal placement, followed by a further optimisation within each region. This same methodology is generally used for routing as well. Routing can be split into two main stages, firstly global routing, and secondly local (detailed) routing.

Global routing is used to assign each net into a particular region on the layout surface, making sure that routing demand does not exceed routing supply. Once all the nets on the layout surface, have been assigned to a region, local routing takes over. Local routing is far more detailed, as it assigns nets to particular routing tracks that run in the routing channel, again satisfying an objective function, but this time it has many more constraints such as crosstalk, routing channel blockage, CP, etc.

Global routing can be modelled during HLS, as it is still fairly abstract and does not have too many constraints. Local routing will not be discussed, as it would be extremely hard to estimate local routing, as there are far too many factors to contend with. The extra accuracy in estimation of the interconnect layout which would be achieved would by too time consuming for it to be feasible during HLS. Presently MOODS only considers macros not the individual cells that make up the macro, and does not contain most of the information needed to estimate Local Routing.

2.6 Factors Involved in Metric Estimation during HLS

Area (slices on an FPGA) and CP of a design will be the target for the optimisation during HLS. Hence all the physical characteristics that affect area and CP on a design once implemented in hardware need to be accurately estimated. There are more optimisation considerations, for example power, but they are out of the scope of this thesis; here we are only concerned with area and delay.

Accurate area estimation can easily be achieved by observing how many placement sites (in terms of slices) a macro occupies when implemented in a design. This placement of the macros might change from one design architecture to the next, but the actual area of a macro does not greatly change. When calculating the area of a design, the area of individual macros is calculated then summed together to form the total area (number of placement sites) needed for the design to be placed on a chip. Dead space does not need to be accounted for when estimating the number of placement sites a design needs in order to be placed on a chip, as the dead space on an FPGA can be minimised to a negligible amount, if the number of slices needed to place a design on an FPGA, is close to the number of placement sites on the designated chip. This reduction in dead space is due to cells belonging to macros not having to be placed in adjacent placement sites (with the exception of macros that use carry logic on the Xilinx definition). Hence the individual cells that make up a macro can be placed into dead space if there is no room for the macros to be placed in adjacent cells, or it is seen to be optimal for the cells to be placed non-adjacently. Hence dead space is not accounted for currently in MOODS, when targeting FPGAs.

Estimating the CP delay is a 2-stage problem, first the delay of the macros that cells that belong to the critical path need to be estimated. Delay estimation of macros is in the cell library. But this delay does not include the delay of interconnect between the macros, which can dramatically increase the CP. Accurately estimating the interconnect properties of a design once implemented on hardware is a much more complex process than area estimation. Further understanding of the layout of the chip in HLS needs to be obtained before accurate interconnect prediction can be achieved. Hence this depiction of the interconnect properties that can affect the CP will be called interconnect prediction.

Interconnect prediction estimates the routing layout of a chip, and different factors that influence the routing need to be considered. Routing is very dependent on placement as if the placement is bad this can cause the routing to be bad (simply shown in Figure 8).

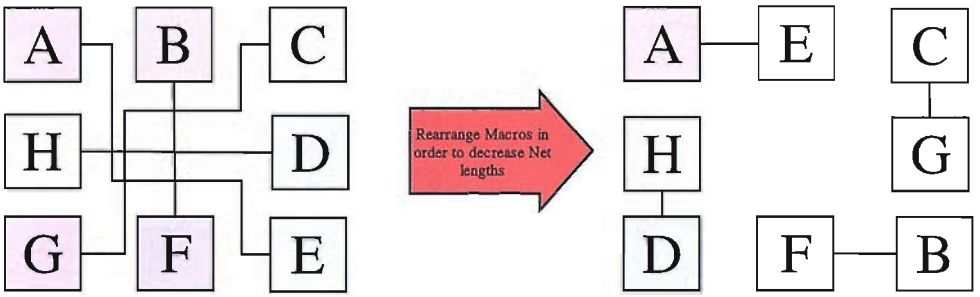


Figure 8. Diagram to Show How Routing can be improved by Simply Rearranging the Placement.

Placement can be affected by the RTL supplied to it, so there can be a knock-on effect from HLS. If the HLS tool supplies an RTL description of a design to an APR tool that is a good platform on which the APR tool can perform its optimisation of the placement and routing, the APR tool will produce a highly optimal design implemented on hardware. For example, consider an RTL representation that can be optimised in an APR tool with respect to interconnect. Again a simple example shall be used to demonstrate this point, before a more detailed example later on in this chapter. Figure 9 shows how an RTL description can aid APR optimisation. It will be assumed that all macros can only occupy the position given in Figure 9. Let A1, A2, B1, B2, and E be registers and C and D be identical adders. C and D are combined to form X (MUX) and Y (adder identical to B and C). This assumes that C and D can be combined and still keep the functionality of the design. All the functional units are of equal size. This combination reduces the unit distance from 16 to 14. The 1 unit in distance in Figure 9 is measured from one placement site to the next placement site. So the RTL, which would represent the architecture from Figure 9 (ii), would produce a more optimal design in terms of interconnect distances, which will be proven in chapters 3 and 4, produces more optimal designs in terms of CP.
Interconnect delay has a large impact on the critical path, hence total delay can be reduced by using interconnect prediction. This reduction can be produced by providing an RTL VHDL version of a design that will form a hardware implementation that has relatively small interconnect lengths, compared to an implementation formed from an RTL VHDL version of the same design that does not use interconnect prediction during HLS. So by using interconnect prediction within HLS the detrimental effect that bad (very complex) interconnect topology has on the critical path of a design implemented on a chip, is reduced, hence reducing the delay of a chip.

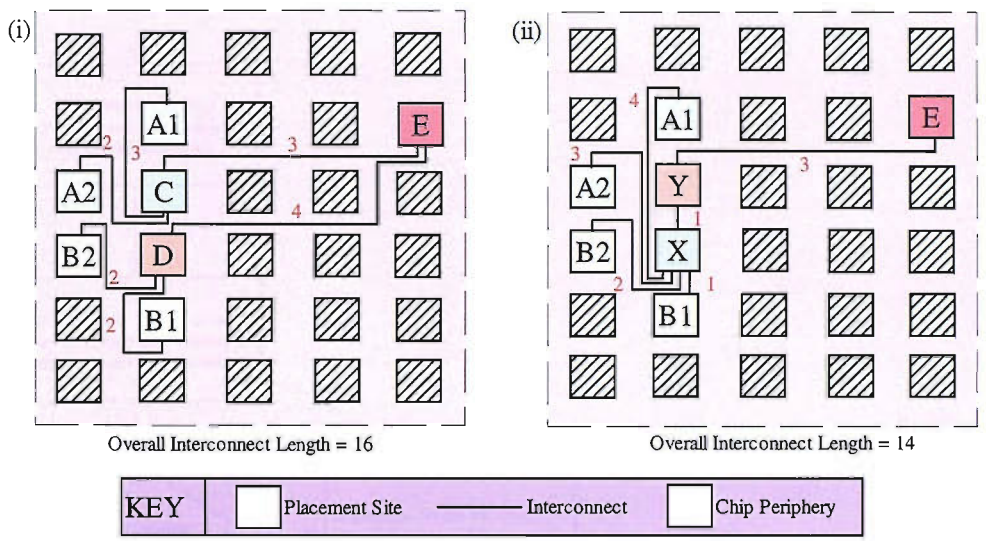


Figure 9. Diagram to Show How Different RTL Descriptions Can Affect Interconnect Lengths.

A HLS tool is designed to speed up the design process by allowing the designer to optimise a design with little knowledge of the RTL. MOODS is such a HLS tool, an overview of this tool shall be discussed as it forms the backbone to this thesis and is involved with all the results presented.

2.7 MOODS Overview

To initiate HLS, the design has to be changed into the desired format on which to perform optimisation. The input data contained in the behavioural VHDL is transformed into an intermediate code (ICODE), which is used to build a data structure. The data structure is in the form of a Data Path and Control Graph: a very brief description will follow. Every node on the control graph represents a state; each state has Instruction Graph (IGR) nodes, which contain a list of instructions that need to be carried out in one clock cycle.

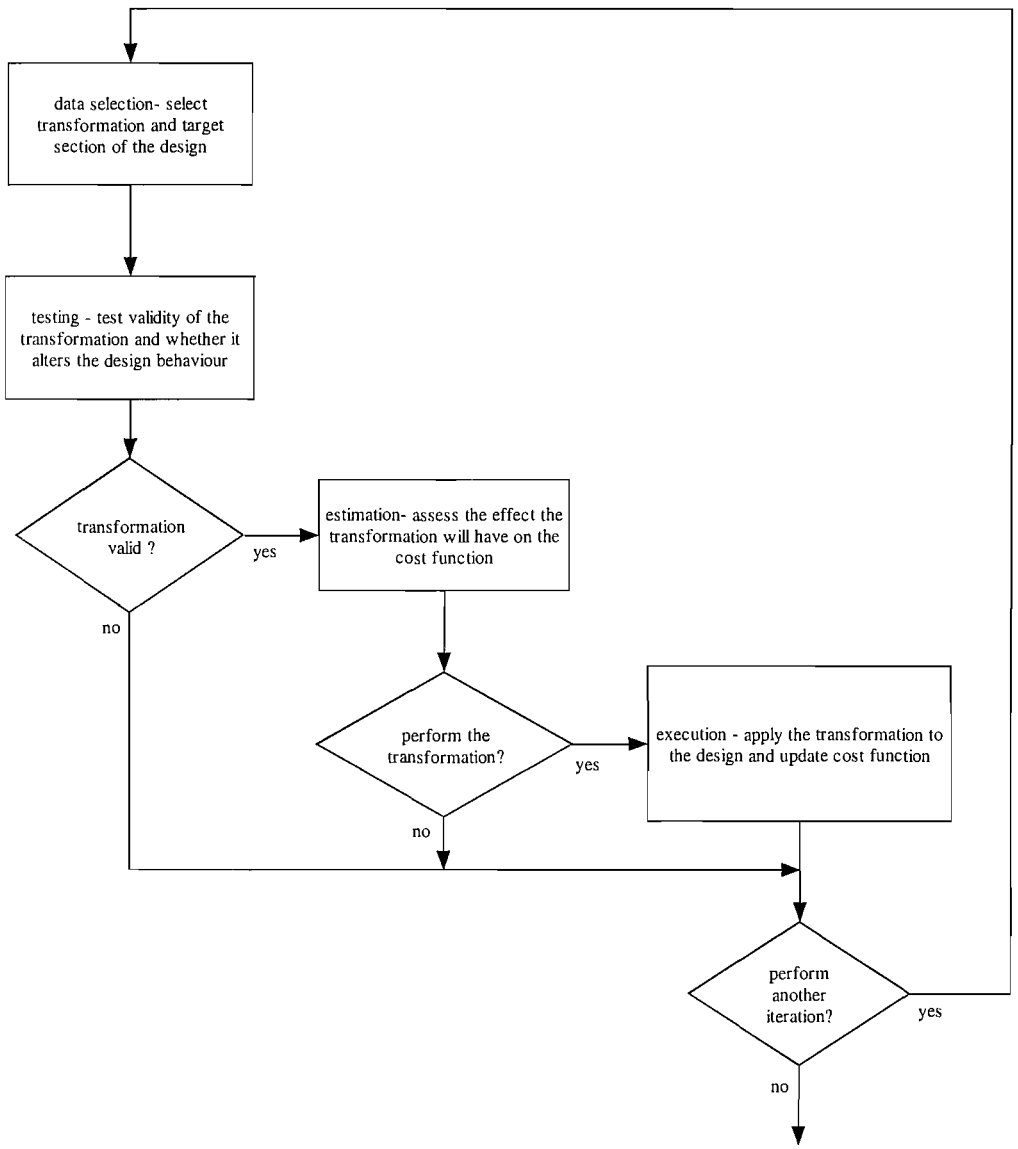


Figure 10. MOODS Optimisation Loop.

All instructions in an IGR node are dependent on each other; hence the instructions are carried out sequentially. Finally all the IGR nodes in the same control node run in parallel. A data path graph node represents a functional unit (adders, multipliers, subtractors,...), register or a multiplexer. These nodes are also known as macros. The CFG and DFG now representing the design are passed through an optimisation loop, which is shown in Figure 10, from [20]. Transforms are used to alter the structure of the design while maintaining the design's behaviour. The two-optimisation algorithms, which are used to find random/suitable transformations, are Simulated Annealing (SA) and Quasi-Exhaustive (QE). Details of these algorithms can be found in [20]. A brief

description is given below, once the cost function and transformations that form the backbone of the optimisation algorithms have been discussed.

2.7.1 Transformations

When exploring the design space, the design architecture is perturbed in order to find the most optimal solution. Transforms are used to alter the design depending on certain conditions.

In order to optimise a design within MOODS, three functions that are needed to be applied to the data structure, namely:

1. Scheduling
2. Allocation
3. Binding

Scheduling decides at what stages behavioural operations (such as addition) are carried out during the design's execution. At each stage that operations are carried out, the resulting values are stored in a register, ready to be used in a later stage. Allocation is used to map operations, storage and interconnect onto specific data path units (ALUs, registers, MUXes). After this mapping, the data path units are still abstract (no structural information), but give a basic netlist of the design. Binding is then used to map these abstract data path units to technology specific cells from a structural library, allowing an estimated view of the final physical implementation.

These transforms affect the scheduling, allocation and binding properties of the design. Whether these transforms are performed depends on the cost function (discussed in the next section), which informs how a transform will affect the overall system. First the node/nodes are selected on which the transform will act. Then these nodes/nodes are tested to see whether the transform, if performed, would be legal. If the test is successful, an estimation of how this transform will affect the design in terms of delay, area, etc is made. Estimation is needed as once a transform is selected it cannot be undone (so as to speed up design exploration). Finally if the transform is deemed to push the design architecture in the right direction it is performed. The acceptance of the transform is dependent on which optimisation algorithm is used, as design degradation can be accepted as well as improvement (depending on the algorithm).

The actual transformations that are used in MOODS will now be discussed in more detail. These transformations can affect the entire data structure or just a small part of it. The

transformation will primarily affect either scheduling, allocation or binding, but generally affects all three. The transformations will now be grouped into three categories, the first category are transformations that mainly affect scheduling. These transformations are used on the control graph, which can in turn affect what transformations can be used on the data path. This will become clear once the transformations have been discussed. The second group contains transformations that mainly affect allocation and binding. The third group mainly affects binding. These transformations are used on the data path, which in turn affect what transformations can be used on the control graph. Appendix 7.2 gives a brief description of the transformations. Greater detail of the description of the transformations can be found in [20].

2.7.1.1 Scheduling Transformations

Scheduling transformations alter the control graph by altering the assignment of instructions to control states. These transformations merge control states in order to increase parallelism or unmerge control states. Parallelism is increased in order that tasks can be carried out at the same time so that total delay is reduced. Unmerging control states is desirable as this allows functional units to be merged, as the functional units in the two disjoint group no longer carry out their tasks at the same time, allowing functional units that carry out the same operation to be merged into one data path unit, which can decrease area among other benefits.

There are four merging transformations, which are:

1. Merge Sequential IGR nodes. These nodes are contained in the control graph and contain instructions on when tasks should be carried out.
2. Merge Parallel nodes after fork, where a fork is a node that has two successor nodes in the control graph
3. Merge fork and successor
4. Group instructions on variable

There are two unmerging transformations

1. Ungroup node by separating groups
2. Ungroup node into time slices

2.7.1.2 Allocation and Binding Transformations

Allocation and binding transforms manipulate the data path by sharing and unsharing data

path nodes, while also mapping library cells. There are two sharing transforms:

1. Data Path unit sharing/ALU creation
2. Register Sharing

There are four transforms that reverse the last two transforms:

1. Unshare single instruction from unit
2. Unshare unit fully
3. Unshare variable from register
4. Unshare register fully

2.7.1.3 Binding Transformation

These transforms are used to see if any other cell in the library could carry out the task better than the current cell chosen for a particular Data Path (DP) or Control Path (CoP) unit. These transforms are:

1. Alternative DP cell selection
2. Alternative CoP cell selection

2.7.2 Cost Function within MOODS

A utility is needed to choose the most beneficial transform with the most appropriate candidates, in order to produce the most optimal design by satisfying the objective function. This utility needs to differentiate between what makes a good transformation and a bad transformation. A cost function will be used for this task; it will represent all the design criteria (area and delay in this thesis) on which the optimisation process will base its decisions. Due to the data structure's architecture, once a design's architecture has been changed through a transformation, the previous design architecture pre-transformation can only be achieved again through further transformations being applied. Hence an estimation of the effect on the cost function caused by the proposed transformation is made. If the transformation is accepted then the cost function is updated. If the design is optimised sufficiently the optimisation loop is exited. But if a further optimisation is required, another transformation is suggested and the whole process is repeated. The next transformation that is chosen is dependent on the optimisation algorithm used, so the different types of optimisation algorithms shall now be discussed.

The cost function is used to decide which transforms are accepted. It achieves this by collating all the values representing the different design criteria into one value. This value

is then used to tell us whether the transformation will be good or bad to the overall system. The cost of a design can be thought of as the energy of the system. Each design metric describes a certain amount of energy in the overall system: the lower the energy, the less impact it will have on a system. So, for example, if a design's area is reduced due to a transform, the smaller area increases the optimality (in terms of area) of the design. This reduces the value (energy) supplied to the Cost Function (CF) and hence a CF can represent a design's optimality as the lower the energy, the more optimal a design's architecture will be.

Each metric (area, power, CP, total delay, etc.) that is represented in the CF needs to be represented by a suitable value that accurately estimates a design's current standing in terms of these metrics. These metrics are calculated as follows: The total area of a design is the sum of all the functional units that the design is composed of.

Total Area = Total Area of Data Path + Total Area of the Critical Path (1) +
Total Area of MUXes

The delay of the control node is the longest delay of all the IGR nodes contained in the respective control node, as all IGR nodes are run in parallel, within the same control node. The total delay is then calculated by multiplying the number of CPs that will be needed to run the design to completion by the delay of the CP. For the last two metrics the delay of the control nodes needs to be calculated, this is show in [20] and is out of the scope of this thesis. All the parameters that are used to calculate the delay of the control nodes (e.g. inherent delays, delay factors, input capacitance and set up times) are stored in the MOODS cell library.

To calculate each individual value that represents how good or bad a transform is with respect to certain criteria, the following equation is used:

$$\Delta E = \frac{C_{estimate} - C_{current}}{C_{initial}} \qquad (2)$$

where $C_{estimate}$ is the estimated cost after the transformation has been applied;

$C_{current}$ is the cost of the current design architecture before the transformation has been applied;

$C_{initial}$ is the cost of the initial design architecture. This value is used as a normalisation factor in order to give a fair comparison with all the design

criteria;

E is the energy of the system, where the delta sign signifies a change in state.

The *Es* for each individual design criteria are then simply added together to give the overall E for the transformation. A negative overall E shows that a transform is beneficial to the overall system. By performing a transform, a certain aspect of a design (area, CP, etc.) will decrease significantly enough to warrant the transformation, while not badly affecting any other criteria. The more negative (lower) the value, the better the transformation will be for the overall system.

When a designer is synthesising a design, certain design criteria might be more important such as reducing the design's area but not the delay, hence the designer can set the priority for area. The priorities set the order in which targets are attempted to be met. So if the area metric has the highest priority then the design will first be optimised until it has met that target or has been deemed to be unreachable. Then the algorithm will try and satisfy the next criteria.

2.7.3 Optimisation Algorithms, Within MOODS

2.7.3.1 Simulated Annealing (SA)

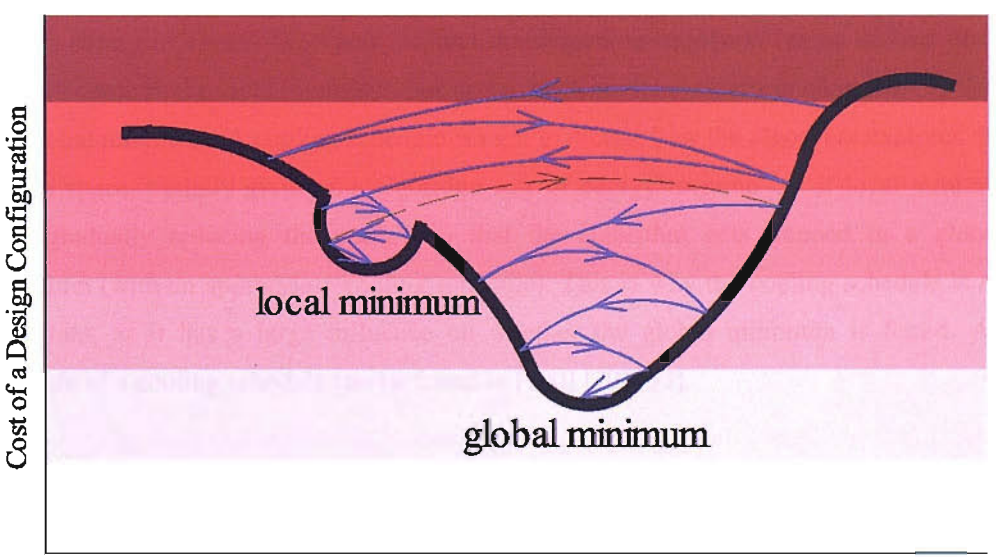


Figure 11. SA design space exploration.

SA is based on the Metropolis algorithm [4]. SA is a general optimisation technique for

minimising a function of many variables. SA can perform comparisons between multiple objectives without stating the complex interactions involved. Hence, this method is very good for tradeoffs between multiple objectives, due to its ability to find a global minimum.

The transformations are made through a random process, which is as follows. Firstly an optimisation type is chosen whether a scheduling (as well as alternative control cell) or allocation (alternative DP cell) transform. If scheduling is the optimisation type, an IGR node is randomly selected, or if allocation is the optimisation type then a DP node is selected. Then a transform is randomly selected but dependent on which class of transformations was selected at the beginning. Finally the data that is needed to carry out the transform is selected, for example if the transform which shares DP units were chosen, then another DP unit needs to be randomly selected.

When deciding whether to accept/refuse a transform, SA can accept improving ($E < 0$) or degrading ($E > 0$) transforms. The probability of acceptance of a degrading transform is a function of the annealing temperature (as the temperature decreases the probability of accepting a degrading transform decreases, allowing SA to settle in a minimum). This means that if SA has settled in a local minimum, allowing a degrading transform enables SA to jump out of the local minimum and carry on exploring the rest of the design space. This is shown in Figure 11, where without the degrading transform (green dashed line), SA will settle in the local minimum, but using the degrading transform allows SA to find the global minimum. A cooling schedule is used to dictate how the algorithm explores the design space, initially giving the algorithm lots of energy to jump out if local minima, then gradually reducing the energy so that the algorithm gets trapped in a global minimum (with an appropriate cooling schedule). This is why the cooling schedule is so important, as it has a large influence on whether the global minimum is found. An example of a cooling schedule can be found in [100] [70] [93].

The cooling schedule for SA consists of 4 variables.

1. Initial temperature
2. Terminating temperature
3. Number of iterations per temperature step
4. Level of degradation per temperature step

The initial temperature has to be high enough to enable full design exploration, without getting caught in a local minimum. The terminating temperature needs to be low enough to find the bottom of the global minimum. The number of iterations needs to be high enough to cover the entire design space to again maximise the chance of finding the global minimum. The degradation needs to be slow enough again to minimise the chance of getting caught in a local minimum. SA requires a high level of user interaction when setting up the cooling schedule. This can require extensive knowledge and experience of the design to obtain the most optimal solution. Hence many iterations of the algorithm may be undertaken to gain the knowledge needed to achieve the optimum solution, which increases the duration of the design process. As SA uses random transforms and selects random nodes, this means the process is slow as the design space is being explored exhaustively. But the design space has to be explored extensively, so as to maximise the chance of finding the global minimum, not a local minimum. The more irregular the landscape of the design space in terms of cost (i.e. if there are lots of local minima) the harder it is to find the global minimum.

To sum up, the advantages of SA as an optimisation algorithm during HLS are:

- The Global Minimum is guaranteed to be found (given enough time).
- It performs comparisons between multiple objectives without stating complex interactions.

The disadvantages of SA as an optimisation algorithm during HLS are:

- Extensive knowledge and experience of the design is needed in order to obtain the most optimal solution.
- Run time is slow as it randomly searches the design space so we need to give it enough time to explore enough of the design space to give the SA a good chance of finding the global minimum of the objective function

2.7.3.2 Quasi-exhaustive (QE)

The QE technique combats the SA disadvantages in two different ways. Firstly the designer is not required to create a cooling schedule, because the heuristic makes intelligent decisions concerning which transforms are chosen. A by-product of the intelligent decision-making is that the algorithm can find a good solution in a short amount of time compared to SA, but the solution could be worse than SA's solution, **if** a

suitable cooling schedule were chosen for the latter. The algorithm has a subset of transformations, which follow a set order, guided by an analysis of the design, rather than random transformation selection.

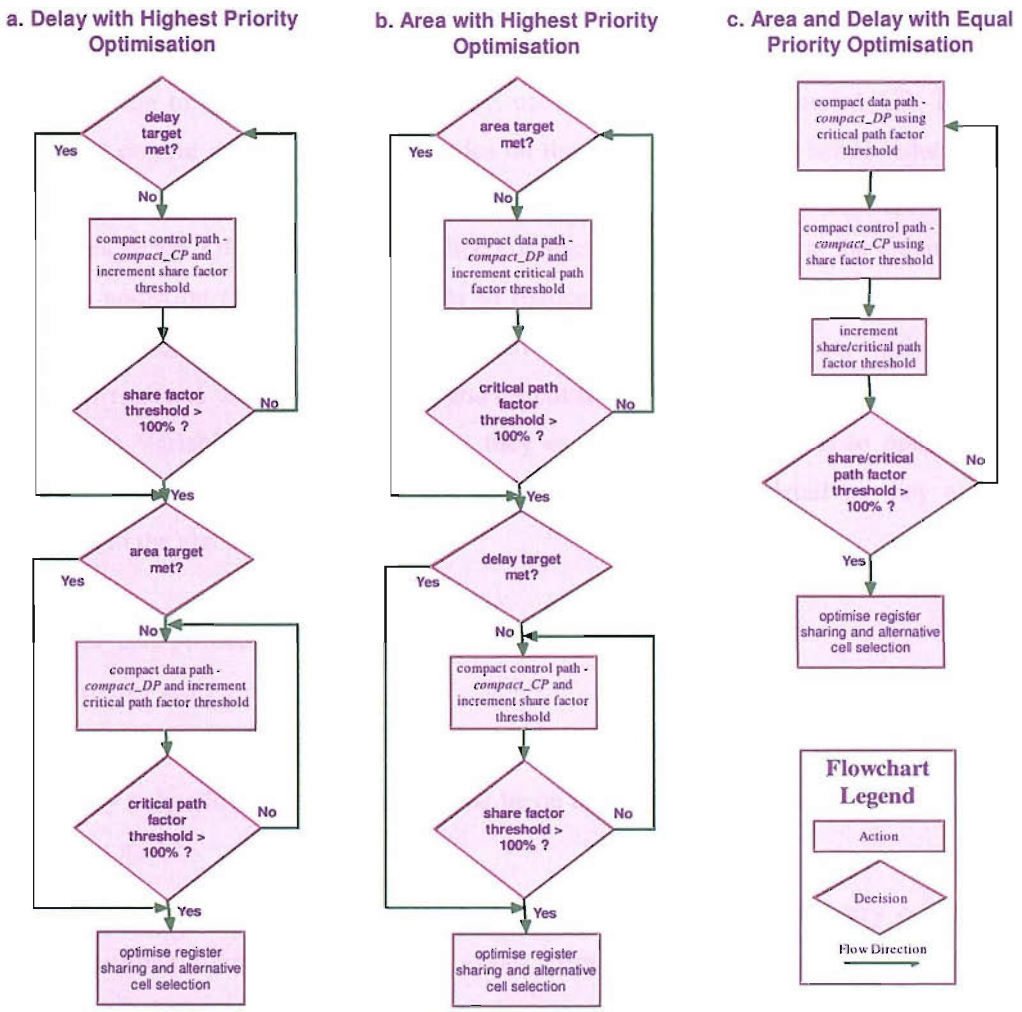


Figure 12. QE Design Flow with Different Design Criteria Priorities.

To allow for the QE to have little user interaction in order to produce optimal RTL VHDL for a design, QE has been derived only to support area, delay and the clock period targets. Using an intelligent strategy for design space exploration means that redundant searches through the design space are eliminated, hence speeding up the algorithm significantly. This limitation on the number of metrics that are entered into the cost function is needed, as the algorithm then can intelligently choose the transform that will benefit these criteria. If more criteria were added and they conflicted with the current metrics this would reduce the usefulness of the algorithm. By conflict, we mean if a path was chosen during the

optimisation that would improve area, delay and CP, but could really degrade another metric, then this would mean the objectives would be in conflict.

QE has three forms: the first is when Delay is the main priority; the second when Area is the main priority; and lastly when Area and delay have equal priority (clock period always has high priority). These different forms of the algorithm are shown in Figure 12. When attempting to meet the delay target, QE uses a process called compact_CP. This compacts the control graph, by merging nodes on the control graph, and hence reducing the length of the control graph and to some degree reducing area. When attempting to meet the area target, QE uses a process called compact_DP. This compacts the datapath, by merging nodes on the datapath in order to reduce the area. Lastly the registers are exhaustively searched, to see if any registers can be shared to optimise area, while also checking for registers with only one input and output to see if they are redundant (Group Instruction on Variable Transform), so that they can be eliminated, so as to optimise delay. Compact_CP and compact_DP are now described in more detail as they are fundamental to the algorithm.

Compact_CP is dependent on two metrics that are:

- *Critical Path Factor*

  This metric is used to identify which control nodes affect the delay, it achieves this by only allowing control nodes that lie on the critical path to be merged.

- *Share Factor*

  The metric measures the number of sharable data path units that carry out the instructions found in the control nodes being considered for merging. The likelihood of a pair of shareable datapath nodes merging will be reduced when the control nodes are merged, this is due to the datapath nodes becoming dependent. Hence the less the share factor, the less consequence there will be on area reduction of a DP.

The two metrics are designed to reduce delay with minimal affect on area minimisation. Compact_CP is recursively performed until either the delay target is met or the share factor threshold reaches 100%. The share factor threshold starts at 0% and is increased systematically (20% by default) on every use of compact_CP. Only nodes with a share factor less than the threshold are considered for transformations geared towards total delay minimisation.

Compact_DP is dependent on another two metrics that are:

- *Share Factor*

  This factor is derived for every datapath node by measuring the difference in area if the respective node was merged with every suitable datapath node. The higher the value of the shareability factor, the higher the likelihood of finding an appropriate candidate for merging. A negative value signifies that a unit is not satisfactory for merging.

- *Critical Path Factor*

  This metric is to minimise the affect on the control graph. The metric identifies how close the unit is to the critical path. The metric is derived by finding how many instructions the unit carries out that lie in a control node belonging to the critical path. A lower value will mean less effect on the critical path hence it is better for delay.

The two metrics are designed to reduce area with minimal effect on delay minimisation. Compact_DP is recursively performed until either the area target is met or the critical path factor threshold reaches 100%. The critical path factor threshold starts at 0% and is increased systematically (20% by default) on every use of compact_DP. Only nodes with a critical path factor less than the threshold are considered for transformations geared towards area minimisation.

2.8 High Level Synthesis with Interconnect Prediction

Up to this point Interconnect Prediction has been discussed in general terms. Now Interconnect Prediction fundamentals will be discussed in much greater detail. In order to estimate the physical properties of a design once implemented in hardware, consideration needs to be taken of how the final physical implementation will be derived. A good floorplan strongly affects the eventual placement implementation, hence a design architecture that can easily form a highly optimal floorplan, will have a high probability of producing a highly optimal placement. Therefore the first topic to investigate will be how to form an optimal floorplan, the floorplan will then allow global interconnect properties to be evaluated during HLS. A brief survey on Placement tools can be found in [56] with industry benchmarks also listed.

Following the question posed in section 2.3: if the higher the accuracy of the metrics during HLS the better the optimised RTL is at producing an optimal physical

implementation with regards to the design objective, why do we not just use actual design metrics (area, power, CP, etc.) post APR tool during HLS? This thinking would imply that a design should be taken through to the APR tool and physically implemented to obtain the actual metric values such as CP, hence total delay, area, etc. Then re-adjust the design accordingly to obtain the final physical implementation that satisfies the design's objectives. But an APR tool takes a relatively long time to carry out all the procedures required so that a design can be implemented on a chip as described in section 2.5.

2.9 Transforms that Benefit from Interconnect Prediction

In order for interconnect prediction to aid the minimisation of delay during HLS, we need to identify what type of transforms will benefit from interconnect prediction. Any transform that merges or duplicates functional units will benefit from interconnect prediction. When considering design architectures it is sometimes better to merge or duplicate [119] [55] functional units in order to:

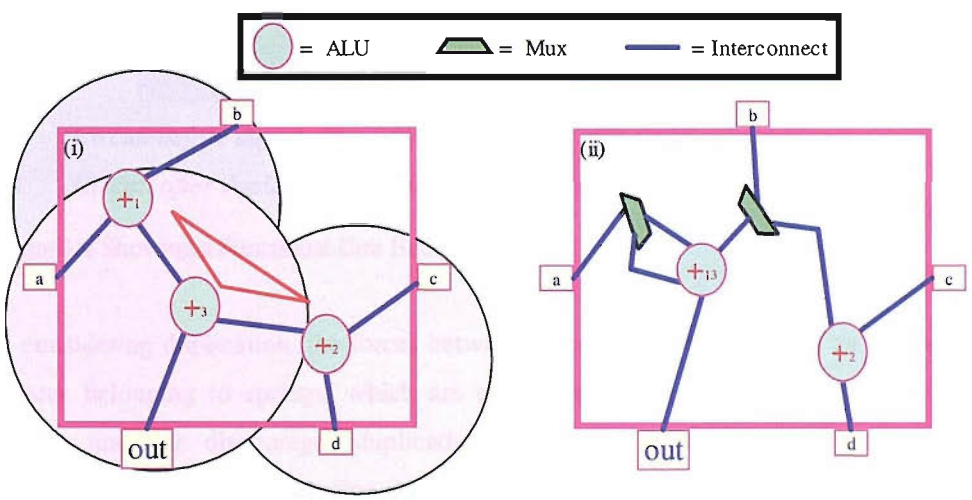
- Decrease clock period
- Reduce routing density
- Decrease area

To know when it is best to merge, duplicate or just leave the functional units alone, the physical estimates are very useful. The methodology presented in this section is founded on the work presented in [2].

2.9.1 Sharing Hardware with Respect to Interconnect Prediction

The merging transformation implemented in MOODS is shown in Appendix 7.1.2.1. To merge a functional unit depends on the following conditions. First whether the operation of the functional units can use the same hardware and in which states they exist after scheduling has been performed. Secondly the interconnect length between the two functional units can be thought of as a gravitational force, where the larger the interconnect length, the less gravitational pull the units have on each other. Conversely the larger the gravitational pull the more likely that the functional units will be merged, this is dependent on whether the functional units are suitable for merging, for example the units will need to be of the same type (i.e. both adders). Two functional units would be merged to reduce the area of a chip. If the area of functional units is large then there is more to gain by merging rather than if the functional units are small. Ideally the area of the functional units being merged needs to be larger than the area of all the components to

be added to legalise the design, this is so that the design's area is reduced. In Figure 13, this would mean that the area of adders one and two should have less area than the new adder and multiplexer (where the multiplexer was added to keep the design functionally correct) that replaces them, in order to make this transformation worthwhile. If both functional units have a high number of Input/Output net values, then this would discourage merging of the functional units, because all the interconnects from both functional units will be brought into one locality which will increase congestion. Merging of two functional units is not warranted if the routing channels in the affected area are heavily congested before merging, because all the interconnects which flowed into the original functional units will now flow just into one functional unit which will increase routing density to an even higher degree. Excessive attempts to minimise the area can cause higher congestion [23], which can lead to the actual area being increased. So another constraint to merging two functional units is a global congestion measure.



(i) The gravitational force surrounding each adder

(ii) The affect of the merged adders

Figure 13. Showing Two Adders Merging.

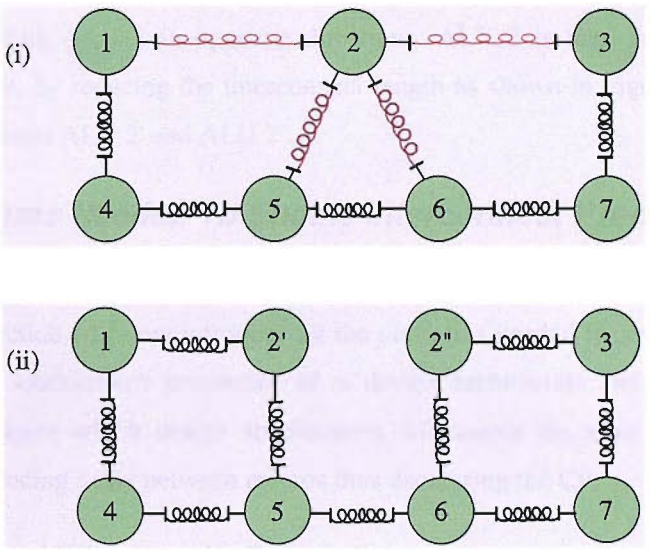
If two operations are merged, this will affect the circuit in three different ways

1. The operations will share the same functional unit;
2. MUX units will be inserted into the design;
3. A Register may have to be added to legalise the design.

This can be seen in Figure 13, which describes a simple design architecture pre- and post-

merging of a functional unit. If merging occurs there will have to be 2 clock periods and a register will be needed in Figure 13(ii), also 2 MUXes (trapezium) have to be added.

2.9.2 Duplicating Hardware with Respect to Interconnect Prediction



(i) *Circuit before duplicating a functional unit*

(ii) *Circuit after duplicating a functional unit*

Figure 14. Showing a Functional Unit Being Duplicated.

When considering duplication, the forces between macros can be thought of as behaving like forces belonging to springs, which are compression and tension, one encourages duplication and one discourages duplication, whichever is stronger will decide the respective course of action. The first force, which shall be represented as C, represents (with respect to this thesis) the Area pre- and post-duplication. The larger the area post-duplication the more that C will oppose duplicating the macro in question. In general when a macro is duplicated this will increase the total area of a design, which can reduce the optimality of the overall system, hence will be used to oppose duplication. The second force that will be considered shall be represented by T, this is the force from all of the functional unit's neighbours. To duplicate the functional unit depends on C and T. If C were less than T this would encourage duplication of the functional unit. If C is greater than T this would discourage duplication. A factor which influences T in a positive fashion is routing demand, i.e. the higher the routing demand in the region of the macros being duplicated, the larger T will be. If routing demand is high in the location of a functional unit, then to relieve congestion the functional unit could be duplicated so that

all the interconnects which lead into that functional unit do not lead into just one locality, this will be affected by the global level of congestion. Figure 14(i) shows seven components, where 2 is a functional unit, say an ALU. Another factor that influences T is the length of interconnects connected to the macro being duplicated, the longer the nets are, the higher the stress on the net, hence the greater the value of T. So Comp 1 and Comp 3 are pulling ALU 2 in opposite directions. ALU 2 is then duplicated, which relieves the stress, by reducing the interconnect length as shown in Figure 14(ii), where ALU 2 now becomes ALU 2' and ALU 2".

2.10 Algorithms Needed To Enable Interconnect Prediction During HLS

The following section will now introduce all the platforms needed for enabling MOODS to evaluate the interconnect properties of a design architecture, which then allows MOODS to compare which design architectures will create the most optimal routing layout, hence reducing delay between macros thus decreasing the CP.

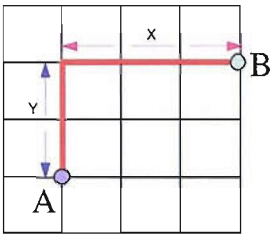
2.10.1 Individual Wire Length Calculation

To calculate the length of the individual interconnects they will need to be modelled in order to estimate the routing layouts of each interconnect. Interconnect can be modelled in two ways when considering the calculation of individual interconnect lengths. Either the interconnects can be described as multi-pin nets which would be more accurate to the actual routing, or they can be thought of as two pin nets, this reduces the computation dramatically without having too much effect on the accuracy, as described earlier. These individual interconnects can then be used to find the average interconnect of the circuit. In order to predict the interconnect properties of a design, decisions on how the nets will be modelled need to be considered. This modelling of the nets is very important as it will affect how the nets are distributed, when predicting the interconnect properties.

2.10.1.1 Two Terminal Nets

A quick but accurate method for 2 pin nets is using bounding box estimators to predict individual interconnects. The size of the bounding box for a *n*-pin net would be just large enough to encompass all *n* pins. The half perimeter rule estimates the Wire Length of a net to be half the perimeter of the bounding box, as shown in Figure 15. The half perimeter rule gives the exact cost for 2-pin and 3-pin nets and is used in many applications for a fast but relatively accurate estimator of Wire Length

[93][70][111][105].



$$E(\ell_{AB}) = X + Y \tag{3}$$

Figure 15. Representing a Net with Two Terminals A and B.

2.10.1.2 Multi-pin Nets

A popular representation for the multi-pin nets is a Rectilinear Steiner Minimal Tree (RSMT). Hence the Wire Length of a net is the same as the Wire Length of the RSMT. [36] gives an extensive survey on heuristics that minimise a RSMT. Heuristics are used as it is an NP-complete problem. Once a RSMT is found, a bounding box can again be used to calculate an interconnect length. *Caldwell et al.* [50] expand on the half perimeter rule and the estimating bounding box which encompasses the interconnect whose length is being measured. The paper shows that the Wire Length is dependent on the aspect ratio of the bounding box, which had not been considered before. Hence the proposed new equation for the expected length of an n terminal net would be:

$$E(\ell) = \sqrt{XYn} \tag{4}$$

where X and Y represent the vertical and horizontal length of the bounding box that encompasses the net.

A criticism of the technique is that it does not take into account the local congestion of the region through which the net passes. If the routing channels are highly connected then the path of a net might not stay within the boundary or it might snake back and forth, increasing the net length. Hence a metric which represented the level of congestion in the region would be useful to act as a factor to increase or decrease the average interconnect depending on whether the level of congestion is respectively large or small. This method requires a floorplan so that the bounding box can be calculated. An alternative model to the RSMT model for estimating the routing layout of a multi-pin net can be found in [116], where the computation is too complex to be feasible for HLS.

2.10.1.3 Summation on Net Models

All nets belonging to the circuit's netlist shall be modelled as two pin nets. This means that costly prediction of multi-pin nets lengths is not needed; whereas multi-pin net lengths need an estimation on the layout of the routing depending on where the terminals lie, two pin nets simply use the half perimeter Wire Length, which involves no prediction of the layout of the routing. This also means that when producing models of interconnect properties, the behaviour of two pin nets is much easier to describe and predict. This abstraction does not cause a significant impact on the accuracy of interconnect prediction as the majority of nets are two-pin nets [111].

2.10.2 Placement of Functional Units

To establish a good platform for metrics that accurately depict physical level properties during HLS, placement information will need to be obtained. As discussed earlier the placement of cells onto an FPGA or ASIC heavily influences the routing that follows, which in turn heavily influences the critical path delay, which needs to meet the Clock Period (CP). During HLS to perform a placement would be impractical, with too many constraints for an accurate estimation, making the placement problem highly intricate. But a floorplan can be used as an abstract representation of the placement problem. That is macros can be used instead as cells when placing the floorplan. This abstraction does not lose too much accuracy, as cells that belong to a macro are highly connected and in most cases are placed together on an FPGA. A floorplan can then be used within HLS to improve the depiction of a design once placed on a chip. This in turn can be used to guide the optimisation process during HLS. The closer an estimated floorplan within HLS is, the more accurate the metrics resulting from the floorplan will be. If the metrics can become more representational of a design post APR, but during HLS, then better decisions can be made during design exploration in order to meet design objectives once placed in Hardware. In order to obtain an accurate floorplan of functional units, accurate estimation of the area during HLS is needed, this can be achieved fairly easily as the area of a macro that carries out a particular task does not change that much if at all.

Producing a Floorplan would be desirable when estimating the physical properties of the design once implemented in hardware [69]. If the Floorplan highly resembles the final hardware implementation of the design, but is produced in a fraction of the time, this would be greatly beneficial. Hence the placement algorithm will need to mimic the behaviour of the APR's placement algorithm as closely as possible. At the higher level,

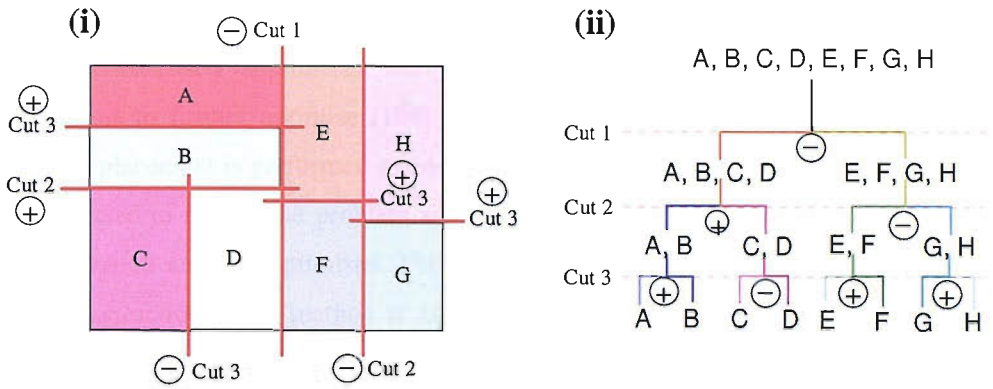
generally the layout of the chip is not known, but this means that decisions are made that do not take into account the actual size, layout of the chip and routing topology. This is important because the layout of the chip influences how the chip will be routed and the routing affects delay. Hence a floorplan during HLS will allow higher accuracy in prediction of interconnect properties. When constructing the floorplan certain considerations need to be taken into account, these are:

- *Minimising Area,* The area is minimised by fitting macros together in a certain combination and minimising the dead space between macros. This problem is more related to the ASIC placement problem, as FPGA cells do not have to be placed in adjacent placement sites of other cells that make up the same macro, allowing greater flexibility (except for macros that use carry logic).
- *Minimising Wire Length,* The shorter the interconnect between the macros, the smaller the delay between the macros.
- *Maximum Routability,* By increasing the routability of a floorplan, this can allow routes to be placed on their optimal path, hence keeping the interconnect length to a minimum.
- *Minimising Delay,* Estimates of the delay of paths is used to minimise the CP.

2.10.2.1 Floorplan Construction Methods

A good background to floorplan construction can be found in [19], but only the most recent and popular methods shall be discussed. To construct a floorplan, and hence reduce the complexity of placement, there are 4 different methods:

1. Construction based methodology builds a floorplan by grouping macros together (according to a set of criteria) until all the macros have been placed;
2. Iterative based methodology starts with an initial floorplan then improves the floorplan iteratively until the floorplan satisfies the objective function or can no longer by improved in optimality;
3. Knowledge Based methodology uses pre-existing optimal floorplans to construct new floorplans;
4. Hybrid based methodology uses a combination of the first two methods.



i) *Different Direction that Construction of Floorplan can Flow*

(ii) *Slicing Tree Representation of Floorplan in (i)*

Figure 16. Flow Between Slicing Tree and Construction of Floorplan.

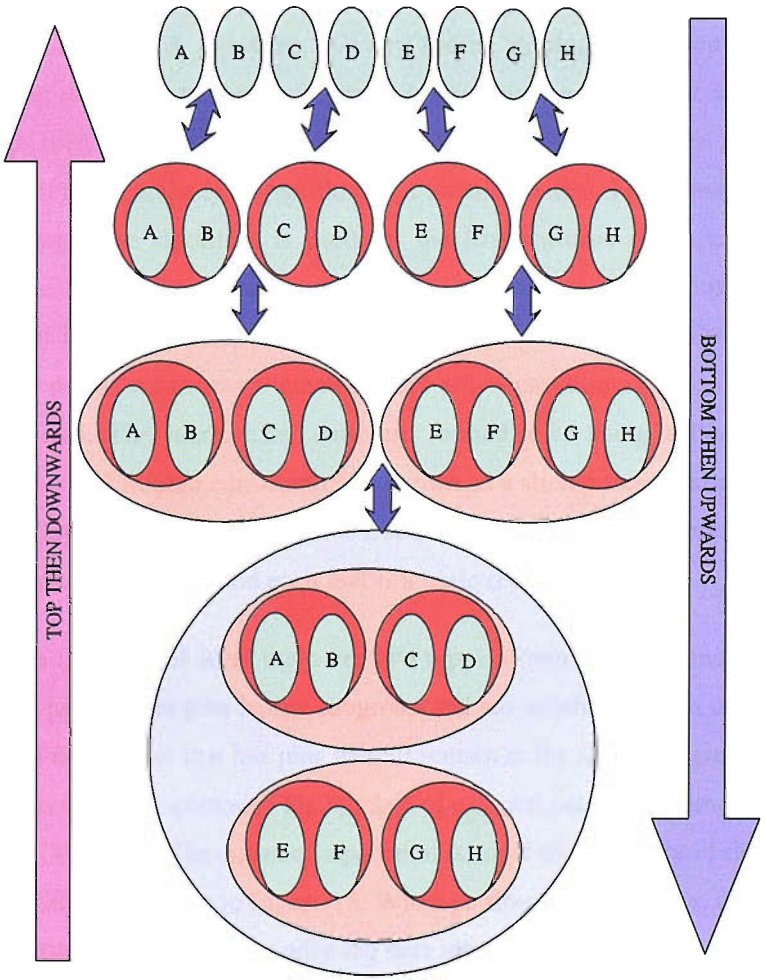


Figure 17. Diagram to Represent Top Down and Bottom Up Strategies

Construction-Based Floorplan

Construction-based approaches are used to generate an initial implementation for algorithms to further optimise [104]. Once the macros have been optimally placed, a detailed placement is performed on each individual block. Construction based approaches can be used to reduce the problem size [63]. So initially the circuit is divided into sub groups which are then optimised. This method can drastically reduce the solution space. One construction-based method is circuit partitioning [53]. This is where macros are grouped together into tightly connected sub groups. When using a circuit partitioning method, there are three approaches; these are hierarchical, flat, or Greedy. The hierarchical method can either be bottom-up, or top-down. A bottom-up strategy means starting at the bottom of the hierarchy and the working up the tree as in Figure 17. The process involves starting with all the macros then gradually merging (clustering [122]) them at each hierarchical level until all the macros are in one group at the top of the hierarchical tree. A top-down strategy means starting at the top of the hierarchy and working down the tree as in Figure 17. Each time a cut is performed to form two sub groups then this forms a hierarchical level again shown in Figure 17. A floorplan can be obtained by recursively bi-partitioning a circuit's netlist. For example if bi-partitioning is being used, the algorithm is used for the first partition then the algorithm is used to partition the resulting sub groups from the initial partition. This is repeated until the required level of abstraction is reached (the circuit cannot be partitioned anymore), and will be discussed in much greater detail when partitioning algorithms are discussed in section 3.5. The hierarchical structure formed after using a bottom up or top down approach to floorplan construction is known as a slicing tree. A slicing tree (Figure 16(i)) is a binary tree with n leaves and n-1 nodes, where each node represents a vertical cut line or a horizontal cut line and each leaf is a basic rectangular macro.

At each hierarchical level there are two types of nets, internal and external. An external net is a net that has pins in one subgroup and has another pin in a different sub group. An internal net is a net that has pins that all remain in the same sub group. When partitioning the objective is to minimise the number of external nets, this is known as minimising the cut-set (Min Cut). The cutset of a partition is equal to the weight of the set of edges cut by the partition. as shown in Figure 18. When partitioning the netlist, this approach is known as a net-based approach, as only the nets are being considered when minimising the cut set, and not the net lengths. If net lengths were taken into account, the distance between macros would have to be known. The distance between macros cannot be obtained solely

through circuit partitioning, the distances would only be calculated once the floorplan has been constructed. When partitioning a circuit, optimising the cut-set is a linear function and this type of approach is called the net-based approach.

A Linear objective function can be formulated as follows:

$$\phi_q(x,y)=\sum_{i,j} a_{ij}(x_i-x_j)^2+\sum_{i,j} a_{ij}(y_i-y_j)^2 \qquad (5)$$

Where $(x_i,\ y_i)$ is the location of one terminal and $(x_j\ ,\ y_j)$ is the location of the other terminal. $(x_i - x_j)^2 + (y_i - y_j)^2$ represents the half perimeter distance. a_{ij} is the weight of the net.

Partitioning is a very quick method for obtaining a globally optimal solution, and it forms the basis for most recent placement tools [113][81].

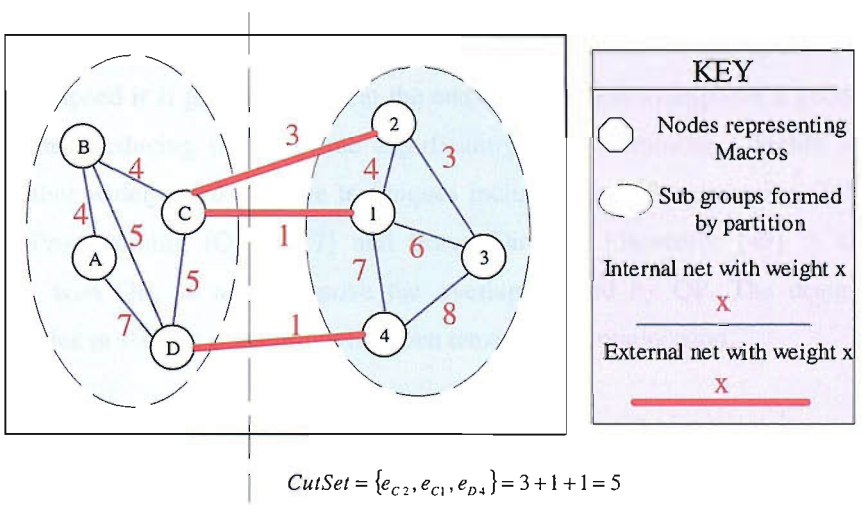


Figure 18. Diagram to Show How External and Internal Nets are Formed and How the External Nets Form the Cut Set.

The greedy approach (also known as a neighbourhood search) strategy constructs the floorplan starting with one node and gradually builds the floorplan by adding one node at a time. The floorplan is initialised with a seed (macro chosen randomly or has a property that makes it a good starting point). Then in a greedy fashion (look for the best suited macro to join the macros placed already, for example the most connected macro to the post placed macros) one macro is added at a time until every macro has been added. The

flat-based approach is where a floorplan is constructed by grouping macros together into clusters all at the same time, but this increases the number of constraints needed, which increase the complexity of the algorithm.

Iterative Based Floorplan Construction

Iterative floorplan construction starts with an initial floorplan, undergoes a series of perturbations until a feasible floorplan is obtained or no more improvements can be achieved. Generally a construction method is used to speed up the optimisation but can cause the optimisation to be trapped in a local minimum. The most common iterative method is Simulated Annealing (SA) [93][88][89][99][62]. The SA algorithm behaves as described in section 2.7.3.1.

The same properties are present when using SA for placement as in during HLS, as it randomly chooses transformations and randomly chooses the macros that the transformations will act on, in an effort to improve optimality of a floorplan. Hence the same advantages and disadvantages apply both for SA in placement and in HLS. Due to SA's lack of speed it is generally used at the end of placement to improve a good initial solution, hence reducing the run time significantly while obtaining a highly optimal solution. Other widely used iterative techniques include Linear Programming (LP) [79], Quadratic Programming (QP) [117] and Force Directed Placement [49] is used in conjunction with QP, so as to remove the overlap formed by QP. The design then naturally settles in state of equilibrium between tension and compression.

The objective function solving the QP problem can be formulated as follows:

$$\phi_q(x,y) = \sum_{i,j} a_{ij}(x_i - x_j)^2 + \sum_{i,j} a_{ij}(y_i - y_j)^2 \qquad (6)$$

Where (x_i, y_i) is the location of one terminal and (x_j, y_j) is the location of the other terminal. $(x_i - x_j)^2 + (y_i - y_j)^2$ represents the half perimeter distance. *aij* is the respective weight of the net. The difference between equation 5 and equation 6 is that in equation 6 the bounding box has a much larger contributing factor due to the x, y coordinates being squared rather than just taking the modulus. The QP problem is equivalent to solving the squared Wire Length [49] and is called a "path based approach. It is a path based approach as now it is considering the length of a path that a net may take. When using QP it produces larger amount of cell overlap, as the objective function does not consider

overlaps only net lengths and the respective weights that apply to the nets. This can be stopped by adding constraints during QP or by performing legalisation after QP [117] [131]. Legalisation means removing the overlap, as multiple macros cannot be placed on the same placement site.

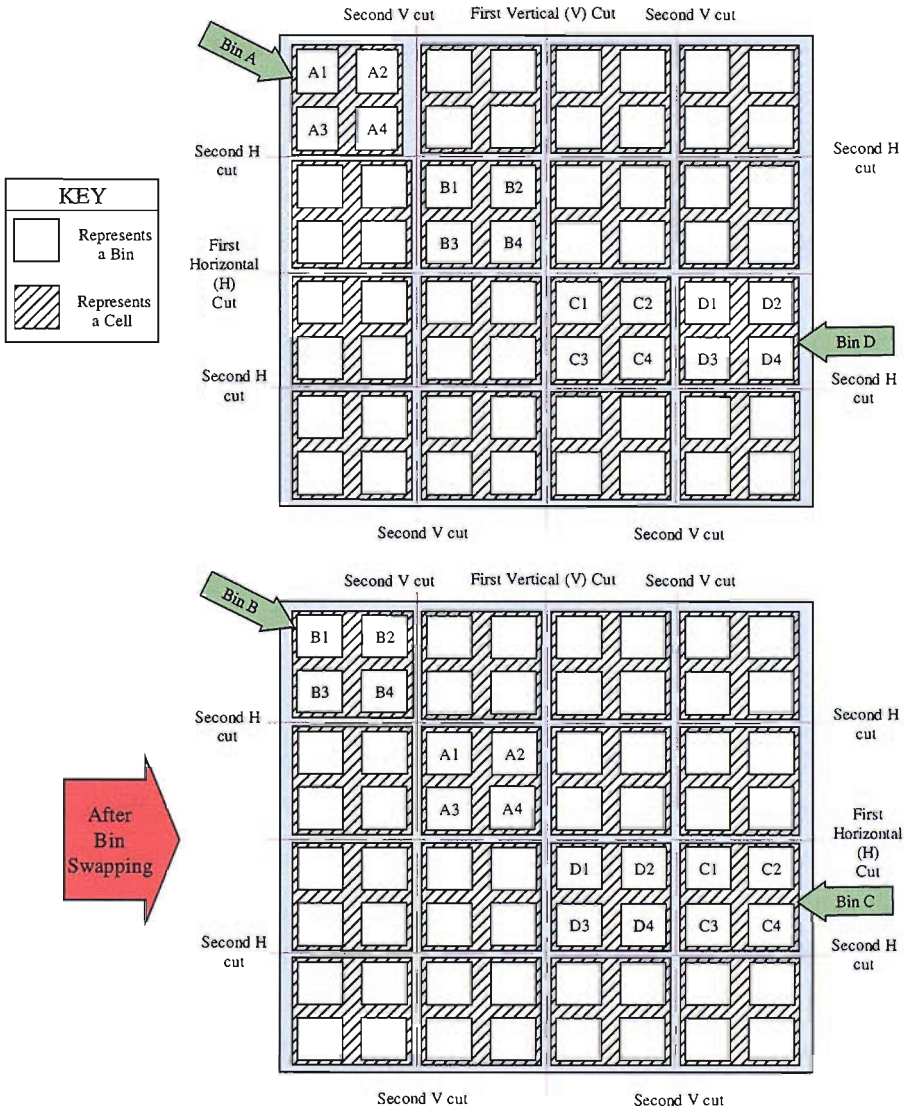


Figure 19. How Bins may be Swapped at a Given Hierarchical Level.

A common method for legalisation (i.e. removal of the overlapping of macros on the floorplan) is Force-Directed Placement, where the floorplan is first optimised using QP then perturbed over many iterations using forces that push the macros apart removing the overlapping of the macros. The forces involved are calculated by measuring the overlaps, the greater the overlap the larger the force pushing the macros apart. The interaction

between the QP objective function and the forces that remove the overlap is a complex interaction. The process is repeated until the floorplan naturally settles in state of equilibrium between tension (provided by the forces pushing macros apart) and compression (provided by QP).

A path-based approach tends to make global nets longer while decreasing the size of local nets, while LP tends to reduce the size of global nets while increasing local nets [114]. This is why a LP solution is better for critical paths. QP does not allow simultaneous optimisation of design criteria such as critical path. *A. B. Kahng* says that linear Wire Length (WL) is better objective for minimisation of WL [57], but QP provides better timing minimisation than min-cut (LP objective) placers.

Knowledge Base Floorplan Construction

Knowledge based approaches to floorplan construction is where previous floorplan/placement constructions are used to form a library from which future floorplans are constructed. This method is only feasible if there is a slight change in the design's architecture.

Hybrid Methods

M. Wang, X. Yang, M. Sarrafzadeh state that recursive partitioning (net-cut objectives) tools are more affective than WL minimisation tools, such as QP at reducing the delay of a circuit [113]. But they say that at different hierarchical levels, the WL objective is better than net-cut at reducing delay. Hence *M. Wang, X. Yang, M. Sarrafzadeh* partitions the circuit using the net-cut objective, then at each level of the hierarchy they allow bins to be swapped around at the end of each stage. Bins can be thought of as regions on a floorplan that group macros together. This methodologies presented in [113] will be discussed in further detail in section 2.10.5. Using bins allows further abstraction by reducing the solution space; e.g. a solution space of 4 square shaped macros has 4! (24) possible arrangements, 16 square shaped macros have 16! (20922789888000) possible arrangements. But if the 16 macros get grouped into 4 sets of 4 macros then there are 4! possible arrangements for each group, hence 4*24 = 96 possible arrangements, these groups also have 4! possible arrangements, hence the total number of possible combinations is 96+24=120, which is vastly smaller than 16!, hence the search space has been reduced significantly.

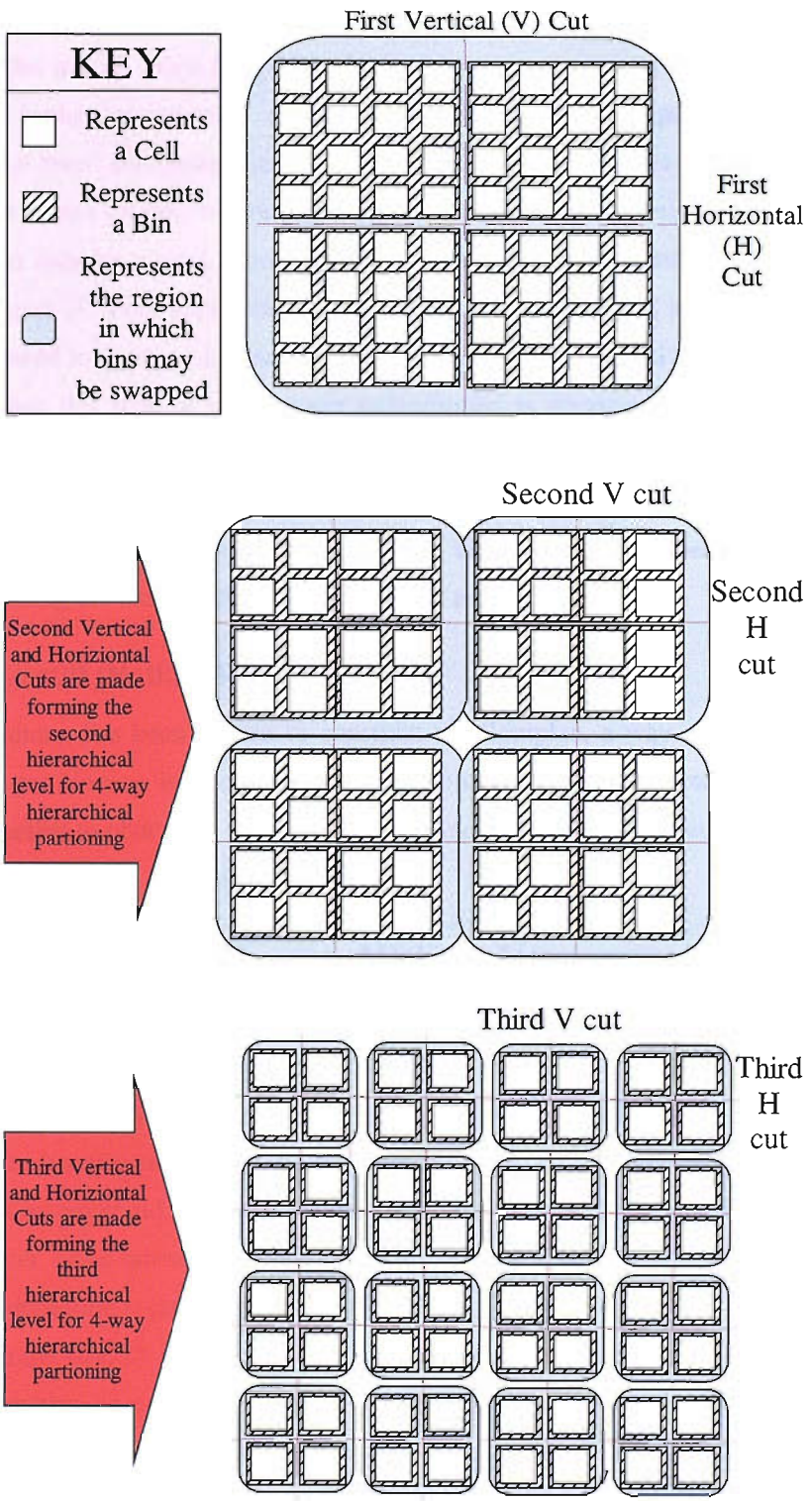


Figure 20. How Bins are Formed at a Given Hierarchical Level.

In the case of the algorithm found in [113], the different regions are formed by

partitioning, where each sub group is assigned to a particular bin. Then in order to optimise the overall Wire Length the bins are swapped around (shown in Figure 19). Once no further optimisation can be achieved the algorithm proceeds to the next hierarchical level. But global bin swapping is not allowed, i.e. 4-way partitioning is used to produce 4 sub groups, to every group, at each partitioning level. Hence those 4 sub groups can only be moved within the boundaries of the original sub group, this can be seen in figure 20. Four approaches to bin placement are proposed, and these approaches are introduced to reduce interconnect distance. The approaches will be covered in section 2.10.5, when this type of interconnect minimisation is discussed. In [77] the cells are partitioned into bins using min cut based partitioning algorithm. This floorplan is then optimised using SA, then the cells inside the bins are detailed placed. As the bins when being swapped are not allowed to traverse along block boundaries when being optimised, hence reducing the likelihood of finding a global minimum.

2.10.3 Circuit (Netlist) Partitioning

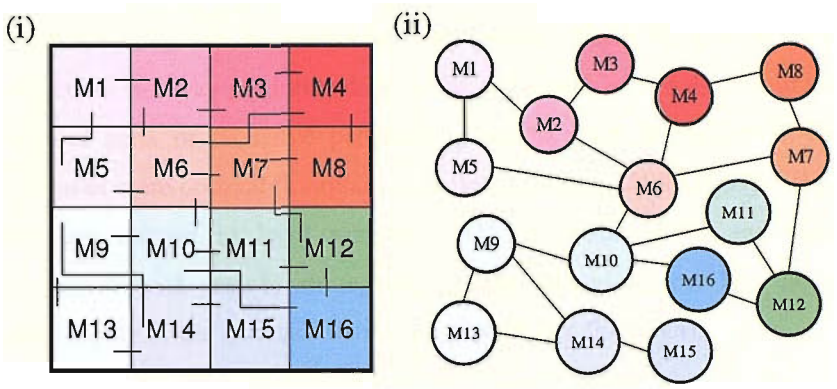
As partitioning has been chosen to construct the floorplan, a way of finding the most optimum partitioning is needed to form an optimal floorplan. Because partitioning a design's netlist to obtain the exact optimal solution is NP-complete, heuristics are used to obtain approximately optimal solutions.

2.10.3.1 Partitioning a Circuit to Minimise Interconnect Length

At a high level the only information available for partitioning a design is the netlist of a design's architecture and the area of the individual components using specific technology libraries. The circuit is represented as a hyper graph (G) as shown in Figure 22(ii), where each vertex (macro) can represent transistors, gates, ALUs or even entire circuits, but in our case a macro could represent a storage unit, e.g. register, a functional unit, e.g. adder or multiplier, or an interconnect unit e.g. multiplexer, that belongs to a design's structure. A formal definition is given below:

Given a graph $G(V, E)$, where each vertex $v \in V$ has size $s(v)$ and edge $e \in E$ has a weight $w(e)$,the problem is to divide the set V into k subsets V_1 , V_2 ,..., V_k , such that an objective function is optimised subject to certain constraints. When a partition divides the set into k subsets this is known as k-way partitioning. If k = 2 this is referred to as Bi-partitioning. G will be partitioned into two sub partitions A and B, where each vertex $a \in A$, each vertex $b \in B$, $A \cap B = \varnothing$ and $A \cup B = G$. Each edge represents the interconnect between the macros. The weight of each edge will be the number of I/O pins

the interconnect required to join the net to a macro.



M = *Macro*

—— = *Interconnect Between Macros*

(i)Macros Placed Together

(ii) Hypergraph (G) Representation of the Design's Netlist

Figure 21. Design's Netlist Partitioned into Macros and Nets Joining the Separate Macros.

Figure 21(i) shows a circuit, which has been partitioned into 16 separate macros (sub-circuits) with nets joining them to each other. Now to minimise the long nets the individual macros get moved around. The nets that go from one macro to another macro are called external nets. Nets that stay within a macro boundary are called internal nets. To minimise the overall interconnect length, when partitioning the design's netlist, reduction of the length of external nets is strived for, (external nets are mostly longer than the internal nets of a circuit). So if M1 in Figure 21(i) were swapped with M16 their nets would need to travel across a large proportion of the chip to connect with their neighbours, which would be undesirable.

To minimise these long nets a Min Cut algorithm [19][73] can be used, which is very fast. The Min Cut algorithm partitions the circuit into a desired number of parts, while minimising the cutset. The cutset of a partition is equal to the weight of the set of edges cut by the partition. The components within the groups created by this partition should be placed close to other components in the same group. This reduces the routing resources occupied, since the lengths of the nets that have a higher weight are reduced.

2.10.3.2 Relative Placement Information Obtained from Partitioning

To obtain a relative placement of all the components, the circuit needs to be partitioned until no group can be partitioned any further. There are two ways of achieving this objective. A k-way partition (flat) can be performed, where the circuit is partitioned into k disjoint parts or recursive partitioning can be used. Recursive Bi-Partitioning (RBP) produces more optimal floorplans than flat placement [99][43][59][101][106][120], which can get trapped in local minima [107]. [77] uses RBP, and then uses a greedy (neighbourhood search) strategy to legalise the placement (by the term legalise, no overlapping of the macros exists when placed on a floorplan).

From the floorplanning discussion RBP has evolved as the most suitable choice for partitioning a circuit's netlist within MOODS. To obtain the relative placement through recursive partitioning the circuit is recursively bi-partitioned until no sub-group can be partitioned any more, hence a top-down partitioning approach. The sub-groups formed by the partition should be evenly sized, so that we can compare different groups fairly. Once the circuit has been recursively bi-partitioned, a hierarchical map can be obtained, i.e. where the macros should be placed in relation to each other.

2.10.3.3 Partitioning Heuristics

There are 5 types of general methods of partitioning a netlist, 4 of which can be found in [41], where an extensive survey on partitioning heuristics can be found. The 5^{th} general method of partitioning can be found in [51]. Only iterative improvement and clustering heuristics shall be considered as they are the most commonly used due to their high optimisation potential while being the quickest heuristics. Iterative Improvement [1][3] and Clustering algorithm [59] have been chosen to partition a circuits netlist within the HLS tool MOODS, due to their speed, optimality and hence popularity [45].

A clustering algorithm groups elements together, by forming natural clusters. These clusters then form the sub-groups of the partitioned group. These algorithms are very fast and simple, while maintaining reasonable optimality. The iterative improvement algorithms start with two different sub-groups. The two sub-groups have either been picked randomly or produced by an alternative algorithm e.g. a clustering algorithm. The iterative improvement algorithm attempts to reduce the cut-set by moving elements from one sub-group to the other, either by swapping nodes in each group, or by just swapping

one node at a time while maintaining an even spread between the two sub-sets.

Madden et al derives an iterative deleting partitioning algorithm [51], in which a vertex is assigned to multiple groups rather than to just one group as normal, then the worst case redundant (having multiple locations) vertices are removed one by one to obtain the final solution, when no more redundant vertices are left. This approach looks promising due to speed, but when bi-partitioning the algorithm behaves like a greedy clustering algorithm. As recursive bi-partitioning will be used, this technique is redundant.

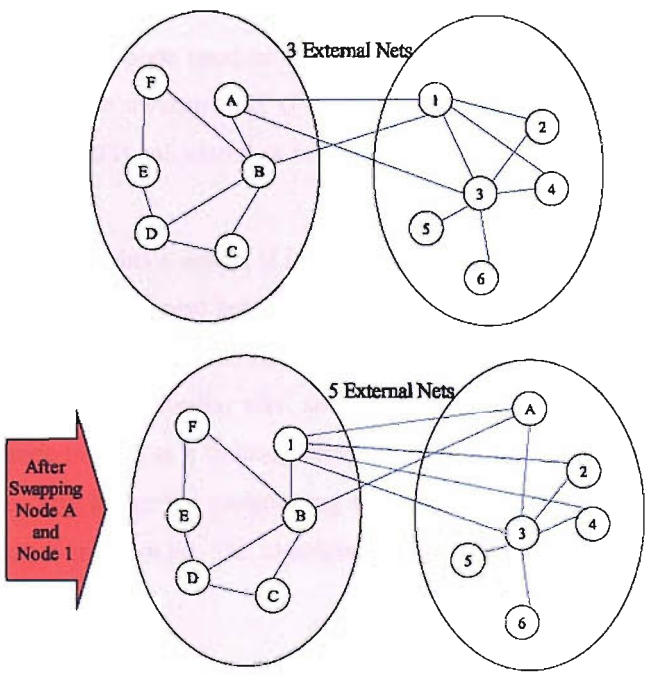


Figure 22. Showing how swapping two nodes can be detrimental to the overall system if internal nets of the candidates for swapping are not considered prior to swapping.

Kernighan and Lin (KL)[1] proposed an iterative improvement algorithm that swaps two nodes between partitions, but which is quite slow, with time complexity $O(n^2 \log n)$. The swapping of two nodes from either side of the partition is performed in order to reduce the number of external nets (i.e. the nets in the cut set). Hence before the swap is performed, the algorithm needs to make sure that by swapping the nodes, there is not an increase in external nets caused by the swap. This can happen if the nodes being swapped were highly connected to nodes within the partition they originally resided in, as shown in

Figure 22. Then if each net has unit weight, the number of external nets is 3, and then if node A and node 1 are swapped, the external nets increase to 5, which is detrimental to minimising the cut set. Thus a value called the D value is used that represents the gain in a system i.e. how much a swap will benefit (reduce) the cut set. The D value is calculated as follows:

$$D_x = E_x - I_x \quad (7)$$

Where E is the external nets that belong to node x and I is the internal nets that belong to x. The more positive D_x, the more that x will reduce the cut set if placed in the other region (if negative, x would increase the cut set if placed in the other region). As the nodes are swapped, D values of each node need to be considered, and this will be the gain G to the system if both nodes are swapped. If G produces a positive value then the swap is beneficial to the cut set. G is calculated as follows:

$$G_{xy} = D_x + D_y - 2c_{xy} \quad (8)$$

Where c_{xy} is the net that joins x and y; if it exists, the net value needs to be subtracted as the net will be in both sets of nets belonging to x and y, hence c_{xy} will be counted twice, and the net will be still external after the swap so does not affect the equation. The nodes being swapped need to be of similar size, so that the groups do not become too uneven. Partitioning tolerance is used as a limiting factor on how uneven sub groups formed from a partition can be. The larger the partitioning tolerance, the larger the imbalance of the sub groups respective area can be. The algorithm stops when no more swaps can be made that will decrease the cut set.

The algorithm cannot handle nodes with non-uniform area. The problem of KL failing to deal with non-uniform area will be shown in section 3.5.4. An extension to KL to rectify this failing can be found in [19], which considers the area when partitioning a circuit. The method duplicates a node until the number of duplicated nodes matched the area of the original node (assuming the area value is an integer). The duplicated nodes then have a very high weighting between the duplicated node so that they are kept together during partitioning (i.e. they are always placed in the same group, unless the sub group is less than the number of duplicated nodes that make up the original node), simulating the original node. But the computation increases dramatically for HLS as the number of nodes, n, will increase dependent on the area of the macros. So whereas the complexity for the KL algorithm was O(n2logn), the complexity now would be O(a2n2logan), where a is the average area. The average area can be very large in HLS (much larger than n for

example), hence making the methodology infeasible for HLS.

The Fiducci Mattaus (FM) algorithm [3] is a very well known and popular iterative improvement algorithm. FM is similar to KL, but only moves one node at a time from one sub-partition to the other. FM is faster than KL with a time complexity of O(n) (where n is the number of nodes) when using a bucket sorting algorithm [91] on data that belongs to a uniform distribution. When the edge costs in the graph do not all have a unit cost (i.e. do not belong to a uniform distribution) the bucket data structure can no longer be used and the time complexity increases to O($n\log n+e$) where e is the number of edges. The nets of the graph in MOODS will not have uniform weighted nets, as all nets are treated as two terminal nets with a weight equal to the number of bits or bit-width between nodes. Hence O($n\log n+e$) is used as the time complexity. An algorithm that swaps two nodes produces a better cutset improvement compared to an algorithm that only moves one node at a time between partitions, as stated in [22]. The reason for this is the observation that small groups that are being partitioned contain macros that are larger than the partitioning tolerance in FM. First it partitions a netlist with a large tolerance with regard to area equality. This high tolerance to large area mismatch allows a lot of freedom for macros to be placed with their highly connected neighbours [64] [102]. The tolerance is then reduced until it is at an acceptable level. But again multiple runs are too time-consuming, so in [78] a look-ahead strategy of possible reverse moves is used to legalise the design. An algorithm called Quick Cut [22] uses the same method as KL, but reduces the time complexity of KL by decreasing the amount of searching through the neighbourhood when selecting the two nodes to swap. The time complexity of Quick Cut is in the worst case O(max(ed, $e\log n$)) (where d is the maximum node degree (number of nets/edges a node has) of G) and average case complexity O($e\log n$), which is less than FM when the following constraint is satisfied:

$$
\begin{aligned}
& e\log n < n\log n + e \\
& e\log n - e < n\log n \\
& e(\log n - 1) < n\log n \\
& e < \frac{n\log n}{\log n - 1} \\
& e < n + \frac{n}{\log n - 1}
\end{aligned}
\qquad (9)
$$

This constraint means that the modified KL algorithm time complexity is much closer to the FM time complexity, and is faster when the average degree of the nodes in the circuit's netlist is small. The *Dutt* and *Deng* approach to netlist partitioning uses a look-ahead strategy [45], where consideration of not just the immediate effect of swapping or moving a node is considered, but the after effect as well. This method shows a good improvement, but the algorithm is too slow to make it feasible. But if finding the most optimum solution becomes the priority then this method is worth considering.

The final solution to obtaining a global minimum is to use multiple runs with different initial partitions, so as not get trapped in a local minimum [110]. This means the design space is explored in more depth; hence there is more chance in finding the global minimum. The algorithm is performed once with an initial partition, and then the algorithm is run again but this time with a different initial partition. The resulting cut-sets are compared to see which resulting partition is the most optimal. Carrying out multiple runs will obviously increase the run time of the algorithm. As long as the algorithm remains stable and comparisons can be made with confidence, the most optimum solution is not of the utmost importance. The reason for this is that the partition will only influence transform acceptance, and not actually contribute to the conversion from the data structure into the RTL VHDL output post HLS. So our main concern is that the RTL VHDL output is the most optimal RTL VHDL, compared to any other possible combination of RTL VHDL that could have been written to produce the same functionality. Hence the initial partition has to be selected in a way that stabilises the partitioning algorithm. But the iterative improvement algorithms have results that form a normal distribution, i.e. they form a bell shape graph (Figure 23) where the majority of the results are found on the centre of the graph. This means there is a high probability that a resulting cut set will be a value close to the centre of the graph making comparisons between cutsets belonging to different architectures fairer. This is very important when comparing multiple designs, as comparison of one design architecture cut set, which has the most optimal value is compared with another design architecture cut set which has the worst possible cut set, can lead to incorrect conclusions on which design architecture offers the most optimality.

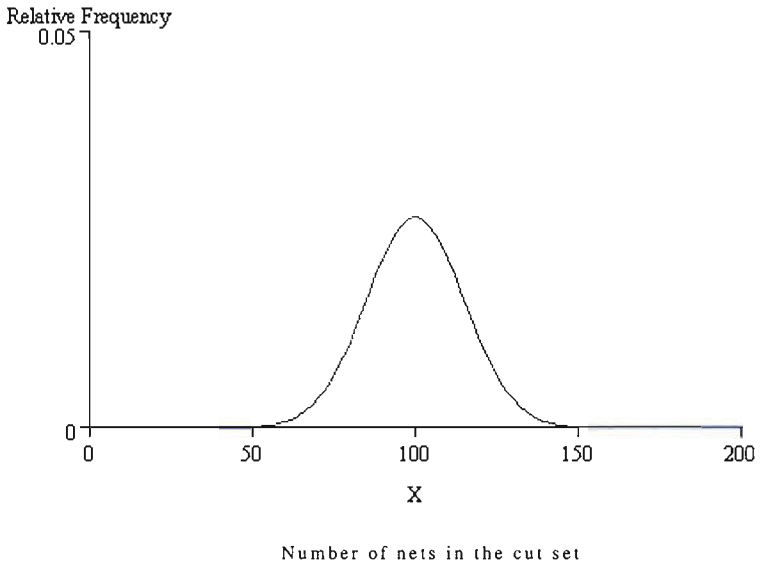


Figure 23. Plot of the normal probability density function (total area under the graph represents 1, that is 100% probability), with μ=100, s = 15.

These incorrect conclusions discussed in the last paragraph could occur when performing HLS. A design architecture pre-transformation might produce a highly optimal cut-set better than the average cut set if many multiple runs were made. But post-transformation may then produce a low optimised cut set, worse than the average cut-set if many multiple runs were made. When both pre- and post-transformation designs are compared, the first design structure could appear to lead to a more optimal partition than the second design structure, when in reality the second design will lead to a more optimal circuit partition than the first partitioned design. This confusion on which design has the highest optimality needs to be avoided, so that we can compare design architectures fairly.

2.10.4 Net Consideration During Recursive Bi Partitioning (RBP)

When partitioning, the aim is to reduce the overall Wire Length in order to have a smaller impact on the critical path. Timing-Aware Weighting can be used to reduce overall net length or just concentrate on the critical path [9]. When considering just the critical path the nets are weighted so that the macros on the critical path are placed closer together to reduce the critical path. *S. Ou* and *M. Pedram* add weights to the nets but also reduce the number of times nets are cut [115]. *B, Halpin, R. Chen* and *N. Sehgal* disagree with this method as it is hard to quantify how the weights are increased/decreased according to their critical priority. *B, Halpin, R. Chen* and *N. Sehgal* use linear programming [108] to improve on the partitioning solution, they obtain impressive improvements in delay, but it

is too computationally expensive. *A. B. Kahng et al.* use this method in [79], but then develop a much faster method in [57], as they simply increase the weight of a net once it has been cut to reduce the likelihood of the net being cut again.

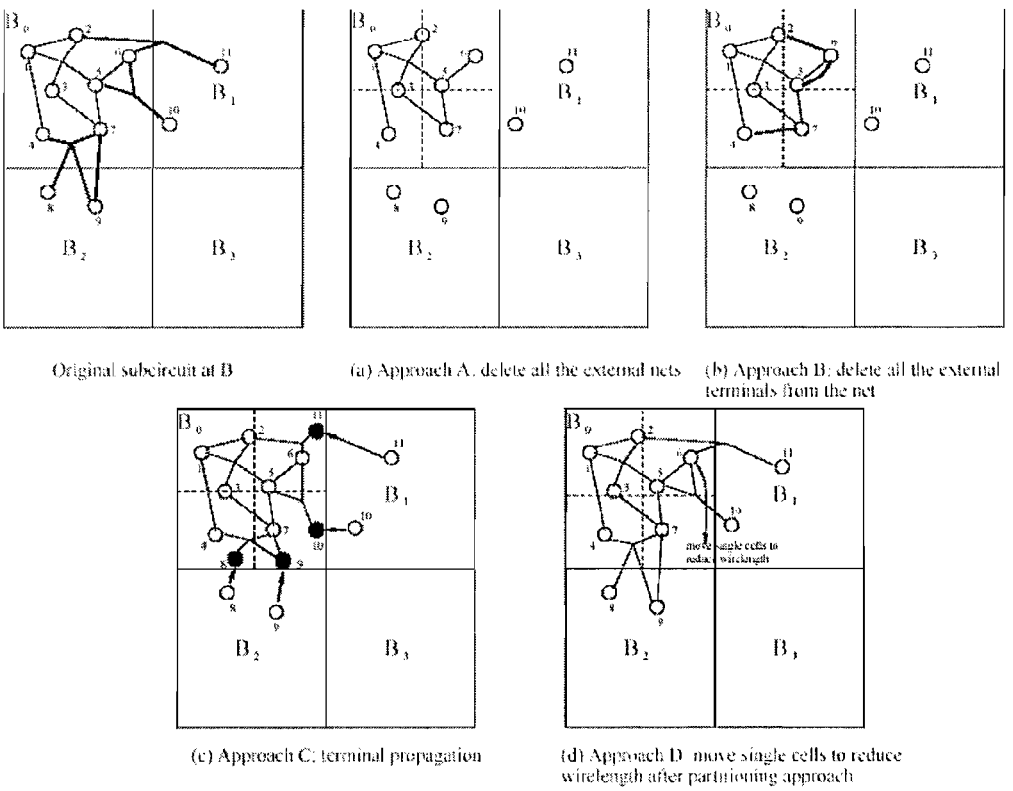


Figure 24. Showing different Approaches to Net Representation During Circuit Partitioning.

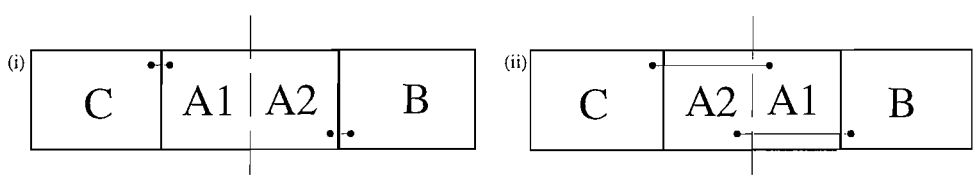


Figure 25. Diagram Demonstrating the Effect of Terminal Propagation

M. Wang et al. uses recursive partitioning to group macros together [113], then experiments with 4 different approaches to how to model the nets during recursive partitioning in order to decrease interconnect length. The following approaches can be seen in Figure 24, which has been extracted from [113]. The first approach (Figure 24(a)) ignores all external nets, the second approach (Figure 24(b)) ignores all external terminals.

Ckts	#cells	App. A	App. B	App. C	App. D
ibm01	12282	4.79	**4.71**	4.98	4.81
ibm02	19321	**13.70**	13.91	14.38	13.99
ibm03	22207	13.12	**12.83**	13.02	12.93
ibm04	26633	17.66	**16.58**	17.54	17.21
ibm05	29347	38.94	**38.21**	39.32	39.12

Figure 26. Comparison of 4 Different Approaches [113] to Wire Length minimisation.

The third approach (Figure 24(c)) is to add dummy nodes to represent external nets; this will encourage nodes to be placed close to their external neighbours, this is called terminal propagation. Terminal propagation [59] is defined as the process through which nodes external to the block being partitioned are propagated as fixed terminals. During partitioning normally only internal nets are considered when partitioning a group. But the external nets that are being ignored will still be affected by how the sub groups are positioned after the cut has been made, as shown in Figure 25. Terminal propagation (TP) is an important factor in minimising global Wire Length. For this to work an appropriate weighting scheme would have to be devised. In order to reduce the overall Wire Length in Figure 25 sub-group A2 should be placed next to B and A1 should be placed next to C. This placement would then reduce the terminal propagation effect. Terminal propagation will be discussed in greater detail in section 4.6, where the Wire Length is minimised by swapping bins around at each hierarchical level, so as to minimise terminal propagation. Finally the fourth approach (Figure 24(d)), rather than swapping whole bins around once a partitioning has been completed, single nodes are swapped one at a time to try to improve the cut set. *M. Wang, X. Yang and M. Sarrafzadeh* find that the second approach provides the best result but this method cannot be used here as it considers multi-pin nets, as shown in Figure 26, extracted from [113]. But their first approach does not perform significantly worse and is favourably compared to the other approaches. So when partitioning they show there is no benefit in considering external nets. But when deciding which side of a partition a sub-group should be placed these external nets will be considered, as having a post bin swapping stage is very useful in reducing the Wire Length (shown in Figure 25). Figure 27 again extracted from [113] shows min cut with and without terminal propagation, and then approaches B and C, which have a post-bin swapping stage. Figure 27 shows how using a post-bin swapping stage can decrease Wire

Length, while also showing that during partitioning the external nets do not need to be considered.

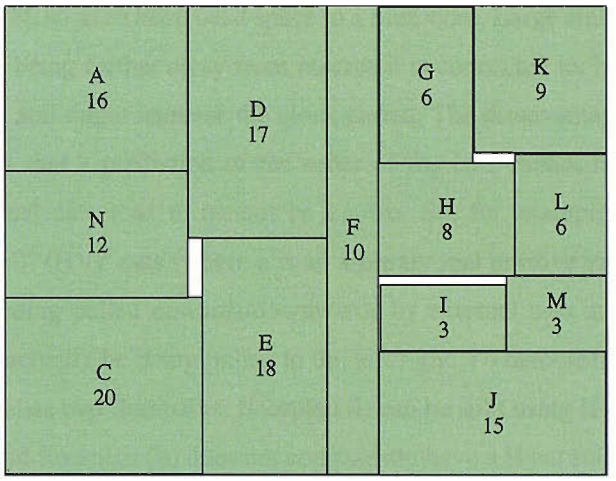
Ckts	#cells	App. A	**App. B**	App. C	App. D
ibm01	12282	4.79	**4.71**	4.98	4.81
ibm02	19321	**13.70**	13.91	14.38	13.99
ibm03	22207	13.12	**12.83**	13.02	12.93
ibm04	26633	17.66	**16.58**	17.54	17.21
ibm05	29347	38.94	**38.21**	39.32	39.12

Figure 27. Table Comparison of Conventional Min-cut Schemes and [113] Approaches.

An extension to the terminal propagation problem is the case when there is ambiguous terminal propagation. Ambiguous terminal propagation [59] arises when terminals lie equally proximate from two sub-blocks of block being partitioned, so that their destination propagation is ambiguous. To solve this case *A. B. Kahng* and *S. Reda* re-partition the circuit at each hierarchical level. *A. B. Kahng* and *S. Reda* achieve this by using recursive partitioning, first placing a group on the same hierarchical level, and then using that group's location on the floorplan to position the other sub-groups on the same hierarchical level. Then the algorithm is moved onto the next hierarchical level. This is a good idea but would be very dependent on the first group's location, so would heavily influence the other groups at the same hierarchical level and below. This could lead to a local minimum, if the initial node chosen forces the algorithm in the wrong direction. The reason for this is that the second pass uses the terminal location from the first pass. This process removes the ambiguity, as the partitioning tool now knows where the external nets lead. But the re-partitioning would be expensive in time. And there is no guarantee that partitioning the second time will cause the terminals to stay in the same place with the new information.

2.10.5 Cut Sequences

Another consideration during floorplan construction is to determine orientation of the cutting of the groups: when to create vertical cuts or horizontal cuts during recursive bi-partitioning that will form the most optimum floorplan. When recursively bi- partitioning a circuit, normally a Horizontal (H)/Vertical (V) cut is used to partition the circuit then followed by a V/H cut respectively then a H/V cut etc...



The integer values are the approximate area of each macro for descriptive purposes only

Figure 28. Representing a Floorplan of a Circuit.

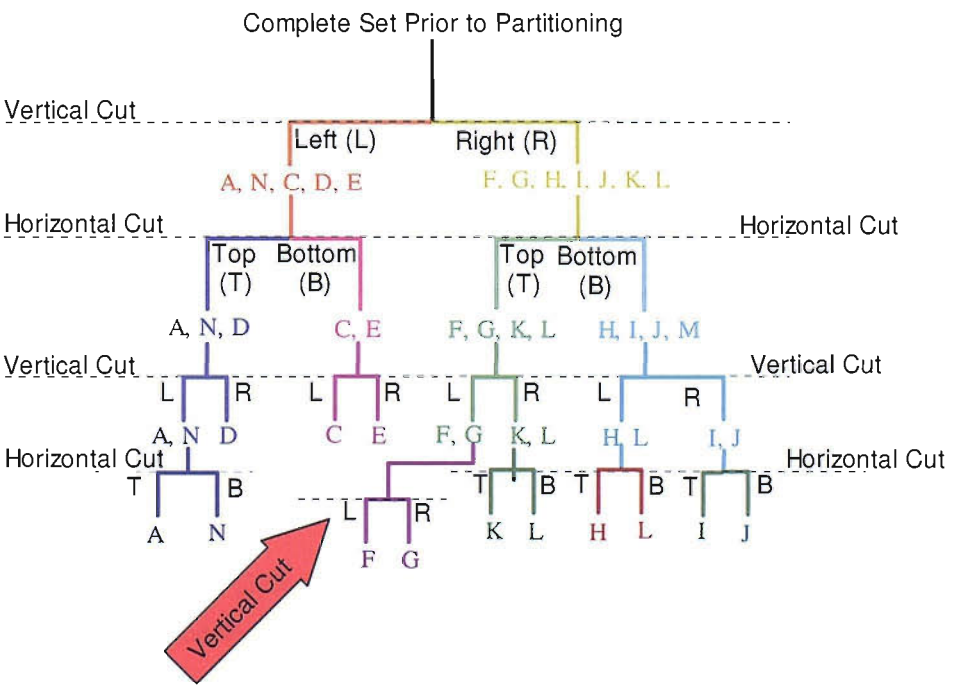


Figure 29. Corresponding Slicing Tree of Floorplan in Figure 10

If 4 components are shaped as squares with area 4, then it would be easy to predict the order of cuts when partitioning the circuit i.e. H => V => H => V. But the order might change according to the geometry of the shapes of the elements, which are in the groups being partitioned, as shown in Figure 28 and Figure 29. When F and G are partitioned they should have a vertical cut instead of a horizontal cut, hence the order of cuts is now V, H, V, V,

instead of V, H, V, H, so as to keep dead space to a minimum. Large amounts of dead space can lead to macros being further away from macros it is connected to, hence this increases interconnect length and might increase the clock period. The disadvantage with the Bottom Up methodology is that a prediction of the order of the cuts cannot be made, hence the direction the external nets lead to cannot be known. So, for example, when presuming $(\text{Horizontal/Vertical})^n$ $(HV)^n$ cuts (where n is an arbitrary real positive value) partitioning, a macro appears to being pulled upwards/downwards by external nets in the cut set, where the macros should actually be being pulled to the left/right. To demonstrate this point, look at Figure 30, which has two floorplans: floorplan (i) can be split using HV cuts one after the other; and the second floorplan (ii) does not consistently have a H cut followed by a V cut.

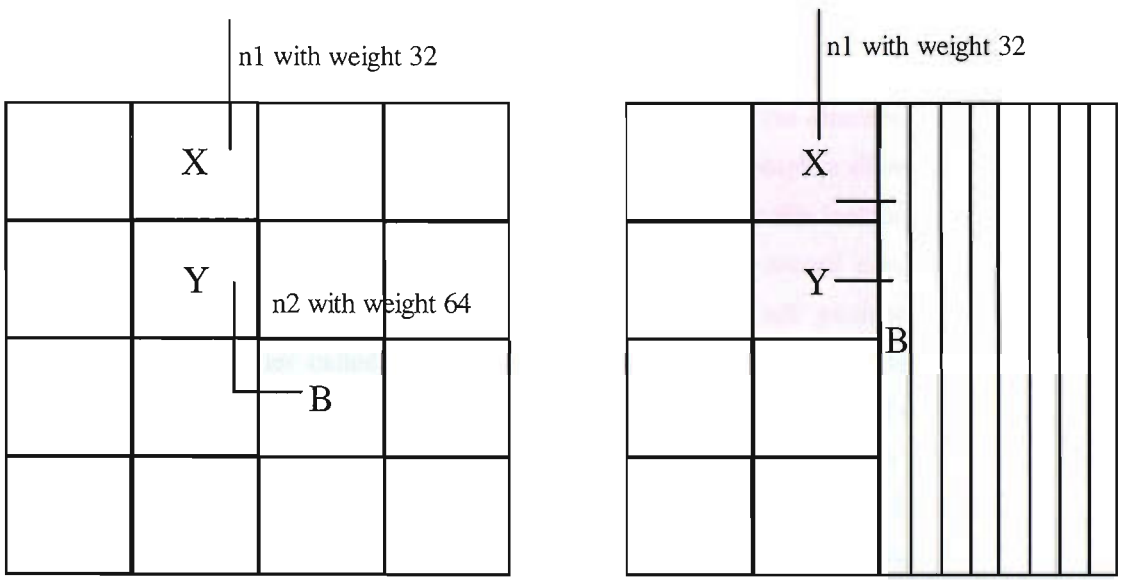


Figure 30. Diagram to Show How Net Lengths can be Affected by Cut Orientation.

Sub-Group *A* can be placed in site *X* or site *Y* in the floorplan (i). From above, a net with weight 32 is connected to *A* and *A* is also connected to *B* with a value of 64. Ignoring the length of the nets only the weight of them, it would be more beneficial to place *A* in position *Y*. But in floorplan (ii) *B* is now to the right (still with the same area but has a different shape). As the macros are long and thin, they have been placed alongside each other by using a vertical cut so as to minimise dead space. Now whether *A* is placed into *X* or *Y* is much more dependent on the external net from above as *B* only influences whether *A* gets placed left or right of a partition. So now *A* should be placed in *X* presuming net n1 is the only external net. But if it is presumed that a H cut is always followed by a V cut and that a V cut is always followed by a H-cut, then *A* would be placed in *X* as in

floorplan (i). This would not be the optimal placement in terms of *A* and its external nets. This is where combination of a top down and bottom up strategy will be useful, because prediction of the order of cuts can be made when proceeding in a top down manner, then using that information obtained on the way down the floorplan can be built in a bottom up manner. This methodology will combine the advantages of both approaches without the disadvantages.

2.10.5.1 Methods in which the Cut Orientations are Chosen

While the design is being partitioned a quick analysis of the sub partitions will be processed. Firstly the square root of the total area of the original group G will be taken. Now depending on different conditions, the cut will either be Vertical (V) or Horizontal (H). The conditions will depend on the aspect ratio (AR) (Height / Width). *Yildiz* and *Madden* show that by considering the aspect ratio, an appropriate cut direction [43] [99] can be chosen. [69] decide on the orientation of the cut once the complete slicing tree has been formed, so that the macros can be matched up evenly. But this method does not consider how the smaller macro's shape will conform to the overall design. So to envisage this global view the complete design and consequent sub partitions will be placed within boundaries called bins. *Tessier* places all the macros into bins whose dimensions are decided by the largest hard macro [47], which is a good way of estimating the order of cuts, without actually having to place the components. The bins will be at least as large as the total area of the macros they contain and will satisfy the dimensions of the hard macros. The bins will also ensure that design will fit within the boundaries of the designated chip.

2.11 Interconnect Prediction

As previously discussed, when constructing a floorplan (estimated) Wire Length values are used to guide placement. This section will now consider what interconnect properties are needed to differentiate between a good design architecture and a bad design architecture in terms of design criteria. It is insufficient to know the positions of the components on a chip. Knowledge of the routing topology is needed to improve analysis of a design architecture during exploration of the design space. Awareness of how the circuit's components are positioned relative to each other is now achieved due to recursively partitioning the circuit's netlist. This section will show how routing models can aid prediction of interconnect length properties of a design, and in Chapter 3 routing models will be incorporated into HLS to improve optimality of a design's delay in terms

of CP, hence total delay. The interconnect length properties will then be developed further in Chapter 4, to show that this same relationship can provide more comprehensive details about the circuit. This will deliver a better understanding of a circuit's hardware implementation at a higher level.

2.11.1 Different Methods for Obtaining a Priori Estimate of Routing Topology

A measure of the complexity of the interconnect topology needs to be achieved. This interconnect topology can affect delay, area, power, etc. The first metric which shall be used to represent interconnect topology is average interconnect length. Reducing the average interconnect length will reduce the size of the local nets hence allowing easier routing [109][84], but still keeping global nets under control.

2.11.1.1 Average Interconnect Length Prediction

To predict average interconnect there are three ways.

(i) A Pre-Floorplan prediction uses statistical prediction based on previous design properties. This concept of prediction can be used for many different designs as it does not need a floorplan. At this stage two-pin nets are presumed so that prediction of multipin nets is not needed (and which in a floorplan would be needed).

(ii) Post-Floorplan Prediction uses a floorplan to predict the length of individual nets. These nets can then used to find the average interconnect. This uses partial prediction as the actual routing of each individual interconnect will need to be predicted.

(iii) Post-Placement Prediction uses more routing information, as now there is a much higher level of information.

A truly a priori method is needed for obtaining estimates of the routing topology. Simultaneously the method needs to be quick enough to test many different architectures in a short enough time to make the process viable for HLS.

Pre-Floorplan Prediction

Statistical Inferences

These approaches try to predict how a circuit will behave without actually measuring the individual interconnects themselves. For the following, statistical inferences are based on a relationship called Rent's Rule. Rent's Rule is an a priori method for estimating

interconnect properties [32] [94] [7] [29] [33] [46] [74] [65]. Only the information provided in the netlist and partitioning the circuit is required to use Rent's Rule, as will be shown in section 3.8.

If any physical information about the design is desired, prior to the APR tool, partitioning the netlist of the design is the fastest method to obtain that information. The most important detail is that Rent's Rule provides physical metrics, which are accurate enough to compare different design architectures. Rent's Rule has been shown to be an accurate interconnect a priori estimate for ASICs [14] [30] [31] [8] [12] [65] and FPGAs [37] [74] [89] [25].

Interconnects between macros shall only be considered, not the interconnects between the cells that make up the macros. The reason for this is that macros are only being considered during HLS not the cells that make up the macros, but when macros are used in the following section, cells can be interchanged with macros without affecting the properties of the relationships discussed.

Rent's Rule gives a complexity measure of the interconnection topology and the quality of the placement, or if placement has not been done, an estimate of the quality of the placement using circuit partitioning. This is the relationship between the average number of terminals (or pins) T of a part of a circuit (a bin) and the average number of cells/macros (Basic Logic Blocks B) inside the bin. t is the average number of terminals per cell/macro, p is the Rent Exponent and their relationship is given by:

$$T = tB^p \tag{10}$$

Equation 10 is called Rent's Rule [12].

The relationship $T = tB^p$ provides the Rent exponent p. The Rent Exponent depends on the complexity of the interconnect topology (with higher values for more complex topologies) and on the quality of the placement (with higher values for less placement optimisation). The Rent exponent can be in the range $0 < p < 1$, but generally p's value is between 0.5 and 0.75. The relationship of the equation follows (depending on p): if p is close to 1 then this increases B^p to a higher value, which in turn makes the RHS of the equation higher. Hence the number of external nets between bins becomes larger, which means the overall interconnect length also becomes larger. As p decreases in size this also

decreases the RHS of equation 1, which means the overall interconnect length reduces. Hence this relationship can be used to predict the routing behaviour because:

- The more complex the interconnect topology, the harder it is to route the interconnects, which increases interconnect length and could increase area.
- If each bin is connected to a high number of bins this means it is much harder to place the bin next to its topological neighbours, which again will increase interconnect length because the neighbours will be placed further away.

The bins can represent transistors, gates or even entire circuits. The average interconnect length of a design's architecture can be calculated using p (derived from Rent's Rule) placed in a equation that maps out the behaviour of average interconnect depending on a design's p value. The derived equation to calculate the average interconnect length is given in section 3.9.

As Rent's Rule gives a level of complexity of the interconnect topology, this can be used to derive metrics that are heavily influenced by how complex the interconnect topology is. For example if the interconnect topology is highly dense in a particular region, this means there is a large amount of interconnect in that region, which means there will be fewer available routing tracks for nets to be placed, compared to a region which is sparse. Have fewer tracks to be routed on, makes it probable that nets will be forced to find sub-optimal routing tracks to be routed on, hence increasing the delay of those nets, which in turn can lead to an increase in the CP if those nets lie on the critical path.

Derivatives of Rent's Rule can then be used in conjunction with an estimated floorplan to further improve the design's architecture. It is shown in [31] that for FPGAs that when a placement was refined the Rent exponent also decreased, hence showing it is a valid metric for a measuring the optimality of a design's architecture. [89] shows there is a strong correlation between Wire Length and the Rent Exponent for a fixed design architecture. D. Chen et al. use Rent's Rule to achieve constant reduction in area and power [71], while also reducing delay in most cases, where they state using the results in [68] that 60% of the total power is taken up by interconnects in deep sub-micron FPGAs.

To reduce the drawback from the non-homogeneous properties of a circuit, the Rent Exponent can be obtained from values available while partitioning the circuit's netlist. This would be highly computationally expensive. To construct a floorplan, circuit partitioning

will have already been performed.

Post-Floorplan Prediction

Once a floorplan has been generated, the floorplan can then be used to estimate distances between macros, which in turn allows interconnect lengths to be enumerated in terms of delay. The length of interconnects can be used to obtain interconnect topology properties of an entire circuit, but require much more computation, due to having to enumerate every single net to obtain a complexity measure of the estimated routing topology [86].

An extensive survey on detailed routing can be found in [48] but these algorithms are out of the scope of this thesis. The routing algorithm is far too complex to obtain a priori estimate of the interconnect lengths. Detailed routing of all the nets would be impractical to obtain the expected lengths. Maze Routing [38] for example becomes redundant, due to it being too slow to be feasible. *Kastner, et al.* uses pre-defined nets to limit the search of all possible edges in which the interconnect can travel [40], but this is still too slow.

2.11.1.2 Conclusion on Proposed Interconnect Predictors

After partitioning the netlist, the only information available is how the elements are related hierarchically. So the only logical method to predict interconnect properties is Rent's Rule, because the relationship only requires knowledge of the interconnections, size and number of the logic blocks and hierarchical information obtained from partitioning. Using Rent's Rule, the average interconnect length of a design can be calculated immediately after partitioning the netlist, this will give a quickly derived metric to be implemented in the cost function of MOODS. To further estimate interconnect topology, knowledge of the congestion is needed as this affects interconnect extensively, hence congestion shall now be discussed.

2.11.2 Congestion Estimation

In order to analyse a circuits interconnect properties an accurate estimate of the congestion of a circuit is needed. But first how congestion is related to interconnect and why it is important to the prediction of the routing layout in HLS shall be examined. Where a region is congested, nets that are located in that region have much more chance of having to detour outside the bounding box [98], hence increasing the net length. The reason for the last statement is when the routing channels of a chip get congested, the design will become harder to route, which will increase time spent routing the chip and could increase the average interconnect length.

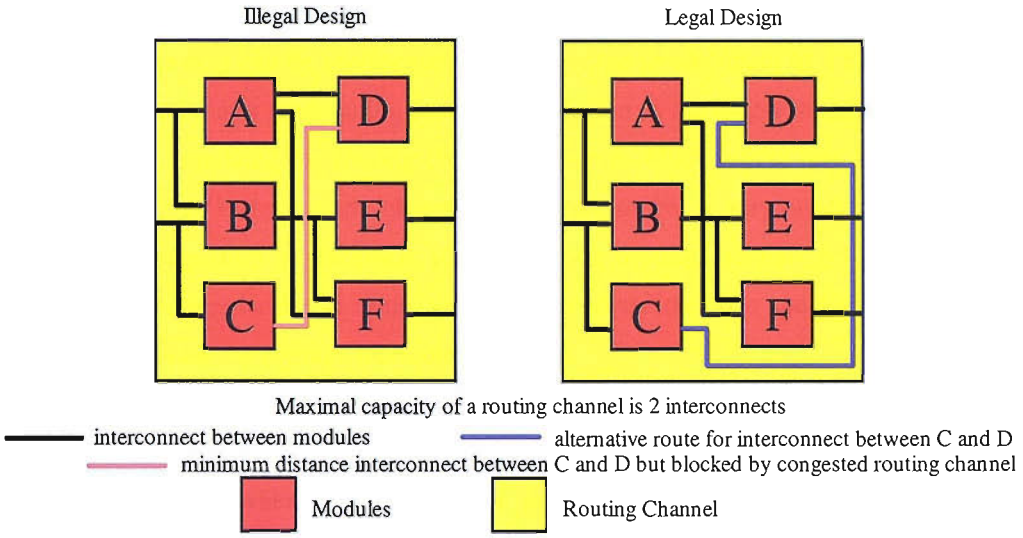


Figure 31. Circuit which Shows Increased Interconnect Length Due to Routing Channel Congestion, with 2 Layer HV Routing.

A chip's routing channels become congested in various ways. The chip can be affected by having longer interconnects between modules, which means more routing channels are monopolised, i.e. the tracks within the routing channel are used, leaving fewer tracks available for unrouted nets. This causes the routing channels in that area to become more densely populated, which leaves less room available for routing other interconnects. This unavailability can lead to a routing channel becoming full, i.e. blocked, which means that interconnect will need to find a different routing channel to connect all the pins that belong to the same interconnect. This alternative route can increase interconnect length, as shown in Figure 31. Congestion of a circuit relates to what interconnect resources are needed and what resources are available, i.e. the closer routing demand is to routing supply the more congested a circuit is. The Four main factors that influence congestion are:

- Area Utilisation
- Availability of Routing Channels
- Number of Nets
- Length of Nets

Area Utilisation represents how dense the surface of the chip will be, i.e. the ratio of the number of occupied slices to the total number of available slices. As an FPGA has fixed routing channels, the widths of the channels do not need to be considered when considering the level of congestion in a region. As the routing channels have a fixed value

they will not influence the calculation of the congestion. The number of nets and the length of those nets can be represented by the average interconnect of a circuit. Average interconnect is a major contributing factor to congestion as the longer the interconnect, the more densely populated the routing channels become, which leaves less room available for routing. The Wire Length objective is found to be a very useful metric when minimising congestion in placement [63]. When the routing channels get congested, the design will become harder to route, which will increase the time spent routing the chip and could increase the average interconnect length.

The higher the congestion of a circuit the more chance there is that a routing channel of a circuit will be full when wanting to route an interconnect in it. This means that the interconnect needs to find a different channel to connect all the pins that belong to the same interconnect. This alternative route can increase interconnect length, as shown in Figure 31. C. Cheung et al. attempt to reduce this by altering the paths of the nets to reduce congestion [121], but that is too low level, so is infeasible for HLS.

Also the more an APR tool has to find alternative paths for interconnect, the longer the run time will be, especially if the routing tool has to rip up all the interconnect and start again. But re-routing the chip takes time and might not work, so in the worst case, a larger chip will be needed to allow for the routing. If a larger chip is not desirable or possible, the RTL design can be altered to decrease the area, which will increase the time to market window and could cause further problems with delay.

2.11.2.1 Definitions of Congestion

Routing demand is a made up of an internal routing demand and an external routing demand. The internal routing demand for a region is proportional to the total routed Wire Length within that region, which can be obtained from the average interconnect in that region, which can be derived from Rent's Rule.

External routing demand needs a probability density function to represent the probability that interconnects will pass through that region from other regions. Figure 32 shows the internal and external nets according to placement regions. *J. Dambre et al.* sum up the individual routing demands in each region to form a congestion guide [13]. The congestion guide can be used in the cost function as an extra variable and has been shown in to be very useful when optimising a design [13]. The more congested a design is the

harder it is to route and the average interconnect length will increase.

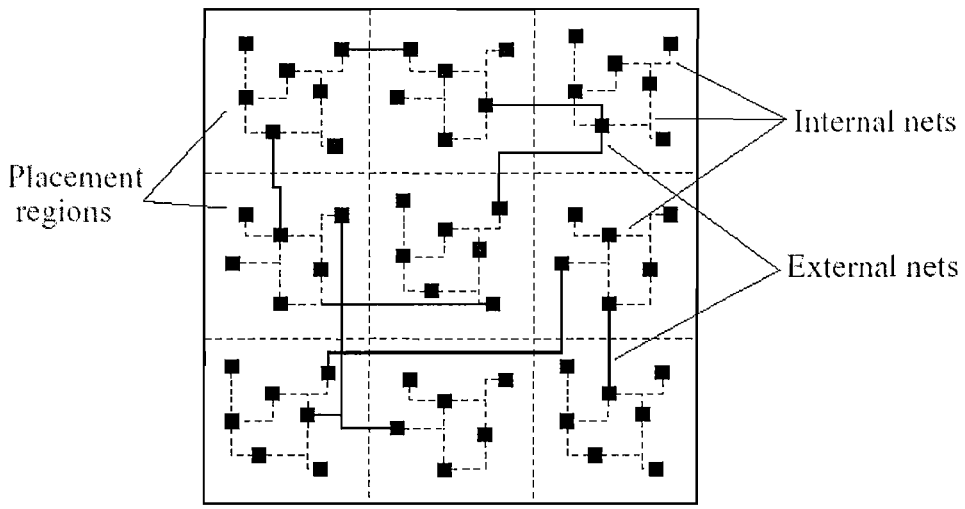


Figure 32. Placement Region in More Detail.

2.11.2.2 Previous Work on Congestion Estimation

Congestion Reduction During Placement

Pandini, et al. do not believe that accurate prediction of congestion of a circuit can be achieved without introducing iteration loops between the Logic/High Level synthesis and APR tools [35], which requires multiple iterations. But MOODS is comparing many different architectures for a design, so this is not feasible. If interconnect length can be predicted at a high level then congestion should also be predicted at a high level, on the basis that congestion is highly dependent on interconnect length and routing channel resources. Routing channel resources would depend on the technology but especially when dealing with FPGAs, the routing resources are fixed, hence can be predicted with a high level of accuracy. *P. N. Parakh* remarks in [118] that Wire Length is a good metric for good placement, but it is an indirect measure for congestion as it only represents demand not supply. But an FPGA's supply is constant with respect to each family of chips, as discussed earlier. Previous methods of congestion derivation will now be shown. Yang, et al. show two approaches to measuring the congestion of a circuit [13]. The first approach measures the peak congestion, which is given by:

$$C_{\max} < \frac{C_1}{\sqrt{N_c}}\left(\frac{1}{2}+2\alpha\right)\frac{\sqrt{N_c}\alpha^{2H}-1}{2\alpha^2-1} \qquad (11)$$

where C_1 is the cost of the cut at the first Bi-Partition;

α is the ratio between the net cuts of two consecutive partitioning operations in an ideal circuit this is equal to 2^{-p}, where p is the Rent Exponent;

N_c is the number of cells in the circuit;

H is equal to $\log_4(N_c)$.

Again this can be obtained without relying on a floorplan, so could be calculated as soon as partitioning has been completed, and hence will give a quick congestion metric for a design.

Yang, et al. then proceed into local congestion, which is more detailed and would require a floorplan. The level of routing demand in the region controls local congestion. Routing demands give two useful properties for the following

- Routability in a placement region
- Congestion guide for the overall Design

X. Yang et al. present an equation to measure the congestion in sub regions [34], which is:

If the macros of sub-groups $G1$ and $G2$ of group G have approximately the same area, then Congestion

$$C_i = \frac{p_i + \log T_{bi}}{\log\left(\frac{B_i}{N}\right)}, i = 1,2 \qquad (12)$$

Where r_i is equal to the local Rent exponent

C_i is the congestion of that region

$\log(T_{bi})$ is the intersection on y axis

N is the number of sub partitions

B_i is the number of logic blocks

P is the Rent Exponent

The problem with this equation is that the circuit would have to be large enough that the local regions are themselves large enough for the Rent exponent to be extracted.

In [18] a probabilistic matrix is used to predict all possible routes that a net can take. To find the congestion they then enumerate every net, which is computationally expensive. This graphically shows however, how congestion prediction is a good prevention measure for congestion limitation in the final implementation. [109] define local nets in terms of a

net that has a terminal in one bin and then has a terminal in one of its neighbour's bins, otherwise it is deemed to be a global net. Firstly, multi-pin nets would have to be considered for this to work and we do not consider them. Secondly, this method would depend very highly on bins being of equal size. This is hard when dealing with macros of varying size, and can lead to unfair representation as shown in Figure 33.

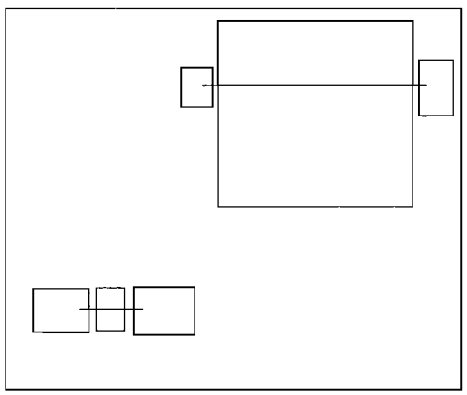


Figure 33. To Show How Different Macro Sizes Affect Global Wire Length.

The congestion improvement using the method found in [109] is predominately based on Wire Length and gives favourable results. [105] use the number of times a net is cut, but this time they divide the Wire Length by the number of edges that cut it. This is a much truer representation of local congestion. They define the overall congestion of a chip as the total over flow of all edges or the number of congested bin edges. Again this depends on size of bins, which is not applicable. They show that Wire Length is a very good objective for congestion, but state that this is not always the case. To combat this *M. Wang* and *M. Sarrafzadeh* introduce a simple look-ahead strategy where they consider how a move will affect the system by introducing the following cost function:

$$\max(d_e, s_e - \delta) - \min(d_e', s_e - \delta) \qquad (13)$$

where d_e is the current demand on edge e

d_e' is the resulting demand on e if move accepted

δ is an adjustable parameter.

Again this strategy would be useful if we were trying to improve a congested floorplan by moving bins around but a global congestion value is wanted. But this paper shows that Wire Length is a good metric for estimating congestion.

P. Kannan, et al. present 4 methods of predicting congestion in a circuit [42]. Three of the methods would need a floorplan, while the fourth uses Rent's Rule [13], and hence does not need a floorplan. They are measuring the peak routing demand. The three methods are based on fGREP[17], RISA[16] and Lou's Method[18]. Enhancements are shown for fGREP and Lou's Method, where they increase the speed of fGREP and accuracy of Lou's Method. Results show that the enhanced fGREP is very accurate while maintaining a high speed. RISA is extremely fast while beating the accuracy of Rent's Rule and enhanced Lou's Method. Lou's Method beats Rent's Rule for speed and accuracy. The times given for Rent's Rule include the time to partition the circuit, but partitioning will have already been completed in our case. Calculation of the congestion metric would only require the values to be inputted into the equation, which would require a negligible amount of time. [98] extends RISA from just reporting the Total Routing Demand to finding the maximum channel width, which is not in the scope of this work. For a quick analysis, Rent's Rule is still the best candidate for measuring Peak Congestion, so it will now be discussed in more detail. But when measuring local congestion, enhanced fGrep or RISA looks very attractive, depending on accuracy or speed respectively.

Congestion Reduction by Utilising Dead space

When trying to reduce congestion on a circuit, a preventative method can be used, that is obtaining a congestion metric and using it to influence the design of the circuit. Or you can cure the circuit of congestion. A circuit can be relieved of its congestion by spreading out the macros. Densely placed designs with little dead space can increase congestion, while designs that are sparsely packed can increase delay so a compromise needs to be found. [58] use an analytical approach to solve this problem, which is too slow. Another congestion minimisation method uses the dead space on a chip for expansion, so as to reduce congestion [109][85]. *B. Hu* and *M. Marek-Sadowska* alleviate congestion in [109] by considering dead space (also called whitespace). The method presented in [109] is used to improve a congested design, and not prevent the design from being congested in the first place. *P. N. Parakh et al.* use Quadratic Programming in [118] to relieve congestion by increasing bin sizes. They form these bins by recursive partitioning. But these types of methods will have little impact on the usefulness of the floorplan in HLS, as they use dead space to reduce congestion, whereas we are not concerned with dead space as it will have little impact on distance between metrics, as the floorplan is an estimate. In the work of this thesis the only concern is that macros are in the same approximate location (with respect to all other macros) as their location when actually

implemented on a chip. These methods are only useful once a floorplan has been realised so as to further improve a detailed placement, which is too low level for High Level Synthesis.

In [87] *N. Selvakkumaran, P. N. Parakh* and *G. Karypis* address the problem of locality of congestion, with the main focus on the non-homogeneity in routing supply, which is not applicable to FPGAs. But they do introduce a measure called "perimeter degree", which is the net degree of a bin divided by the bin's perimeter. The usefulness of this can be seen in Figure 34 [87]. The perimeter degree is a much fairer representation of the congestion in that region than just the net degree. B has more routing resources available, thus it will have a lower perimeter degree compared to A, which has the same net degree but has less routing resources. They then use Rent's Rule to prove the following relationship:

$$\begin{aligned} \text{Perimeter Degree} &= P/Perimeter \\ &= kB^p/sqrt(area) \\ &= const * B^{(p-0.5)} \end{aligned} \tag{14}$$

where p is the Rent Exponent, the Perimeter is the perimeter of the region on which the perimeter degree is being measured in. Finally B is the size of the region same as in Equation 10 (kB^p), where the size is measured as the area of B, hence the *sqrt(area)* is equal to $B^{0.5}$. Obviously this would depend on the bins being square.

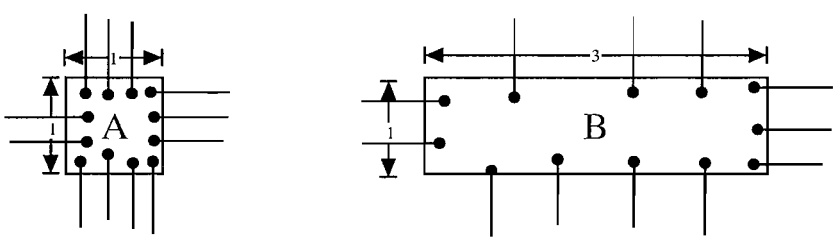


Figure 34. Considering a Macro's Area when Considering Congestion can Alleviate Congestion.

Effectively the perimeter degree is based on Rent's Rule which is the same basis for estimating the average interconnect length. *N. Selvakkumaran, P. N. Parakh* and *G. Karypis* then introduce a heuristic to avoid finding the Rent Exponent. To get to this stage, circuit partitioning will already have been performed, so the Rent Exponent can be

obtained for free. They use this perimeter degree as a constraint to alleviate congestion by expanding the bins, by reducing the perimeter degree it alleviates congestion as shown in Figure 34.

2.12 Conclusion of Congestion Methodologies

The critical path is of most concern, but as the congestion metric will be a global measure, the critical path will be dependent on congestion. The higher the congestion, the larger the delay of the interconnect belonging to the critical path, which is detrimental to the design. Where [13] splits congestion into internal and external routing demand, internal routing demand is of only concern where the region is the entire chip, as a global congestion measure is desired. The external routing demand would be the IO pins, hence the internal routing demands for a circuit is predominately influenced by the average interconnect length of a circuit.

2.13 Floorplanning

So far, average interconnect and congestion have been the only physical properties that have been considered for estimation. They have not needed a floorplan to calculate their values. But if merging two macros is being considered and the distance between these macros is needed, knowing their relative position is not enough as we do not know in which direction the macro's neighbours lie. Two methods can be used to obtain an approximate distance of a macro and with any other macros on a chip.

Firstly using the hierarchical information, the number of cuts between the two macros could be used. So, a larger number of cuts could imply that the macros are quite far away from each other. But this can be misleading, as it would depend on where each sub group is placed. The second method is to construct a floorplan. The main application of this floorplan is to obtain a fast, global view of the macros. The information from this floorplan is basically to show whether macro's respective distance is at a value in which it would be desirable to perform transforms during HLS. These transforms are either to merge macros on the data path, or to unmerge a macro that has previously been merged.

The method of constructing a floorplan has already been decided (Recursive Bi-Partitioning, but how a floorplan is actually represented has not yet been discussed. The floorplan will be constructed during HLS, as a simple representation containing the dimensions and locations of all the macros. Hence when constructing the floorplan a

representation is needed that is fast to construct, in which information is easily available, and which does not require a large storage space. This floorplan is needed so that distances between macros can be calculated accurately, in order to aid decision making when considering duplicating or merging functional units (discussed in greater detail in section 4.11). The following section will now discuss these individual properties and describe relevant floorplans that would have these desirable properties.

2.13.1 Floorplan Representations

The floorplan representation includes the geometry and size of the components, including the gaps in between the elements if they do not fit exactly together. The floorplan will need to include the type of component, either a hard macro, in which the geometry of the element cannot change or a soft macro, in which the geometry of the element can change, also the orientation of the element will need to be considered. Hard and soft macros have to be considered, as FPGAs have hard macros such as adders (any macro that uses carry logic) and FPGAs also use soft macros for any other macro that does not use carry logic. Both types of macros need to be considered, so that the floorplan in MOODS will be similar to the FPGA's final placement.

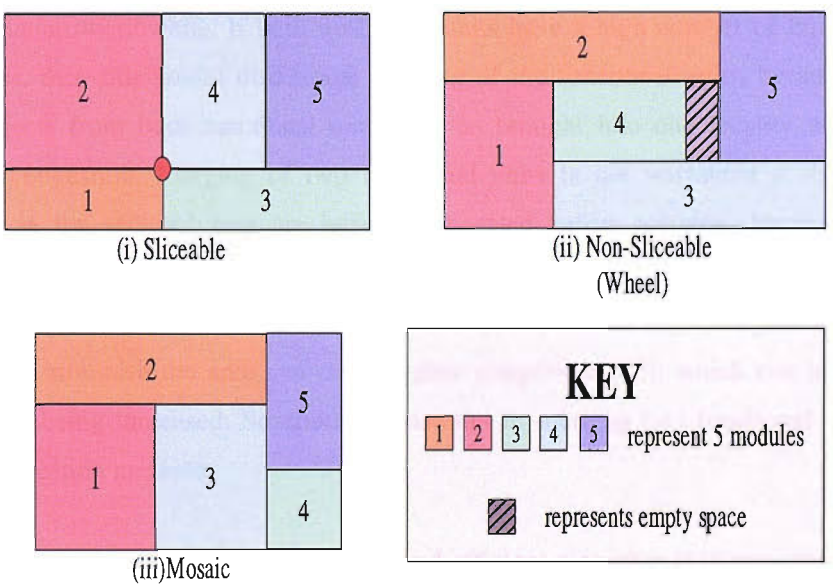


Figure 35. Different Floorplan Representations.

There are 3 types of floorplan representations: sliceable, non-sliceable and mosaic as shown in Figure 35. A sliceable floorplan is where the macros can be organised in a set of slices which recursively bisect the layout horizontally and vertically (hence a non-sliceable floorplan is where you cannot). Figure 35(ii) is also called a Wheel. A mosaic

floorplan is similar but the floorplan cannot have any gaps as in Figure 35(ii) and cannot have a crossing cut as shown in The merging transformation implemented in MOODS is shown in Appendix 7.1.2.1. To merge a functional unit depends on the following conditions. First whether the operation of the functional units can use the same hardware and in which states they exist after scheduling has been performed. Secondly the interconnect length between the two functional units can be thought of as a gravitational force, where the larger the interconnect length, the less gravitational pull the units have on each other. Conversely the larger the gravitational pull the more likely that the functional units will be merged, this is dependent on whether the functional units are suitable for merging, for example the units will need to be of the same type (i.e. both adders). Two functional units would be merged to reduce the area of a chip. If the area of functional units is large then there is more to gain by merging rather than if the functional units are small. Ideally the area of the functional units being merged needs to be larger than the area of all the components to be added to legalise the design, this is so that the design's area is reduced. In Figure 13, this would mean that the area of adders one and two should have less area than the new adder and multiplexer (where the multiplexer was added to keep the design functionally correct) that replaces them, in order to make this transformation worthwhile. If both functional units have a high number of Input/Output net values, then this would discourage merging of the functional units, because all the interconnects from both functional units will be brought into one locality which will increase congestion. Merging of two functional units is not warranted if the routing channels in the affected area are heavily congested before merging, because all the interconnects which flowed into the original functional units will now flow just into one functional unit which will increase routing density to an even higher degree. Excessive attempts to minimise the area can cause higher congestion [23], which can lead to the actual area being increased. So another constraint to merging two functional units is a global congestion measure.

Figure 35(i) by the dot. To produce a fast and efficient algorithm it is very important to decide how the data will be stored and accessed, enabling ease of manipulation. An extensive analysis of recent advancements in floorplan optimisation can be found in [90][15][96] and [24]. [92] introduces bounds on the number of solutions for a particular floorplan representation, the reason being that the smaller the solution space the easier it is to find the most optimal solution. [15][24] compare the attributes of all the recent methods in optimising floorplans. All the methods presented in these papers are focused

on area optimisation (as expected), but the run time of the algorithm is more crucial. This is due to many designs being constructed and RBMT should have produced an optimal hierarchical structure from which an optimal floorplan can be constructed. After RBP a circuits netlist, a slicing tree (structure) [88] [69] is produced. If the placement algorithms discussed in the last paragraph were implemented, the slicing tree would have to be transformed into a different format, which would require time (hence increase the run time). Slicing floorplans, due to their nature, are very fast and easy to implement. Wong shows that slicing floorplans produce good floorplans for soft macros [39], and in the case of FPGAs the majority of the macros are soft. Incremental changes are easily applied to slicing trees [97], which is a highly desirable property, as this speeds up the transition from one slicing tree to the next slicing tree, when the Data Path (DP) is changed after a transformation has been applied during HLS.

M. Lai, D. Wong show that a slicing tree can represent a non-slicing floorplan [21]. They use XY-compaction to transform a slicing tree into a non-slicing floorplan. XY-compaction pushes every macro to the left until no macro can move any more to the left, then every macro is pushed downwards until no macro can move downwards. This is repeated until every macro cannot be moved either to the left or downwards. Alternatively the initial push could be downwards then to the left which might give a different floorplan. The drawback of this method is that it might separate macros from each other, where partitioning has derived the macros should be placed together to minimise Wire Length.

2.13.2 Calculation of Macro Proximities

To enhance interconnect prediction to include individual interconnects, knowledge of how the macros are placed with respect to each other is needed, which cannot be achieved by simple partitioning. To enable this better understanding of how the individual components are connected a floorplan has been produced. Two-pin nets are then used to model interconnects to reduce computation with a negligible loss in accuracy. The half perimeter of the bounding box is then used to estimate the actual length of the respective interconnect. This method of calculation will also be used to measure the distance between two macros placed on the floorplan, whether connected or not. The distance between two macros when not connected will need to be known in the case when decisions are being made as to whether to merge a functional unit. This will now be discussed in the next section.

2.14 Conclusion on Interconnect Predictors within HLS

In this chapter physical estimation has been discussed, with the main focus on Interconnect Prediction. The entire prerequisite that are needed to allow Interconnect Prediction during HLS were introduced, and the following conclusions were made:

The Rent Exponent *(p)* obtained from Rent's Rule relationship shall be used to obtain a measure of the estimated complexity of the routing layout of a design's architecture once implemented in hardware. The higher the complexity of the routing layout, the larger the interconnects will be due to less likely that the interconnects will find an optimal path. To obtain *p* a slicing tree of a design architecture is derived. Recursive Bi-Partitioning (RBP) has been chosen to construct the slicing tree. RBP minimises the cut set when performing the partitioning. *p* will then be input into an average interconnect equation that will take area of the design into account as the larger the area of a design, the greater the interconnects will need to span.

Estimated Average interconnect length (AIL) will give a measure of the congestion of a circuit, as the higher the AIL, the greater the demand on the routing channels. Finally to obtain interconnect lengths or distances between macros (which is needed when deciding on whether to perform merging or duplicating functional units with respect to interconnect lengths) the hierarchical slicing tree formed from RBP is used to construct a floorplan, which will then give locations of all the macros in a design. The estimated interconnect properties from Rent's Rule (derived from RBP) and the floorplan will then be made available to MOODS to influence decisions on which candidates are most appropriate for merging and duplicating within a design architecture. Through the increase in accuracy of the physical characterisation of the design architecture, these transformations will have an improved effect on the optimality (reducing CP) of a design's physical implementation in hardware, hence produce timing closure.

Chapter 3 Pre-Floorplan Interconnect Prediction

Foundations now have been laid on which to find suitable Interconnect Prediction Methods to guide our HLS tool. Once all the new interconnect metrics are incorporated into MOODS, the new metrics will improve the optimality of the eventual design architecture. The main focus of the metrics will be the improvement of the delay optimisation in terms of total delay and clock period. The first priori estimate of interconnect properties of a design will be the average interconnect length, derived using Rent's Rule as an accurate predictor of the average interconnect length of a design once placed on the designated chip. In the case of this thesis, the targeted chip for the designs to be placed on will belong to the Xilinx Virtex series. When average interconnect prediction is shown to be accurate (high enough for high level synthesis) this will show that prediction of the general routing layout can accurately be achieved at a high level. This can then be used to guide design space exploration during HLS, as routing layout can drastically affect a design performance in terms of delay. When the designs optimality in terms of delay shown to improve using interconnect prediction, this will show that Interconnect Prediction is a beneficial addition to HLS. At this stage the hierarchical information obtained from Recursive Bi-Partitioning (RBP) (i.e. how macros should be grouped together) obtained during HLS will be used to aid the placement in Xilinx, this will be shown to improve designs in terms of delay. This is due to placement forming the foundation for all the routing that follows; hence a good placement can provide a good platform for routing. Also the design improvement shows that during HLS the groupings formed during circuit partitioning are accurately portraying an optimal placement. The Quasi-Exhaustive (section 2.7.3.2) algorithm will be used for this as it gives a fair comparison between all the metrics belonging to the cost function, allowing an excellent way of discovering whether Interconnect Prediction within HLS can provide Clock Period Reduction. This will now lead onto the next chapter, where a floorplan generated from the hierarchical information provided by RBP (during HLS) will be used to make much more detailed analysis of interconnect characteristics of a design, hence aiding decision making during HLS.

This is where using the estimated location of macros and the routing topology will be used to consider whether a certain transformation will cause a detrimental effect on the routing topology which in turn will cause a detrimental affect on the delay as the interconnect on the chip plays a large role in the total delay produced by a chip. These metrics will need to be formulated so that MOODS can effectively use them when performing design space exploration.

In this chapter interconnect predictors that do not require a floorplan to be realised will be discussed in terms of how the metrics are obtained, and in what way the metrics influence design exploration, with results to validate the methodologies. In the next chapter interconnect predictors that need a floorplan will be introduced. As discussed in earlier chapters, to achieve a quick analysis of a design's routing layout, circuit partitioning will form the basis of all interconnect prediction in this thesis. But in order that circuit partitioning and all the estimation tools that follow are accurate, the Xilinx Virtex Series Architecture needs to be understood.

3.1 Architectural Considerations

When obtaining a metric to be placed with the cost function of MOODS during synthesis, the metric needs to represent the routing characteristics and the type of technology will have to be considered, for example whether the design will be implemented on an FPGA. Hence the architecture of the chip that the design is to be placed needs to be quantified in terms of area and delay. This means knowing the number of placement sites that are available, and how many placement sites are needed for a macro to be placed on the designated chip. These values are then stored in the cell library in MOODS, this library is then accessed when providing the structural information to the black boxes that represent macros during HLS. In terms of delay, the length of time that a macro takes to complete its task when implemented on a chip needs to be known. Finally how many IOBs (as shown in Figure 5 in section 2.4) does the chip contain, so that the designs off chip nets do not exceed the number of IOB sites.

The series of chips that will be targeted in this thesis belong to the Virtex FPGA Series, which comprises 9 distinct chips, with the same basic architecture. The Virtex FPGA series has been chosen as it offers a wide range of system gates from 50000 (XCV50) to 1000000 (XCV1000), enabling many different design architectures of varying sizes to be implemented on the chips. The chips allow a highly optimal placement and routing of

designs and are a very popular brand of chip. The Xilinx Virtex Structure is the same as the structure in Figure 6 in section 2.4. A functional block on a Virtex chip is called a Configurable Logic Block (CLB). Each CLB on a Virtex chip is comprised of two slices. A slice consists of two independent SRAM Look Up Tables (LUT), embedded Multiplexers, Carry Logic and two Registers. Each LUT can be configured into a 16 x 1 RAM unit or a 16 bit shift Register. When a LUT is configured into a RAM unit this is known as distributed RAM. There is also dedicated Block SelectRAM (BRAM). BRAM are placed in two columns, the first column is placed in between the first column of CLBs and the IOBs to the left, and the second column is placed in between the last column of CLBs and the IOBs to the right.

The routing consists of 5 layers of horizontal and vertical routing channels. Programmable Interconnect Points (PIP) are used to connect the routing of channels together, where they are pre-programmed to define in which direction a signal goes [23]. The PIPs are located in switch matrices (boxes) that are located where the horizontal and vertical routing channels overlap. Routing can also pass through CLBs using pass transistors. Hence by decreasing the interconnect complexity, this resource will not have to be used, thus decreasing the number of slices needed to place the design.

There are different types of routing on a vertex chip: general purpose routing and global routing. The general purpose routing makes up the majority of the interconnect resources. The general purpose routing can be split into 3 types:

1. 24 single-length lines between adjacent switch matrices in 4 directions
2. 72 buffered hex line routes switch matrices to other switch matrices 6 blocks away again 4 directions.
3. 12 longlines, in each column and 24 longlines in each row, are buffered bi-directional wires that distribute signals across the chip. If the longlines run in the vertical/horizontal direction the wire runs from the furthest left/top of the chip to the furthest right/bottom of the chip.

Global routing is used to distribute the clock and any signal with high fanout. There are two standards of global routing; the primary global routing contains 4 dedicated global nets with dedicated input pins for clocks, driven by global routing buffers to reduce skew. The secondary global routes contain 24 backbone lines, 12 across the top of the chip and 12 across the bottom of the chip. 12 distinct signals can be distributed using these lines

via 12 longlines in each column. There is one more type of dedicated routing, which is for the carry logic, there are two dedicated routes per CLB for vertical carry signals to adjacent cells.

The interconnect resources just described dominate the area and delay of an FPGA, so FPGAs are prime candidates for the use of interconnect prediction, as reducing the interconnect lengths will reduce the delay of an FPGA. When considering the layout of interconnects (routing layout) there is one major factor that influences the optimality of the routing layout, which is Wire Length. The longer nets are, the more routing channels they occupy, which means, that when other nets are wishing to be placed, there is more chance that these longer nets occupy routes that would provide the shortest distance for nets that have not been placed yet. Hence forcing the nets to find longer routes, increases interconnect delay, which can lead to an increase in the critical path if the nets lie on it.

3.2 Pre-Floorplan Interconnect Prediction

The interconnect properties are needed for accurate estimation of the density and performance of the design, especially when considering deep submicron designs. Wire length estimation of the interconnect lengths in a design implemented in hardware is fundamental to measuring the affects of the interconnect topology. Wire length estimation can include average interconnect of a circuit or individual interconnects between modules. Wire length estimation has become increasingly more relevant, due to the switching delay of gates reducing significantly with respect to interconnect delay. Hence the interconnect delay can no longer be ignored, (making interconnect delay a greater influence) when deciding which design architecture is to be implemented. Minimising the detrimental affects of interconnect on the Clock Period, will produce a highly optimal design in terms of delay. The longer interconnects between macros are, the larger the delay between the macros connected by these interconnects will be. Hence if the length of interconnect between macros is reduced, the delay will also be reduced. To enable delay reductions to be made, accurate estimates of interconnect lengths between macros needs to be obtained. These estimates can then be used when making decisions on whether to perform a transform on the design architecture (such as merging or duplicating functional units) during synthesis. The distance between macros cannot just be used by itself for the interconnect prediction, as the routing is also affected by congestion, where the more congested a chip is the harder it is to route a net on its optimal path, which will mean a higher chance for the net length to be sub-optimal, hence causing an increase in the delay

of the net.

3.2.1 Average Interconnect Length Metric

A property that can be affected by the average interconnect of a circuit is the interconnect congestion of a circuit, which can influence circuit area and interconnect length. A higher number of longer interconnects can lead to more nets having sub-optimal paths. The higher the congestion of a circuit the more chance there is that a routing channel will be full (when wanting to route a net into it), this can cause the delay to increase as stated earlier. And the more the nets have to look outside their locality, the greater detrimental effect they will have on their neighbours which could cause a chain reaction, affecting the entire chip. Circuit area can also be influenced by congestion if the routing channels become so full that the interconnect overflows the chip, hence the circuit would need to be put on a larger chip. This will increase the design's area or the design's architecture would need to be changed to allow it to be placed on the current chip.

The higher the complexity of interconnects on the routing layout, the higher the congestion of the circuit will be. Rent's Rule can be used to represent the interconnect complexity (Rent Exponent *p)* of a design's architecture. Interconnect complexity of a design gives a detailed guide of the density and performance of a design [6]. Rent's Rule achieves this by using circuit partitioning to obtain a relationship between the nets cut at each hierarchical level and the size of the sub groups at that hierarchical level. This relationship then allows an estimate of the complexity of a design's routing architecture, which is represented by the Rent Exponent *(p),* the less complex the routing architecture the better the design is in terms of routing potential. Just p by itself is not a true representation of an interconnect layout when considering delay between macros, as it does not take into account the size of the overall design architecture.

The area of the design architecture needs to be taken into account as the larger the area of a design architecture, the larger the interconnect delays on the nets, as nets will need to cover more distance. To take area into account, the Average Interconnect Length (AIL) of a circuit is used. AIL is calculated using equations which were derived under assumptions on how p is obtained. AIL now gives a fair representation of the interconnect complexity of any given design architecture, and can be used to measure how good a design will be in terms of interconnect when physically implemented. Hence AIL will be the first metric that will be used to influence decision making during the optimisation process within MOODS.

If the average interconnect length is relatively high for a particular design's architecture, this can lead to a chip having a larger area, as the longer interconnects need more available routing channels or even CLBs (using pass transistors). If there are no more routing channels available, a larger chip will have to be used to allow routing of all interconnects.

The larger the average interconnect length, the larger is the Clock Period (CP), hence total delay of the circuit, due to the larger interconnects between components. The average interconnect length of a circuit is a beneficial metric to be used in HLS, due to the one value expressing numerous characteristics in which to compare alternative design architectures. This metric will be used within MOODS cost function to determine whether to accept transformations on the design architecture during synthesis.

Congestion will be represented in the cost function by the Average Interconnect Length (AIL) of a design (while also providing the average delay between macros), as the AIL gives a complexity measure of the routing layout of a proposed design architecture. A Congestion measure will also be used, when testing to see if macros should be merged or duplicated during synthesis. The higher the congestion the more undesirable long nets are between macros, as a high level of congestion will make interconnects even longer. So if any transformation would increase the length of interconnects between the macros, then the higher the congestion and the less likely we are to accept that transform.

3.3 Accurate Area Estimation

In order to obtain accurate estimates of the number of slices that a macro would occupy on a Xilinx Virtex chip, a tool called COREGEN developed by Xilinx was used. COREGEN allows a designer to map macros into cells that can be placed on an FPGA. Hence this tool gives accurate information on how many slices a macro needs to be placed on a Virtex chip. The different types of macros that are supported by MOODS are then mapped using Coregen, the relation between the area of the macro and the number of inputs plus the bit width of each of the inputs is formulated. Once this formulation has been completed it is placed in a library for MOODS to access when constructing the black boxes. As there are two registers within each slice, some registers maybe combined with other functional units that the registers store the output of. By observing which registers get observed when a design is placed on a Xilinx chip it was noticed that registers with one input and one output were generally absorbed into the same slices as the functional

unit they store the value of, so the register that has been absorbed will no longer contribute to the total area estimated by MOODS. But if the register has multiple outputs it was also observed that if the number of output nets numbers less than approximately ten, the register would still be absorbed. The register absorptions can be seen in Figure 36.

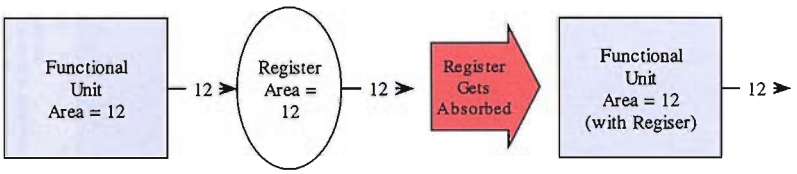


Figure 36. Register Absorption.

Figure 105 (Appendix 7.7) shows the total area (the total number of slices needed to place a design) estimations of a design against the actual area results, with the % error value. The estimated area is produced by MOODS using the method in the previous paragraph, the actual area is measured once the design has been run in Xilinx and has completed the Place and Route stage. The average % error for all the designs equals 2.666012 with the range [-8.40662, 17.88377]. The % error is sometimes negative as the number of registers absorbed is overestimated. A positive average % error is to be expected due to logic being optimized during the Logic Optimisation stage in Synplify. The reason for the high overestimate is that MOODS does not always remove all redundant logic, to tackle this problem is out of the scope of this Thesis. But as is shown in Figure 37, majority of the % error differences are within 10% with a few outliers.

Now that reasonably accurate area estimates are provided in MOODS (for macros being placed on a Xilinx Virtex chip), partitioning of the circuit netlist can now be made, where confidence that the size of the groups being partitioned is within acceptable accuracy. The accuracy of the partitioning and the floorplan that will follow will be increased with more accurate area estimation (improvement of the previous area prediction of MOODS). But to ensure that the floorplan will be as accurate as possible, a method for influencing how the design is placed during the Placement stage in Xilinx will be introduced at the end of this chapter. This method will then show if the groupings produced by partitioning are accurate. If influencing placement with information obtained during HLS improves the design optimality, this will show that the groupings created during partitioning are

optimal groupings and have been predicted accurately, as Xilinx will not necessarily choose the most optimal design.

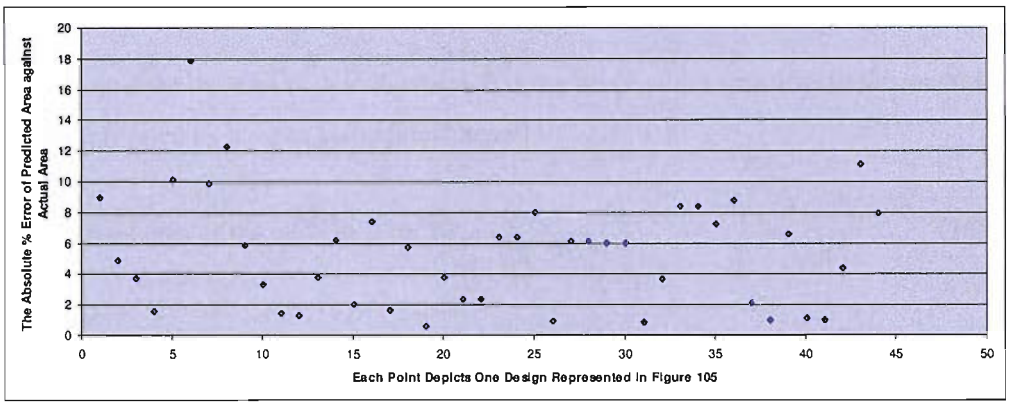


Figure 37 to show the Absolute % Error of the Predicted Area against the Actual Area

3.4 Partitioning Algorithm within MOODS

Partitioning is used to obtain information about where every macro is, relative to every other macro in the circuit. This information will show which macros should be placed close together. When partitioning the design's netlist, the elements are grouped together according to how highly connected they are with their neighbours, so as to reduce the number of longer nets that span the width or length of a circuit.

In this chapter partitioning of a netlist provided by MOODS will be discussed, to obtain physical level information. During the explanation of the methods used to partition a circuit's netlist, macros will be referred to as nodes on a graph, where interconnects of the macros are represented as nets that connect the nodes, the weight of a net is the number of bits that are transferred between macros. When partitioning a netlist the design is separated into multiple disjoint groups.

The two partitioning algorithms that are used are a neighbourhood search (greedy) based algorithm and an iterative improvement algorithm. The latter will never be used by itself as it is too slow, so the cluster based algorithm will be used then further improvement can be applied by the iterative improvement algorithm, if any exists. But first some definitions will be provided that will be used in the remaining part of this chapter.

3.4.1 Definitions Needed for Partitioning

Let ∂_a represent the area of $a \in A$, where A is the set of nodes (macros) in the first sub

partition formed by a cut at hierarchical level l

Let the total area of the node in A be $T\partial_A = \sum_{a \in A} \partial_a$. (15)

Let ∂_b represent the area of $b \in B$, where B is the set of nodes (macros) in the second sub partition formed by a cut at hierarchical level l

Let the total area of the node in B be $T\partial_B = \sum_{b \in B} \partial_b$. (16)

The total area of both sub partitions A and B need to be equal or similar in size, i.e. $T\partial_A \approx T\partial_B$.

Let E_a represent the nets, which start at $a \in A$ and finish in B, i.e. external nets. Let I_a represent the nets, which start at $a \in A$ and finish in A, i.e. internal nets.

When considering moving a node from one sub partition to another in order to reduce the size of the cut set, assurance is needed that when the node is placed in another sub-partition that the number of external nets that are created (if any) are not more than the original number of external nets that existed before the movement. This assurance is so that we do not produce a sub-optimal cutset. Hence knowledge of the benefit the move would provide in reducing the number of external nets of each sub-partition is needed. The value that shall be used is called the D - value, which is given by:

$$D_a = E_a - I_a \quad (17)$$

The smaller the D-value, the less beneficial the move would be. When the D-value is negative the move would be adverse to the overall solution. The reason for this is that if D is negative this means that there are more internal nets than there are external nets. If this node in question was moved to the other sub partition then the external nets would become internal nets and visa versa. Hence now there are more external nets than there are internal nets, meaning the optimality of the solution has been degraded.

When two nodes are swapped $a \in A$ and $b \in B$, the gain produced by this swap is given by

$$g_{ab} = D_a + D_b - 2c_{ab} \quad (18)$$

where c_{ab} represents the sum of weights of the edges which connect a and b. Equation 18 is comprised of the two D values of a and b, but if there are any nets that connect a and b

then these will have no effect on the gain to the overall system as the nets will still remain unchanged after the swap (i.e. both are external). Hence the value of the nets in question needs to be removed from the equation. c_{ab} is multiplied by two as the nets will be included in both D values.

In order to choose the 2 nodes to swap in order to produce the most beneficial effect on the cutset, we choose to swap the nodes which cause the greatest gain value given by

$$\overline{g_{ab}} = \max(g_{ab}), \forall a \in A, \forall b \in B \tag{19}$$

Now we have swapped these nodes, we need to adjust the D-values of all the nodes adjacent to a and b,

$$D_x' = D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\} \tag{20}$$

D_x is the original D value before swapping a into the sub group B. If nodes have nets connecting to a or b, the corresponding D values need to be changed as external nets will become internal nets and visa versa. c_{xa} is the value of the net that was connected to a, where b is now entering the same sub group as x. Hence c_{xa} needs to be multiplied by two and is positive as first we need to remove the effect of the net as an internal net, then we need to add the effect of the net now being an external net. c_{xb} is the value of the net that is connected to b, where b is now entering the same sub group as x. Hence c_{xb} needs to be multiplied by two and is negative as first we need to remove the effect of the net as an external net, then we need to add the effect of the net now being an internal net. The approach just explained is the same for Equation 20.

$$D_y' = D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\} \tag{21}$$

If $a \in A$ is being swapped with $b_1,..,b_n \in B$ the gain value of this swap would be

$$g_{ab_1,b_2,...,b_n} = \overline{g_{ab_1,b_2,...,b_{n-1}}} - 2c_{ab_n} + 2\sum_{i=1}^{n-1} c_{b_i b_n} \tag{22}$$

where b_n is the latest node to be considered. The maximum gain is given by

$$\overline{g_{ab_1,..,b_n}} = \max(g_{ab_1,...,b_n}), \forall a \in A, \forall b_1,..,b_n \in B, n \le |B| \tag{23}$$

The difference in A and B areas is given by

$$\varepsilon = |T\partial_A - T\partial_B|. \tag{24}$$

To ensure that ε is not too great an upper and lower bound to ε are used to keep the difference in total areas of A and B satisfactory.

Let ℓ_f represents the lower bound of ε and is given by

$$\ell_f = -\left(\frac{\sum_{z \in G} \partial_z}{2|G|}\right), \text{ where } |G| \text{ is the cardinality of } G. \qquad (25)$$

ℓ_f was found experimentally and is only an approximate bound. But it was found to provide enough flexibility for the partitioning to reach an optimal solution, without the sub groups being too different in size.

Let ℓ_t represents the higher bound of ε and is given by

$$\ell_t = \frac{\sum_{z \in G} \partial_z}{2|G|} . \qquad (26)$$

Hence $\ell_f < \varepsilon < \ell_t$.

The definitions in this section have provided all the tools that are needed for the algorithms presented in section 3.4.2 and 3.4.3. The definitions are mainly used to decide whether swapping a node to an alternative sub group is advantageous. The remaining definitions are then concerned with keeping the two sub-groups equal.

3.4.2 Greedy Algorithm

The first partitioning algorithm, which is going to be used for partitioning a netlist, is the Greedy (Gr) Algorithm, modified from [27]. The algorithm was designed for speed and ease of implementation. The algorithm is outlined in figure 38, and a practical example is shown in Figures 39 and 40. Figure 39 shows a graph of 14 nodes that represent macros with unit area. The graph is desired to split into two distinct sub graphs, using the greedy algorithm as described in Figure 38. Figure 40 shows the process of partitioning the graph using initial node 1 and cut *A* shows the resulting partition. Now an outline will be given explaining how the partition forming cut *A* in Figure 39 is derived. Let node 1 be chosen to be the initial seed, hence node 1 is placed in set *A*, and every other node is in set *B*. Next we need to choose the most highly connected node to set *A*. The nodes in set *B* that are connected to nodes inside *A* are deemed to be connected to *A* (hence external nets), and we wish these nets to be minimised in order to reduce the cut set.

Greedy Partitioning Algorithm

1. Find ε (Eqn 26) and mid point(*mp*) of the area in *G* i.e. $(\sum_{z\in G}\partial_z + 1)/2$
2. If a node $n \in G$ is greater than *mp* enter *n* into *A* and go to 7
3. Choose node $n \in G$ to enter *A*, let n_A represent the last node to enter *A*
4. Find D values of all the neighbours of n_A, and place them in a vector *V*
5. Order *V* in descending order according to *D* values
6. While *V* is empty go to 3
7. Let n_i = first element of *V* and removed from *V*

   If $(mp + \ell_f \le T\partial_A + \partial_{n_i} \le mp + \ell_t)$

   n_i enters *A* and $n_A = n_i$ Go to 8

   Else If $(mp + \varepsilon < T\partial_A + \partial_n)$

   n_i enters *B*

   Go to 6

   Else

   n_i enters *A* and $n_A = n_i$

   Go to 4
8. If the *D* value of n_i is greater than 0

   Go to 4

   Else

   Go to 9
9. Nodes which have not entered *A* are placed in *B* and End of Algorithm

Figure 38. Constructive Circuit Partitioning Algorithm

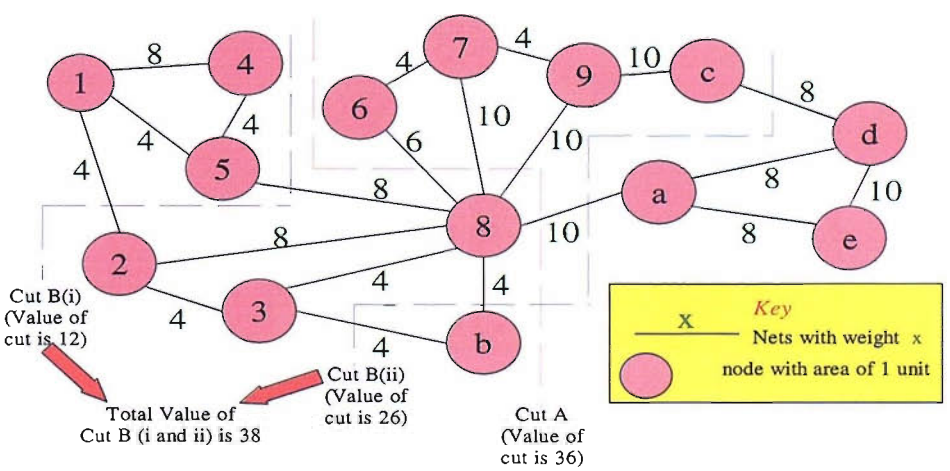


Figure 39 Representing the Effect on the cutset Depending on the Choice of Initial Node

The next step is to find a node in B that if brought over to *A* will produce the smallest cut set than if any other node was placed in *A*. To make this choice we find the *D* value of all

the nodes that are connected to *A*. If there is no node that resides in *B* and is connected to *A*, then we just chose the node with the lowest number of weight of nets. The *D* values that are in the "Respective *D* Values" column of Figure 40 are calculated as follows:

Node 4, 5 and 2 are connected to *A* through node 1. Node 4 has one external net that starts in *B* and ends in *A*, namely the net that connects node 4 with node 1. Node 4 also has one internal net that starts in *B* and finishes in *B*, namely the net that connects node 4 to node 5.

$$\Rightarrow D_4 = E_4 - I_4 = 8 - 4 = 4$$

(i)

A	V	
	Node Numbers connected to *A*	Respective *D* Values $D_a = E_a - I_a$
1	4 5 2	4, -8, -8
1 4	5 2	0, -8
1 4 5	2 8	-8, -44
1 4 5 2	8 3	-28, -4
1 4 5 2 3	8 b	-20, 0
1 4 5 2 3 b	8	-12
1 4 5 2 3 b 8	Reached Mid Point	

(ii)

A	V	
	Node Numbers connected to *A*	Respective *D* Values $D_a = E_a - I_a$
8	2 3 5 6 7 9 a b	0, -4, 0, 2, 2, -2, -6, 0
8 6	2 3 5 7 9 a b	0, -4, 0, 10, -2, -6, 0
8 6 7	2 3 5 9 a b	0, -4, 0, 6, -6, 0
8 6 7 9	2 3 5 a b c	0, -4, 0, -6, 0, 2
8 6 7 9 c	2 3 5 a b d	0, -4, 0, -6, 0, 0, -10
8 6 7 9 c 2	3 5 a b d 1	4, 0, -6, 0, -10, -8
8 6 7 9 c 2 3	Reached Mid Point	

Initial node is 1 resulting in cut A in figure 39

Initial node is 8 resulting in cut B in figure 39

Figure 40. Showing the Derivation of Sub Groups Formed by a Partition Using the Greedy Algorithm (figure 38) with Different Initial Nodes.

The same method for obtaining the *D* value for node 4 is used for obtaining the *D* value

for nodes 5 and 2;

$$\Rightarrow D_5 = E_5 - I_5 = 4 - (4+8) = -8$$

$$\Rightarrow D_2 = E_2 - I_2 = 4 - (4+8) = -8$$

So node 4 has the most positive D value, hence is chosen to join node 1 in set A. Now all the nodes that are connected to node 4 are now connected to A. The D Values are now calculated for the newly connected nodes to A, and the already connected nodes if they were connected to node 4. The next node will then be chosen depending on which node has the highest D value, and so on until the area of A is within an acceptable range of the mid-point of the total area of A and B.

3.4.3 Key Improvements to the Greedy Algorithm

Initially the algorithm did not use D values when considering the next node to enter A. Only the nets incident with A were considered. This could cause a bad choice when selecting the next node to enter A, owing to the chosen node having a large number of nets not already in A. These nets then result in a high weighting being added to the cutset. This problem becomes especially important during the latter phase of the algorithm. The first improvement to the initial algorithm was the ordering of the Neighbourhood Vector (V) in decreasing order using the Quicksort algorithm [91] for sorting and Select algorithm [91] for selecting a good pivot. The Quicksort algorithm requires O($n$log$n$) comparisons, where n is the total number of nodes. Hence the first node will always be the best candidate for entering into A, this means the complete list of nodes in V does not need to be looked at to find the best candidate, therefore reducing the run time. Further explanation can be found in [19]. This causes another problem because re-ordering the whole of V every time would be highly undesirable. Firstly, all those D values that have not been changed do not have to be re-ordered. So only order the nodes that have been affected by the latest entry into A, which might include nodes from $V \cap B$, then merge this list with the existing nodes in V. The technique of the merge requires in the worst case n-1 comparisons and at best $n/2$ comparisons. This is a very efficient technique due to the fact that every time a new/changed node is inserted into V, the whole vector does not need to be ordered again - only a fraction of V. For the method to be efficient, the number of nodes in the vector to be merged needs to satisfy the constraint:

$$m \log m \; 1 < (\log n) \qquad (27)$$

where m is the number of nodes in the vector which is joining V, and n is the size of V. Hence the smaller m is, the more efficient the sorting method will be. If the constraint is not satisfied then we use Quicksort and Select algorithm as before for all the nodes, but this occasion will seldom occur.

3.4.3.1 Data Structure Improvement

When very large netlists were processed, a problem was revealed due to considerable reduction in speed of the execution of the program. There is a repetition of most of the numbers in the set, so when ordering this vector the searching process had to look at multiple nodes with the same D value, this is illustrated in Figure 41, where x represents a node in V and $D[x]$ is equal to the D value of x. So the V vector would be as follows with the node that has the largest D-value first, then descending in value.

V : { 555, 56, 221, 1999, 46, 765, 64, 256, 141, 1566, 232, 356, 799, 999, 567, 679, 888, 1788, 1222, 1445, 987, 777, 675, 23, 1764, 1111, 123, 432, 901, 1301, 32, 451, 987, 741, 158, 469, 816, 1725, 1425, 247, 453, 324, 874,.....}

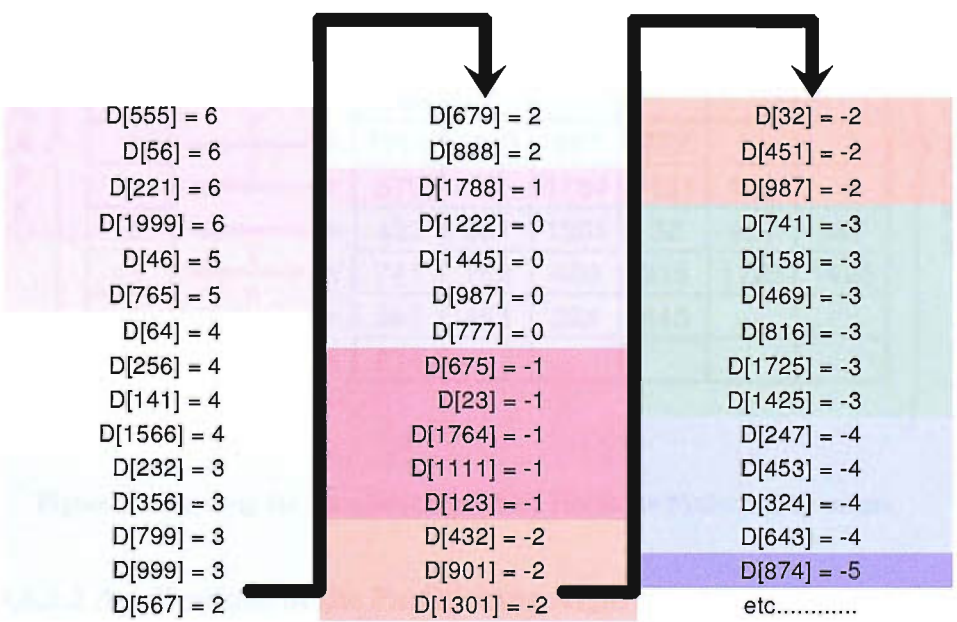


Figure 41. Diagram Representing the Elements in V in Descending Order.

Now if the next node to be added has a D value of –3, assuming the search starts at the beginning of the set, then 34 comparisons need to be made before the correct location is found for this node. To prevent this from happening the type of storage arrangement was changed. Instead of storing the node numbers in the V vector, the D values were stored in

the vector, which would look as follows:

V : {6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, ...}

These *D* values then point to other vectors for each respective node, which has that particular *D* value as shown in Figure 42. So now if the next node with *D* value to enter is –3, there are now only 9 comparisons. This data structure means inserting or deleting nodes is a lot faster. When inserting a node, if the vector pointed to by that node's *D* value is non-empty, no further searching needs to be done on where to place the node's *D* value in *V*. If deleting a node, only the vector, which is pointed to by the old *D* value, needs to be searched in order to delete the relevant node.

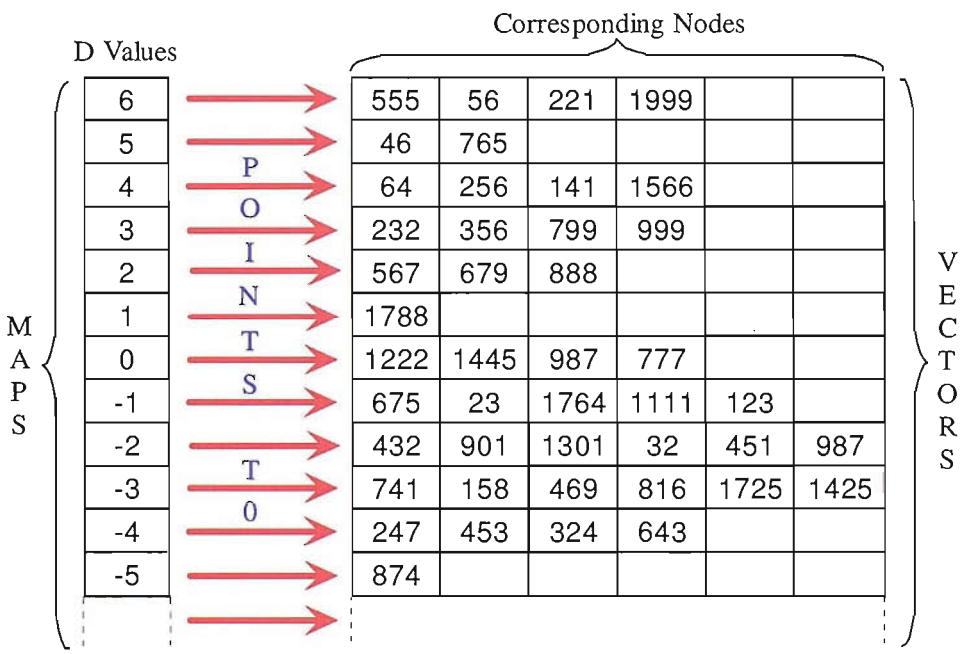


Figure 42. Showing the Data Structure which Holds the Nodes and *D* values.

3.4.3.2 Applications of the Partitioning Algorithm

The Greedy algorithm can be used in two ways. Firstly the greedy algorithm can be used for speed, i.e. just pick a random starting point run the algorithm through once and use the result to partition the circuit. The problem with this methodology is that the results are very dependent on the initial partition. The results can and most often change depending on the starting node chosen. This behaviour is demonstrated in Figures 39 and 40. For example Cut A in Figure 39 has a value of 36, when initial node 1 is chosen. If the initial node was 8, which is in the centre of the graph, the resulting cut is shown as cut B in

Figure 39, and the value of the cut is 38, the derivation of this cut is shown in Figure 40. The reason the cut has a higher value is that it has expanded from the centre fairly evenly to the left and the right, which has caused a graph to be split into 3 distinct (disjoint) groups. This increases the size of the cut, due to forming a bottle neck, where nets have to pass through a small number of nodes before reaching the remaining nodes in the graph. Node 8 is a bridge between the left and right of the group. Hence avoidance of bottlenecks is preferable; as this will prevent a bridge, hence reduce the cut set.

3.4.3.3 Seed Selection

Obviously it would be desirable to choose the starting position that would yield the best result. Hence a fast heuristic is desired to choose a good initial node, which will produce an optimal partition.

The different criteria when choosing the initial node are

1. area
2. max weight
3. random

Two methods, are used to choose the best seed for the cut-set minimisation problem. Firstly choose the module with the largest area. This has more chance in reducing the run time of the algorithm because it will be closer to the mid point of the total area, when the algorithm will stop once the mid point has been reached. Hence in the majority of cases, choosing the node with the largest area will speed up the algorithm.

Secondly choose the module that has the highest total sum of interconnect weights: the more neighbours that are considered when choosing the best solution the more chance the most optimum partition will be obtained. Also, if the weighting is high then it is probably best to make sure that it is placed with all its neighbours to reduce the cost of the system under scrutiny. If two or more nodes have an equal area, then out of those nodes, whichever node has the highest connectivity can be chosen as the seed. If this time two or more nodes have the same connectivity, the initial seed can be chosen at random out of the nodes selected that have the highest connectivity and largest area.

As an alternative to selecting a single seed, multiple initial seeds can be selected on separate partitions. The partition that produces the smallest cutset is chosen to be the partition of the circuit. The seeds can be chosen by an exhaustive search or alternatively

the seed can be picked at random throughout the set for an allocated number of times.

Modified Kernighan Lin Partitioning Algorithm

1. Calculate error value ε
2. Create the partition the group into two distinct sub groups A and B, which satisfies :

$$\varepsilon \le \left| \frac{\sum_{z \in G} \partial_z}{2|G|} \right|$$

$$A \cup B = \phi$$

$$A \cap B = G \quad \text{and}$$

3. Set GL = 0 and find all values of D_a for $a \in A$ and D_b for $b \in B$
4. Order the nodes in containers (as shown in figure 9)
5. Find $\overline{g_{ab}}$ and Δ (all marked nodes for swapping are ignored)

   If $((\overline{g_{ab}}) \le GL)$ Then

      If $(D_a > 0)$ Then

         Replace b with a Dummy node with 0 degree and 0 area without loss of generality

         Find $\overline{g_{ab}}$ and Δ (all marked nodes for swapping are ignored)

      Else if $(D_b > 0)$ Then

         Replace a with a Dummy node with 0 degree and 0 area without loss of generality

         Find $\overline{g_{ab}}$ and Δ (all marked nodes for swapping are ignored)

   If $(\overline{g_{ab}} \le GL)$ Then

      Go to 9

   Else

      Go to 6

6. Calculate

   $\ell_b = \ell_b + \varepsilon$

   $\ell_t = \ell_t + \varepsilon$

   If $((\overline{g_{ab}} > GL) And (\Delta = 0)) Or ((\ell_b \le \Delta \le \ell_t) And (\overline{g_{ab}} > GL))$ Then

      Mark a and b for swapping. Then

      Go to 8

   Else

      Go to 7

7. Find $\overline{g_{ab_1 \ldots b_n}}$, where b_n is the latest node to be added (all marked nodes for swapping are ignored)

   If $((\ell_b \le \Delta \le \ell_t) And (\overline{g_{ab_1 \ldots b_n}} > GL))$ Then

      Identify b_n for swapping

      Go to 7

   Else if $((\ell_b \ge \Delta) And (\overline{g_{ab_1 \ldots b_n}} > GL))$ Then

      Remove b_n from consideration Then

      Go to 7

   Else if $((\ell_b > \varepsilon) Or (\overline{g_{ab_1 \ldots b_n}} \le GL))$ Then

      Change D_a to –9999, so the node is not chosen again Then

      Go to 5

   Else

      Go to 7

8. Derive new D values using D_x And D_y for the Neighbours of the nodes to be swapped and re-order the nodes and corresponding D values, Then go to 5
9. If (any candidate nodes for swapping) Then Swap nodes and re-order the nodes and corresponding D values, Then Go to 5 Else Go to 10
10. End Of Algorithm

Figure 43. Iterative Improvement Circuit Partitioning Algorithm

As speed is the main consideration the initial seed will be chosen based on which node has the largest area. Generally this node will have the largest number of nets as well (except if a multiplier). The more times the algorithm is run the longer it takes to obtain the final solution, but with an increased chance in obtaining an optimal solution or close to it.

3.4.4 Modified Kernighan Lin Algorithm (MKL)

The KL algorithm presented in [19] was used as a comparison in order to see how results differ when using different partitioning algorithms. KL will always have as good if not better cut set than the Greedy algorithm as it improves on the Greedy algorithm's cut set. KL does not consider area when partitioning the circuit. But an algorithm, which considers area and net cost is needed. When partitioning the circuit each sub-partition should be approximately of equal size. The algorithm that has been developed to fulfil these criteria is outlined in Figure 43, using definitions from section 2.2. Consider Figure 39; when the initial node is 8, the resulting cut set is given by cut *B*. This cut can be further improved by applying the MKL to cut *B*, as shown in Figure 44. This solution now outperforms both cuts produced by the Greedy algorithm.

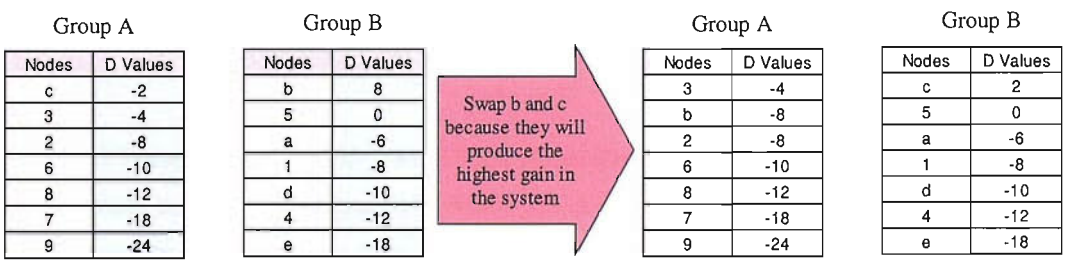


Figure 44. Iterative Improvement using MKL of the Solution in figure 42.

3.4.5 Key Improvements to the Kernighan Lin Algorithm

To decrease run time, the methods given in [22] have also implemented in this algorithm, to make the complexity comparable with Fiduccia Mattheyses algorithm [3]. KL algorithm was chosen rather than FM, as a partitioning algorithm that swaps two nodes at a time provides a better solution than an algorithm that only moves one node at a time between two sub-sets. The reason such a good solution is required is to see whether a better partitioning method (better the cut-set) for each respective hierarchical level results

in more accurate interconnect prediction, hence producing better designs when using interconnect prediction. There is a similar problem to that of the greedy algorithm, in that KL can settle into a local minimum and the partition is dependent on the initial partition of the circuit. The greedy algorithm may be used to improve the initial partition. This approach gives a very good initial partition, which the iterative improvement algorithms can enhance [41]. Due to the fact that the initial partition will be ordered, the number of runs of the algorithm should drastically be reduced. The drawback to this methodology is that the greedy algorithm might have too much influence over the KL and force the partition into a local minimum, which is not the best result achievable if the Greedy Algorithm were not used. But having the Greedy Algorithm produce the initial cut should stabilise the process of partitioning the circuit, as the end partitioning will be dependent on the initial cut. Due to the resulting partitions being a basis for all the predicted physical information, this stability is very useful during design exploration, because different architectures for a design can be compared fairly. Another benefit of running the greedy algorithm first is that some of the D values will have already been calculated, namely the ones in set A and in V. The D values of the remaining nodes that are in B but not in V, can be calculated just by summing all the Input/Output nets of each respective node. The destination of the nets need not be searched; as when a node enters A it brings all its neighbours that are not already in A into V. Hence if a node is not in V then a node in B cannot be connected to any node in A, hence all nets must be internal.

Alpert and Kahng show that the FM (which is similar to KL) cutset results follow an approximately normal distribution [41], where the average results are significantly worse than the best results. Again multiple runs of the KL algorithm could be made, but the algorithm is required to be fast, so that many different architectures for a design can be compared. Hence the number of iterations of KL needs to be kept to a minimum.

3.5 Recursive Partitioning and Representation

Whether passing on the hierarchical grouping information to an APR tool or providing a floorplan, the relative placement/groupings of all the components needs to be known. To obtain this information the circuit needs to be recursively partitioned until all sub-partitions contain only one module. At this point, all the relative positions will be known, so formation of a slicing tree representing of all the components can be derived, forming the foundations for the floorplan. The slicing tree is formed during the recursive partitioning. A binary tree is shown in Figure 45, which is a simple representation of a slicing tree. Every time the circuit is partitioned, all the nets that cross the partition

become redundant as far as partitioning is concerned, so need to be removed in order not to influence the partitions further down the hierarchy.

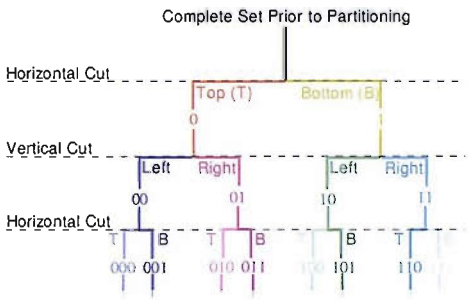


Figure 45. Formation of a Slicing Tree Represented as a Binary Tree.

Hence when partitioning it would be useful to have some way of deciding on which side of the partition the modules are placed. When the redundant nets are removed the weights of the nets are stored within each sub-partition, so that the values are accessible when producing the floorplan.

The nets are used to decide on which side a partition should be placed, enabling quick decisions to be undertaken, while providing a valuable metric to reduce interconnect length, as the modules with the highest interconnects should be placed as close together as possible. The process is explained in more detail in chapter 4, as these values have no effect during the recursive partitioning phase, but will help reduce terminal propagation.

3.6 Average Interconnect Relationship

In section 2.11, the Rent's Rule relationship was chosen to estimate a circuit's average interconnect length. The relationship can be used to represent the interconnect complexity of the circuit, which is then used to obtain an estimate of the average interconnect of a circuit. The average interconnect of a circuit depends on two main characteristics: the size of the circuit being partitioned and how complex the interconnect topology is. The size of the circuit is trivial but the second property is a much harder factor to characterise. As stated earlier, the Rent exponent represents this interconnect complexity. So a method that can find the Rent exponent of a circuit needs to be used that can account for the non-homogeneous nature of a circuit's netlist. To represent the non-homogeneous nature of a circuit the different levels of the hierarchical tree are used to derive the Rent exponent. Hence methods for obtaining the Rent exponent in this manner shall now be discussed.

3.7 Methods for Obtaining the Rent Exponent

The Rent Exponent can be obtained before or after a circuit is placed. Another extraction procedure for the Rent Exponent uses previously built designs, but that is more beneficial for verification, although it is shown to be better than the pre-placement extraction in [30]. For HLS, the Rent Exponent of a design's architecture needs to be obtained before the design has been implemented in hardware, so only pre-placement Rent Exponent Extraction will be discussed.

3.7.1 Pre-Placement Rent Exponent Extraction

There are two types of ways of finding the Rent Exponent: either by flat (k-way) partitioning [31], or hierarchical partitioning [112] [75] (Recursive Bi-Partitioning). As Recursive Bi-Partitioning (RBP) has been chosen to partition a circuit's netlist, the Rent exponent of a design architecture will be extracted using hierarchical partitioning.

Figure 46 shows the Rent exponent, p, being extracted using RBP. The circuit is partitioned into 4 subcircuits. This is achieved by Bi-Partitioning (BP) a group, then BP the two sub-groups formed by the previous partition. The average number of cells and the average number of external nets for all subcircuits are recorded and used to calculate p. Hence a hierarchical structure is formed, where each level consists of 4 subcircuits (of equal size) at the next (lower) level of hierarchy. 4 sub circuits are used as at this point as there are only three distinct directions an external net can take, for example if an external net starts in the lower left quadrant then it can only go to the left, right or diagonal direction, this simplifies analysis, and allows estimation of the behaviour of the external nets to be performed. If there were more sub-groups the complexity of the problem would dramatically increase. A common way of achieving a four way partitioning is to Bi-partition the circuit twice, there is a methodology that has been formulated using the assumption that the circuit is being Bi-Partitioned. The sub groups formed from partitioning the circuit shall be called bins. Later on when constructing the floorplan, these bins will have a dimension in which the macros inside the bins can be located. Every time the circuit is partitioned, each bin gets split up into 4 parts, until each cell is assigned to a single grid point in a Manhattan Grid as shown in Figure 47 (i) and (ii). The gridlines in Figure 47 (ii) correspond to routing channels and each grid point (node) represents where one logic block (this can be a macro, group of macros or even an entire circuit) can be placed.

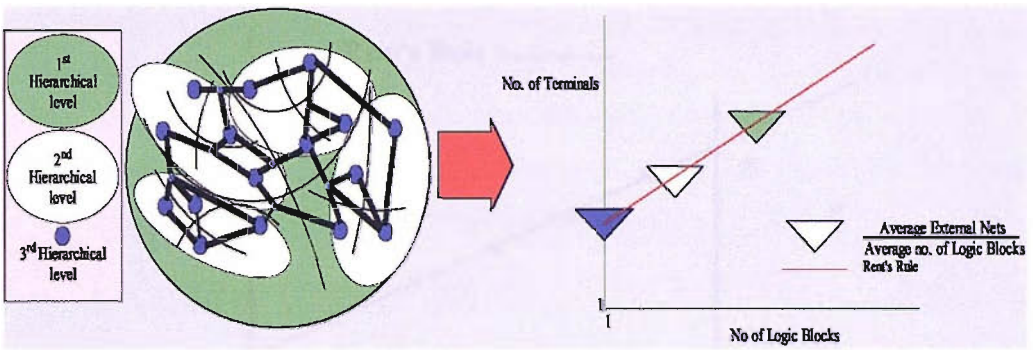


Figure 46. Extraction of Rent Exponent Using Circuit Partitioning.

The circuit is split in this manner, so that the number of cells in the circuit is a power of 4 (when split k times the circuit has $4k$ cells at the k^{th} hierarchical level). This is used to derive the hierarchical-based average interconnect length equation. Hence it is a condition that needs to be satisfied when partitioning the circuit. The average interconnect equations from hierarchical partitioning can also be derived presuming that the circuit is bi-partitioned twice in order to form these 4 subcircuits. Equations formulated in this way should provide the best results, as the partitioning methodology being used is recursive bipartitioning. These equations will be discussed further in the next section but first obtaining p will be examined.

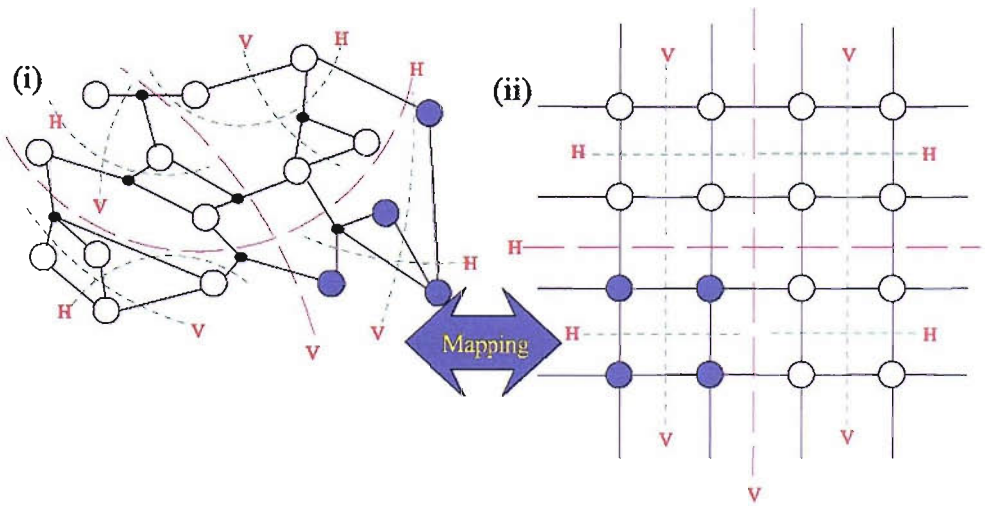


Figure 47. Transformation from a Topological Graph of a Design to a Manhattan Grid.

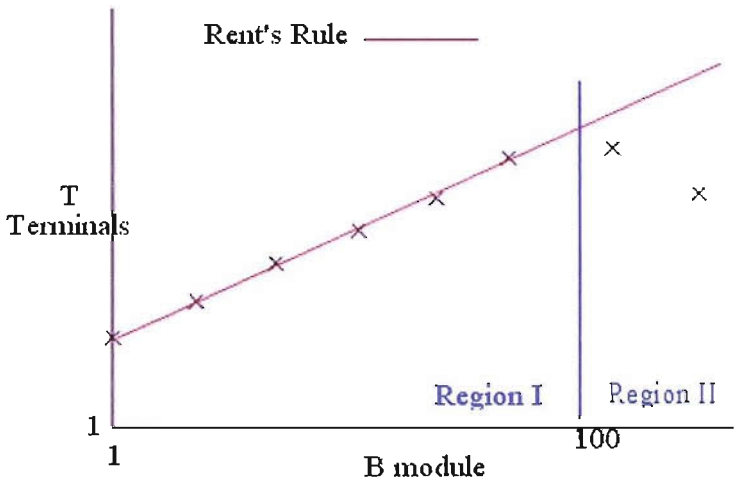


Figure 48. Graph Showing Log-Log Plot of the Average Number of Terminals (or pins) *T* of a Part of a Circuit (a module) and the Average Number of Logic Gates (Basic Logic Blocks *B*) Inside the Modules.

Rent's Exponent *(p)* is computed by plotting the average number of terminals, *T*, of a part of a bin versus the average Basic Logic Blocks, *B*, inside the module relation in a log-log plot (Figure 48), during a top-down partitioning process. Figure 48 shows the relationship, which is used to extract the Rent Exponent for an ISCAS89 benchmark 's953' used in [12]. The graph's points stop following the trend at 100 gates per sub-circuit in region II. The reason for this is that as the number of gates approaches the total number of gates on the chip, the number of terminals becomes constrained by the limited number of input/output terminals at the edge of the chip. This results in a rapid decrease in the number of estimated terminals [95]. A best line is placed through the plotted points (this is found using least squares, Appendix 7.3); the gradient of this line is the Rent exponent (p). The Rent exponent requires several levels of partitioning in order to obtain enough points to achieve reasonable accuracy, and this process will be shown in the next section with an example of a Rent Exponent extraction from Benchmark2 (Appendix 7.6).

3.7.2 Actual Extraction of the Rent Exponent

To obtain the Rent Exponent, we shall be using a Bi-Partitioning approach taking a reading of the number of external nets cut at each hierarchical level. This approach must be taken, as in general there are not enough levels of hierarchy if only external nets caused at every fourth hierarchical level are used to calculate the Rent Exponent, in order

to provide enough points for reasonable accuracy for least squares. Hence Figure 49 shows a hierarchical structure formed from Bi-Partitioning. This should not affect the calculation of the Rent exponent due to 4 way partitioning in general being formed from 2 way partitioning as stated earlier. But this assumption will be tested in Section 3.8 when the application of the Rent Exponent is introduced.

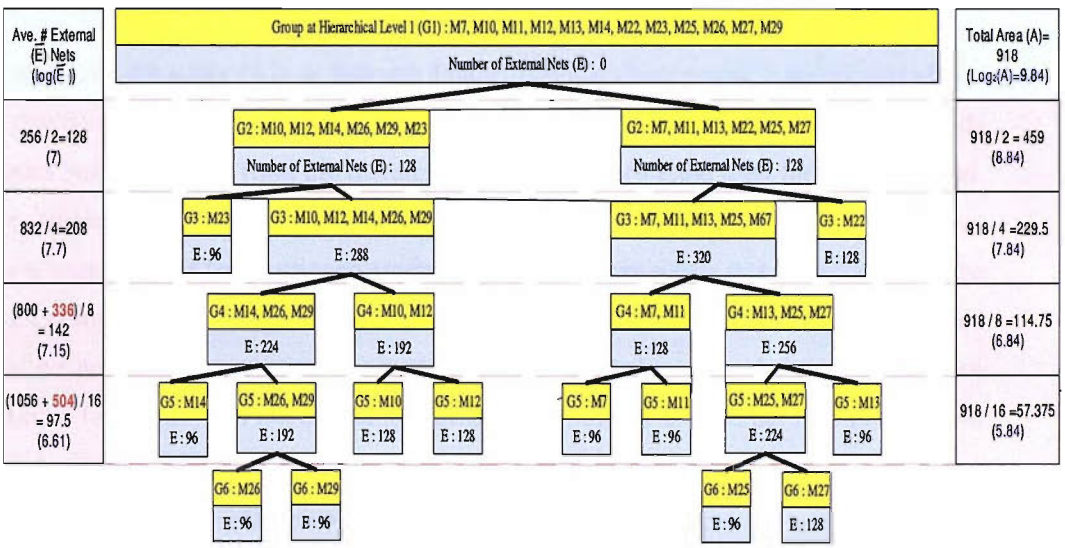


Figure 49. Benchmark2 Recursively Bi-Partitioned, with all Relevant Information Needed to Obtain the Rent Exponent. Where the criterion of the partitioning algorithm is to minimise the cutset. The weight of each net is the bit width of the respective net.

Instead of using the number of components in each group as stated in section 3.3.1, the total of each group shall be used. This will give a fairer representation because sometimes there are large differences between the total areas of the groups at a certain hierarchical level. i.e. if one group, say *'A'*, has one module with an area of 500 and the other group, say *'B'*, has 10 modules with a total area of 250, and if we were to use the number of modules it would look as if group *B* is 10 times larger, which would be an unfair comparison as in reality group *A* is twice the size of group *B*. The average area for each group will be calculated as follows:

$$\text{Average Area at Hierarchical Level } l = \frac{TotalArea}{2^{l-1}} \qquad (28)$$

Let Q_l represent the total number of external nets belonging to groups that have cardinality of one. In Figure 49 this value is represented by the numbers in bold.

$$\text{Average \# of External Nets at } l = \frac{Total\#\,ExternalNets + 1.5\sum_{r=1}^{l-1} Q_r}{2^{l-1}} \qquad (29)$$

The reasoning for using Q_l is as follows; Figure 49 shows benchmark 2 partitioned all the way down until each sub-group only contains one node, but if the tree is uneven this leads to nodes being removed from the hierarchical structure before the final hierarchical level. For example at hierarchical level 3, M22 and M23 are isolated in their own respective groups, hence cannot be partitioned anymore. But the external nets that are supplied to the overall system cannot be ignored. An estimate of the extra number of external nets that would be produced by splitting this node further is needed, so that the external nets are accounted for. If they are not accounted for then the number of external nets lower down in the hierarchy would appear to be lower than they actually were, hence producing an unrealistic relation. Q_l is calculated as follows:

The 1.5 constant is experimentally validated in the next section, when the average interconnect equations are introduced and the AI values derived are then compared to the actual average interconnect length post PAR. The external nets are summed from one hierarchical level to the next. This is different to how the Rent Exponent is normally calculated. But it was found that summing the external nets produced a better correlation on average compared to not summing the external nets. This is again shown in the next section when the Average Interconnect Length Equations are discussed.

Once the bottom of the tree has been reached we can then fit a line of best fit using least squares, the very bottom of the tree is not used so as to reduce the influence of the estimated nets formed after a macro has reached the bottom of a branch. The logarithm values in Figure 49 can be seen on the graph for the benchmark 2 in Figure 50. The gradient is then obtained from the line of best fit, and hence p has been calculated and is 0.64. This is also a graph for Matrix 2 benchmark to show a graph with a larger data series. The number of coordinates used when deriving the linear model depends on the size of the design. The more points used the more reliable the test and solution. Hence the number of data points is equal to:

(log(total number of nodes of the complete graph / log(2)) - 1, as this is the value that

represents the number of bi-partitions needed to split the graph until each node resides by itself in a sub group. The value is decremented by one so as to make sure that the data sample is inside Region I.

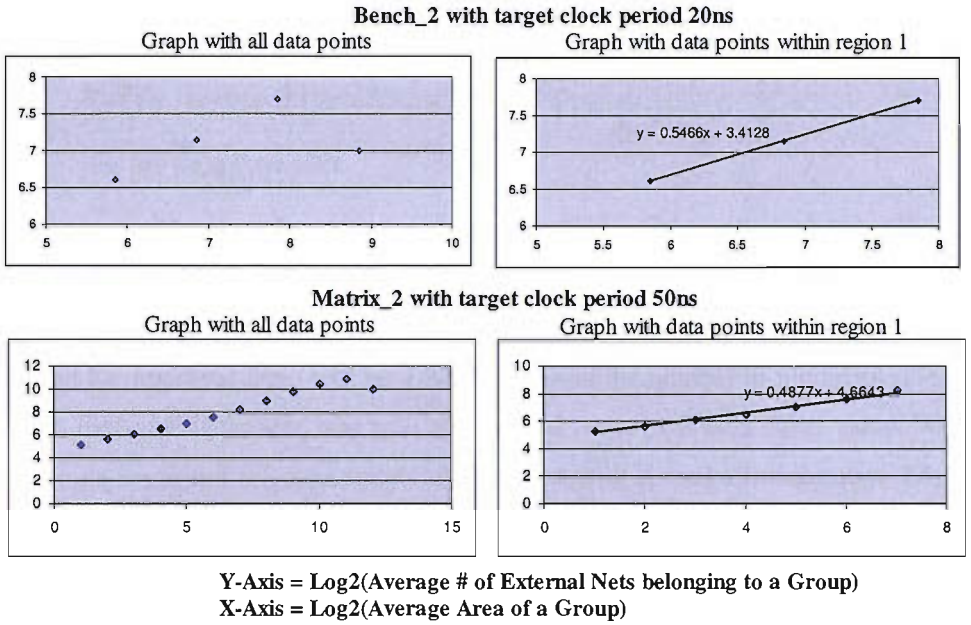


Figure 50. Graph Showing Log-Log Plot of the Average Total Area of Each Group Verses the Average Number of External Nets of Each Group at Each Hierarchical Level from Figure 49.

3.8 Average Interconnect Length Equations

The Rent exponent can now be used to estimate the interconnect lengths ranging from individual interconnects to the average interconnect of an entire circuit. The first interconnect length characteristic we will consider is the average interconnect length (AI) of a circuit. There are two types of models for predicting Wire Length, one is based on hierarchical placement and the other on flat placement. As discussed earlier hierarchical placement is formed from recursive bi-partitioning, while flat placement is formed from one multi-way partition. The following equations are used to predict the average interconnect of a design once implemented in hardware. When deriving Equation 3, it is assumed that, at each hierarchical level, each group is partitioned into 4 parts, as in section 3.3.1.

Donath's Average Interconnect Equation

Donath's Average Interconnect Length Equation [12] is given by:

$$L_{avg} = \frac{14H(K,p,1) - 2H(K,p,3)}{9H(K,p,2)} \tag{30}$$

where $$H(K,p,x) = \frac{2^{K(2p-x)} - 1}{2^{2p-x} - 1}$$

K is the number of Hierarchical Levels.

p is the Rent Exponent.

As shown in Figure 49, a partitioned circuit might not be split evenly, so to use the number of hierarchical levels produced while partitioning would cause an unfair representation, because if the tree is uneven this will cause more hierarchical levels. Instead K= $\log_e$(total area) will be used to represent the number of hierarchical levels, as this is the value if the tree was split evenly, and hence will be a fairer value. When the equations are tested for their suitability for predicting Average Interconnect, Log to the base 2 will also be tested as Bi Partitioning is being used to partition the design.

Davis's Model Average Interconnect

Equation Davis's Model [14] is given by:

$$L_{avg} = \frac{\frac{p-0.5}{p} - \sqrt{N} - \frac{p-0.5}{6\sqrt{N}(p+0.5)} + N^{p}\left(\frac{-p-1+4^{p-0.5}}{2(p+0.5)p(p-1)}\right)}{N^{p-0.5}\frac{-2p-1+2^{2p-1}}{2p(p-1)(2p-3)} - \frac{p-0.5}{6p\sqrt{N}} + 1 - \frac{(p-0.5)\sqrt{N}}{p-1}} \tag{31}$$

This equation is based on a flat placement, where N is the total number of nodes in the circuit. N is equal to the total area, using the same argument for K when used in conjunction with Equation 30.

An extension to Equation 30 is also found in [12] and is given by:

$R_d(p) = \left(\frac{2p-3}{2p+1}\right)\left(\frac{4^{2p} - 3^{2p+1} + 32^{2p} - 1}{4^{2p-1} - 3^{2p} + 32^{2p-1} - 1}\right)$, where R_d represents interconnects travelling along the diagonal.

If (p>0.5) Then

$R_a(p) = \left(\frac{2p-3}{2p+1}\right)\left(\frac{3^{2p-1} - (2p+7)2^{2p} + (4p+5)}{3^{2p} - (p+3)2^{2p} + (4p+3)}\right)$, where R_a represents interconnects travelling in a vertical or horizontal direction.

$$L_{avg} = \left(\frac{4R_a(p)+2R_d(p)}{6}\right)\left(\frac{H(K,p,1)}{H(K,p,2)}\right) \quad (32a)$$

Else

$$L_{avg} = \left(\frac{R_d(p)}{3}\right)\left(\frac{H(K,p,1)}{H(K,p,2)}\right) \quad (32b)$$

Where

$$H(K,p,x) = \frac{2^{K(2p-x)}-1}{2^{2p-x}-1}$$

Bi-Partitioning Average Interconnect Equation [12]

The final average interconnect equation presented, when derived, takes into account bi-partitioning, rather than 4-way partitioning. This is the same method used to extract the Rent Exponent, and the equation is given by:

$$L_{avg} = \left(\frac{16+2^{p+1}-5.2^{2p}}{6(2-2^{p-1})}\right)\frac{H(K,p,1)}{H(K,p,2)} - \left(\frac{4-2^{p}-2^{2p-1}}{6(2-2^{p-1})}\right)\frac{H(K,p,3)}{H(K,p,2)} \quad (33)$$

Equations 30-33 show that the only values needed are the Rent exponent and the total area of components in the circuit. Both the Rent Exponent *(p)* and area can be obtained after RBP, which means after RBP the average interconnect length of a circuit can be calculated simply by inputting p and K= $\log_4$(total area) into the derived interconnect equation. So when RBP is performed on the circuit netlist within MOODS during design space exploration, the average interconnect length can then be obtained and used during HLS to aid design space exploration, in terms of minimising the detrimental effect of routing by choosing a design architecture with a relatively low average interconnect length with respect to other design architectures. To obtain the most accurate predicted AI value, the assumed conditions used to derive the model should be the same as the actual conditions when obtaining the Rent Exponent. When obtaining p through partitioning the netlist, an average Wire Length equation based on hierarchical placement models (Equations 30, 32 and 33) is preferred, rather than a flat placement model (Equation 31), as shown in [30] and [31]. The average interconnect equations will now be tested to see which equation offers the best power of prediction of a design's average interconnect length when placed on an FPGA.

The values are calculated as follows:

Each benchmark was synthesised in MOODS with the design criteria in Figure 51, where 1

is the highest and 2 is the lowest priority. The clock period always has priority 1. 17 different benchmarks were used, which makes a total of 204 data points in the series. The average correlation is the correlation for each benchmark summed then divided by 17. The value will show which equation and partitioning set up produced the most optimal average interconnect predictor. Figures 52-54 are tables to show the Average Pearson's Correlation Coefficient, between the predicted Average Interconnect post MOODS, and the actual average interconnect post PAR in Xilinx. Y represents the different factors that are used to multiply the macros external nets once the macro has reached the bottom of a branch and can no longer be partitioned.

Area Target	Area Priority	Delay Target	Delay Priority	CP Target
1	1	1	1	5
1	1	1	1	10
1	1	1	1	20
1	1	1	1	50
1	2	1	1	5
1	2	1	1	10
1	2	1	1	20
1	2	1	1	50
1	1	1	2	5
1	1	1	2	10
1	1	1	2	20
1	1	1	2	50

Figure 51. Table Showing the Different Criteria used During Optimisation of Benchmarks

The next three tables in Figures 52, 53 and 54 use the Pearsons Correlation Coefficient, this is used to demonstrate how good a relationship there is between the predicted average interconnect length and the actual average interconnect delay. The closer the Pearsons Correlation Coefficient is to 1, the better the relationship, 1 being perfect. Column 1 with the heading Y contains the coefficient that is used when multiplying Q_{l-1} in Equation 29. The second column contains the value which is used in calculating K in Equations 30 to 33. So increasing x, reduces the effect of K on the overall equation, and in this case this means that the interconnect complexity has a larger influence compared to the size of the design. The final 5 columns are the 4 different equations plus a variant to Equation 33 in how K is calculated, where L4 is Log to the base 4 and L2 is Log to the base 2. The

reason for this variant is to see whether L2 might be a better estimate for K, because when deriving Equation 33 the Bi-Partitioning was used as the main assumption.

Y	$\sqrt[x]{Area}$	NS(g) Eqn(32)	NS(g) Eqn(33)L2	NS(g) Eqn(33)L4	NS(g) Eqn(31)	NS(g) Eqn(30)
0.5	x = 1	0.022	-0.092	-0.004	0.087	0.105
	x = 2	-0.04	-0.003	0.06	0.071	0.154
	x = 3	-0.074	0.037	0.081	0.071	0.36
	x = 4	-0.093	0.059	0.092	0.07	0.638
	x = 5	-0.104	0.068	0.092	0.071	0.652
1	x = 1	-0.119	0.016	0.055	0.149	0.164
	x = 2	0.024	0.052	0.078	0.175	0.247
	x = 3	0.059	0.068	0.085	0.19	0.423
	x = 4	0.074	0.077	0.088	0.197	0.662
	x = 5	0.054	0.022	0.051	0.206	0.655
1.5	x = 1	**0.136**	-0.03	0.261	-0.001	-0.021
	x = 2	0.128	0.25	**0.271**	0.129	0.185
	x = 3	0.126	**0.368**	0.221	0.18	0.397
	x = 4	0.124	0.268	0.203	0.202	**0.664**
	x = 5	0.095	0.211	0.164	**0.25**	0.649

Figure 52. Table to show the correlation between predicted average interconnect and actual interconnect, NS(g) represents predicted AI lengths without summing the external nets at each level while just using the greedy algorithm to partition.

It was found that reducing the area improved the correlation between the predicted average interconnect delay and actual average interconnect delay. The reason for this is that the influence of the size of the chip is being reduced, while increasing the influence of the interconnect complexity represented by the Rent Exponent. In all of the tables Equation 32 performs very poorly, the reason for this is that the data series is not large enough. Equation 33 has the number of levels as log(area) / log(2) compared to log(area) / log(4), this is because the equation is derived assuming the circuit is being bi-partitioned. This is shown to be valid in all the tables. When the number of hierarchical levels is given as log (area) / log (2) it out-performs the correlation when the number of hierarchical levels is given as log (area) / log (4).

Y	$\sqrt[x]{Area}$	S(g) Eqn(32)	S(g) Eqn(33)L2	S(g) Eqn(33)L4	S(g) Eqn(31)	S(g) Eqn(30)
0.5	x = 1	**0.176**	-0.072	-0.071	0.332	0.358
	x = 2	0.157	-0.115	-0.143	0.366	0.457
	x = 3	0.143	-0.138	-0.144	0.384	0.602
	x = 4	0.135	-0.142	-0.144	0.392	0.721
	x = 5	0.124	-0.071	-0.059	0.389	0.74
1	x = 1	-0.346	0.14	0.09	0.42	0.427
	x = 2	-0.163	0.105	-0.058	0.509	0.595
	x = 3	-0.025	-0.016	-0.088	0.551	0.719
	x = 4	0.035	-0.056	-0.102	0.569	0.754
	x = 5	0.151	-0.006	-0.029	0.575	0.749
1.5	x = 1	-0.041	0.11	**0.478**	0.32	0.267
	x = 2	-0.053	0.517	**0.478**	0.542	0.609
	x = 3	-0.062	**0.585**	0.38	0.612	0.727
	x = 4	-0.068	0.48	0.331	**0.639**	**0.754**
	x = 5	-0.088	0.449	0.291	0.588	0.686

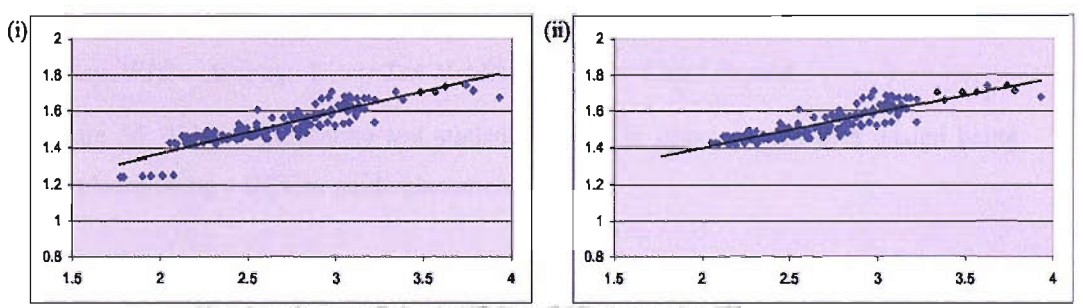
Figure 53. Table to show the correlation between predicted average interconnect and actual interconnect, S(g) represents predicted AI lengths summing the external nets at each level while just using the greedy algorithm to partition.

The correlation values in Figure 52 are significantly worse than in the other two tables where external nets are summed when finding the rent exponent. For both Figures 53 and 54, Equation 30 produces the best correlation with Y=1.5 and Input Area = $\sqrt[4]{Area}$. There is very little improvement of the highest correlation value in Figure 54, compared to the highest correlation value in Figure 55 compared to the extra computation needed to partition the circuit using an iterative improvement algorithm. Hence just the Greedy algorithm will be used to perform circuit partitioning.

Figure 55(i) shows the correlation from all the benchmarks. With a correlation of 0.89, this means there is a very high relationship between the predicted Average Interconnect Length of a design and the delay of the design when implemented in hardware. Hence when a design has a high predicted average interconnect length during the transformation stage in High Level Synthesis, the relationship tells us that the delay of that design has a high probability of also being large when implemented on an FPGA. This information can then be used to decide whether the transformation in question will be good for the system. Hence the values are unimportant – it is the relationship that is of most concern.

Y	$\sqrt[3]{Area}$	S(b) Eqn(32)	S(b) Eqn(33)L2	S(b) Eqn(33)L4	S(b) Eqn(31)	S(b) Eqn(30)
0.5	x = 1	0.148	-0.085	-0.079	0.382	0.41
	x = 2	0.101	-0.16	-0.138	0.399	0.489
	x = 3	0.079	-0.148	-0.127	0.414	0.622
	x = 4	0.068	-0.138	-0.121	0.422	0.663
	x = 5	0.045	-0.039	-0.021	0.422	0.659
1	x = 1	-0.275	0.138	-0.076	0.493	0.497
	x = 2	-0.032	-0.067	-0.215	0.57	0.646
	x = 3	0.137	-0.176	-0.243	0.61	0.74
	x = 4	0.18	-0.214	-0.255	0.628	0.762
	x = 5	**0.295**	-0.138	-0.167	0.635	0.735
1.5	x = 1	0.113	0.181	**0.546**	0.372	0.326
	x = 2	0.102	**0.631**	0.446	0.583	0.656
	x = 3	0.093	0.586	0.329	0.641	0.765
	x = 4	0.087	0.447	0.276	**0.66**	**0.767**
	x = 5	0.107	0.407	0.22	0.616	0.697

Figure 54. Table to show the correlation between predicted average interconnect and actual interconnect, S(b) represents predicted AI lengths summing the external nets at each level while using the greedy algorithm to partition, and iteratively improving the solution using the modified Kernighan Lin Algorithm



Y-Axis = Average Delay in NS Post PAR stage using Xilinx PAR Tool.
X-Axis = Estimated Average Interconnect Length in Gate Pitches During High Level Synthesis

Figure 55. Graph to show the Relationship between Predicted Average Interconnect and Actual Average Interconnect Delay.

To remove the influence of the outliers on the average correlation, the lower quartile, median and upper quartile respectively are (0.71, 0.86, 0.88), which shows a high correlation for the majority of the benchmarks. Figure 55(ii) has the data points belonging to the benchmark GCD removed to observe the graph without the few outliers, so as to

see the correlation more clearly.

3.9 Passing on Information to the FPGA APR tool

At this point we can retrieve information that is useful when passing the design through Synplify and eventually onto Xilinx [67]. We cluster macros together to form one big macro, a sub-group, which is highly connected without too many external nets. This means the information about the hierarchical structure is passed down to the placement tool through the RTL Synthesis. The problem with this method is that it reduces the freedom of the APR tool, by forcing it into a local minimum. So if we want to pass on hierarchical information to the APR tool we can use a User Constraint File (UCF), which allows the user to add Location and Timing constraints to a design when it is placed and routed in Xilinx. The UCF is a better choice as this allows suggestions on where macros should be placed in relation to each other without adversely affecting the APR's placement optimisation.

3.9.1 User Constraint File (UCF) Application

3.9.1.1 Location Constraints

Metric Type	AREA	AREA	AI	AI	WN	WN	W10	W10	CP	CP
Restriction	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Average	0.65	0.59	0.15	0.04	1.63	0.06	1.71	0.89	2.19	1.88

Data Source Figure 106 (Appendix). AI = Average Interconnect Delay, WN = Worst Net Delay, W10 = Average Worst Ten Net Delays, CP = Clock Period.

Figure 56. Table representing test statistics of the % improvement of a design being placed using a UCF to guide placement.

The UCF is a constraint file that is used to apply user constraints to a design when being placed and routed using the Xilinx APR tool. As shown in Appendix 7.1, there are two constraints we are concerned with, that is the LOC and AREA_GROUP constraints. These constraints are used to pass information about instance locations onto the Xilinx tool. LOC constraints designate specific areas on the chip where instances should be placed, but this is not beneficial, as this will take control away from Xilinx and not allow it to optimise to the maximum potential. The second constraint is more desirable, as AREA_GROUP still allows Xilinx to have full control but suggests which instances should be placed next to each other. Thus AREA_GROUP can be used to pass high level information to the APR tool. The actual information passed to the APR tool is the

hierarchical information obtained when circuit partitioning was performed. That is which macros should be placed together according to hierarchical groupings during RBP.

In the UCF, if an instance is placed in a group but then, later in the file, the same module is placed in another group; the last entry will negate the previous entry. This means there is no point in writing the complete hierarchy. If the complete hierarchical structure is written in the UCF starting with the top of the hierarchical tree, then all the groupings that are suggested higher up the tree will be ignored, as these same nodes will be repeated lower down the tree. So two methods have been used to pass on the hierarchical information. One method is to pass the groups at the bottom of the hierarchy onto Xilinx. The second method is to pass some of the groups onto Xilinx depending on the following criteria:

If the average number of external nets is less than the average number of external nets at that particular hierarchical level, then the group is passed on. The results of these two methods can be seen in the table of Figure 56. In every single case the average % improvement is higher when passing on the complete hierarchical information compared to using a criterion to pass on restricted hierarchical information, hence the complete hierarchical information should be passed onto Xilinx. But as can be seen there is no degradation on average, on any of the design metrics. The CP is improved by a significant amount when considering that the designs are exactly the same, other than being influenced by the hierarchical information supplied by MOODS. The methodology when using location constraints in the UCF can be as follows, find the minimal CP then once that has been achieved a UCF file can be used to see if the solution can be improved anymore, and as can be seen from the results this extremely likely.

3.9.1.2 Timing Aware Constraints

This topic will be discussed in Chapter 5 with future work.

3.10 Average Interconnect Length Influencing Decision Making

At this point a metric that describes the predicted average interconnect for a design has been derived. The average interconnect is now placed in the MOODS cost function so that the optimisation algorithm QE can optimise the design. The cost function objective function is shown below:

$$\Delta E = \frac{C_{estimate} - C_{current}}{C_{initial}}$$

where $C_{estimate}$ is the estimated cost after the transformation has been applied;

$C_{current}$ is the cost of the current design architecture before the transformation has been applied;

$C_{initial}$ is the cost of the initial design architecture; this value is used as a normalisation factor in order to give a fair comparison with all the design criteria;

E is the energy of the system, where the delta sign signifies a change in state.

Using the average interconnect length of a design in the cost function, we shall show how this new metric has influenced the final design implementation in terms of the clock period. Even though QE heuristic is designed for minimising area and delay, the same transforms that achieve those objectives will also achieve the objective of reducing the average interconnect of a circuit. As the average interconnect for a design does not vary significantly compared to area or total delay, different factors were tested when calculating AI's E of a respective design architecture, to see which factor would produce the most favourable design architecture. The effect on the area and CP of the different benchmarks can be seen in the tables of Figures 57 and 58, due to the different factors used when calculating E, the factor is represented by an integer value, where the larger the factor the more influence the AI will have on the cost function. S10 represents 10 * (AI's E), S25 represents 25 * (AI's E), etc. The % improvement is calculated as follows:

Let X^I Represent a metric (e.g. Area) of a synthesized design obtained after PAR using the Xilinx tool. I represents the set of constraints that were used when exploring the design space as the design is being synthesized in MOODS. These different constraints decide how much influence interconnect prediction has on the final design. These different constraints will be discussed later in this chapter and the next, when the actual constraints are included into the synthesis process within MOODS.

The % Difference is calculated as follows:

$$\%\ Difference = \left(\frac{X^N - X^I}{X^N} \right) \tag{34}$$

This equation shows what improvement (+ve) or degradation (-ve) of a design metric, has been caused by using Interconnect Prediction during Synthesis. The % Difference is averaged to give an overall measure of how effective Interconnect Prediction during synthesis is. This measure will be used for all the remaining results.

Scaling Factor	S10	S25	S50	S75	S100	S150
Average	0.46	0.49	0.36	0.51	-0.86	-0.85
First Quartile	0	0	-0.1	-0.1	-0.19	-0.2
Median	0.58	0.97	0.67	0.67	0.59	0.67
Third Quartile	3.1	4.26	3.1	4.11	4.11	4.11

Data Source Figure 107 (Appendix). S represents the scaling factor used when calculating E of AI in conjunction with the cost function, where * is an integer.*

Figure 57. Table representing test statistics of the % improvement of area of a design that has been synthesised using different AI metric scaling factors

Scaling Factor	S10	S25	S50	S75	S100	S150
Average	-24.54	19.15	18.27	20.55	12.93	13.84
First Quartile	0	0.65	0	2.77	0.65	0.65
Median	2.98	16.03	6.62	13.31	9.37	9.37
Third Quartile	23.53	28.23	29.84	32.3	34.95	34.95

Data Source Figure 108 (Appendix). S represents the scaling factor used when calculating E of AI in conjunction with the cost function, where * is an integer.*

Figure 58. Table representing test statistics of the % improvement of CP of a design that has been synthesised using different AI metric scaling factors

Figures 57 and 58 contain the average, median, lower quartile and upper quartile. The reason why there are 4 statistical measures are as follows: An average value is a good indication of the value in most cases. But this can be varied dramatically by the outliers whether negative or positive. For example we could have the following set of results -5, -5, -5, -5, -5, 0, 0, 0, 50, which would give an average value of 3.57, which would indicate that there is a positive improvement due to the effect of the outlier 50. But in reality if a there is only 1/7 chance of the design improving and 5/7 chance that the design will degrade, the methodology would be termed a failure due to the poor chances of success. For example the lower quartile (-5), median (-5) and upper quartile (0) provide a more realistic interpretation of how much actual improvement is being provided by this particular example.

When analysing the % improvement of area of a design, we are looking for a small change in area. This will show that when using interconnect topology to optimise the CP

of a design, there will be no major degradation of the area of a design. This will be the case throughout the analysis of the % improvement of area when interconnect prediction is used to guide HLS.

As can be seen in Figure 57 there is in fact a slight improvement of area on average until the scale factor has too much influence in the cost function. For the CP shown in Figure 58, the CP can be seen to increase in third quartile as the scale factor gets higher. The problem is that AI metric has too much influence, so does not always produce an overall optimal design in terms of area and delay. But overall there is a very good improvement of the minimal CP across the board. Figure 58 shows that in the first quartile there is no degradation while a large improvement in the third quartile, which steadily increases under the influence of the AI metric in the cost function. This shows that if the metric is used then there will be little chance of degradation (if any, the degradation would be very small) in the design's optimality in terms of delay and area. Actually there is a good chance of a large improvement in the design's optimality. Scale factor 25, 50 or 75 have all very similar results in terms of average % error. But the median value for scale factor 25 is a significant improvement over the medians belonging to the other scale factors, while the degradation in area is the least. Hence a scale factor of 25 shall be chosen for the AI metric in the cost function.

3.11 Conclusion on Pre-Floorplan Interconnect Prediction

Having seen in the last section that interconnect prediction can aid HLS, greater influences of interconnect properties are desired, in order to improve the optimality (in terms of CP) of a design produced by MOODS. Up to this point only a global measure of the interconnect layouts has been used to guide the quasi-exhaustive optimisation algorithm within MOODS. But if the distances between macros can be estimated then these distances can be used to decide whether merging or duplicating functional units would be beneficial or detrimental to the overall system. A further metric will also be introduced into the cost function within MOODS. The metric will measure the effect a transform has on the interconnect lengths of the macros involved in the transformation, whether the interconnects get smaller or larger if the transformation is performed, and to what degree. This will be a local measure of the interconnect layout compared to the global measure of the average interconnect metric. In the next chapter it will be shown that estimations of individual interconnect lengths between modules is an integral part of using interconnect prediction information when performing transforms in MOODS. The

original method for calculating individual interconnect lengths was to use equations derived from Rent's Rule that represent the expected interconnect length at a given hierarchical level. In order to find the expected individual Wire Length of a net, first the hierarchical level at which the net was cut would be found, then placed into an equation. But the drawback to this method is demonstrated in Figure 59.

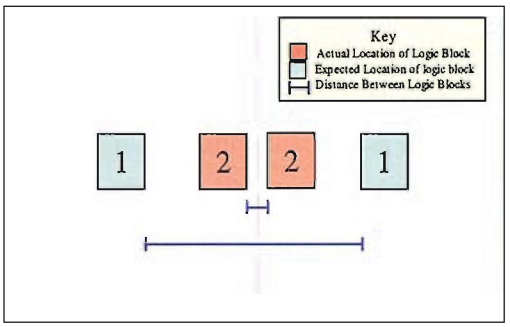


Figure 59. Positioning of Components Within the Hierarchy

The diagram shows that even though two logic blocks might be on either side of a partition, they can still be located next to each other. So if the expected interconnect length at hierarchical level *k* was used then the estimated interconnect length would be over-estimated. The higher up the hierarchical level, the more the expected interconnect length could be overestimated. So in order to solve this problem a floorplan will be generated. In the next chapter a floorplan will be constructed, which will then allow individual interconnect lengths to be calculated, enabling much greater physical information about a design to be used during HLS. This more extensive information will be shown to provide much more optimal designs in terms of Area and Delay (CP and Total Delay).

Chapter 4 Post Floorplan Interconnect Prediction

The next stage in using interconnect prediction within Moods (during synthesis) is to improve the clock period optimality of a design. Hence a measure of how transforms (namely transforms that merge or duplicate functional units) affect interconnect topology needs to be obtained. A transformation that causes a bad routing topology can cause a degrading effect on the delay of a design, hence interconnect prediction will aid in avoiding these transformations. In this thesis this includes interconnect length and congestion as stated throughout this thesis. To enable a measure of how transforms affect interconnect distances of macros belonging to a design during synthesis (within MOODS), individual interconnects will need to be calculated. Thus far, only the relative location of the macros are known, therefore to use bounding box methods to calculate interconnect distances, knowledge of the approximate location of macros on a chip is necessity. To find the approximate location of macros on a chip, an algorithm is provided in order to construct a floorplan from the slicing tree formed by Recursive Bi-Partitioning (RBT) in the last chapter. Once a floorplan has been constructed, the approximate location of all the macros will be known. Candidates will then be chosen for merging or duplicating functional units will be partly chosen based on the proximity of the candidate macros to their neighbours. The next few sections will now discuss the formation of the floorplan representation and all the considerations this entails.

4.1 Placement of Functional Units

At the highest level, generally the layout of the chip is not known, this means that decisions are made which cannot take into account how components are placed relative to each other. The only information available during HLS in MOODS is the circuit's netlist and the structural information of the macros needed for correct design functionality (data path units). This does not describe how close macros are and where they are located in relation to all the other macros that make the design architecture.

The layout of the chip influences how the chip will be routed and the routing delay. So a Floorplan available during HLS would be useful to predict how the functional units will eventually be placed on a chip. Having the knowledge of the approximate locations

of macros on a chip enables interconnect delay to be taken into account when deciding whether to perform transformations (namely merging and duplicating transforms), as how a transformation affects distances between macros (larger the distance, larger the delay) will now be available during HLS. So if a transformation causes distances between macros to increase to the point that the transform makes a design's delay decrease in optimality, then the transform will be declined.

To obtain the initial placement a floorplan can be used: a floorplan is an abstraction of a placement, i.e. the placement is generalised to reduce the detail to enable faster optimisation. As HLS is using an abstracted view of the physical level, a floorplan (topological) representation will be used in order to estimate the final placement. The macros will be represented as non-disjoint shapes on the floorplan, even though on an actual Xilinx Virtex chip the cells that make up a macro do not need to be in adjacent slices unless using carry logic. This abstraction is acceptable as Macros when placed will be tightly packed [47]. A Macro's internal routings will be concentrated in the locality of the macro, so has less effect on the global scale. The Floorplan will need to be fast in construction, while still being accurate enough to make good architectural decisions when optimising a design. The Floorplan will allow further design metrics to be produced, which will enhance the level of interconnect prediction in MOODS, which will then be used to improve design optimality.

The Floorplan will provide the knowledge of how the modules will be placed with respect to each other, most importantly which modules will be placed close together or far away from each other. The floorplan construction will also provide the basis for the estimation of the routing topology. The information obtained from the floorplan will then form the basis for all physical metrics. In order for this information to be as accurate as possible, the construction of the floorplan during HLS should mimic (as closely as possible) the floorplan constructed during APR. Remember a floorplan is an abstraction of the placement problem, hence the floorplan produced during HLS will have a higher level of abstraction compared to the floorplan during the PAR stage in Xilinx. To obtain this information various methods can be used, these methods were discussed in chapter 2. A floorplan can be randomly constructed then iteratively improved by rearranging the macros until the overall system is sufficiently optimized in terms of the design's objective function, or the floorplan can be constructed by

grouping modules together that are highly connected, in an effort to reduce interconnect lengths, hence increasing the optimality of the floorplan in terms of delay.

4.2 Constructing a Floorplan Representation

The first decision about deriving the floorplan is whether to form the floorplan in a top down or bottom up fashion. A top down approach is building the floorplan in a top down manner as shown in Figure 17. The floorplan is constructed while traversing down a slicing tree formed by RBP. Hence RBP first partitions the macros into sub-groups, then the sub-groups formed from this partition are placed into bins (in which the location and dimension of the region that the macros belonging to the sub-group are placed) on either side of the partition depending on an objective function such as the net cut objective or the Wire Length objective. The macros are then fixed in this region, unless a bin swapping stage is invoked later in the construction of the floorplan.

A bottom up approach starts at the bottom of the slicing tree and then gradually builds the floorplan up using a cluster based algorithm. Hence the macros at the bottom of the tree are placed first, then at the next hierarchical level they are combined with other macros to form bins. These bins will then combine with other bins of equal size at the next hierarchical level, this process is repeated until the top of the slicing tree is reached. At each hierarchical level there is no knowledge of the location of any of the other bins at the same hierarchical level. The locations are not known, as the groups that were partitioned to form the current sub-groups the macros reside in, have not been given a location yet. This can cause problems when minimising the net length between macros/sub-groups, as sub-groups can be placed on a non-optimal side of a join. This sub-optimal placement is caused by a sub-group being unaware of the approximate locations of other sub-groups that it is highly connected to, causing the sub-group to be placed on the wrong side of a join, further away from the sub-groups that it is highly connected to. Again on which side of the join the sub-groups in bins are placed is dependent on an objective function.

4.2.1 Floorplan Construction Considerations

The first decision that needs to be made is how the macros will be represented. Firstly the macros can be of two types: a hard macro, in which the geometry of the shape cannot change or a soft macro, in which the geometry of the shape can change. For the Virtex Xilinx Series the only macro that has a shape that is a hard is one that uses carry

logic. Fast carry logic is contained in each CLB and is used to increase the efficiency and performance of adders, subtractors, accumulators, comparators and counters. The routing of the carry signal can only move in a vertical direction to the CLB above it, unless at the top or bottom of the chip, where the carry logic moves to the next adjacent CLB to the right. The reason for the restriction in routing is that the carry logic uses separate logic and routing for fast generation of the carry logic.

The second decision is how these macros will be represented in our floorplan. There are only two real choices. One choice is to have each macro represented by a rectangle or a square (rectangular); this will allow the programming to be much simpler, hence much faster when run. Or to allow macros to be represented by shapes such as rectangular, L-shapes, T-shapes, Z-shapes or even allow non-regular shapes with sides like steps as shown in Figure 60. The macros could even have unrestricted dimensions (with respect to still fitting on the chip it is being placed on), where the macro is split into unit blocks with a number the same as the number of slices that the macro requires once placed in Xilinx. When Xilinx places macros (that do not use carry logic) the dimension are not fixed, as the cells that make up the macro do not have to be placed in adjoining slices. Hence the cells can fit into slices, into which macros with fixed dimensions (cells that have to be placed in a particular arrangement) would not. This allows dead space to be reduced while also allowing cells belonging to macros to be placed closer to cells they are connected with. Hence representing macros as a group of cells (Figure 60(iii)) would represent macros in Xilinx better, hence mimicking the placement of a design produced by Xilinx. But the abstraction of the floorplan in MOODS would be lost; hence the computational complexity would increase dramatically, with a slightly better imitation of the final Xilinx placement. The floorplan is going to be used to decide whether transforms during synthesis are good or bad with respect to interconnect delay, hence the floorplan only needs to be sufficiently accurate to correctly judge this decision. Again for complexity reasons the step-like shape will not be considered, even though allowing the shape to take this non-regular shape, will enable the macros to mould round other macros (shown in Figure 60), mimicking the way cells in macros would be placed in Xilinx. This will increase complexity for a negligible increase in the accuracy of the floorplan being used in Synthesis. L-shapes or T-shapes will not be considered as cells of macros on a Xilinx FPGA do not take this form. Using either the unrestricted dimensions or step-like dimensions as representations of a macro's shape, an increase in run time would occur. This increase would be caused by the combination

of the shapes becoming a much more complex problem, in an attempt to reduce the dead space on the floorplan. The dead space will not have much impact on the distances between macros, and as this is the main reason for the floorplan, the reduction in dead space is not worth the extra complexity. Hence all macros will be represented as rectangular shapes. Now how the floorplan is to be constructed needs to be considered.

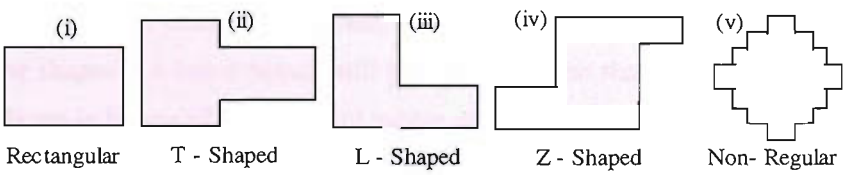


Figure 60. Different Shape Representations on a Floorplan

Now that the representation of the macro has been decided, consideration of what macro dimensions (long thin rectangle, square) will produce the most floorplan optimality is needed, so as to mimic a design's optimal placement in Xilinx. Using this knowledge a floorplan construction method can be chosen so that it is most likely to produce macros with the most optimum shape. When forming the floorplan, the most desirable shape for each sub section is a square as it has the smallest area to perimeter ratio. If this ratio is small, then the interconnect lengths within the macro boundaries will be smaller, as all the macros will be close to the middle, and hence should be in close contact with each other, reducing the average interconnect length. Macros/Bins that have elongated shapes are harder to join together without leaving dead space, which will cause an unnecessary increase in design size. Cells belonging to macros are tightly connected [47], hence it is important to have as much surface area for the routing of interconnects belonging to a macro. A square has a larger surface area than a rectangle with the same perimeter. This reasoning also applies to Bins lower down in the hierarchy, as they are also highly connected due to the nature of RBP. By highly connected we mean the number of interconnects with respect to the size of the Bin. Having a higher routing supply is desirable as this will relieve congestion if demand is high.

4.2.1.1 Dead Space Minimisation

To minimise dead space, the shapes of groups at the same hierarchical level need to closely match each other in width and length. If not, when the uneven bin is placed together with the bin in the next level of the slicing tree, dead space will occur, the assumption is that the larger the mismatch in shape, the larger the dead space. This

assumption is based on the following: bins are combined at every level of the hierarchy, which causes a problem when a bottom up strategy is used, as the dead space is caused due to a bad formation of a bin further down the tree. This can be seen in Figure 61, where shapes formed at early stages will not necessarily be the best fit further up the hierarchical tree. The reason A and B have an elongated shape is that a macro in either group A or in group B contains a hard macro and has that shape fixed. But a top down approach will tackle this problem, as it will start at the top of the hierarchical tree, and the shapes in a lower branch will be dictated by the shape of the group above them (as shown in Figure 62), so this will reduce dead space.

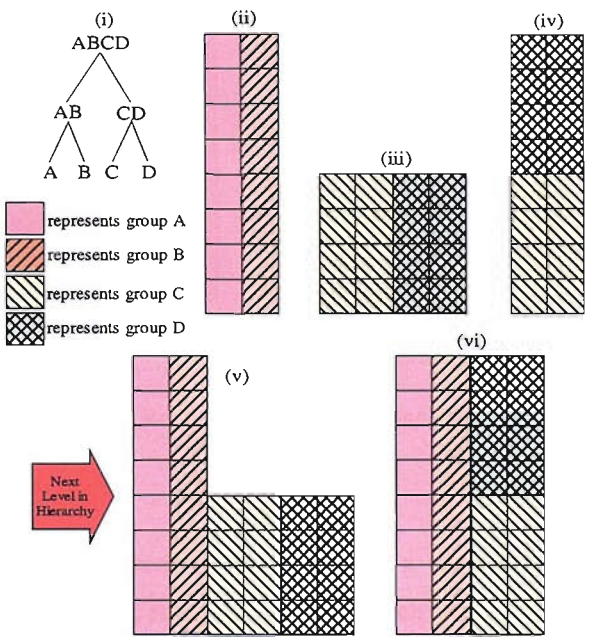
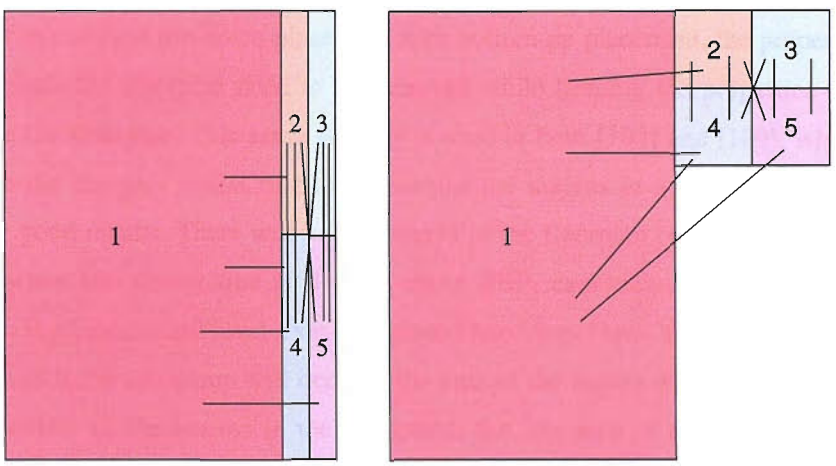


Figure 61. Demonstrating the Deterministic Effect on Macro Shape

4.2.1.2 Average Interconnect Minimisation

To minimise the average interconnect length the floorplan needs to be placed such that the number of longer interconnects is reduced. The hierarchy level that has the highest number of interconnects will be found at the lowest level, given the nature of recursive partitioning. Hence the interconnects belonging to the macros at the bottom of the hierarchy need to be kept to a minimum so as to reduce the average interconnect, as these interconnects will have the greatest effect. If a bottom up strategy is used, the smaller macros found at the lower levels of the hierarchy are shaped according to local restrictions, hence reducing the average interconnect length. If a top-down method were used, where the shapes of the smaller macros would be dictated by global restrictions,

this would increase the average interconnect length (but decrease the global interconnect length), the assumption being that the macros at the bottom of the hierarchy are shaped according to the placement sites left by the larger macros further up the hierarchy tree that have already been placed. The effect of the last statement is that the placement sites left can force the macros that are still needed to be placed, into a sub optimal placement.



(i) *Top Down Strategy* *(ii)* *Bottom Up Strategy*

Figure 62. Floorplans Representing the Same Design, but Using Different Strategies

Hence a bottom up approach would produce a floorplan with lower average interconnect length compared to a top down approach. Also, using a top down method may lead to higher congestion, as an interconnect is constricted into a smaller region to reach its destinations, as shown in Figure 62. The nets in Figure 62(i) will have optimal paths all within close proximity. This means that there will be more nets wishing to be placed within that region, hence increasing demand on the routing channels, which in turn will increase congestion. The congestion can then lead to a detrimental effect on the critical path delay, hence the clock period of a design. The nets in Figure 62(ii) have a much expanded region in which to find an optimal path, hence will be easier to route as the route will cause less demand on routing channels in a particular region.

Through the cases stated in this section the conclusion is that a mixed approach between bottom up and top down is needed, this will provide the least dead space while reducing the average interconnect length and minimising global interconnect. The degree of

optimality of a floorplan is important, as the more optimal the floorplan representation during HLS, the greater chance of mimicking the optimal floorplan of the design once implemented in hardware. This hybrid floorplan construction algorithm shall now be discussed.

4.3 Top Down Bottom Up Placement Algorithm

In order to combine top down placement with bottom up placement, the properties that will degrade the floorplan need to be removed while keeping the properties that will improve the floorplan. This same strategy is used in both [103] and [109], where they partition the design's netlist first then combine the macros in a bottom up fashion to produce good results. There will be two stages to the floorplan construction algorithm. Firstly when the slicing tree is derived using RBP, each sub-group formed from a partition at a hierarchical level k shall be placed into bins. These bins shall represent the region which the sub-group will occupy; the area of the region will be large enough to accommodate all the macros in the sub-group, (i.e. the area of the sub-group will be equal to the total area of the sub-group placed in it). The dimensions of the bins will also be used to determine the most appropriate orientation of cut to produce the next hierarchical level in RBP. Decisions on whether a cut is horizontal or vertical are made depending on the aspect ratio, as decided in section 2.10.6 to minimise dead space.

When a group within bin B is partitioned into two sub groups, these groups are both placed into a bin, both bins lie within the region of Bin B, without overlapping. The regions of the bins at each hierarchical level will not have a specific location, but decisions are made on which side of the cut, sub groups are placed on. The decision on which side of the cut sub groups are placed are made on order to minimise net length, hence reducing the delay between macros. The algorithm that is used to determine the dimension of every bin and what the cut orientations should be is shown in Figure 63, and will be discussed in the following section.

When a sub group contains one macro, the macro temporarily takes the dimension of the bin that the group resides in; this dimension will then be used when assigning locations to all the macros that are at the bottom of a branch in the slicing tree. This assignment of location is in the final stage of the algorithm and uses a Cluster-based algorithm in a Bottom-up manner. That is the smallest macros (highest hierarchical levels) are placed first, then larger macros that have not been assigned a dimension are

placed while traversing back up the slicing tree, this process will be covered in more detail in section 4.6. The reason for the Bin dimension being used is so that the shapes of the macros initially being placed have some consideration of the macros that will be placed around them. Once the hierarchical tree has been completed, a bottom up strategy will commence, where all the groups are pieced together. Once a macro has been given a dimension it is fixed (i.e. the macro shape becomes hard) until the floorplan formation has been completed. This is to reduce the complexity and the shape should be reasonably optimal as it was heavily influenced by top down partitioning. When a macro is at the bottom of a branch its shape is assumed to be soft unless the shape is hard due to being bound to a particular library cell (e.g. an ALU that uses carry logic on an FPGA). So presuming the shape of the macro is soft, the shape is moulded to fit the other sub group on the adjoining branch at the same hierarchical level. Once a macro's dimensions have been decided the macros shape becomes hard, so as to reduce complexity.

4.4 Bin Creation

Figure 64 shows the algorithm that is used to form the bins that are used to shape the groups of macros during RBP. These bins are used to decide how to cut the groups to form their sub groups. But first the initial dimension of the complete design needs to be decided.

Let A represent the area of all the total area of the design.

Initially

$A = d + remainder$, where d is an integer value

If *remainder* is equal to 0 then the dimension of the border is d x d If

remainder is not equal to 0 then the dimension of the border is

$(d + 1)$ x d

Let h_x and w_x represent the height and width (respectively) of the Bin x, being partitioned to form two bins at the next hierarchical level.

Bin Creation Algorithm

1. Let the Bin being partitioned be P and the two bins being formed from this partition be Bin A that contains macros $a_1,...,a_n \in A$, and Bin B that contains macros $b_1,...,b_n \in B$
2. If $\left(\frac{h_P}{w_P}\right) \geq \frac{1}{2}$ Then

   Go to 3

   Else

   Go to 4
3. Horizontal Cut has been chosen

   w_l will stay constant

   Find the Maximum height of all the hard macros (if any) in A.

   Let that value be mh_A.

   Find the Maximum height of all the hard macros (if any) in B.

   Let that value be $\boldsymbol{mh_B}$

   If $\left(\left(\frac{Area(A)}{w_P}\right) > mh_B\right) Or \left(\left(\frac{Area(B)}{w_P}\right) > mh_B\right)$ Then

   A Vertical Cut must be chosen

   Go to 4

   Else

   Remain with a Horizontal Cut

   $h_A = \frac{Area(A)}{w_P}$

   $w_A = w_P$

   $h_B = \frac{Area(B)}{w_P}$

   $w_B = w_P$

   Go to 5
4. Vertical Cut has been chosen

   $w_A = \frac{Area(A)}{h_P}$

   $h_A = h_P$

   $w_B = \frac{Area(B)}{h_P}$

   $h_B = h_P$
5. If Aonly contains one Macro, assign dimensions of A to the Macro If Bonly contains one Macro, assign dimensions of B to the Macro
6. End of Algorithm

Figure 64. Bin Dimension Assignment Algorithm

Let the dimensions of the designated chip be C_h x C_w and for this case, let the remainder be equal to 0, and finally let the dimensions of the initial border be d_x x d_y, which are also the dimensions of the whole design. If the initial border is not square then

If $(C_h C_w > d_x d_y)$ then the design is too big for the chip

Let i be the initial dimensions of the design before any partitions are made,

$$If\ d_y < C_h \Rightarrow h_i = d_y \qquad If\ d_x < C_w \Rightarrow w_i = d_x$$

$$w_i = \frac{C_h C_w}{d_y} \qquad h_i = \frac{C_h C_w}{d_x}$$

If $d_y >= C_h$ and $d_x >= C_w \quad \Rightarrow \quad h_i = C_w$ and $w_i = C_w$

Then the new dimension is passed onto the next iteration of the algorithm.

When the bins are given dimensions they are also allocated a region on one side of the partitioning cut. If it is a horizontal cut the bins whether placed below or above the cut and if the cut is in the vertical direction, then the bins are either placed to the right or the left of the partitioning cut. The allocation of the bin on either side of the cut is very important in terms of reducing interconnect length. If the bin is placed on the wrong side of the cut, this can increase the overall net length of the macros involved. Simple diagrams that show how arrangement of bins on a floorplan can affect the overall net length are shown in Figure 25 (section 2.10.5).

4.5 Minimising Wire Length through Bin Assignment

When deciding on the arrangement of the individual macros on the floorplan, information (in terms which macros should be grouped together) obtained by circuit partitioning would be beneficial, as this will tell us where all macros should be placed in relation to every other macro on the chip. The lower down in the hierarchy a grouping of macros is, the higher the priority that the macros in the group should be placed in close proximity. During RBP with bin assignment, an approximate geometric relation of the bins/macros is known. This relationship will be used in conjunction with all the external nets caused at each respective hierarchical level, k, to decide on which side of a cut bins are placed. In section 3.5, the weights of each net in the cut set of each individual group at each hierarchical level were stored. At this stage we are only using the net-cut cost, not the actual Wire Length between the macros. Yang, et al. found that the net-cut cost was globally consistent with the Wire Length [26]. These net values can

now be used during the placement of the macros to decide whether a macro is placed on the left/top or the right/bottom when a vertical/horizontal cut is made. Using these external nets during this phase will reduce terminal propagation (section 2.10.5).

To decide where a component will be placed, the following representation will be used:
Let A represent a bin, such that $a_1, a_2, .., a_n \in A$, where $a_1, a_2, .., a_n$ are macros that lie within A.
Let B represent a bin, such that $b_1, b_2, .., b_n \in B$, where $b_1, b_2, .., b_n$ are macros that lie within B.
Let δ_{AB} represent the sum of all the weights of the nets that connect bin A with bin B.
Let φ_{Al} represent the $\sum_{B \in C} (-1)^s \delta_{AB}$, where C is the set of all bins that are connected to A, at hierarchical level l. s is dependent on which direction the external net lies and which of the next 3 methods is being used. This value is the summation of all the external nets that are sourced or have a destination in A.

3 methods have been investigated. In the examples below for the 3 methods, the equations are in accordance with figure 65.

1. **Two separate values which represent the Vertical *(y)* and Horizontal *(x)* direction separately**

Let φ^X represent φ in the x direction, and φ^Y represent φ in the y direction, the more negative φ^X, the further to the left the bins should be placed; the more negative φ^Y, the further upwards the bins should be placed. So for Figure 65, the φ_A values would be:

For y direction at Cut 8 $\quad \varphi^Y_{A8} = (-1)\delta_{AD} + \delta_{AF} + (-1)\delta_{AH}$ (35)

$= (-3) + 2 + (-1) = -2$

For x direction at Cut 7 $\quad \varphi^X_{A7} = \delta_{AE} + (-1)\delta_{AG} + \delta_{AI}$ (36)

$= 3 + (-3) + 1 = 1$

Hence Bin A wishes to be placed in the bottom right hand corner as shown in Figure 65, but this would also depend on the value φ^X of Bin C and φ^Y of Bin B, which have been ignored for simplicity. But if for the y direction at Cut 7, Bin B had $\varphi^X_{B8} = -3$, Bin B

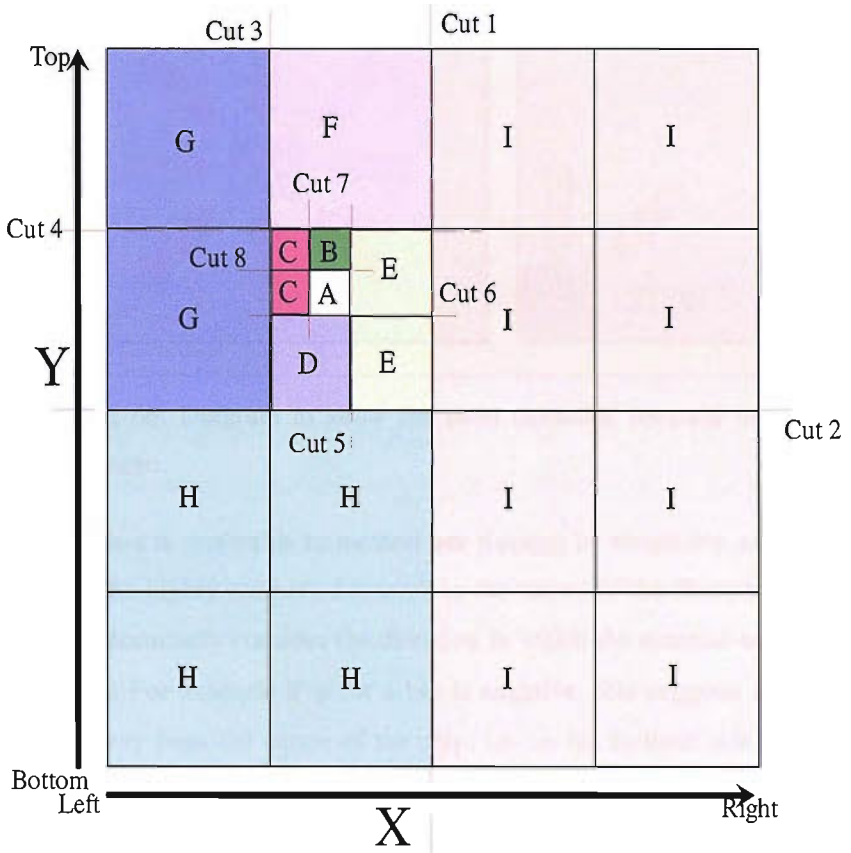
would be placed on the bottom and Bin *A* would be placed on the top. A more detailed explanation follows:

The nets when cut by a vertical bisection are added to the horizontal φ^X, and if the cut is a horizontal cut the nets are added to the vertical φ^Y. When a group of nodes is partitioned with a horizontal cut, then the nets in the cut set are added ($s = 2$) to the φ^Y value for the sub-partition placed on the bottom and subtracted ($s = 1$) from φ^Y for the sub-partition placed on the top, this means that the more positive the value of φ^Y, the more the Bin will be drawn upwards. When a group of nodes is partitioned with a vertical cut, then the nets in the cut set are added to φ^X value for the sub-partition placed on the left and subtracted from φ^X for the sub-partition placed on the right. This means that the more positive the value of φ^Y, the more the Bin will be drawn to the right. So if a group of nodes is partitioned, then the sub-partition with the highest value should be placed towards the top or right of the cut. The method is advantageous due to the fact that if, for example, a vertical cut partitions the circuit, the sub-groups produced will always lie on the left or right with regards to each other, depending on which side of the partition the sub-groups are placed. The downside is that if the weights of the external nets in a cut set are given to the *x*-direction δ value, then those nets will not influence the y-direction. So this would cause ambiguous terminal propagation, as discussed in section 2.10.4.

When two bins are formed from a partition, if one bin *A* is placed on the left and the other bin *B* is placed on the right of a vertical cut, then throughout the floorplan we know that the macros in bin A will always be on the left of the macros that reside in bin B. This means that the external nets (if any) that join A to B will always pull A towards the right. This makes equations (35 and 36) an accurate model for these forces.

δ_{AD} has a negative value as bin D is below bin A, this will make φ^Y_{A8} more negative with a larger number of nets that connect A to D and H, hence pulling bin A downwards. The higher the value of δ_{AD} the stronger the attraction for A to be placed closer to F. δ_{AF} has a positive value as bin F is above bin A, this will make φ^Y_{A8} more positive the larger the number of nets that connect A to F, hence pulling bin A upwards. The same methodology applies to calculating φ^X_{A7}, when positive, Bin A is being pulled to the right and if φ^Y_{A8} is negative, Bin A is being pulled to the left. This same method

for calculating φ^X and φ^Y is used for all bins during RBP, pulling all highly connected bins into the same region, reducing interconnect lengths between the bins. If two sub-groups (Bins) have the same value of φ then the group that has the highest number of external nets gets assigned to its desired region. If both sub-groups have the same number of external nets then the regions are assigned arbitrarily.



Cut	Cutset, but only including external nets that have a terminal in A
1	1
2	1
3	3
4	2
5	3
6	2
7	5
8	5

Figure 65. To Show the Relative Placement of Macros According to Partitioning

2. One value which represents both the *x* and *y* direction

To combat the disadvantage of method one, a combination of external nets belonging to both horizontal and vertical cut sets shall be put into the same φ. This is a straightforward representation, where the external nets that lead away from the centre will be negative and nets that lead into the centre are positive. The φ_A for Figure 65 is shown below:

$$\varphi_{A8} = \delta_{AD6} + \delta_{AE5} - \delta_{AF4} - \delta_{AG3} + \delta_{AH2} + \delta_{AI1} \quad (37)$$

$$= 2 + 3 + (-1)2 + (-1)3 + 1 + 1 = 2$$

The sub group with the highest φ is placed towards the centre of the floorplan. This is simple but allows the bins with the highest number of external nets to be placed in the centre where they are most accessible, as shown in Figure 66.

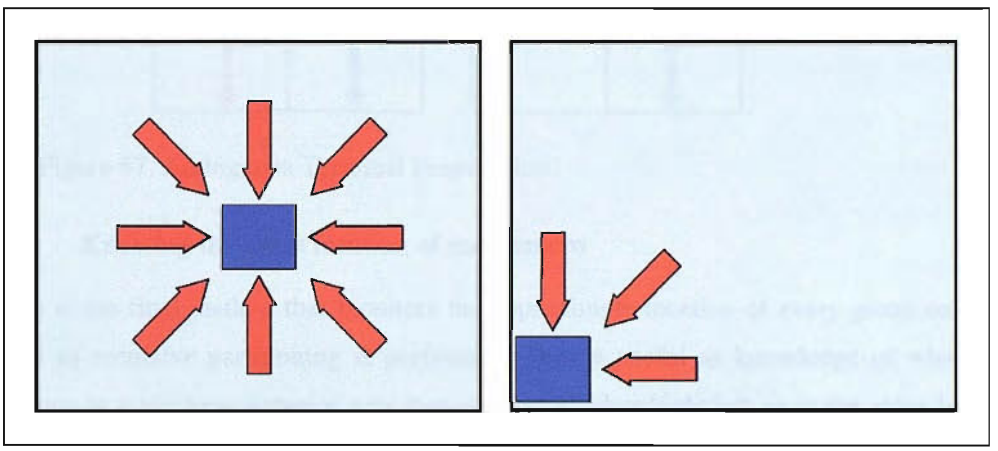


Figure 66. Diagram to show the most desirable location for a highly connected macro

This method is preferable to method one through its simplicity, as our only objective is to place the highly connected macros in the centre of the floorplan. Again this method does not accurately consider the direction in which the external nets that belong to bins pull them. For example if φ for a bin is negative, this suggests that the bin should be placed away from the centre of the chip, i.e. on the furthest side of the cut away from the centre of the chip. But the reason why a bin's φ is negative could be that the bin is highly connected to a bin from up above (presuming the bin is above the centre of the chip). Placing the bin on the left side of the partition will have little if any effect on reducing the interconnect distance between those highly connected bins. The bin might have actually been better placed on the right side of the partition, where it had other highly connected bins, but that were not as highly connected as the bin from above. So when φ signals that the group should be placed away from the centre, it should really be placed on the side of the partition closest to the centre. Another problem with this approach is that it could lead to higher congestion in the centre of the chip, as the majority of interconnects will be routed in the centre of the chip. Hence it would require too many resources in the centre of the chip to make this a practical approach.

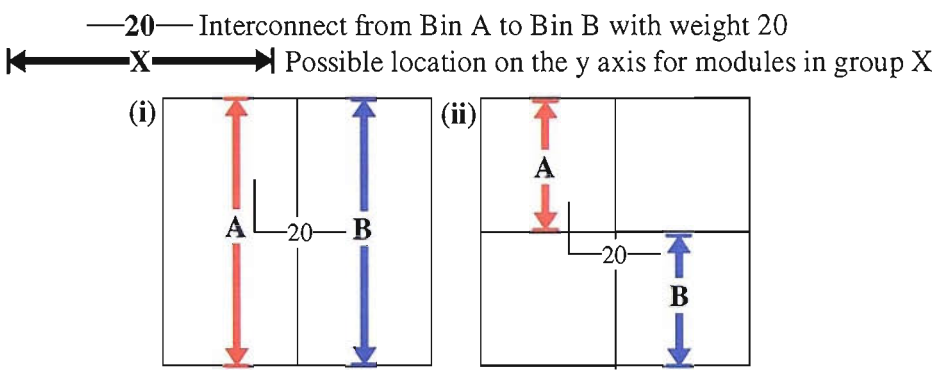


Figure 67. Ambiguous Terminal Propagation

3. Knowing the exact location of each macro

This is the final method that monitors the approximate location of every group on the chip as recursive partitioning is performed. This is useful as knowledge of whether macros in a bin have external nets that are above/below/right/left or at the same level. To achieve this computation is marginally more expensive and storage would increase to hold all these extra values (true/false if the net has been added to φ^X, φ^Y and approximate locations of bins). But the gain in accuracy of the placement information would improve the overall solution, and will remove the ambiguity of terminal propagation (Figure 67). Figure 67 demonstrates the result when Bin A does not know where Bin B will be placed until once the Bins have been placed, this means that Bin A is placed on the top and Bin B is placed on the bottom where this is not the most optimal placement (assuming there are no other external nets). As for method one, nets cut at a vertical cut are used to calculate φ^X, and nets cut from a horizontal cut are used to calculate φ^Y. But the nets are then added to the δ_{ABX} (if originally partitioned by a horizontal cut) or δ_{ABY} (if originally partitioned by a vertical cut) further down the slicing tree. For the explanation of this method the original partition is made by a horizontal cut shall be presumed and also using Figure 67. So the first cut is a vertical cut, where the weight of the net going from bin A to bin B is 20 so this is added to δ_{ABX}. Then RBP carries out the partitioning on the next hierarchical level, both A and B get placed on the higher bin of the respective partition. The next cut is a vertical cut so has no influence on δ_{ABY}. Then finally A is placed on the upper bin of partition after a vertical cut. The weight of the net is then added to δ_{ABY}. This means that A will also pull B in an upwards direction which can now be accounted for where method (1) could not account for this.

4.5.1 Conclusion on Wire Length Minimisation Strategies

Method 3 has been chosen for its better floorplan optimisation potential, while having little increase in computation. This is important as this will increase the predictive power of the floorplan, as it should more closely match the optimal floorplan of the APR tool. The drawback to this method is storage, but for the extra accuracy only two extra bits are needed to store whether a net has been added to the horizontal, vertical or both external values and the approximate locations of the bins. Each individual data path node will have an external vertical and horizontal value. When a sub-group's location on either side of a horizontal cut is being decided, the vertical external (+ or – depending on the direction of the external nets) values for all the macros belonging to the group are added together. This value is then used to decide on what side of the partition the bin containing the group is placed. If the cut is horizontal then the sub-group with the highest vertical external value is placed on the top, hence the other sub-group is placed on the bottom. If the cut is vertical then the sub-group with the highest horizontal external value is placed on the right, hence the other sub-group is placed on the left.

The orientation of the macros after the initial partition is not important due to the fact that the design's floorplan can be flipped along the x or y- axis and there would be no loss in generality. Until either 2 or more Vertical Cuts and 2 or more Horizontal cuts, no components have actually been moved away from the perimeter of the chip, but during the next phase some of the sub-partitions will be moved into the centre (away from the periphery of the chip). Hence after the second cut is the best time to remove the IOB pins. The IOB pins are the pins that join nets from off the chip; this is why they are on the periphery of the chip. As they are on the periphery of the chip, they do not occupy any slices, so they are modelled with no area, hence will not affect RBP when taken out. At the same time, which sub-partition the IOB pins belong to is stored, so that later on pin locations can be placed in the most advantageous region, at the edge of the chip.

4.6 Formation of the Floorplan

The actual formulation of the floorplan starts at the bottom of the hierarchy and works up towards the top, placing all the macros together until all groups have been placed together. The decisions on whether the groups shall be placed on the top/bottom or to the left/right have already been made, so it is simply placing the macros together in order to realise the floorplan. The algorithm is shown in Figure 68.

Let k represent the number of hierarchical levels produced by Recursive Bi-Partitioning (RBP) the circuit netlist.

Let $X_l = \{x_1, x_2, ...x_C\}$ be the set of all bins formed during RBP (using the bin creation algorithm) at hierarchical level l and C is the cardinality of X, where each bin contains at least one macro.

Let $a(x_n)$ represent the area of bin x_n

Let $h(x_n)$ represent the height of bin x_n
Let $w(x_n)$ represent the width of bin x_n
Let $s(x_n)$ represent the number of macros that lie within bin x_n
Let $t(x_n)$ represent the type of shape of bin x_n *(i.e.* either hard (fixed dimension) or soft (variable height, variable width))
Let $y_n \in X_{l-1}$ and $z_n \in X_{l-1}$ be the two children bins of $x_n \in X_l$ when partitioned given that $s(x_n) > 1$.

For all of the following functions, the orientation of the cut and which side of the cut a bin is placed was decided in the Bin_creation Algorithm (section 4.4). When a bin has been chosen to be placed on the right or on top of a cut, the bin is said to have been given priority. There are only two functions shown in the algorithm but repeated many times, it has been shown this way to clarify the different type of combinations of bin types.

Function 1: Both_Bins_Hard()
Place Macros together, according to what type of cut has been decided and which bin has priority. If a vertical cut, match up the bottom of the bins with each other, if a horizontal cut, match up the furthest left point. Assign combined dimensions of y_n and z_n to bin x_n so that the dimension of x_n can be used for the preceding hierarchical level, $t(x_n) = Hard$.

Floorplan Generation Algorithm

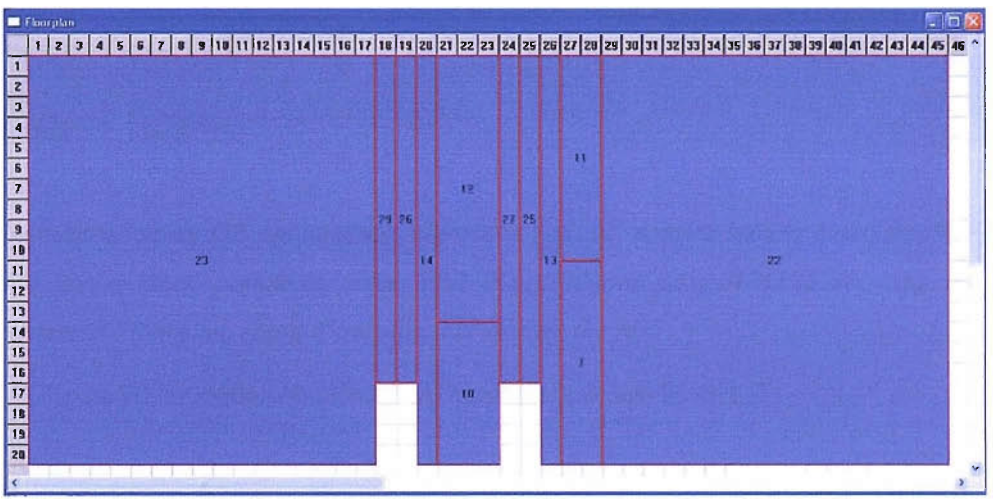
1. Let hierarchical level be $l = k - 1$ and $n = 1$
2. While $s(x_n) = 1$, $x_n \in X_l$ and n does not equal C then
   $n = n + 1$
3. If $n = C$ go to
4. Get y_n and z_n
5. If $s(y_n)=1$ and $s(z_n)=1$ then
   If $t(y_n)=$ *Hard* and $t(z_n)=$ *Hard* then
   Both_Bins_Hard()
   Else $t(y_n)=$ *Hard* or $t(z_n)=$ *Hard* then
   Soft_ Bin()
   Else $t(y_n)=$ *Soft* and $t(z_n)=$ *Soft* then
   Soft_Bin()
   Else If $s(y_n)>1$ and $s(z_n)=1$ then
   If $t(z_n)=$ *Hard* then
   Both_Bins_Hard()
   Else $t(z_n)$ *is Soft* then
   Soft_Bin()
   Else If $s(z_n)>1$ and $s(y_n)=1$ then
   If $t(y_n)=$ *Hard* then
   Both_Bins_Hard()
   Else $t(y_n)$ *is Soft* then
   Soft_Bin()
   Else $s(z_n)>1$ and $s(y_n)>1$ then
   Both_Bins_Hard()
6. If $n = C$, go to 7
   Else $n = n + 1$, go to 2
7. If $l = 1$ go to 8
   Else $l = k - 1$ and $n = 1$ and then go to 2
8. Floorplan Fully Constructed, hence Algorithm Finished

Figure 68. Floorplan Construction during HLS

Function 2: Soft_and_Hard_Bins()

Choose the bin with $t(\ ) =$ *hard*, if both $t(\ )$ are soft, choose the bin with priority. Assume the bin chosen is y_n, obtain dimensions of the bin. (if $s(y_n)=1$, the dimensions were assigned during Bin_Creation Algorithm). If a Vertical cut, let $h(z_n)=h(y_n)$, then adjust $w(z_n)$ so that $a(z_n)<h(z_n)*w(z_n)$ and the total area of the group z_n can fit inside the Bin. If a Horizontal cut, let $w(z_n)=w(y_n)$, then adjust $h(z_n)$

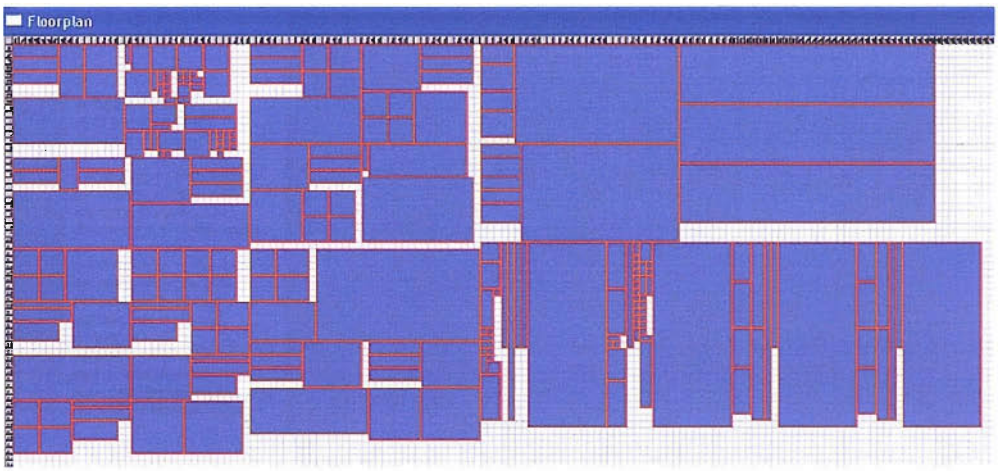
so that $a(z_n) < h(z_n) * w(z_n)$ and the total area of the group z_n can fit inside the Bin. Place the bins together depending on the type of cut and which bin has priority. If a vertical cut, match up the bottom of the bins with each other, if a horizontal cut, match up the furthest left point and then assign the combined dimensions to bin x_n, $t(x_n)$ = *Hard.*



Synthesised using QE optimisation algorithm with all metrics having equal priority and target clock period of 20ns. FLR file produced post MOODS showing an estimated Floorplan, using Floorplan.exe to view the file. Macros 7, 22, 27, 10, have respectively absorbed 6, 5, 8, and 9.

Figure 69. Floorplans obtained from benchmark design Bench_mark_2;

The floorplan is written to a floorplan (FLR) file that lists all the coordinates of all the macros. Another program which has been introduced is called Visual_Floorplan. This program takes in a Floorplan (FLR) file which contains all the locations of the macros on the floorplan. The floorplan for a design can then be viewed, and examples are shown in Figures 69 and 70. This is accessible as part of the design process in MOODS. Having a visual aid allows a better understanding of how the design will eventually look rather than just coordinates, which are harder to comprehend.

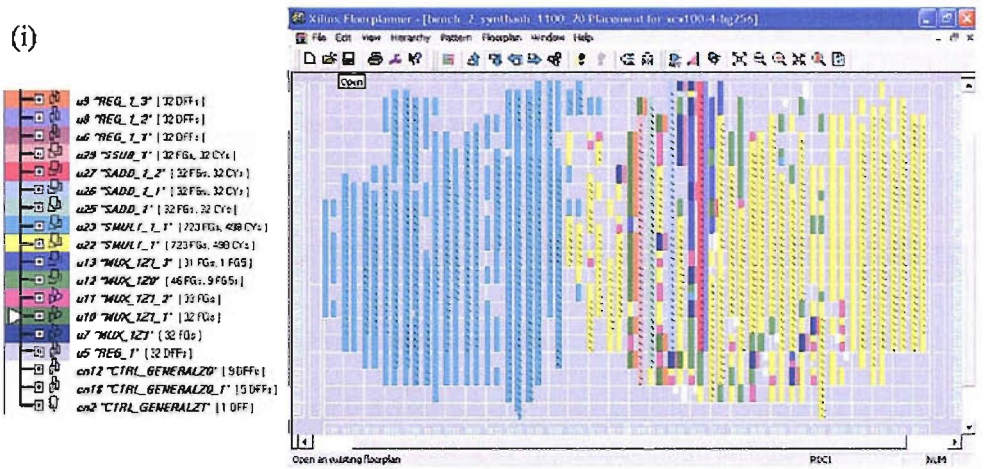


Synthesised using QE optimisation algorithm with all metrics having equal priority and target clock period of 50ns. FLR file produced post MOODS showing an estimated Floorplan, using Floorplan. exe to view the file.

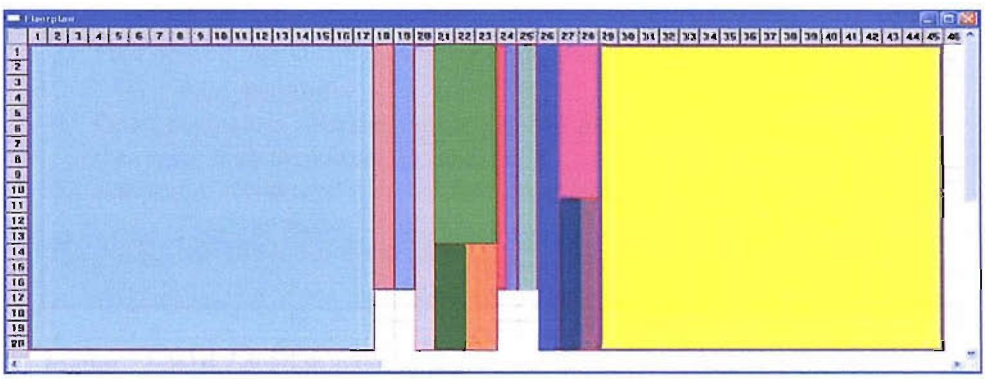
Figure 70. Floorplan obtained from benchmark design Matrix_2

Figure 71 shows that there is a good imitation of the floorplan produced in MOODS compared to the final floorplan after the place and route stage in Xilinx. The imitation is not exact, but as long as the macros are in approximately in the same position as the macros placed on the floorplan produced from the NCD file, then the decisions based upon the floorplan in MOODS can accurately take into account individual nets. There can be some dead space on the floorplan as shown in Figure 70. This may have an effect on the interconnect lengths between macros, and hence could lead to false information, that is, macros that are separated by dead space could move into this dead space to shorten the distance between macros. So the next stage is to test whether dead space does have an effect on the accuracy (and hence improve design optimality) of the floorplan, by removing as much of this dead space as possible, this is covered in Chapter 5.

(i)



(ii)



(i) Synthesised using QE optimisation algorithm with all metrics having equal priority and target clock period of 20ns. NCD file produced post Xilinx Place and Route, using Xilinx Floorplanner 6.3i to view the file

(ii) Same floorplan as in Figure 69 but colour coded to match Figure 71(i)

Figure 71. Comparison of Floorplans obtained from benchmark design Bench_mark_2;

4.7 IOB Pin Assignment

The final stage of the floorplan is to assign the IOBs to pin sites around the periphery of the chip. These pin sites then can be passed on to Xilinx using the UCF file. The main algorithm is shown in Figure 72, followed by a more detailed description of selected functions, but first some definitions.

Let $P = \{p_1, p_2, ..., p_F\}$ be the set of all input signals in the entity declaration, where p_n has t number of nets with bit width w.

Let $M(p_n) = \{m_1, m_2, ..., m_C\}$ be the set of all macros that are connected to p_n, where

(x_n, y_n) is the centre point of the space that m_n occupies on the design floorplan.

Let h represent the chip height, Let w represent the chip width

Let m_r represent the pin sites wanted in region r

Pin Placement Algorithm

1. For all $p_n \in P$ repeat 2 to 3
2. Find Average Location (A) of $m_i \in M(p_n)$, calculation as follows:

$$A = \sum_{n=1}^{C} \frac{(x_n, y_n)}{C} = (x_a, y_a)$$

3. Find desired region in which to place p_n using following criteria:
   Find_Region()
4. Order the regions with the highest demand first, then in descending order to the region with the lowest demand.
5. Assign the actual region in which to place pins
   Pin_region_Assignment()
6. Assign specific locations to all pins

Figure 72. IOB Location Assignment

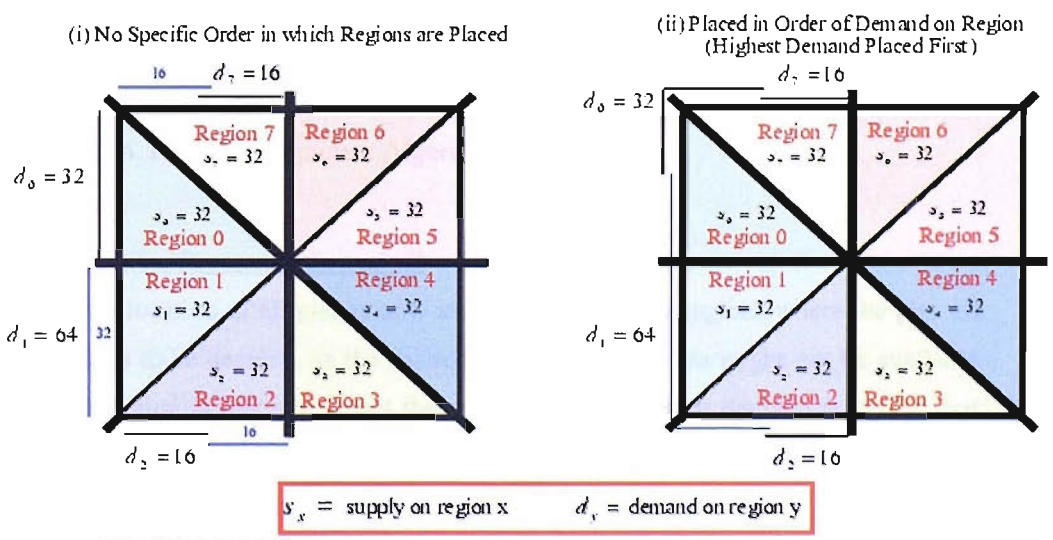


Figure 73. Region Assignment

Function 1: Find_Region()

The regions are assigned using the algorithm described in Figure 74, where a positive

gradient is the diagonal from bottom left hand corner to the top right hand corner. The different regions are shown in Figure 73.

Find Desired Region

Let $pos\_grad = (h/w) * x_a$ represent the diagonal
Let $neg\_grad = h - ((h/w) * x_a)$ represent the negative diagonal

If $(0 <= x_a <= w/2) and (0 <= y_a <= h/2)$ then
    If $(y <= pos\_grad)$ then
        Assign region 7
    Else
        Assign region 0
Else if $(0 <= x_a <= w/2) and (h/2 < y_a <= h)$ then
    If $(y <= neg\_grad)$ then
        Assign region 2
    Else
        Assign region 1

Else if $(w/2 < x_a <= w) and (0 <= y_a <= h/2)$ then
    If $(y <= neg\_grad)$ then
        Assign region 5
    Else
        Assign region 6
Else $(w/2 < x_a <= w) and (h/2 < y_a <= h)$ then
    If $(y <= pos\_grad)$ then
        Assign region 4
    Else
        Assign region 3

Figure 74. Region Assignment Algorithm

Function 2: Pin Region Assignment()

The desired location of all pins is now known, so the actual regions where the pins are placed needs to be decided, as the desired location for the pins might not be available. Now each region is ordered so that the region with the highest demand is placed first. Hence the region with the highest demand will have pins placed in it first: p_1 is placed before any other set of pins.

Pin Region Assignment

If $(s_n - d_n) >= 0$ then

$b_{pr} = b_n,\ s_n = s_n - d_n,\ e_n = b_n + d_n,\ d_n = 0$

Else

$b_{pr} = b_n,\ b_n = e_n,\ s_n = s_n - d_n,\ d_n = d_n - s_n,\ s_n = 0$

Next search through adjacent regions for pin sites

Let $i = n - 1$ *(adjacent region anti-clockwise direction)* and

$j = n + 1$ *(adjacent region clockwise direction)*

While $d_n > 0$

If $(s_i - v_i) >= (s_j - v_j)$ then

If $d_n > s_i$ then

$s_i = 0,\ b_i = e_i,\ d_n = d_n - s_i$

Else

$b_{pn} = e_i - d_n,\ e_i = e_i - d_n,\ s_i = s_i - d_n,\ d_n = 0,\ i = i - 1$

End

Else

If $d_n > s_j$ then

$s_j = 0,\ b_j = e_j,\ d_n = d_n - s_j$

Else

$b_j = b_j + d_n,\ e_i = e_i - d_n,\ s_i = s_i - d_n,\ d_n = 0, j = j+1$

End

End

End While Loop

End

Figure 75. Pin Region Assignment Algorithm

For an explanation of this concept Figure 73 shall be used. Figure 73(i) is where the pins are placed without any specific order, where by matter of chance region 0 is placed first followed by region 2, then region 7, then finally region 1. In Figure 73(ii) the pins are placed such that the pins that lie in the region that has the highest demand are placed first, hence the pins that lie in region 1 are placed first followed by region 0, then either region 2 or region 7 followed by either region 2 or region 7 with respect to the last choice. When placing the pins into regions 0, 1, 2 and 7 in Figure 73(i), as region 1 is placed last the pins become very disjoint forcing the some of the pins wishing to be placed in region 1 to placed far away. But Figure 73(i) shows that placing pins that lie in the region that has the highest demand makes the pin placement much less disjoint. Also the pins are not placed that far away from their desired location. Now all $p_n \in P$, that were assigned to a pin placement region on the chip, have an exact region in which to be placed, all there to do now is to use the starting point of each region b_{pr} to assign

IOBs exact locations that are available on the chip. The algorithm that actually finds the specific locations of the IOBs is shown in Figure 75. When the Pin locations are passed onto Xilinx using the UCF, the new locations do not affect the design metrics, so the concepts discussed in this section are more applicable for an ASIC.

4.8 Individual Wires Calculation Considerations

Using the floorplan that is now accessible during HLS, it is desirable to find the individual Wire Lengths of the interconnects between the macros. The individual Wire Length will be used when deciding on resource allocation, whether to merge or duplicate components and the clock period. A floorplan is now available in MOODS allowing an estimation of the bounding box to become possible. *Bodapati et al.* expand on estimating the bounding box and use both the half perimeter and the area for estimating the Wire Length, depending on the aspect ratio [10], as shown in 2.5.1.1. The problem with the proposed method in [10] is that too much detail of the routing would be needed to implement it. Also the algorithm should be extended to include a factor that will represent how congested a region is.

The problem with individual Wire Lengths is that if a circuit is split into different areas each sub-group will have a different Rent exponent *(p),* as demonstrated in [34]. But these individual interconnect lengths shall be used when considering candidates for the merging or duplicating transforms within MOODS. When considering candidates for the duplication transform, the candidate's interconnects will span many different regions, i.e. spanning the entire circuit or the candidate would not be suitable for duplication, hence the p for the whole design can be used so no more calculation would be needed. The merging transform will be more sensitive to variations in local congestion, but to compute the local congestion every time would be too time consuming. We shall use the global congestion (derived from the average interconnect length) as a measure for how congestion will affect a transform. The global congestion will be slightly less accurate than a local congestion measure for describing the congestion in the region in which the new and old interconnects (evolved from a transformation) are/were located, this is due to local congestion taking into account the non-homogeneous properties of a circuit. But the global congestion measure will give an accurate measure of the congestion on the entire chip. If the global congestion is high and the merging transform is wishing to be performed, a merging transform generally elongates the interconnect lengths of the macros that are connected to the

candidates for merging. The reason for this is that two data path units can be separated, giving a larger degree of freedom where the macros can be placed, hence allowing the macros to be placed in close proximity to the macros that are connected to them. Once the candidates for the transform have been merged the new macro will most likely be placed in the middle of all the macros that are connected to it. This means that global congestion will affect these new interconnects, as interconnects will be travelling on a large portion of the chip, hence making these interconnects more susceptible to global congestion.

The penultimate consideration for individual wires is the number of pins each interconnect has got. The larger the number of pins, the larger the density of the interconnect will be, which, as stated before, can have adverse affects. The problem is that at a higher level, the number of terminals each net has is not known without estimation, only the combined number of bits each interconnect contains. It has already been shown that two-terminal nets are sufficiently accurate to model all the nets on a chip, so we shall model all individual interconnects as 2-pin nets, with a weight equal to the number of bits of each interconnect.

The final consideration is from which two points the distance between two macros is calculated. As stated earlier the half perimeter rule estimates the Wire Length of a net to be half the perimeter of the bounding box that encompasses the terminals of the net as shown in Figure 16 (section 2.10.1.1). But where these terminals will reside is not known, hence this approach does not fully lend itself to HLS. So a way of deciding what this bounding box will encompass is needed. There are two different places on a macro from where the interconnect can be measured, either measuring the interconnect length from the closest points of both macros to each other (Figure 76(i)) or measuring interconnect length from the centre of both respective macros [60] (Figure 76(ii)). As shown in Figure 76(i), measuring the interconnect length from the closest points of the macros with respect to each other does not consider the size of the macro, where Figure 76(ii) does. Taking the size of a macro into account when measuring the interconnect distance is important because interconnects will not necessarily be sunk in the closest point to all the pins on the net, so the larger the macro the more probable the net will have a greater distance to travel. If there was only a one to one mapping from input to output then the second choice might be adequate, as it will orientate the shape to make the output closer to its destination macro. But if there are two output macros these

become a less realistic representation, as the orientation will now be dependent on two locations. Even though taking the points to be in the centre of the macros will result in longer interconnect. This should give a fairer representation with which to make comparisons.

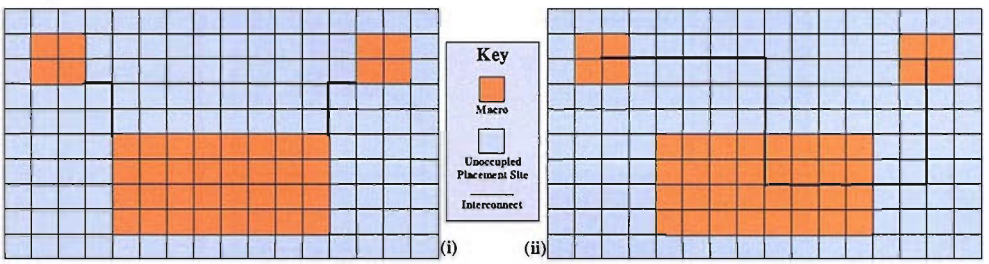


Figure 76. Where to Measure the Interconnect Distance From on a Manhattan Grid

4.8.1 Actual Individual interconnect Calculation

From the points discussed in the previous discussion the most desirable choice from where to measure interconnects is the centre of the macros. The interconnect distance between two macros will be calculated by using the half perimeter rule between the centre/closest points of the two respective macros.

Centre Point Calculation

The centre points are found simply. For clarity, Figure 77 can be used to understand the process. First to work out the centre point on *B* we need to know the most extreme points in the *X* direction. These points are 6 and 12, which are added together and divided by 2, hence making the mid point of *B* in the *X* direction 9. Then to find the mid point of *B* in the *Y* direction the two extreme points are 1 and 5, which are added together and divided by 2 to give 3. This gives the centre point as (8, 3) for *B*. The same procedure gives the centre point for *A* as (3, 8).

Now to find the distance from the centre of *A* to the centre of *B* the following half perimeter equation is used:

$$D_{AB} = |x_a - x_b| + |y_a - y_b|, \tag{38}$$

where the centre co-ordinates of macro *C* are (x_c, y_c); the calculation of these points will be discussed next.

Hence the estimated interconnect length of AB in Figure 77 using the formula for D_{AB}. is 5.

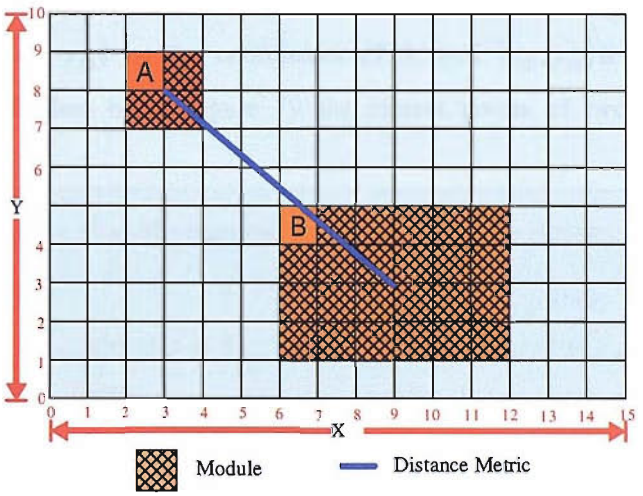


Figure 77. Calculation of Individual Interconnect Distance

Closest Point Calculation

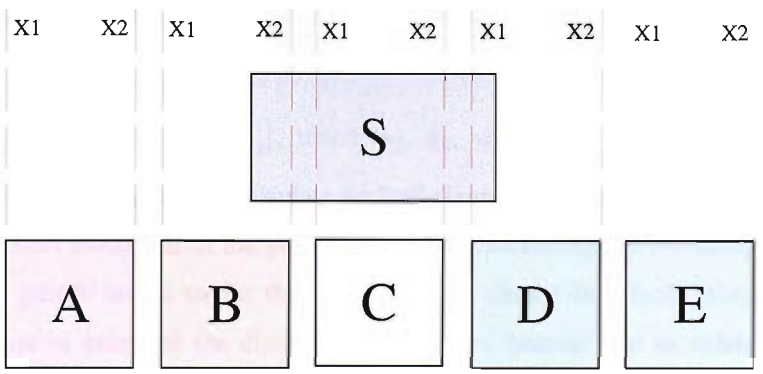


Figure 78. Diagram Showing Different Relative Positions of Macros

To calculate the closest point there are three conditions that need to be considered, depending on the two macros positioning with respect to each other and whether the X or Y coordinate is being considered.

Using Figure 78, we can see where the three conditions occur. Firstly A is completely to the left of *S*, so the closest point (with respect to the X-coordinate) will be the furthermost point to the right of *A*, and the furthermost point to the left of *S*. *E* is

completely to the right of *S*, so the closest point (with respect to the X-coordinate) will be the furthermost point to the left of *E*, and the furthermost point to the right of *S*. *B*, *C*, and *D* shall be classed as the same condition as there is some overlap in each case. To calculate when each of these conditions occurs, the following method is used:

Let $(x_{A1}, y_{A1}) \& (x_{A2}, y_{A2})$ be the coordinates of *A*, and $(x_{B1}, y_{B1}) \& (x_{B2}, y_{B2})$ be the coordinates of *B*, then using Figure 79 the closest points of two macros can be calculated.

Calculation of the Coordinates of the Closest Points Between Two Macros

If $x_{A2} < x_{B1}$ then

$x_{w1} = x_{A2}$ and $x_{w2} = x_{B1}$

Else if $x_{A1} > x_{B2}$ then

$x_{w1} = x_{B2}$ and $x_{w2} = x_{A1}$

Else

$x_{w1} = (x_{A1} + x_{A2})/2$ and

$x_{w2} = (x_{B1} + x_{B2})/2$

If $y_{A2} < y_{B1}$ then

$y_{h1} = y_{A2}$ and $y_{h2} = y_{B1}$

Else if $y_{A1} > y_{B2}$ then

$y_{h1} = y_{B2}$ and $y_{h2} = y_{A1}$

Else

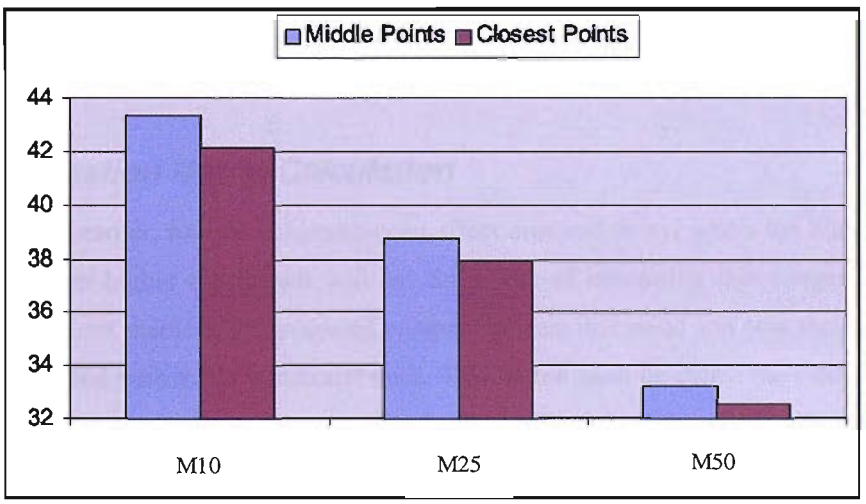
$y_{h1} = (y_{A1} + y_{A2})/2$ and

$y_{h2} = (y_{B1} + y_{B2})/2$

Figure 79. Algorithm for Calculating the Closest Points Between Two Macros

Again $D_{AB} = |x_{w1} - x_{w2}| + |y_{h1} - y_{h2}|$, where h_1, h_2, w_1, and w_2 are the closest points between two macros, is used to calculate the half perimeter distance between two macros. From the points discussed in the previous section, calculating the bounding box using the middle points seems to be the most suitable choice in which to calculate the bounding box to calculate the distance between two macros. But to validate whether this would be the correct choice, experiments were run to compare the two bounding box methods to see which method produces the best results. The experiments were run using the algorithm that will be discussed in much greater detail in section 4.8, but a quick description of the algorithm is that when two candidates are chosen for merging they are first tested to see whether the transformation would be detrimental to the overall system. If they pass this test the process is allowed to proceed onto the estimation stage to see whether the cost function is improved. The test is to see how close the macros are to each other, if they are close enough then the test is passed, the benefits of this test will again be discussed in section 4.8. The more accurate the

bounding box method, hence the half perimeter distance, the better the decisions will be in regards to whether candidates for merging will be allowed to proceed onto the estimation stage.



Source Figure 109

The values M10, M25 and M50 represent different constraints on which to accept merging transformations M50 are the most lax and M10 being the most strict. This concept will be discussed in much greater detail later in this chapter.

Figure 80. Graph showing the average % improvement of CP when using different net sources.

So to decide which bounding box method will be chosen graph shown in figure 80 will be used to see which bounding box method shows the greatest improvement to a design's architecture in terms of delay. The calculation of the % improvement of the Clock Period will be calculated using Equation 34 in Section 3.10. To deem whether macros are close enough to merge, three different limits are used in which the macros must be closer than this value to be allowed to proceed onto the next stage. The limits are unimportant, as they are only being used to see which point calculation method discussed in this section consistently outperforms the other. The average % improvement shown in the graph consistently shows that using the centre (middle) point for two macros produces the best bounding box (in terms of producing the best design optimality in terms of CP), for calculation of the half perimeter distance for all different limiting factors. Hence when calculating the half perimeter distance, the bounding box

shall be calculated using the centre points of the macros involved.

As stated earlier, the congestion of a circuit can affect the length of the interconnect, so when calculating individual interconnect lengths, congestion of the circuit needs to be considered. So the next section will introduce a value to represent the congestion of a circuit.

4.9 Congestion Metric Calculation

As discussed earlier, routing congestion can affect area and delay, where the higher the congestion the higher the impact will be. So a way of measuring this congestion is needed. Previous methods for analysing congestion were discussed and now those ideas shall be realised with a few additional ones. This metric shall be called the Congestion Metric (CM), and this will be used to influence Merging or Duplicating Functional Units.

As stated in section 4.5.3 only a global congestion measure shall be used to represent the effect of congestion on a transform. Also as previously discussed, due to the target architecture being the Xilinx Virtex Series, the routing channel supply does not change within this family of chips. Hence the routing channels can be treated as constant factors. When average interconnect increases so does the demand on the routing channels. So as the supply is constant, average interconnect can be used to represent the congestion. The current set up is that the average interconnect length is multiplied by the individual interconnect distance to form a new distance D that now reflects congestion as well. So as when the routing supply is already congested the larger the average interconnect value, the worse the congestion will become, hence making the path of the individual interconnects longer. An extension to the congestion measure would be to find a suitable normalising factor, which would have been pursued if more time had been available. Currently the normalizing factor is the average interconnect value obtained from the initial design architecture. The normalised average interconnect will then be multiplied by the individual interconnect distance to form D. So the more congested the circuit (i.e. > 1) the larger the distance will become and the less congested the circuit (i.e. < 1) the smaller the distance will become.

Another metric that was considered was how the percentage of a chip occupied affected the routing. Firstly, a merging transformation will be considered. If a chip is quite close

to being full, merging will allow more space for macros to be moved about; hence making it more likely macros will be placed in an optimum location. But if the percentage of the chip being occupied is high, then merging would not be wanted as the congestion will be high and merging will just increase the congestion as it will draw more interconnect into one area. Hence, these two factors cancel each other out, so this metric will not be used when calculating the congestion metric.

Before discussing how exactly the individual interconnect lengths between macros will aid transformations of a designs architecture during synthesis, a final metric shall now be added to the MOODS cost function. This metric will improve the functionality of the cost function through better understanding of the effects of transformations on the physical behaviour of a design.

4.10 Individual Interconnect Aware (IIA) Metric in Cost Function

The last metric to be added to MOODS is called Individual Interconnect Aware (IIA) and is concerned only with interconnects between the macros involved with the transformation and all their immediate neighbours. The metric will measure the difference between the average interconnect of these nets pre and post (estimate) transformation using the following equation (equation 4, section 2.7.2):

$$\Delta E = \frac{C_{estimate} - C_{current}}{C_{initial}}$$

where $C_{estimate}$ is the estimated cost after the transformation has been applied;

$C_{current}$ is the cost of the current design architecture before the transformation has been applied;

$C_{initial}$ is the cost of the initial design architecture. This value is used as a normalising factor in order to give a fair comparison with all the design criteria;

E is the energy of the system, where the delta sign signifies a change in state.

$C_{estimate}$ for IIA will be the average interconnect length of all interconnects that belong to the macros being brought in to replace the old macros removed by the transformation. $C_{current}$ for IIA will be the average interconnect length of all the macros

removed due to the transformation. And finally $C_{initial}$ will be the half perimeter of the bounding box that encompasses the entire design. To explain the derivation and reasoning behind the calculation of E of IIA, let us consider a Merging Transformation. If the two candidates for merging have been selected and the transformation has passed all tests (section2.7), satisfying the conditions for the transformations to be performed, the estimation of the design metrics such as area, delay are entered into the cost function as above. The IIA first needs to calculate the Average Interconnect Length (AIL) for all interconnects belonging to the modules being removed through optimisation. To calculate the AIL for a macro a, the following equation is used:

$$AIL_a = \frac{\sum_{b \in B} x_{ab}}{n} \tag{39}$$

Where B is the set of all macros connected to macro a and n is the cardinality of B. The average AILs for all the old macros is found and given to $C_{current}$, and this same method is used for the AIL of the new macros that would be formed (if the transformation is performed) and the value is given to $C_{estimate}$. Finally the normalising factor of $C_{initial}$, is the half perimeter of the bounding box that encompasses the entire design, before the transform is performed. The larger the area of the design, the larger the difference between the present IIA and the estimate IIA is needed, in order to have a large effect on the CF. The reason this is a good property can be seen in Figure 81. Figure 81(i) will have much more effect on the global interconnect, hence CP, as the routing demand within the bounding box requires a higher proportion of routing resources of the chip. This is compared to the bounding box in Figure 81(ii) where the proportion of the routing demand on the routing resources of the chip is comparatively small. To model this effect, the chip perimeter shall be used as the normalising factor $C_{initial}$.

This will then give a view of how this transformation is affecting the general system, not just the immediate macro or macros involved. By this we mean that a transform might look desirable when we are only considering the macro or macros directly involved, but when looking at the bigger picture the transform might be detrimental to the overall system, i.e. the transformation might be causing unfavourable effects on the respective macro's immediate network neighbours. Now that all the interconnect prediction metrics have been discussed, consideration on how the Quasi Exhaustive Heuristic should be influenced by these metrics needs to be made.

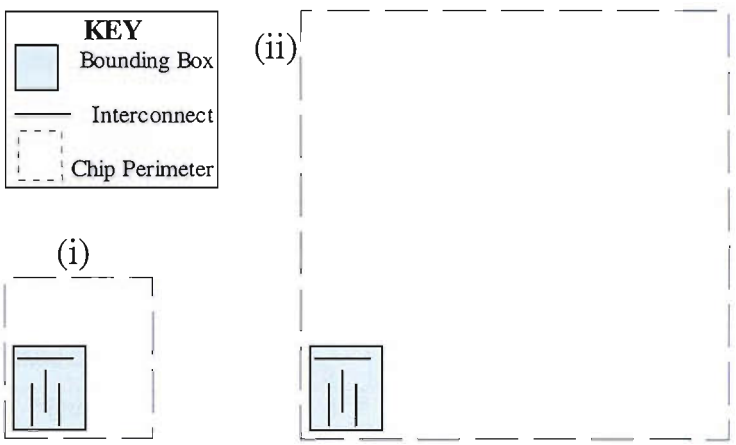


Figure 81. Diagram showing the effect of the same size bounding box on two different sized designs.

Scaling Factor	A0	A0.125	A0.25	A0.5	A1
Average	1.02	1.36	1.56	1.45	1.03
First Quartile	-0.19	0.11	0.11	0.11	-0.1
Median	0.52	0.59	0.59	0.12	0.36
Third Quartile	1.92	1.94	1.94	1.6	1.71

Data Source Figure 110 (Appendix). A represents the scaling factor used when calculating E of IA in conjunction with the cost function, where * is an integer. All results obtained when using interconnect prediction use AI in the Cost Function with scaling factor 25.*

Figure 82. Table representing test statistics of the % improvement of area of benchmarks that has been synthesised using different IIA metric scaling factors

With the AIL metric in the cost function (section 3.11) a scaling factor was used to increase the influence of the metric, this will be again used when using the IIA in MOODS's cost function. The influence of the IIA needs to be reduced as the IIA favours the duplication of functional units significantly compared to merging functional units, as generally, if two functional units are merged this stretches the interconnects directly involved in the transformation. Conversely if a duplication transform is being performed the interconnects should be reduced as duplication allows the duplicated macros to be separated and placed in a closer proximity to their network neighbours. So if the IIA is left with too much influence, only duplication transforms will be performed which will have a detrimental affect on the optimality of the area of a design. To find

the most suitable scaling factor, the QE algorithm was run with the AI in the cost function with the scaling factor 25 chosen in the last chapter and then a range of scaling factors for the IIA metric starting with 1 and then reducing.

Scaling Factor	A0	A0.125	A0.25	A0.5	A1
Average	28.02	33.41	34.38	34.43	32.76
First Quartile	7.97	5.34	5.34	5.34	13.37
Median	19.29	35.44	35.44	41.25	32.06
Third Quartile	42.2	42.73	47.3	49.78	46.6

*Data Source Figure 111 (Appendix). A * represents the scaling factor used when calculating E of IA in conjunction with the cost function, where * is an integer. All results obtained when using interconnect prediction use AI in the Cost Function with scaling factor 25.*

Figure 83. Table representing test statistics of the % improvement of CP of benchmarks that have been synthesised using different IIA metric scaling factors

When using IIA in the cost function, Figure 82 shows that the metric is beneficial for reducing the area of designs. The first quartile only drops below 0 where the scaling factor is 1. In Figures 82 and 83, scale factors 0.25 and 0.5 are very close; they further improve the CP and Area. But as our main focus is on delay, a scaling factor of 0.5 shall be chosen for IIA, as it has the best CP improvement on average (34.43%) and as we are looking for consistency this is the best candidate. But as the average improvement of CP is so close further investigation on which Scaling Factor is best would be beneficial.

4.11 Interconnect Predictors Influencing Transform Candidates Selection

All the metrics now made available shall be used to aid decision-making when the Quasi Exhaustive Heuristic (QE) algorithm is performed.

4.11.1 QE with Interconnect Prediction

Access to substantially more interconnect layout information is now available. This information will now be incorporated into the QE heuristic. More in-depth design information will enhance the QE's decision making so that it can choose more favourable transform actions to increase the optimality of a design passed through

MOODS. The algorithm is split into three main parts. First there is compacting of the control graph, and secondly there is merging functional units on the data path. If desired, a final duplication phase can be performed on the data path. The compaction of the control graph was explained earlier, as it is part of the QE. The compaction of the data path will be the same function as described for the QE, except that the selection procedure for the candidates of a datapath transformation will depend on estimated physical characteristics. When duplication is required, a simple function will look through all eligible macros in the datapath and again will use a selection procedure based on estimated physical characteristics. Both selection procedures will now be discussed.

4.11.2 Selection Criteria

In order to select appropriate candidates for either merging or duplication, some form of selection criteria according to area and delay is required. But first, they need to be eligible for merging or duplication. To test which, a function called test_trans [20] is used. Once a node or nodes have been chosen and found to be eligible for merging or duplication they then are tested for favourable properties with which to perform the respective transform. The selection criteria will be performed as follows, depending on which transform is being performed.

4.11.2.1 In the case of Merging

For this test, the clock period or delay of the nodes will not be considered. For the merging test only the distance between the two macros that are being considered for merging is of interest. For the distance between the macros, the half perimeter rule will be used, as we are only considering two pin nets. In this process, the aim is to look at the two macros under consideration and if they are close enough, then the test is successful; if not then the test is unsuccessful. Hence merging will be prevented between the two macros being tested. In order to know when macros are close enough, an upper limit for the distance D between these macros is needed. If the distance between the macros is greater than this upper limit M, then the macros are deemed too far away to make merging feasible.

The Distance D between two macros is measured as shown in section 4.6.4. Once this distance has been calculated, congestion needs to be considered. If the congestion is high, the nets belonging to the macros involved in the transformation will generally be a

bit longer; a merging transformation will increase these nets even further. Merging two data path units will generally increase the interconnect lengths of the nets belonging to the macros involved in the transformation. To model this behaviour, *D* is multiplied by the average interconnect length (discussed in section 4.7), hence the larger the average interconnect length the larger the value of *D,* hence making it harder for *D* to fall under the limit. This factoring of *D* means that the candidates have to be even closer for the merging transform to be performed, if congestion is high, compared to a design with low congestion. Multiplying by the average interconnect length will be beneficial as the higher the congestion of a circuit; the less desirable it is for merging to take place.

Merge Limit		M10	M25	M50	M75
Average	1.02	-13.11	-12.94	0.97	1.04
First Quartile	-0.19	-25.58	-27.49	-0.19	0
Median	0.52	0.11	0.47	0.67	0.67
Third Quartile	1.92	1.6	2.27	1.82	1.94

(i)Data Source Figure 112 (Appendix). % improvement of Area

Merge Limit		M10	M25	M50	M75
Average	28.02	43.36	42.12	38.7	37.92
First Quartile	7.97	35.21	32.3	23.79	23.79
Median	19.29	41.89	40.95	34.75	35.01
Third Quartile	42.2	52.68	50.59	44.69	45.15

(ii)Data Source Figure 113 (Appendix). % improvement of CP

M represents the limiting factor used when deciding whether to merge two data path units, where * is an integer. All results obtained when using interconnect prediction use AI in the Cost Function with scaling factor 25.*

Figure 84. Table representing test statistics of the % improvement of physical metrics of benchmarks that has been synthesised using restrictions on candidates nominated for Merging.

The upper limit M now needs to be decided. This value will be decided by running multiple experiments and the results of these experiments are found in Figures 84(i)(ii). Figure 85(i) shows the % improvement in Area of a design when using different M values. The lower the M value, the less likely merging will be performed, this can be seen when looking at the highly detrimental affect on the area design when using a very small M value, as the smaller the M, the more merging transformations will be prevented, which would have reduced the area. But the higher the M value, the less detrimental the effect on the area becomes, until using the Distance Metric Calculation

actually improves the area on average, even more so than just using the AI metric in the Cost Function (CF). When considering CP, Figure 84(ii) shows that the tighter the restriction on merging, i.e. the lower the value of M, the higher the improvement in the CP. But as there was a large increase in area in Figure 84(i) when M dropped below 50, M50 is chosen to be the limit for the distance at which two functional units may merge as it produces designs with a small average CP but does not increase the area.

4.11.2.2 In the case of Duplicating

Duplication is used for undoing transforms that were made earlier in the optimisation process, where unforeseeable badness could not be accounted for. The lack of foresight is due to design transformations being thought good for the design objective function, at the current time of execution. But changes in the design architecture later on in the optimisation process may reduce the optimality of previous transforms. When trying to find nodes to duplicate, this will be an exhaustive search after every completion of the Compact_DP in QE. An exhaustive search is not too slow as only one node at a time can be chosen for duplication. If a data path unit is duplicated, the resulting data paths cannot be duplicated so there is no need to test them. Hence the new data path units do not add to the search space. When duplicating, only one node at a time will be considered. If a node is chosen, it will be split up into multiple nodes, but as stated earlier, this can have its drawbacks as well as its benefits.

Again, a test needs to be provided in order to make the best decisions. The test is similar to the merging test, in which candidates are tested to see how, if duplicated, the transformation will affect interconnect length. But this time to measure whether a candidate is appropriate for duplication, the average interconnect distance of all the macros that are involved with the transformation are calculated. The Average Interconnect Length (AIL) is calculating for the data paths that will be removed. AIL is used, as a smaller AIL means interconnects that are connected to the macro that is the candidate for duplication will be less spread out. So duplicating the functional unit will not greatly aid the interconnect lengths that are already small. But if the AIL of the macros connected to the candidate for duplicating is large then this means the interconnect belonging to these macros is spread out, this means that duplicating the candidate will allow the resulting data path units to be placed closer to the data path units that were connected to the original data path (shown in Figure 15, section 2.9.2), hence reducing the average interconnect of all the data path units involved in the

transformation. Hence we require that the AIL is larger than a minimal limit U so as to know when a transformation is beneficial in terms of interconnect length.

Average Interconnect length	M0+U0	M50+U5	M50+U10	M50+U25	M50+U50
Average	1.02	-3.9	-0.3	-0.32	-0.43
First Quartile	-0.19	-0.19	-0.5	-0.5	-0.5
Median	0.52	0.67	0.59	0.59	0.59
Third Quartile	1.92	1.82	1.94	1.94	1.54

(i)Data Source Figure 114 (Appendix). % improvement of Area

Merge Limit + Duplication Limit	M0+U0	M50+U5	M50+U10	M50+U25	M50+U50
Average	28.02	33.42	35.24	35.43	33.94
First Quartile	7.97	12.42	22.27	21.5	13.47
Median	19.29	34.85	35.15	35.15	32.55
Third Quartile	42.2	44.48	44.1	44.1	44.1

(ii)Data Source Figure 115 (Appendix). % improvement of CP

M represents the limiting factor used when deciding whether to merge two data path units, where * is an integer. All results when using interconnect prediction use AI in the Cost Function with scaling factor 25. Limiting factor for Merging of 50*

Figure 85. Table representing test statistics of the % improvement of physical metrics of benchmarks that has been synthesised using restrictions on candidates nominated for Duplication

Again the limiting factor U needs to be decided. This value is will be decided by running multiple experiments and the results of these experiments are found in Figures 85(i)(ii). From Figure 85(i), even though duplication does degrade the area on average, the degradation is very minimal; except when there is a scaling factor of 5, but this means a lot of duplication transformations are allowed to go ahead. The CP is further improved the larger U is, due to the algorithm becoming more selective, until U becomes too large and restricts the heuristic too much. Using Figures 85(i)(ii), a limit of 25 or 10 are both valid limits as both produce designs with good CPs. But encouragement of duplication is not desirable unless necessary, as the size of the design could increase substantially depending on the transformation. So a limit of 25 shall be chosen for U.

4.12 Summary of Results

All the physical characteristics that will be estimated in order to improve design exploration in MOODS have now been discussed. Through additional metrics implemented in the Cost Function that represent interconnect properties, it has been shown that these metrics in conjunction with the QE algorithm increase the optimality of a design in terms of Clock Period (CP). This is due to the metrics allowing better decisions to be made, when selecting which candidates will be best for transformations within synthesis, with respect to reducing the interconnect distances, and hence CP. The CP should then be reduced through the reduction of interconnect distances. The next stage is to experimentally validate the use of interconnect prediction with some more benchmarks. The results of these further experiments can be seen in Figures 86 – 91. The design procedure can be found in Appendix 7.6, which includes information on all the benchmarks used. The benchmarks are chosen so that there is a variety of different design architectures. 2 different designs will be synthesised with two different CP targets set during synthesis (20ns and 50ns). Each design will be synthesised using different levels of interconnect prediction to guide decision making during synthesis. Each level of interconnect prediction has been discussed in the last chapter and this chapter and they are as follows:

N = No interconnect Prediction is involved during Synthesis.

A = Average Interconnect Metric is used in MOODS Cost Function (Section 3.11).

B = Methodology A and restriction on merging (Section 4.8.3.1).

C = Methodology B and duplication is introduced with restrictions (Section 4.8.3.2).

D = Methodology A and Individual Interconnect Aware (IIA) (Section 4.7.1).

E = Methodology B and IIA.

F = Methodology C and IIA.

Firstly % improvement of area will be discussed (shown in Figure 86). When using method A the area is increased slightly on average compared to a design produced using no interconnect prediction to guide synthesis *N*. B again increases the area, but this can be expected as the constraint *M* is not allowing some merging transformations to be performed, hence in some cases not allowing a reduction in area. C then increases area further as now using duplicating functional units will cause the area to increase. Method D is now using the IIA metric; this metric will restrict merging as discussed in section 4.7.1 because merging in most cases causes the IIA metric to increase. Hence the area degradation increases even further compared to just using AI in the Cost

Function. E increases the area further on average due to the merging restriction, but method F reduces the increase in average % difference of the area. One reason for this could be that the IIA metric is calculated in the same way as the value used to decide whether interconnect directly involved in the potential transformation are stressed (long) enough for the transformation to be beneficial. In all cases the average % increase in area for designs using interconnect prediction during synthesis is a slight increase but nothing substantial.

Heuristic Method.	A	B	C	D	E	F
Average	-0.68	-1.17	-2.43	-1.63	-2.54	-1.6
First Quartile	-1.65	-2.06	-4.84	-1.65	-4.78	-3.77
Median	0.47	0.13	-0.27	0.12	0.12	0.12
Third Quartile	1.54	1.23	0.76	1.58	1.1	1.26

Data Source Figure 116 (Appendix).

Figure 86. Table representing test statistics of the % improvement of Area of benchmarks using different levels of interconnect prediction during synthesis.

Heuristic Method.	A	B	C	D	E	F
Average	0.83	0.48	1	1.5	1.11	1.41
First Quartile	-1.98	-1.91	-1.31	-2.77	-1.64	-0.76
Median	0	0.74	1.67	-0.07	1.6	1.8
Third Quartile	3.88	3.69	3.51	3.17	4.58	5.62

Data Source Figure 117 (Appendix).

Figure 87. Table representing test statistics of the % improvement of Average Interconnect Delay of benchmarks using different levels of interconnect prediction during synthesis.

Using Figure 87, the % improvement of the average interconnect delay obtained post PAR can be seen, on average, to increase when using interconnect prediction during synthesis for all methods. Method A shows a small improvement, but when restriction on merging is implemented this drops, but then increases again when duplication is implemented. The improvement of the Average Interconnect Delay steadily increases through designs A to C. This pattern then repeats for D through to E, but shows that using the IIA metric in the cost function in MOODS in conjunction with AI metric is very beneficial.

Heuristic Method.	A	B	C	D	E	F
Average	-2.33	-1.33	-1.92	0.19	1.65	1.68
First Quartile	-14.16	-10.33	-19.2	-7.93	-9.15	-8.97
Median	-0.38	1.39	1.4	0	4.55	1.46
Third Quartile	8.34	8.89	9.08	8.27	9.56	8.47

Data Source Figure 118 (Appendix).

Figure 88. Table representing test statistics of the % improvement of Worst Net Delay of benchmarks using different levels of interconnect prediction during synthesis.

Heuristic Method.	A	B	C	D	E	F
Average	-2.64	-2.8	-1.53	-0.76	-0.49	-0.15
First Quartile	-9.35	-9.35	-8.25	-5.7	-4.98	-5.06
Median	-1.96	-0.93	-0.22	0	0.83	1.51
Third Quartile	2.52	6.72	5.11	5.26	7.18	5.76

Data Source Figure 119 (Appendix).

Figure 89. Table representing test statistics of the % improvement of Average Worst Ten Net Delay of benchmarks using different levels of interconnect prediction during synthesis.

Heuristic Method.	A	B	C	D	E	F
Average	18.82	21.73	22.28	18.42	21.6	23.03
First Quartile	2.98	3.17	4.88	1.18	5.3	5.3
Median	13.28	17.44	20.44	13.45	17.13	18.02
Third Quartile	22.27	33.52	33.96	32.67	32.32	33.77

Data Source Figure 120 (Appendix).

Figure 90. Table representing test statistics of the % improvement of Minimum Clock Period Delay of benchmarks using different levels of interconnect prediction during synthesis.

The average % improvement of the Worst Net Delays shown in Figure 88 steadily increases, the higher the level of interconnect prediction used to guide HLS. Where there is a degradation in the Average Worst Net Delay and where IIA is introduced into the cost function, there is an improvement in the average % difference. This same pattern is seen for the average % difference of the 10 worst net delays in Figure 89, except that the average % difference never shows an improvement. So this shows

that the IIA metric is a good metric for reducing the delay of global interconnect, as it reduces the degree of degradation to the global nets.

Heuristic Method.	A	B	C	D	E	F
Average	21.83	24.65	24.74	20.8	24.47	25.05
First Quartile	2.98	5.34	4.88	1.18	5.3	5.3
Median	16.3	19.3	20.44	13.45	17.73	18.02
Third Quartile	40.02	41.75	39.36	44.2	37.06	35.79

Data Source Figure 121 (Appendix).

Figure 91. Table representing test statistics of the % improvement of Total Delay of benchmarks using different levels of interconnect prediction during synthesis.

The next two Figures, 90 and 91, are where the expected highest metric improvement should be seen, as this is the main focus to reduce the CP, and hence the total delay by using interconnect prediction. Figure 90 shows there is a large improvement in the CP when using Interconnect Prediction; in the majority of cases the CP is improved and if there is degradation in the CP it is minimal. The average % improvement increases steadily from A to B to C, showing that the higher the level of interconnect prediction used during HLS, the better the CP will become. This same pattern repeats with methods D, E and F. But there is a drop in the % of improvement from A to D. But as discussed earlier, using the metric IIA in the cost function is most beneficial when using duplication, as it has the highest average % improvement compared to any other method. Throughout the table the first quartile never drops below 0, while the third quartile is extremely high. This shows that when using interconnect prediction, it is very unlikely the minimal CP will degrade, while it is very likely the minimal CP will be improved significantly. The length of CP alters insignificantly between designs that have the same criteria during synthesis (delay, area and CP target), hence Figure 91 repeats the pattern of Figure 90. The results presented in this section show that Interconnect Prediction is a valid HLS methodology for providing optimal design in terms of CP.

4.13 Does a Better Partitioning Algorithm Improve Optimality?

The next test is to see whether using an iterative improvement algorithm to improve the cut set produced by the Greedy algorithm during Recursive Bi-Partitioning, improves the eventual optimality of a design, compared to just using the Greedy algorithm by itself to perform partitioning. The global net statistics in Figure 92(i) seem to improve

when using an iterative improvement-partitioning algorithm, but the CP is better without. Figure 92(ii) shows that to obtain the highest level of optimality the Greedy algorithm should be used by itself, which is useful as this requires a lot less computation.

Metric Type	AREA	AREA	AI	AI	WN	WN	W10	W10	CP	CP
Partitioning Method	G	B	G	B	G	B	G	B	G	B
Average	**-1.17**	-1.6	0.48	**0.88**	-1.33	**0.86**	-2.8	**-0.69**	**21.73**	19.36
First Quartile	-1.75	**-1.26**	**-1.71**	-2.38	-9.31	**-7.02**	-8.53	**-8.25**	**4.8**	3.49
Median	0.19	**0.28**	1.39	**1.75**	**1.39**	-0.11	-0.93	**-0.29**	**17.7**	13.67
Third Quartile	**1.3**	1.05	3.69	**4.71**	**9**	7.99	6.77	**8.96**	**33.86**	30.87

Metric Type	AREA(G)	AREA(B)	CP(G)	CP (B)
Average	**-1.17**	-1.6	**21.73**	19.36
First Quartile	-2.06	**-1.63**	**3.17**	0.79
Median	0.13	**0.24**	**17.44**	13.45
Third Quartile	**1.23**	1.02	**33.52**	30.65

Data Source Figure 122 (Appendix). G represents circuit partitioning undertaken by Greedy Algorithm (G) and B represents circuit partitioning undertaken by Greedy algorithm and the Modified Kernighan Lin Algorithm. AI = Average Interconnect Delay, WN = Worst Net Delay, W10 = Average Worst Ten Net Delays, CP = Clock Period

Figure 92. Table showing test statistics of the % improvement of the physical metrics of a design using interconnect prediction during synthesis derived from different partitioning algorithms.

4.14 Simulated Annealing with Interconnect Prediction

As time was limited, only AI in the cost function using Simulated Annealing (SA) to perform the optimisation during HLS was tested (with a scale factor of 1). As can be seen from Figure 93 there is an improvement across the board, except for the first quartile for the average worst ten net delays. This shows that even with SA, which is a very random process, interconnect prediction can improve the optimality of the physical metrics once a design has been placed on an FPGA.

Metric Type	AREA	AI	WN	W10	CP
Average	0.5	0.6	0.36	1.19	4.58
First Quartile	0	0	0	0	0
Median	0	0	0	0	0
Third Quartile	0.07	0.72	4.19	2.89	4.62

Data Source Figure 123 (Appendix). *Matrix_calc has been left out as the design did not fit on a Virtex chip. AI = Average Interconnect Delay, WN = Worst Net Delay, W10 = Average Worst Ten Net Delays, CP = Clock Period*

Figure 93. % Improvement of the physical metrics of a design using interconnect prediction during synthesis within MOODS. The Simulated Annealing Heuristic was used for the optimisation. AI with a scaling factor of 1 is in the Cost Function and a merging limit of 50.

4.15 The Advanced Encryption Standard (AES)

The AES is a cryptographic algorithm called Rijndael. Rijndael was chosen for AES because of its combination of security, performance, efficiency and ease of implementation.

4.15.1 The Structure of Rijndael

A thorough explanation of Rijndael can be found from [52]. Rijndael has a variable key and block length, the general values of these lengths are 128, 192 and 256. But the key or data length can be any value as long as it is a multiple of 32. The algorithm was designed so that it acts on bytes rather than on single bits, and the other main design considerations were:

- Resistance against all known attacks;
- Speed and code compactness on a wide range of platforms;
- Design simplicity.

4.15.2 Implementation of Rijndael

The algorithm is designed for encryption rather than decryption. Thus encryption requires less computation (hence is smaller and faster) than decryption. An example application of the encryption algorithm is smart cards, and this is where hardware designs are very applicable. Smarts Cards encrypt data and send out the information to stationary receptors; hence the size is not so much of an issue. This algorithm has been written with fixed and small (128 bit) key/block length, thus reducing the amount of components that are needed. The target architecture is the XCV1000 chip. The design is

semi-pipelined: after half the key has been expanded, the next data block is passed through, so as to increase throughput. A fully pipelined design was constructed but the number of slices exceeded the number of slices available on the largest chip in the Virtex series, unless the block RAM was used on the Virtex chip, but MOODS does not support this option.

Heuristic	AREA	AI	WN	W10	CP	1st Output #CPs	Repeated Output #CPs	Clock Freq.	Rate (Mbs)
QE	12080	3.64	18.14	15.32	23.84	35	18	41.95	298.28
QE + A	11900	3.76	16.29	14.65	23.9	35	18	41.85	297.59
QE + B	11834	3.6	17.78	15.06	22.96	35	18	43.56	309.73
QE + C	11834	3.6	17.78	15.06	22.96	35	18	43.56	309.73
QE + D	12028	3.65	15.97	14.2	24.87	35	18	40.21	285.94
QE + E	11897	3.58	14.54	13.6	23.95	35	18	41.76	296.93
QE + F	11877	3.71	17.94	15.26	23.9	35	18	41.84	297.55

AI = Average Interconnect Delay (ns), WN = Worst Net Delay (ns), W10 = Average Worst Ten Net Delays (ns), CP = Clock Period (ns).

Figure 94. Table showing the % improvement of the physical metrics of Rijndael using varying levels of interconnect prediction during synthesis within MOODS.

Figure 94 shows that in all cases the area, worst net delay and average worst ten net delays reduce when interconnect prediction is used during synthesis. Also it can be seen that restricting the candidates available for merging and duplication decreases the clock period, while also reducing the area, compared to just using the metrics in the cost function. The highest throughput is achieved when the Average Interconnect Length metric is used in the cost function. The average interconnect delays do not vary significantly.

Chapter 5 Conclusion and Future Work

5.1 Conclusion

The work presented in this thesis has shown that interconnect prediction is a beneficial process in which to significantly improve the CP and hence the total delay of a design. At the same time as improving the CP of a design, the area is also improved in the majority of cases. The first step in testing whether interconnect prediction was possible during High Level Synthesis (HLS), was to compare the predicted Average Interconnect Length in MOODS and the actual Average Interconnect Delay post Place and Route (PAR) and to see if the correlation between them was high enough to show a linear relationship, and hence to allow accurate prediction of the average interconnect of a design during HLS. Average Interconnect Length was chosen as it gives a global measure of the interconnect topology. When used in conjunction with a floorplan it can be obtained freely, as partitioning is needed to obtain a floorplan and also to calculate the Rent Exponent used in the average interconnect equation. It was then shown that this average interconnect length obtained during HLS correlated very well with the actual average interconnect delay post PAR stage in Xilinx. This correlation meant that there was a strong linear relationship between the predicted values and actual values using 23 different benchmarks.

Next the hierarchical information obtained through partitioning the final design produced by MOODS improved the optimality of the design when used to guide PAR. This showed that the interconnect prediction metrics that are fundamentally based on the circuit partitioning were accurate if Xilinx optimises a design sufficiently well. If the hierarchical information provided made the designs worse, then it would show that the circuit partitioning is forming sub-optimal groups and is not creating an optimal slicing tree. Once this relationship was found, this meant that the predicted Average Interconnect Length (AI) Metric could be used to accurately model the actual average interconnect during design space exploration. The next stage was then to prove that AI was a good metric when used in the cost function during synthesis, in terms of producing a design that reduced delay but did not adversely affect any other design characteristics. As the results showed the Clock period (CP) and hence the total delay

was significantly reduced, while also improving the majority of the design's overall physical characteristics. Then a restriction on which candidates were eligible for the merging transform was applied, defined in terms of distances between the candidates. This improved the delay of most designs even further. Finally a duplication transform was introduced to the Quasi Exhaustive Optimisation algorithm, again with a restriction on the candidate proposed for duplication. This methodology, in conjunction with the Individual Interconnect Aware Metric in the MOODS Cost Function, produced the highest level of optimality in terms of delay.

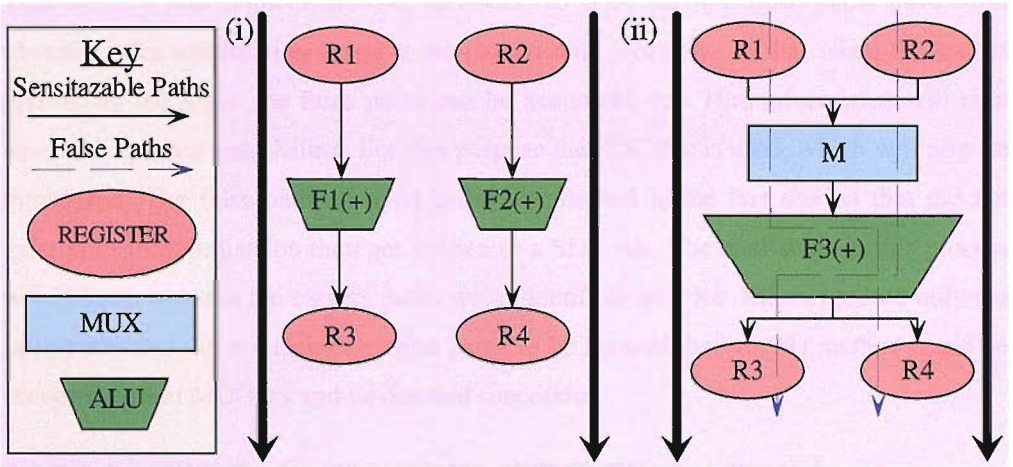
5.2 Future Work

5.2.1 Timing-Aware Circuit Partitioning

For future work, more timing-driven partitioning could be implemented, where weights according to the hierarchical level could be assigned. To decrease the length of the longer wires the weights of the external nets can be increased at the top end of the hierarchy. One possible avenue is to increase the weight of the nets that belong to the critical path, so as mimic the minimisation of the critical path during PAR. Another approach could be to have higher weights at the beginning of the partitioning then systematically reducing them so as to reduce global nets. The final approach could be to weight a net every time it is cut during recursive partitioning, so as to reduce the number of times the net is cut hence reduce the net length.

5.2.2 False Paths

A major part of this thesis concerns delays caused by interconnect, but delay can also be attributed to false paths. Obviously this is not a real delay as the name suggests. A False Path is due to merging of functional units: a path between two registers that did not exist before merging, but exists after the transformation will be a false path. This path will not affect the functionality of the design, but when trying to optimise a design's architecture that is dependent on the clock period, this false path can cause a timing error. This error occurs because this path could appear to be the longest path. To avoid this error the functionality of the design can be used to discover the false paths.



(i) Independent Paths with Two Sensitisable Paths;
(ii) Shared Components with Two Sensitisable Paths Two False paths.

Figure 95. Creation of False paths.

Figure 95 demonstrates how resource sharing can create false paths. From Figure 95(i), it can be seen that there is no resource sharing, which prevents false paths from forming. If the delay was estimated, the correct delay should be derived assuming that the estimated delay measurement was 100% accurate. In Figure 95 (ii) resources are shared. Where the first diagram had two true paths, which were R1 - F1 – R3 and R2 - F2 – R4, due to resource sharing in Figure 95 (ii) there are 4 paths of which 2 are false: they are R1 - M - F3 - R4 and R2 - M - F3 - R3. When the delay is estimated, 4 paths would be measured which takes longer to compute. But the more important factor is that one of the false paths could be longer than the sensitisable paths, which would give an incorrect delay value of the circuit.

Firstly a map of all the datapaths is constructed at the beginning of the synthesis process before any transforms have been performed. This map just shows how the datapath is connected. At this stage there will be no false paths. This map is then kept until the optimisation has been completed, and the final design architecture has been chosen. The next step is an easy process where the first map constructed from the initial design architecture is compared to the current design architecture. A neighbourhood search is used for the comparison. If a path is present between two registers which was not there in the beginning, then this is a false path. If Figure 95 is used the paths in (i) would be simply compared to the paths in (ii). All paths that were not found in (i) but are found in (ii) are false paths.

This method was implemented in MOODS, so now that the false paths have been identified this information needs to be passed onto Synplify, so that when the tool is optimising the logic, the false paths can be accounted for. This information will then need to be passed onto Xilinx. For this purpose the SDC file is used, which will now be introduced. The false paths derived using the method in the last section that did not exist prior to optimisation then get written to a SDC file. The next stage in this process would be to test that the correct paths were identified, and that Xilinx ignored only the false paths and did not cause any true paths to be ignored, before this method could be accepted within MOODS and be deemed successful.

5.2.2.1 Synplify Design Constraints (SDC) File Application

The SDC is Synplify's constraint file. This file shall be used to inform Synplify, and then Xilinx which paths are False Paths, hence these paths are ignored when timing analysis in Synplify and Xilinx is performed, hence allowing an accurate measurement of the critical path delay, and hence clock period to be measured. The problem with using the SDC was that when these false paths are identified to Synplify, those nets are then ignored when estimating the delay. But when passing these paths onto Synplify the paths have to start and finish in a Register or IOB. But as Figure 95 shows, this can cause Synplify to ignore nets that belong to true paths.

5.2.3 Finalise the Floorplan

The floorplan during HLS in MOODS has been shown to be sufficiently accurate for interconnect length estimation, but it would be interesting to see whether reducing the dead space on the floorplan would increase the accuracy of the interconnect prediction obtained from the floorplan. As discussed in section 2.13.1, Lai and Wong show that a slicing tree can represent a non-slicing floorplan [21]. They use XY-compaction to transform a slicing tree into a non-slicing floorplan. XY-compaction pushes every module to the left until no module can move any more to the left, then every module is pushed downwards until no module can move downwards. This is repeated until every module cannot be moved either to the left or downwards. Alternatively the initial push could be downwards then to the left which might give a different floorplan. The drawback of this method is that it might separate modules, which were shown, by partitioning, to be better placed together, to minimise the Wire Length. Hence to obtain the final placement, an XY compaction algorithm can be introduced to reduce the dead space. To ensure minimal degradation of the original floorplan solution in terms of

Wire Length, the XY compaction will be directed towards the middle of the circuit as shown in Figure 96.

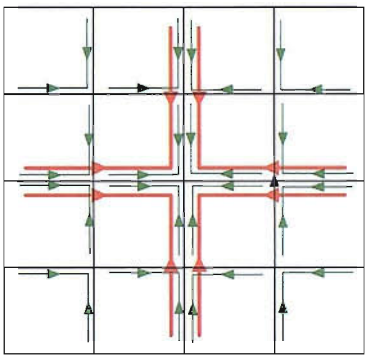


Figure 96. Direction of XY Compaction.

5.2.4 Iterative Recursive Partitioning

Good quality results have been obtained using PLE, showing that using PLE will improve a design's optimality. But in order to make this approach more desirable, a method for speeding up the partitioning of the circuit netlist is needed, as this is the most expensive part of our physical estimation process. The recursive partitioning needs to become iterative, because many different architectures for a design are analysed during synthesis in MOODS. Every time the Rent exponent is needed for a different architecture of a design, the whole process of recursively bi-partitioning (RBP) the entire circuit is repeated, even if only a small part of the design is altered. As mentioned earlier, MOODS changes the design architecture iteratively when searching through the design space, hence the different design architectures change little between iterations. Hence when calculating the Rent exponent, repeating the entire process of recursively bi-partitioning the circuit from scratch would be impractical. A structure needs to be set up so that if part of a design's structure is altered, then only the affected region needs to be re-partitioned. When the partitioning phase can be performed iteratively, then the derivation of the AI will become a great deal quicker and much more viable for HLS.

In order to achieve this speed up in PLE, a short cut needs to be thought of that will partition a circuit's netlist in the fraction of the time. The first method could be a look-ahead strategy, predicting how a merging or a duplication transform would affect the hierarchical structure. This might be possible if the network has a very low interconnection and the modules in question do not occupy a large proportion of the total chip space. These properties would mean that if the transform in question was

performed, the difference in the netlist would probably have very little effect on the rest of the hierarchy, but this could not be relied on. The second approach is to reduce the number of partitions needed to obtain enough points for the Rent Exponent p calculation. [61] obtains impressive results and shows that p can be calculated with a small number of partitions without a degradation in accuracy. This method would obviously speed up the partitioning stage to obtain p but is still too slow, and would not help floorplan construction. Also forming an optimal cut at the top of the hierarchical tree takes up a significant amount of the RBP, due to the number of possible solutions increasing dramatically with size of group being partitioned. The third approach could be using an iterative floorplan algorithm such as SA or Force directed placement. But this would still mean partitioning the floorplan to obtain p. If this method was used when calculating the average interconnect it would be best to just enumerate all the paths. This method would be fairly fast but would greatly depend on the initial solution, so as not to have to perturb the initial floorplan too much. The final approach takes advantage of the deterministic nature of a hierarchical tree. As recursive partitioning has a deterministic effect, the cuts at the top of the hierarchical tree in Figure 97 will affect the cutsets further down the tree. Hence to deduce the hierarchical structure of a design architecture if a transform is performed, knowledge of how the newly formed modules affect the whole hierarchy is needed.

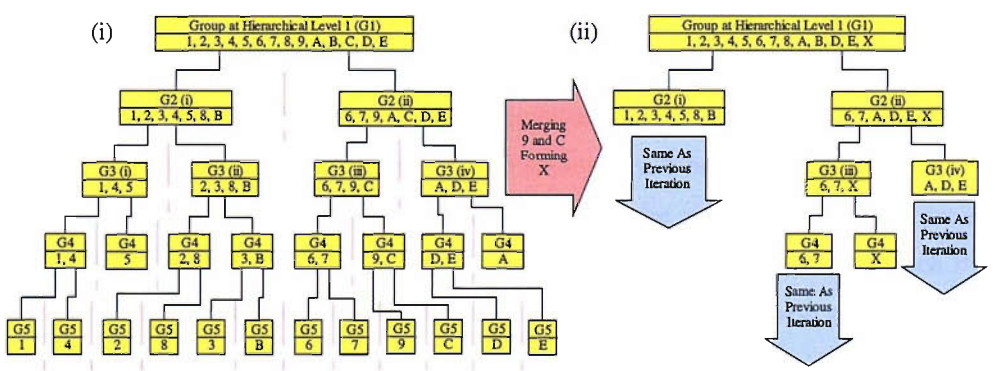


Figure 97. Deterministic Iterative Partitioning Algorithm.

For simplicity, we are going to presume that when merging module 9 and module C, one module replaces them, which will be called X. The dashed lines in Figure 97(i) represent the nets that are removed once a partition is formed. For example, the two groups at hierarchical level 2 have no more communication with each other once

partitioned. When modules 9 and C are merged, repartitioning of the design is needed. But if after partitioning the initial group, G2 (i) is the same as the previous iteration, the partitions beneath this group will be the same. After the first cut, nets belonging to 9 and C (now X) do not have any effect on G2(i) subgroups. As in the previous iteration those nets will be removed. This means no more partitions of this group are needed. The next stage is to partition G2(ii), this time G3 (iv) is the same as G3 (iv) from the previous iteration. Again no more partitions are needed. If this design was very big and the first cut formed an identical group to the previous design architecture, then this would significantly reduce the amount of computation needed. If an identical group is not formed then this would suggest that the design architecture has changed too much, so it would be best to re-partition to obtain an accurate floorplan.

To maximise the potential for this event to happen, the initial partitions shall be chosen in the following manner. The two groups formed from the previous design iteration at the respective hierarchical level will be used as the initial groups for the min cut algorithm in the current iteration. But the initial partition will be modified so as to remove and add the appropriate nodes. So as to leave one side unchanged, if feasible new modules are placed back into the same partition the merged modules were taken from.

Due to the min cut solution being dependent on the initial partition, the more similar the initial partitions, the more stable the results will be. As the results will be more stable, the comparisons of PLE between design iterations will become more accurate. Hence using the last method described means maximising the potential for the maximum amount of the design to have the same hierarchical structure, hence produce a similar slicing tree. Obviously if the majority of the slicing tree remains unchanged, then the resulting analysis of the new tree structure, caused by the current design architecture, will be more dependent on the part of the design that has been altered. Hence this stability makes the analysis more efficient when deciding if the new architecture is beneficial or not.

References

1. B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning of electrical circuits", *Bell Systems Technical Journal,* pages 291–307, 1970.
2. W. E. Dougherty and D. E. Thomas, "Unifying Behavioral Synthesis and Physical Design", in *37th Design Automation Conference,* pages 756-761, 2000.
3. C. Fiduccia and R. Mattheyses, "A linear time heuristic for improving network partitions", in *Proc. Design Automation Conf,* pages 175–181, 1982.
4. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines", *Journal of Chem. Phys.,* vol. 21, pages 1087-1092, 1953.
5. Wackerly, Mendenhall, Scheaffer, "Mathematical Statistics with Applications", Wadsworth Publishing Company, 1996.
6. A. B. Kahng, "Toward Accurate Models of Achievable Routing" *ISPD'02,* April 7-10, 2002.
7. D. Stroobandt, "A new Design Methodology Based on System-Level Interconnect Prediction", *Interconnect hierarchy: technology and system design IMEC workshop,* April 24, 2002.
8. J. Dambre , P. Verplaetse , D. Stroobandt , J. Van Campenhout, "Expanding Rentian Analysis: Getting more out of Donath's hierarchical model for interconnect prediction", in *Proc. of the 2002 international workshop on System-level interconnect prediction,* April, 2002.
9. A. Ranjan, K. Bazargan and M. Sarrafzadeh, "Fast Hierarchical Floorplanning with Congestion and Timing Control", *IEEE International Conference on Computer Design (ICCD),* pages 357-362, September 2000.
10. S. Bodapati, F. N. Najm, "Pre-layout estimation of individual wire lengths", in *Proc. of the international workshop on System-level interconnect prediction,* pages 93-98, April 08-09, 2000.
11. M. Sarrafzadeh and M. Wang. "Interaction Among Cost Functions in Placement".In *InternationalConference on VLSI and CAD,* 1999.
12. D. Stroobandt and J. V. Campenbout, "Accurate Interconnection Length

Estimations for Predictions Early in the Design Cycle", *VLSI Design - Langhorne,* vol. 10 part 1, pages 1-20, 1999.

13. X. Yang, R. Kastner and M. Sarrafzadeh, "Congestion Estimation During Top-Down Placement", in *International Symposium on Physical Design,* Napa, CA, USA, 2001.

14. J. A. Davis, V. K. De and J. D. Meindl, "A Stochastic Wire-Length Distribution for Gigascale Integration (GSI)", *IEEE Trans. Electron Devices,* vol. 45, no. 3, pages 580-589, 1998.

15. J.-M Lin and Y.-W Chang, "TCG: A transitive closure graph-based representation for non-slicing floorplans", in *Proc. DAC,* pages 764-769, 2001.

16. C. L. E. Chang, "RISA: Accurate and efficient Placement Routability Modeling", in *Proc. of ICCAD,* 1994.

17. P. Kannan, S. Balachandran, and D. Bhatia, "fGREP – Fast Generic Routing Demand Estimation for Placed FPGA Circuits", in *11th International Workshop on Field-Programmable Logic and Applications, FPL.* Springer-Verlag, Berlin, August 2001.

18. J. Lou, S. Krishnamoorthy, and H. Sheng, "Estimating Routing Congestion using Probabilistic Analysis", in *Proceedings of the 2001 International Symposium on Physical Design (ISPD),* 2001.

19. S. M. Sait, and H. Youssef, "VLSI Physical Design Automation Theory and Practice", *World Scientific Publishing Co. Pte. Ltd.,* 1999.

20. LME Design Automation Ltd, "MOODS Internals Version 1.0", *Southampton University,* 19th July, 2001.

21. M. Lai, D. Wong, "Slicing tree is a complete floorplan representation", *Proceedings of the DATE 2001 on Design, automation and test in Europe,* pages 228-232, March 2001, Munich, Germany.

22. S. Dutt, "New faster Kernighan-Lin-type graph-partitioning algorithms", *Proc. of the IEEE/ACM international conference on Computer-aided design,* pages 370-377, November 07-11, 1993, Santa Clara, California, United States.

23. Xu and F. J. Kurdahi, "Accurate Prediction of Quality Metrics for Logic Level Designs Targeted Toward Lookup-Table-Based FPGA's", *IEEE Trans. Very Large Scale Integration,* vol. 7 no. 4, pages 411-418, December 1999.

24. Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans", in *Proc. DAC,* pages 458-463, 2000.

25. A. Singh, G. Parthasarathy, M. Marek-Sadowska, "Interconnect Resource Aware Placement for Hierarchical FPGAs", *ICCAD 2001,* November, 2001.

26. X. Yang, M. Wang, K. Eguro and M. Sarrafzadeh, "A Snap-On Placement Tool", I*nternational Symposium on Physical Design,* pages 153-158, ACM, April 2000.

27. M. Stoer and F. Wagner, "A Simple Min-Cut Algorithm", *JOURNAL-ACM,* vol. 44, no. 4, pages 585-59 1, 1997.

28. Y. Cheon , D. F. Wong, "Physical Hierarchy: Design hierarchy guided multilevel circuit partitioning", in *Proc of 2002 International Symposium on Physical Design,* April, 2002.

29. Christie and D. Stroobandt, "The Interpretation and Application of Rent's Rule ", *IEEE trans. Very Large Scale Integration Systems,* vol. 8 part 6, pages 63 9-648, 2000.

30. X. Yang, E. Bozorgzadeh and M. Sarrafzadeh, "Wire Length Estimation Based on Rent Exponents of Partitioning and Placement", in *International Workshop on System-Level Interconnect Prediction,* pages 25-31, 2001.

31. P. Verplaetse, J. Dambre, D. Stroobandt, J. Van Campenhout, "On partitioning vs. placement rent properties", in *Proc. of the 2001 international workshop on System-level interconnect prediction,* pages 33-40, March 31-April 01, 2001, Sonoma, California, United States.

32. J. Dambre, P. Verplaetse, D. Stroobandt, and J. Van Campenhout, "On Rent's rule for rectangular regions", in *Proc. of International Workshop on System-Level Interconnect Prediction,* March, 2001.

33. P. Verplaetse "Refinements of Rent's Rule allowing Accurate Interconnect Complexity Modelling", in *Proc. Of the IEEE Int'l Symp. On Quality Electronic Design,* pages 25 1-252, March, 2001.

34. X. Yang and M. Sarrafzadeh, "Fast Congestion Prediction based on Rent's Rule", *submitted for publication.*

35. D. Pandini, L. T. Pileggi, and A. J. Strojwas, "Understanding and Addressing the Impact of Wiring Congestion During Technology Mapping" *ISPD '02,* April 7-10, 2002, San Diego, California, USA.

36. J. L. Ganley, "Computing optimal rectilinear Steiner trees: a survey and experimental evaluation", *Special volume on VLSI, Discrete Applied Mathematics,* Vol. 90, Issue 1-3, pages 161-17 1, Jan. 1999.

37. A. Singh and M Marek-Sadowska, "FPGA interconnect planning", *Proc. of the 2002 international workshop on System-level interconnect prediction,* pages 23-30, 2002 , San Diego, California, USA.

38. J. A. Nestor, "A New Look at Hardware Maze Routing" *GLSVLSI'02,* April 18-19, 2002, New York, New York, USA.

39. F. Y. Young, and D. F. Wong, "How Good Are Slicing Floorplans," *Proc. Int'l Symp. on Physical Design,* 1997.

40. R. Kastner, E. Bozorgzadeh and M. Sarrafzadeh, "Predictable Routing", in *IEEE International Conference On Computer Aided Design,* pages 110-1 14, 2000.

41. C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey", *Integration, the VLSI Journal,* pages 1–8 1, 1995.

42. P. Kannan, S. Balachandran and D. Bhatia, "On Metrics for Comparing Routability Estimation Methods for FPGAs" *DAC 2002,* June, 2002, New Orleans, Louisiana, USA.

43. M. C. Yildiz and P. H. Madden, "Improved Cut Sequences for Partitioning Based Placement", *Proc. Design Automation Conference,* pages 776-779, 2001.

44. C. Papachristou and M. Nourani, "False Path Exclusion in Delay Analysis of RTL-Based Datapath-Controller Designs", in *European Design Automation Conf.,* Geneva, Switzerland, September, 1996.

45. S. Dutt and W. Deng, "Probabilistic Approaches to VLSI Circuit Partitioning"', *IEEE Trans. CAD,* Vol. 19, No. 5, pages 534-549, May, 2000.

46. P. Verplaetse, D. Stroobandt and J. Van Campenhout, "A Stochastic Model for the Interconnection Topology of Digital Circuits", in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems,* vol. 9, no. 6, pages 938-942, December, 2001.

47. R. Tessier, "Fast placement approaches for FPGAs", *ACM Transactions on Design Automation of Electronic Systems (TODAES),* vol. 7, issue 2, pages 284-305, April, 2002.

48. J. Hu and S. S. Sapatnekar, "A Survey on Multi-net Global Routing for

Integrated Circuits," *Integration: The VLSI Journal,* Vol. 31, No. 1, pages 1-49, November 2001.

49. H. Eisenmann F.M. Johannes, "Generic global placement and floorplanning", *Proc. of the Design Automation Conf.,* Pages 269–274, 15-19 June 1998.

50. A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wire Length Estimations for Row-Based Placement", *IEEE Trans. on CAD,* 18(9), 1999.

51. P. H. Madden, "Partitioning by Iterative Deletion", in *Proc. Int'l Symp. on Physical Design,* pages 83-89, 1999.

52. Daemen, J. and V. Rijmen, *AES Proposal: The Rijndael Block Cipher,* http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaeldocV2.zip, 2000.

53. Frank M. Johannes, "Partitioning of VLSI circuits and systems", *Proc. Of the 33rd annual conf. on Design automation,* June 1996.

54. S. Bhattacharya, S. Dey and F. Brglez, "Fast True Delay Estimation During High Level Synthesis", *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems CAD,* vol.15, part 9, pp. 1088-1105, 1996.

55. Giancarlo Beraudo, John Lillis, "Timing optimization of FPGA placements by logic replication", *Proc. of the 40th conf. on Design automation,* June 2003.

56. S. N. Adya, M. C. Yildiz, I. L. Markov, P. G. Villarrubia, P. N. Parakh, P. H. Madden, "Benchmarking for large-scale placement and beyond", *Proc. of the 2003 Int'l Symp. on Physical design,* April 2003.

57. A. B. Kahng, I. L. Markov, S. Reda, "Boosting: Min-Cut Placement with Improved Signal Delay", *Proc. of the conf. on Design, automation and test in Europe,* Volume 2, Feb. 2004.

58. Charles J. Alpert, Gi-Joon Nam, Paul G. Villarrubia, "Free space management for cut-based placement", *Proc. of the 2002 IEEE/ACM int'l conf. on CAD,* November 2002.

59. A. B. Kahng, S. Reda, "Placement feedback: a concept and method for better min-cut placements", *Proc. of the 41st annual conf. on Design automation,* June 2004.

60. P. Gupta, L. Zhong, N. K. Jha, "A High-level Interconnect Power Model for Design Space Exploration", *Proc. of the 2003 Int'l conf. on CAD,*

November 2003.

61. J. Dambre, D. Stroobandt, J. Van Campenhout, "Fast estimation of the partitioning rent characteristic using a recursive partitioning model", *Proc. of the 2003 int'l workshop on SLIP,* April 2003.

62. A. Marquardt, V. Betz, J. Rose, "Timing-driven placement for FPGAs", *Proc. of the 2000 ACM/SIGDA eighth Int'l Symp. on FPGAs,* Feb 2000.

63. A. H. Farrahi, D. J. Hathaway, M. Wang, M. Sarrafzadeh, "Quality of EDA CAD Tools: Definitions, Metrics and Directions", *Proc. of the First Int'l Symp. on Quality of Electronic Design,* March 2000.

64. A. E. Caldwell, A. B. Kahng, I. L. Markov , "Can recursive bisection alone produce routable placements?", *Proc. of the 37th conf. on Design automation,* June 2000.

65. K. M. Buyuksahin and F. N. Najm, "High-level power estimation with interconnect affects," *IEEE Int'l Symp. on Low Power Electronics and Design,* Italy, pp. 197-202, July 26-27, 2000.

66. P. Zarkesh-Ha, J. A. Davis, and J. D. Meindl, "Prediction of Net-Length Distribution for Global Interconnects in a Heterogeneous System-on-a-Chip," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.* **8**, 649–65 9 (December 2000).

67. C. Jego, E. Casseau, E. Martin, "Interconnect Cost Control during High-Level Synthesis", Laboraty, LESTER, University of Brittony.

68. F. Li, D. Chen, J. Cong, "Architecture evaluation for power-efficient FPGAs", *Proc. Of the 2003 ACM/SIGDA Symp. on FPGAs,* February, 2003.

69. A. Ranjan, K. Bazargan, S. Ogrenci, M. Sarrafzadeh, "Fast floorplanning for effective prediction and construction", *IEEE Trans. on VLSI Systems,* Vol. 9, Issue 2, pages 341–35 1, April 2001.

70. C. Chang, J. Cong, Z. Pan, "Physical hierarchy generation with routing congestion control", *Proc. of the 2002 Int'l Symp. on Physical design,* April 2002.

71. D. Chen, J. Cong, y.Fan, "Low-Power High-Level Synthesis for FPGA Architectures", *ISLPED,* August, 2003.

72. Cong, T. Kong, J. R. Shinnerl, M. Xie, X. Yuan, "Large-Scale Circuit Placement: Gap and Promise", *Proc. of the 2003 Int'l Conf. on CAD* - Volume 00, Nov. 2003.

73. F. M. Johannes, "Partitioning of VLSI circuits and systems", *Proc. of the 33rd annual conf. on Design automation,* June 1996.

74. A. Nayak, M. Haldar, A. Choudhary, P. Banerjee, "Accurate Area and Delay Estimators for FPGAs", *Proc. Of the conf. On Design, Automation and Test in Europe,* March, 2002.

75. P. Verplaetse, "Partitioning properties of general graphs", *ELIS Technical Report, Paris 03-01,* Feb. 2003.

76. C. J. Alpert, J. Huang, A. B. Kahng, "Multilevel circuit partitioning", *Proc. of the 34th annual conf. on Design automation,* Volume 00, June 1997.

77. A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C. Koh, P. H. Madden, "Recursive bisection based mixed block placement", *Proc. of the 2004 Int'l Symp. on Physical design,* April 2004.

78. S. Dutt, H. Theny, "Partitioning around roadblocks: tackling constraints with intermediate relaxations", *IEEE/ACM International Conference on CAD,* pages 350–355, Nov. 1997.

79. A. B. Kahng, S. Mantik, I. L. Markov, "Min-max placement for large-scale timing optimisation", *Proc. of the 2002 international symposium on Physical design,* April 2002.

80. Y. Cheon, D. F. Wong, "Design hierarchy guided multilevel circuit partitioning", *Proc of the 2002 international symposium on Physical design,* April 2002.

81. J. M. Kleinhans, G. Sigl, F. M. Johannes, K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimisation", *IEEE Trans. on CAD of Integrated Circuits and Systems,* pp 356-365, Vol. 10, Issue 3, March 1991.

82. C. Ababei, N. Selvakkumaran, K. Bazargan, G. Karypis, "Multi-objective Circuit partitioning for cutsize and path-based delay minimization", *Proc. of the 2002 IEEE/A CM Int'l Conf. on CAD,* Nov. 2002.

83. P. Metaxas, G. E. Pantziou, A. Symvonis, "Parallel H-V Drawings of Binary Trees", *Technical report, Dartmouth Colledge,* January 1993.

84. Q. Liu, M. Marek-Sadowska, "A study of netlist structure and placement efficiency", *Proc. of the 2004 Int'l Symp. on Physical design,* April 2004.

85. U. Brenner, A. Rohe, "An Affective Congestion Driven Placement Framework", *Int'l Symp. on Physical Design,* April, 2002.

86. S. Balachandran, D. Bhatia , "A-priori Wire Length and interconnect estimation based on circuit characteristics", *Proc. of the 2003 international workshop on System-level interconnect prediction,* April 2003.

87. N. Selvakkumaran, P. N. Parakh, G. Karypis, "Perimeter-degree: a priori metric for directly measuring and homogenizing interconnection complexity in multilevel placement", *Proc. of the 2003 international workshop on System-level interconnect prediction,* April 2003.

88. W. Choi, K. Bazargan, "Hierarchical Global Floorplacement Using Simulated Annealing and Network Flow Area Migration", *Proc. of the Conf. on Design, Automation and Test in Europe,* Vol. 1, March 2003.

89. J. Pistorius, M. Hutton, "Placement rent exponent calculation methods, temporal behaviour and FPGA architecture evaluation", *Proc. of the 2003 Int'l workshop on SLIP,* April 2003.

90. S. Sathiamoorthy, "LaySeq: A New Representation for Non-Slicing Floorplans" *ACM/IEEE Int'l Symp. on Physical Design,* 2002.

91. http://encyclopedia.laborlawtalk.com/Sorting_algorithm.

92. C. Shen and C. Chu, "Bounds on the Number of Slicing, Mosaic and General Floorplans", *IEEE Trans. on CAD,* vol. 22, no. 10, pages 1354-1361, October 2003.

93. C. Chang, J. Cong, and D. Pan , "Interconnect-Driven Floorplanning with Fast Global Wiring Planning and Optimization", *Proc. SRC Techcon Conf.,* Sept. 21-23, 2000, Phoenix, AZ.

94. P. Zarkesh-Ha, K. Doniger, W. Loh, P. Bendix , "Prediction of interconnect adjacency distribution: derivation, validation, and applications", *Proc. of the 2004 international workshop on SLIP,* Feb. 2004.

95. T. Wan, M. Chrzanowska-Jeske, "Prediction of interconnect net-degree distribution based on Rent's rule", *Proc. of the 2004 Int'l workshop on SLIP,* Feb. 2004.

96. F. Balasa, S. C. Maruvada, "Using Non-Slicing Topological Representations for Analog Placement", *IEICE Trans. Fundamentals,* Vol. E84-A, No. 11, Nov. 2001.

97. H. H. Chan, I. L. Markov, "Practical slicing and non-slicing block-packing without simulated annealing", *Proc. of the 14th ACM Great Lakes symposium on VLSI,* April 2004.

98. S. Balachandran, D. Bhatia, "A-Priori Wire Length and Interconnect Estimation Based on Circuit Characteristics", Conf. On *SLIP,* April 2003.

99. J. Cong, G. Nataneli, M. Romesis, J. R. Shinnerl, "An area-optimality study of floorplanning", *Proc. of the 2004 Int'l Symp. on Physical design,* April 2004.

100. J. Cong, T. Kong, D. Xu, F. Liang, J.S.Liu, and W. Wong. Relaxed simulated tempering for VLSI floorplan design. *In Proc. Asia and South Pacific Design Automation Conference Design,* pages 13-16, 1999.

101. A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C. Koh, P. H. Madden, "Recursive bisection based mixed block placement", *Proc. of the 2004 Int'l Symp. on Physical design,* April 2004.

102. A. E. Caldwell, A. B. Kahng, I. L. Markov. "Iterative Partitioning With Varying Node Weights", *VLSI Design,* 2000.

103. P. G. S as son e, S. K. L im, "A Novel Geometric Algorithm for Fast Wire-Optimized Floorplanning", *Int'l Conf. on CAD,* p74, November 09 - 13, 2003, San Jose, CA.

104. Andrew B. Kahng, Stefanus Mantik, Igor L. Markov, "Min-Max Placement for Large-Scale Timing Optimisation", *Proc of the 2002 Int'l Symp. On Physical Design,* April, 2002.

105. M. Wang, M. Sarrafzadeh, "On the Behavior of Congestion Minimization During Placement", *Proc. Of the 1999 Int'l Symp. On Physical Design,* April 1999.

106. S. Areibi, "Recursive and Flat Partitioning for VLSI Circuit Design" *The 13th Int'l Conf on Microelectronics,* Rabat, Morocco, October 29 2001.

107. N. Selvakkumaran, G. Karypis, "Multi.Objective Hypergraph Partitioning Algorithms for Cut and Maximum Subdomain Degree Minimization", *Proc. of the 2003 Int'l Conf. on CAD,* Nov. 2003.

108. B, Halpin, R. Chen, N. Sehgal, "Timing driven placement using physical net constraints", *Proc. of DAC,* Pages 780–783, 18-22 June 2001.

109. B. Hu, M. Marek-Sadowska, "Congestion minimization during placement without estimation", *Proc. of the 2002 IEEE/ACM Int'l Conf. on CAD,* Nov. 2002.

110. A. E. Caldwell, A. B. Kahng, I. L. Markov, "Optimal Partitioners and End-Case Placers for Standard-Cell Layout", *IEEE Trans. On CAD of Integrated Circuits and Systems,* Vol. 19, No.19, November 2000.

111. J. Westra, C. Bartels, P. Groeneveld, "Probabilistic congestion prediction", *Proc. of the 2004 Int'l Symp. on Physical design,* April 2004.

112. H. Van Marck, D. Stroobandt, and J. Van Campenhout, "Towards An Extension of Rent's Rule for Describing Local Variations in Interconnection Complexity". *Proc. of the Fourth International Conference for Young Computer Scientists,* pages 136-141, 1995.

113. M. Wang, X. Yang, M. Sarrafzadeh, "Dragon2000: standard-cell placement tool for large industry circuits", *Proc. of the 2000 IEEE/ACM Int'l Conf. on CAD,* Nov 2000.

114. G. Sigl, K. Doll, F. M. Johannes, "Analytical placement: A linear or a quadratic objective function?", *Proc. of the 28th Conf. on ACM/IEEE design automation,* June 1991.

115. S. Ou, M. Pedram, "Timing-driven placement based on partitioning with dynamic cut-net control", *Proc. of the 37th conference on Design automation,* Pages: 472 – 476, 2000.

116. T. Hamada, C. Cheng, P. M. Chau, "A Wire Length Estimation Technique Utilizing Neighborhood Density Equations", *DAC, pages* 57-61, *1992.*

117. N. Viswanathan, C. C. Chu , "FastPlace: efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model", *Proc. of the 2004 Int'l Symp. on Physical Design,* April 2004.

118. P. N. Parakh, R. B. Brown, K. A. Sakallah, "Congestion driven quadratic placement", *Proc. of the 35th annual conference on Design automation,* Vol. 00, May 1998.

119. W. Mak, "Min-cut partitioning with functional replication for technology mapped circuits using minimum area overhead", *Proc. of the 2001 international symposium on Physical design,* April 2001.

120. N. Selvakkumaran, A. Ranjan, S. Raje, G. Karypis, "Multi-resource aware partitioning algorithms for FPGAs with heterogeneous resources", *Proc. of the 41st annual conference on Design automation,* June 2004.

121. C. Cheung, Y. Wu, D. Cheng, "Further improve circuit partitioning using GBAW logic perturbation techniques", *Proc. of the conference on Design, automation and test in Europe,* March 2001.

122. B. Hu, M. Marek-Sadowska, "Fine granularity clustering for large scale placement problems", *Proc. of the 2003 Int'l Symp. on Physical design,* April 2003.

Appendix

A.1 Definitions

1. **Critical Path** – The Critical Path (Definition 1) is the largest Register to Register delay of a circuit.
2. **Clock Period** - The Clock Period is the minimal period of the clock wavelength, which has to be greater than the delay of the critical path. If the clock period is less than the critical path the functionality of the design is destroyed.
3. **Metrics** – A Metric is a value (technology specific) that represents a physical property, which can then be used to compare with their respective design constraint. I.e. the metric for the size of a Xilinx Virtex chip (FPGA) would be the number of slices needed for all the components that are needed for the design to be implemented on the FPGA. The constraint would then be the number of slices available on a particular Vertex chip.
4. **Signals Nets / Nets** – Signal Nets are signal nets are defined as sets of points that are to be electrically connected together.
5. **Routing Plan** – Routing plan is the layout of the routing on a chip;
6. **Routing Channel** - The routing channel runs between the cells on a chip and this is where nets are placed;
7. **Dead Space** – A placement site that is not occupied is known as Dead Space.
8. **Cost Function** – A cost function (also known as an objective function) represents all the design criteria on which the optimisation process will base its decisions.

A.2 Description of Transformations Applied during Optimisation within MOODS

A.2.1 Scheduling Transformations

Scheduling transformations alter the control graph by altering the assignment of instructions to control states. These transformations merge control states in order to increase parallelism or unmerge control states. Parallelism is increased in order that tasks can be carried out at the same time so that total delay is reduced. Unmerging control states is desirable as this allows functional units to be merged, as the functional units in the two disjoint group no longer carry out their tasks at the same time, allowing functional units that carry out the same operation to be merged into one data path unit, which can decrease area among other benefits.

There are four merging transformations which are:

1. Merge Sequential IGR nodes. These nodes are contained in the control graph and contain instructions on when tasks should be carried out.
2. Merge Parallel nodes after fork, where a fork is when a node has two successor nodes in the control graph
3. Merge fork and successor
4. Group instructions on variable

There are two unmerging transformations

1. Ungroup node by separating groups
2. Ungroup node into time slices

A.2.1.1 Merge Sequential IGR nodes

This transformation merges two control nodes. N.B. one control node is assumed to take one clock cycle. The nodes have to be sequential such that the second node is performed after the first without any feedback loops. The transformation takes the instructions from the second node and places them into the first node; hence the second node is now redundant and can be removed from the control graph. The instructions being merged cannot share any data path hardware unless the instructions which share the hardware are mutually exclusive. Instructions that are mutually exclusive are not active at the same time; hence the two functional units that carry out the instruction are not used at the same time. If this were the case and the IGR nodes were still merged, this would lead to the functionality of the design being incorrect. Finally the clock

period cannot be violated if the instructions are combined into the same state. If the CP is violated then this means that the output signal that the instructions have created will be produced after the CP has already moved the system to the next state. This is a matter of logistics: for example if a product is ready after a transportation lorry has left the depot then when the lorry arrives at its destination without any cargo, the chain will be broken and the product cannot fulfil its purpose.

A.2.1.2 Merge parallel nodes after fork

This transform merges control states, but only in the case that the nodes being considered proceed directly after a fork node. A fork node is simply a node that has two outputs. The two nodes that are being considered for merging can only have one input and have the same activation conditions i.e. they run in parallel.

A.2.1.3 Merge fork node and successor

This transform merges a fork node with its successor. Again the successor node has to follow directly after the fork node. There cannot be any hardware sharing between the nodes and the clock period cannot be exceeded due to the new control state containing all the merged instructions.

A.2.1.4 Group instructions on variables

The transform only acts on register nodes that only have one input and one output, which it then tries to remove them, as they may be redundant. This redundancy occurs because when the data structure is built, registers are placed after every data path unit that carries out an operation. Some of these registers are not needed because through the process of optimisation they do not store anything. The value that would have been stored is immediately used by a data path unit carrying out the next operation in the same stage. To remove the register, the register first needs to be bypassed. This means the instructions that write/read the input/output of the register are placed into the same control state as the functional unit that writes to the register. This will then make the register redundant. Bypassing the register will mean more operations will be carried out one after the other in the same CP. If this new path created becomes the new critical path it cannot exceed the CP, for reasons stated earlier. The instructions that are placed in the same instruction group cannot share the same hardware.

A.2.1.5 Ungroup node by separating groups

This transform simply splits one control state into multiple control states, by ungrouping instructions and placing them into separate nodes. No new data dependencies can be formed between the instructions.

A.2.1.6 Extract single instruction

This transform selects an instruction to be extracted from a control node and places it into a new control node. If any instructions are initiated sequentially, hence dependent on each other and must follow the extracted instruction, they are also placed into the new control node. The instructions that are extracted cannot share the same registers, as this would mean that the registers that the instructions point to would store information at the same time, hence destroying the functionality of the design.

A.2.2 Allocation and Binding Transformations

Allocation and binding transforms manipulate the data path by sharing and unsharing data path nodes, while also mapping library cells. There are two sharing transforms:

3. Data Path unit sharing/ALU creation
4. Register Sharing

There are four transforms that reverse the last two transforms:

5. Unshare single instruction from unit
6. Unshare unit fully
7. Unshare variable from register
8. Unshare register fully

A.2.2.1 DP unit sharing/ALU creation

This transform shares the functionality of two data path units into one ALU. The new ALU needs to be capable of carrying out the functions of the old data path units, so as to keep the same functionality of the design. Also the old units cannot be run concurrently.

A.2.2.2 Register sharing

This transform tries to allow variables that are stored in two registers to be stored into one register, hence removing a register. The registers under consideration cannot have variables that need to be read before another variable is written to them. That is the variable lifetimes cannot overlap or occurs only in mutually exclusive conditional branches.

A.2.2.3 Unshare single instruction from DP unit

This transform removes an instruction from a previously merged DP unit. This instruction is then placed into a new data path node leaving the other instructions in the original DP node. Again the cell library is used to obtain the new ALU physical properties.

A.2.2.4 Unshare DP unit fully

This transform unshares all the instructions in a previously shared DP and places each instruction into its own ALU.

A.2.2.5 Unshare variable from register

This transform selects one variable from a register that stores multiple variables and places it into a new register.

A.2.2.6 Unshare Register Fully

This transform unshares all the variables in a previously shared register and gives each variable its own register unit.

A.2.2.7 Binding Transformation

These transforms are used to see if any other cell in the library could carry out the task better than the current cell chosen for a particular DP or CP unit. These transforms are:

1. Alternative DP cell selection
2. Alternative CP cell selection

A.2.2.8 Alternative DP cell selection

This transform offers alternative cell implementations to the current cell that has been selected for a functional DP unit.

A.2.2.9 Alternate control cell selection

This transform offers alternative cell implementations to the current cell that has been selected for a control node.

A.3 Least Squares

The least squares model [5] can be represented as:

$$E(Y) = \beta_0 + \beta_1 x + \varepsilon \qquad \text{(A1)}$$

where x represents the values on the x axis

Y represents the values on the y axis

E(Y) represents the expected value of Y given x

β_0 is equal to the intercept on the x – axis

β_1 is equal to the gradient of the graph

ε represents the random error (i.e. cannot predict an exact model for nature)

Definition from [5]

If the model relates E(Y) as a linear function of β_0 and β_1 only, the model is called a simple linear regression model. To estimate the parameters of this linear model we use least squares, which fits a line to the data. Least Squares is used because it is an accurate but convenient method.

The least squares estimators for the simple linear regression model are:

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

Where $\beta_1 = \frac{S_{xy}}{S_{xx}}$

$$S_{xy} = \sum_{i=1}^{n} x_i y_i - \frac{1}{n} \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i$$

$$S_{xx} = \sum_{i=1}^{n} (x_i)^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2$$

A.4 APR Tool (Xilinx)

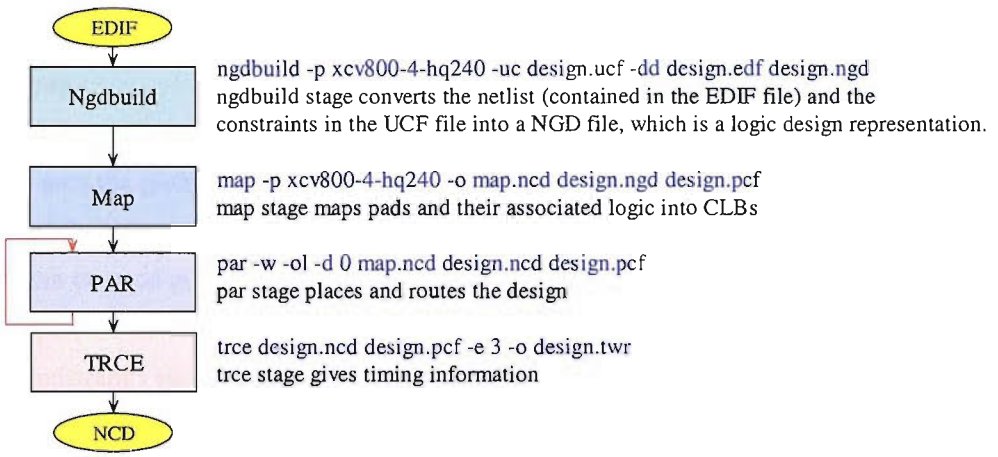


Figure 98. Design Flow Within the Xilinx Tool.

An overview of Xilinx's design flow is shown in figure 98. The architectures of Xlilnix's FPGA's are shown in [25] and [37], and further information can be found on their website 'www.Xliinx.com'. UCF file is used to influence placement of a design's components when the design is being placed and routed in Xilinx, hence enabling a representation of the design in hardware at a higher level, i.e. during design exploration in MOODS. Xilinx placement can be influenced by using a called the User Constraint File (UCF). This file allows timing and physical constraints to be entered into Xilinx at the NDGbuild stage.

The constraints contained in the UCF override any previous constraints, which maybe in the EDIF file. The two different constraint entries, which have been considered, to be placed in the UCF are, LOC and AREA_GROUP constraints. A LOC constraint fixes an instance into a specific location on a chip. AREA_GROUP constraints group Instances together.

AREA_GROUP constraints are of the following format :

INST logic_name_1 AREA_GROUP = Group_A;

INST logic_name_2 AREA_GROUP = Group_A;

INST logic_name_3 AREA_GROUP = Group_A;

INST logic_name_4 AREA_GROUP = Group_A;

This information will now inform Xilinx to keep Instances logic_name_1, logic_name_2, logic_name_3 and logic_name_4, close to each other.

If you want to assign Group A, a designated area on the board you can then write the following :

AREA_GROUP Group_A RANGE = CLB_R1C2:CLB_R5C6;

So the area the group will be placed in, is in between Row 1 and Row 5, Column 2 and Column 6.

Stars can be used instead of numbers. The stars represent any value within the limits of the chip.

LOC constraints are of the following format:

INST logic_name LOC=CLB_R1C1:CLB_R5C5;

So the Instance 'logic_name' will be placed in between Row 1 and Row 5, Column 2 and Column 6.

A.5 False Paths

A.5.1 Cutting Down the Design Space

Rapid increase in the design complexity has increased the need for High Level Synthesis, whose benefit is that it can quickly search design alternatives. The problem is that design exploration is infeasible without fast and accurate delay and area estimation. One way in doing this is to reduce the design space in which to search up to the point where more thorough techniques can be applied without the computation becoming infeasible. Reducing the design space reduces the amount of paths, which need to be analysed. Hence less computation is needed, simultaneously increasing the accuracy by filtering out the paths, which cause bad estimations. A method used in the past was to use a topological based method, which ignores interconnect delay and this compromises accuracy. Though a topological methodology is a lot faster due to lees computation, interconnect delay is becoming much more of a factor when considering design architecture alternatives, because of the decreasing delay in gates.

A.5.2 Sensitisable/False Paths

The next few definitions are from [54], they help to understand what a sensitisable path is, and how it relates to a circuit.

Let $p = (f_0, g_1,.., g_{m-1}, f_{m-1})$ be a path in the combinational circuit, where f_i is a lead and g_i is a gate.

Leads f_0 and f_{m-1} are the primary input and output respectively. All inputs to g_i other than f_{i-1} are called side-inputs of gate g_i.

A logic value is the controlling value of a gate if the logic value at an input of the gate determines the gate output independently of the other inputs, and the converse is called a non-controlling value.

E.g. if g is an AND gate, c(g) = 0 and n(g) = 1, because 1 does not alter any value it is with but 0 will change the value if not 0, so it is controlling the output of the gate.

Definition 1

f_i dominates g_{i+1} if any one of the following conditions is true.

1 The only controlling input to g_{i+1} is f_i

2 There are more than one controlling inputs to g_{i+1}, but f_i arrives before the other controlling inputs.

3 Every input to g_{i+1} is non-controlling. However, f_i is the last input to stabilise and it is the last to arrive.

Definition 2

A path p is sensitisable if there is at least one input vector v under which every lead f_i on p dominates g_{i+1}, $0 \leq i \leq m\text{-}2$.

Definition 3

The true delay of a circuit is the delay of the longest sensitisable path in the circuit. A clock period of a circuit greater than or equal to its true delay is a correct clock period for the circuit.

Sensitisable paths are often referred to as true paths, and unsensitisable paths are often referred to as false paths. There is much interest in these types of paths. Because if the delay is read from a false path and it is the longest path in the circuit, it will result in the wrong delay. Time is being wasted measuring the delay from a false because it has no influence on the clock delay. If a clock period is measured at 50 ns, but the real clock period is 40 ns. Then every period 10 ns are being lost. When comparing different architectures false paths might lead to a false comparison hence a bad decision.

A.5.3 Recognising False Paths

[44] discusses false paths and is a good grounding for path analysis. There are a few papers which use the term false path, when talking about delay estimation. Delay and area estimation is easier at RTL due to fewer components. By understanding how the design functions enables manipulation of the design without destroying the functionality. This is the advantage of being at a higher level, rather than at the logic level where it would take to much time sifting through all the gates etc. One major cause of false paths is resource binding (components, wires are shared), causing false paths. Figure 99 shows the path in which you measure the clock period. The clock period is the largest delay between one register and its successor in the circuit. The diagrams in this section are drawn from [44].

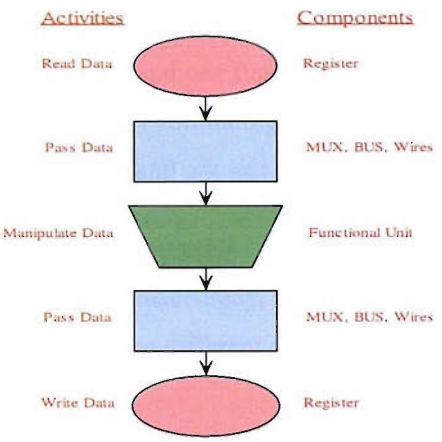


Figure 99. Register-to-Register Transfer Path.

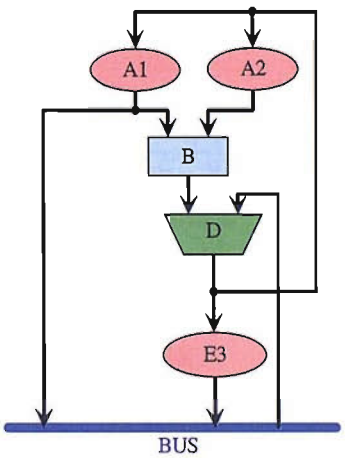


Figure 100. Affect of Binding Decision.

[44] presents six situations where resource binding can cause false paths:

A.5.3.1 Example 1: Certain Binding Decisions

From figure 100 the true paths are A1 - B - D - E3, A1 - BUS - D - E3 and A2 - B - D - A1 call these paths p1, p2 and p3 respectively. Let p1 occur in state 1 and p2, p3 occur in state 3. A false path is A1 - BUS - D - A1 call this path f1. The circuit has two states. The problem occurs when in the first state instead of just going to E3 the path also loops round to A2, ready for p2 in state 2. This reduces the amount of functional units required (i.e. resource sharing p1 and p2 have the same functional unit), but this is what causes the false paths e.g. f1. When the clock delay is being computed, the delay of f1 would be measured.

A.5.3.2 Example 2: Multi-Functional ALU's

If an ALU has more than one function, and one function has a larger delay than another. This can lead to a false path due to over estimating the delay, but this can be remedied by making the estimator more accurate and not just going for the worst case scenario.

A.5.3.3 Example 3: Testable Datapaths

Different types of test registers cause inaccuracy, because the testing registers should not be used in the analysis, and they have different delays.

A.5.3.4 Example 4 : Chaining

Looking at the scheduling graph (figure 101 (i)) both g and h need an ALU(+) which they share. By chaining the ALU(-) to the output of MUX3 instead of creating either another MUX or another ALU(+), creates a false path, from ALU(*) to the ALU(-).

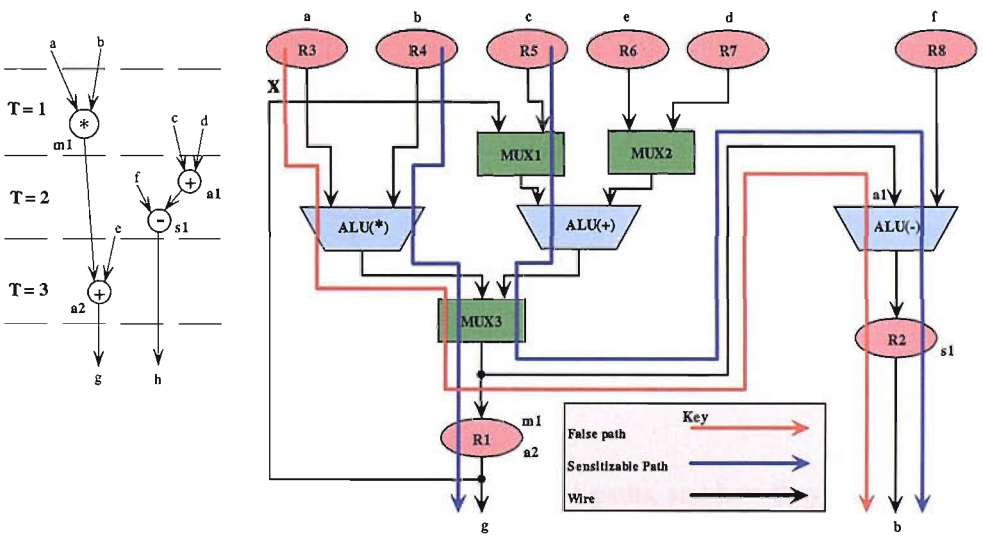


Figure 101. False Paths Created by Chaining.

A.5.3.5 Example 5: Redundant Components

Redundant components occur when components are added one by one i.e. sequentially as shown in figure 102. Consider figure 102 (a) & (b). First b is passed through Mux2 and then goes to ALU(+) and also ALU(*) using w1. But then c is considered, where c and b occur in the same state. b and c cannot go through the same MUX at the same time. Hence w2 is introduced to replace w1 to pass b to the ALU(*). But because w1 is still there, the synthesis tool thinks a MUX is needed. Hence MUX3 is introduced to choose between w1 and w2. Figure 102 (c) shows what the design should look like.

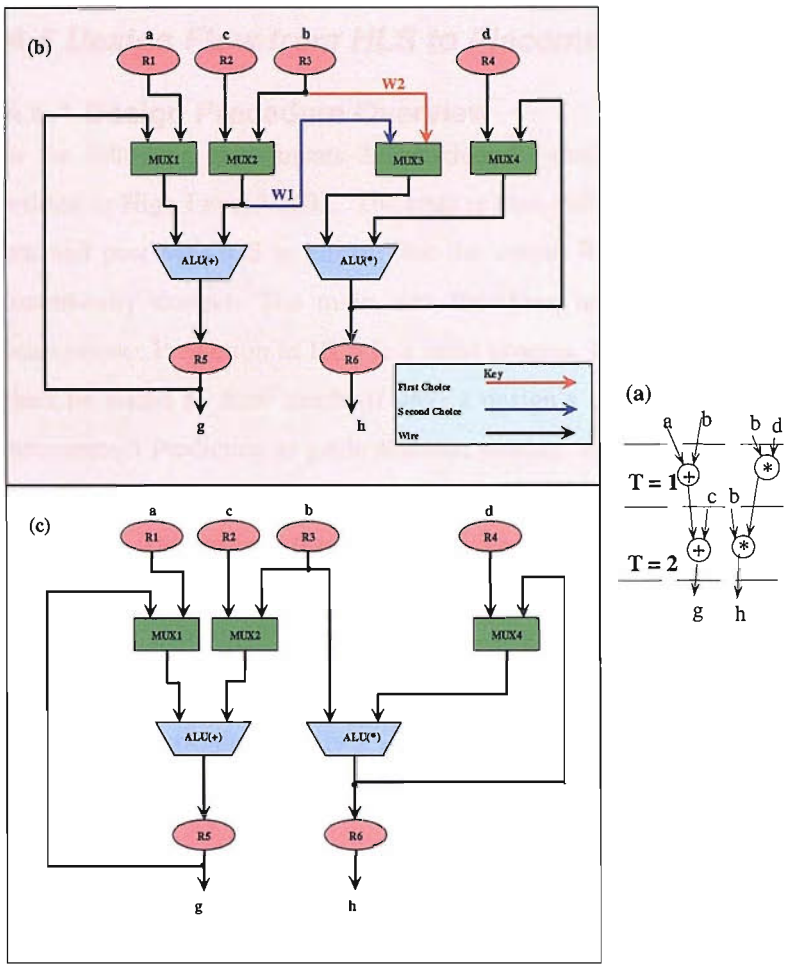


Figure 102. Redundant Components due to Error in High-level Processing

The examples give a deeper understanding of false paths, and how they are caused. The problem of false paths was tackled by giving weights to paths. The weights correspond to register-to-register delays, and construction of a Propagation Delay Graph (PDG). During the analysis, the information about the data transfer is always accessible (edges of the graph) so the tautology (whether true or false) of the path can be decided. This is the benefit of High-level synthesis because there is easy access to functional information.

A.6 Design Flow from HLS to Placement and Routing

A.6.1 Design Procedure Overview

In the following experiments 23 benchmarks shall be used. The benchmarks are all written in High Level VHDL. The code is then validated using Modelsim with stimuli, pre and post MOODS to ensure that the output RTL VHDL from MOODS remains functionally correct. The main aim for these benchmarks is to prove that using Interconnect Prediction in HLS is a valid process. Hence the optimality of the designs shall be tested by how much (if any) a design's optimality is improved when using Interconnect Prediction to guide decision making. When constructing these benchmarks their size and design nature was considered in order to try to make the designs as varied as possible. This ensures that the process works for differing design architectures rather than a select few. This is very important as HLS is used for many different design architectures and should not be biased to any particular sect. The benchmarks consist of 10 purely arithmetic benchmarks with if statements. 5 sorting algorithms, 5 matrix arithmetic algorithms. Finally a GCD and a quadratic algorithm. All these algorithms can be found on the accompanying Compact Disc.

The first set of results shall be obtained by using Quasi Exhaustive heuristic in MOODS to optimise our benchmarks. This will form the control set in which Interconnect Prediction will have to improve on to be come a viable methodology.

Once the optimised RTL VHDL has been outputted from MOODS, this code will then be inputted into Synplify. The Synplify tool has been chosen to convert the RTL VHDL to the EDIF, as Synplify uses a mapper that has been developed in close cooperation with Xilinx, hence the mapper can convert an RTL design to an EDIF, optimally mapping a design, taking full advantage of the Virtex chip architecture. The Xilinx Virtex Series family shall be the designated chip. The size of the chip will be designated by MOODS, when the average interconnect is calculated post synthesis. Hence the average interconnect will be calculated post synthesis for every optimised design architecture. If the size of the design is overestimated and the design could fit onto a smaller chip than the one designated, this means the area estimate was too large. But in the case of the Clock Period (CP), the CP will only be affected within approximately 1ns, so makes little difference to our observation of the CP. The EDIF outputted by Synplify is then passed onto Xilinx. When Xilinx is run, if the CP target (passed on from Synplify in the

NCF) has not been achieved the actual achieved CP (in the PAR file) is used as the target for the next run of Xilinx. After the first run the CP is systematically reduced every time Xilinx is run. Once the CP cannot be reduced any further that is the minimum CP, and then all desired physical characteristics are recorded. A detailed explanation and all the files and programmes involved with the design process now follow.

A.6.2 Automated Design Flow

Chip Pin	Package	Speed	No of Slices	# of User IOBs
XCV50	BG256	-4	768	180
XCV100	BG256	-4	1200	180
XCV150	BG256	-4	1728	180
XCV200	BG256	-4	2352	180
XCV300	BG432	-4	3072	316
XCV400	BG432	-4	4800	316
XCV600	BG432	-4	6912	316
XCV800	BG432	-4	9408	316
XCV1 000	BG560	-4	12288	404

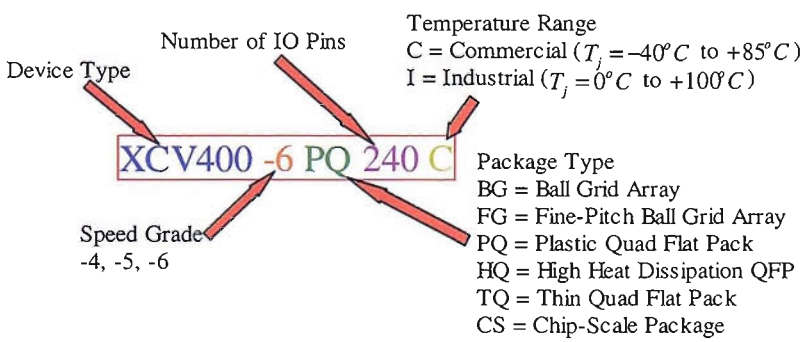


Figure 103. Available Chips within the Virtex Family

The desired architecture for all the benchmarks is the Xlinx Virtex Series, as chosen in Chapter 3. The possible chips are and physical limitations are shown in figure 103. The first stage is to write the behavioural code of a design. The behavioural code is then simulated on Modelsim, which is a HDL simulator. The code is then test to see whether the design shows the correct functionality with an appropriate testbench. If the code has

any errors, the code is modified until all errors are removed. Now that the code is correct the remaining design process from HLS to the design being ready to being implemented on an FPGA chip is automated. The structure of the automation can be seen in figure 104.

MOODS outputs 4 files, the first file is the RTL VHDL that contains the RTL description of the design that has undergone synthesis. The second file is the Floorplan (FLR) file, this contains the average interconnect length of a design, the total area of the design, the locations of all the macros that lie on the floorplan and finally the designated chip for the design. The smallest possible chip in the Xilinx Vertex Family that the design can fit on is chosen. This method of chip selection is chosen due to the tighter the restriction on area forces the design to be more compact than if the design was placed on a larger chip. This forces the routing to also be placed in a more constricted region, hence increasing congestion, hence increase average interconnect length, increasing the probability of a negative impact on the Clock Period (CP).

As the main focus of this thesis is to show how interconnect prediction can improve a designs routing layout, the more chance there is a negative impact on the CP, the more of a role interconnect prediction will have in order to reduce the affect of routing on the critical path. The User Constraint File (UCF) contains location and timing constraints of macros and pin placements (if selected) which can be used during Xilinx, to aid in the optimisation of the placement of routing of a chip. The second stage after MOODS has synthesised a design, a Synplify project file is written so that the Synplify tool can convert the RTL VHDL into an EDIF, so that Xilinx can proceed with APR. The FLR file is used to provide which chip architecture the design is placed on. The desired frequency of the CP is set at 50 MHz. This frequency is purposely set higher than the actual expected frequency of all the designs (normally 30-50 ns), so that every design has the same effort in trying to obtain a relatively small frequency. The pin package is dependent on the chip selected and is chosen according to figure 103. The speed of the chip is chosen to be 4 because. The fanout of the nets is set to a maximum of 100, which is default value of the Synplify tool. At the same time the Xilinx batch file is written which contains all the programs that are needed to run the APR, but this batch file will be discussed when discussing the design being run in Xilinx.

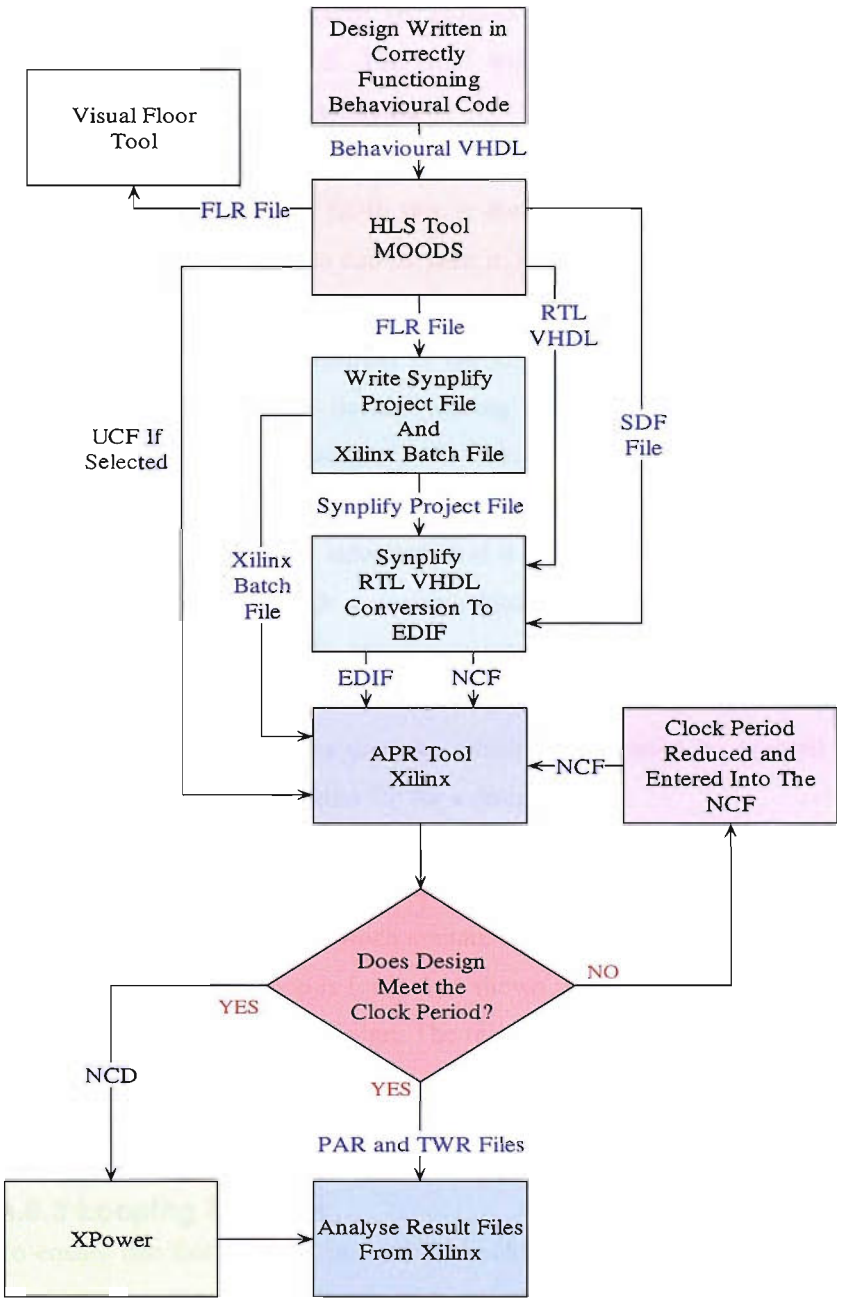


Figure 104 Automation Procedure

Now that the Synplify project file has been written, the design is now ready to proceed to the RTL⇒ EDIF conversion. So the batch file now proceeds to the command line which contains the location of the Synplify executable followed by the project file that is used to provide all the constraints previously discussed, and the location of all the input files and the destination of the output files. During the conversion an SDF is supplied, this tells Synplify which paths to ignore when analysing delays of paths. This

is required for when false paths are introduced to a design through combining of functional units during HLS. The SDF will then prevent false paths by analysed, allowing the true CP to be calculated.

When Synplify outputs the EDIF this is then used in the Xilinx batch file. A break down of the Xilinx batch file can be seen in figure 98 (Appendix A.4). Two constraint files are passed on to Xilinx, one file from MOODS, and another file from Synplify. The constraint file from MOODS is the UCF that contains suggested groupings of macros. The groupings are decided during circuit partitioning within MOODS, the process was discussed in section 3.10. The AREA_GROUP constraints are used, as they do not force the groupings to be placed in a particular location, allowing Xilinx freedom to use hierarchical information if it aids placement, or to ignore hierarchical information if it will degrade a design's placement. Also is the user desires, they can pass on pin locations, which are dependent on where the macros the IOB pins are connected to are placed on the MOODS floorplan. The other constraint file is the SDF, which in this design process contains which paths should be ignored during timing analysis and what the maximum CP for a design is.

With the following constraint files the batch file is run, at the end of the batch file PAR and TWR files are produced which contain all the placement and timing formation of a design. At this point a loop is formed as shown in figure 104, this is due to wanting to obtain the smallest CP for a design. The reason we wish to obtain the smallest CP for a design is that different design architectures formed from different design objectives in HLS can be accurately compared.

A.6.3 Looping Process

To ensure that fair comparisons can be made between the CP of two different design architectures of the same design and objective function (during HLS), the absolute minimum CP needs to be found for every single design architecture. This is why we run a design in Xilinx with a CP constraint in the NCF. If the design manages to reach at CP, the CP constraint is reduced by two ns. If the design fails to meet the CP the design is run through once more but with the CP constraint increased by one. Again all timing and physical information is obtained through the PAR and TWR file.

A.7 Results

Bench Marks	Predicted area	Actual Area	%Error
bench_1(20)	920.477	845	8.932189
bench_1(50)	888.477	847	4.89693
bench_2(20)	921.715	889	3.679978
bench_2(50)	1211.48	1193	1.549036
bench_3(20)	1145.48	1040	10.14231
bench_3(50)	1176.48	998	17.88377
bench_4(20)	1210.72	1102	9.865699
bench_4(50)	1178.72	1050	12.25905
bench_5(20)	1416.1	1338	5.83707
bench_5(50)	1733.15	1678	3.286651
bench_6(20)	1559.1	1582	-1.44753
bench_6(50)	1571.15	1551	1.299162
bench_7(20)	1653.91	1594	3.758469
bench_7(50)	1748.15	1647	6.141469
bench_8(20)	1603.67	1572	2.014631
bench_8(50)	1677.95	1562	7.423175
bench_9(20)	1538.43	1514	1.613606
bench_9(50)	1581.95	1496	5.745321
bench_10(20)	1909.43	1898	0.602213
bench_10(50)	1936.95	1867	3.746652
GCD_GCD(20)	123.861	121	2.364463
GCD_GCD(50)	123.861	121	2.364463
matrix_calc_1(20)	2788.48	2978	-6.364
matrix_calc_1(50)	2788.48	2978	-6.364
matrix_calc_2(20)	6035.28	5589	7.98497
matrix_calc_2(50)	7837.85	7765	0.938184
matrix_calc_3(20)	4072.63	4338	-6.11734
matrix_calc_3(50)	4072.63	4338	-6.11734
matrix_calc_4(20)	2842.46	3023	-5.97221
matrix_calc_4(50)	2842.46	3023	-5.97221
matrix_calc(20)	8353.91	8283	0.856091
matrix_calc(50)	10156.5	10541	-3.64766
insertion(20)	691.53	755	-8.40662
insertion(50)	691.53	755	-8.40662
main_quadratic(20)	2262.08	2110	7.207583
main_quadratic(50)	2127.13	1956	8.748978
merge(20)	4390.07	4302	2.047187
merge(50)	4349.35	4308	0.959842
heap(20)	4151.35	3897	6.526815
heap(50)	4067.58	4023	1.108128
quick(20)	5936.15	5879	0.972104
quick(50)	5909.96	5665	4.324095
B_alg(20)	4812.5	4332	11.09187
B_alg(50)	4621.25	4281	7.947909

Figure 105. Table A1: % error Between the Estimated Area and Actual Area of a design once placed on a Virtex Chip

Name of Bench mark (CP target in MOODS)	AREA	AREA C	AI	AI C	WN	WN C	W10	W10 C	CP	CP C
bench_1(20)	-1.89	0	0.04	-4.69	-9.02	-33.93	-7.39	-16.86	2.11	-2.06
bench_1(50)	0	0	-3.35	3.72	0.62	12.23	5.84	15.32	0.06	0.83
bench_2(20)	-1.8	0	0.78	-8.18	8.14	-16.69	-0.09	-11.11	-0.37	-2.95
bench_2(50)	0	0	0.65	0.65	-10.67	-10.67	-6.31	-6.31	3.43	3.43
bench_3(20)	0	0	6.48	3.44	11.83	11.93	0.36	-3.91	2.59	5.11
bench_3(50)	0	0	-1.32	-3.61	-23.49	-19.42	-10.23	-10.29	2.61	-0.08
bench_4(20)	-0.09	-0.09	-1.11	3.36	7.21	23.52	-4.67	10.42	1.75	1.97
bench_4(50)	0.1	0.1	1.43	-3.2	15.34	11.06	-0.79	-4.21	-2.05	-4.4
bench_5(20)	1.05	0	0.08	-1.76	-5.73	0.23	0.23	-4.89	-1.07	-2.03
bench_5(50)	0.3	0.3	-1.27	-4.17	-5.76	3.75	-6.22	-14.11	-5.06	-4
bench_6(20)	0.06	0	2.64	1.84	-10.28	-10.88	0.68	3.31	1.26	0.28
bench_6(50)	0.06	0	-2.5	1.51	4.29	3.57	11.35	6.49	10.57	6.89
bench_7(20)	-0.94	-1	1.89	3.65	6.26	7.71	6.33	6.99	1.4	3.14
bench_7(50)	0	0	6.38	1.51	3.67	-23.08	3.24	0.97	-1.68	-0.47
bench_8(20)	-0.06	-0.06	-7.64	-1.76	-21.99	-10.53	-11.66	-7.23	-0.13	0.74
bench_8(50)	-0.06	-0.06	3.03	3	0.79	12.69	4.72	9.1	1.46	3.02
bench_9(20)	-0.07	-0.07	4.12	4.12	-5.62	-5.62	12.33	12.33	18.92	18.92
bench_9(50)	-0.07	0	-2.8	1.21	-3.04	0.47	-2.2	-5.09	11.1	14.83
bench_10(20)	-0.79	0	6.2	-0.47	1.49	8.8	1.29	1.8	-2.2	-5.59
bench_10(50)	-0.91	0	12.97	9.96	-3.05	6.99	8.31	10.85	-0.33	-2.07
GCD(20)	0	0	-6.45	-3.48	-3.85	-4.09	-12.2	-0.69	2.9	-0.51
GCD(50)	0	0	-6.45	-3.48	-3.85	-4.09	-12.2	-0.69	2.9	-0.51
matrix_calc_1(20)	1.21	1.04	2.61	2.44	28.88	33.16	14.85	17.17	-0.12	3.01
matrix_calc_1(50)	1.21	1.04	2.61	2.44	28.88	33.16	14.85	17.17	-0.12	3.01
matrix_calc_2(20)	1.91	1.02	-0.62	0.56	-31.57	-14.94	3.13	7.05	0.32	-2.45
matrix_calc_2(50)	1.93	1.08	-2.61	0.99	-0.28	2.99	4.28	3.09	-1.42	-3.06
matrix_calc_3(20)	2.26	2.31	4.14	4.83	-11.97	-10.3	8.17	6.52	-0.02	0.05
matrix_calc_3(50)	2.26	2.31	4.14	4.83	-11.97	-10.3	8.17	6.52	-0.02	0.05
matrix_calc_4(20)	2.58	2.35	4.4	3.56	17	3.93	11.55	8.79	-4.41	-4.02
matrix_calc_4(50)	2.58	2.35	1.91	1.05	21.76	9.45	23.58	21.2	26.27	26.54
matrix_calc(20)	2.39	1.57	-4.56	-2.91	30.04	28.92	13.67	8.67	-2.25	-0.11
matrix_calc(50)	2.52	1.4	-4.74	-3.06	10.2	5.72	7.84	-0.24	-0.18	-3.02
insertionsort(20)	0.93	0.79	-2.5	2.73	3.19	-0.08	-0.45	4.62	0.82	5.01
insertionsort(50)	0.93	0.79	-2.5	2.73	3.19	-0.08	-0.45	4.62	0.82	5.01
quadratic(20)	2.46	1.71	-4.37	-5.2	40.64	29.99	5.62	3.56	2.72	-2.63
quadratic(50)	1.89	1.58	0.24	0.72	-1.58	1.85	-0.91	2.93	-0.02	-0.54
mergesort(20)	1.65	1.67	-2.26	-2.52	1.53	-6.1	-3.72	-11.09	-0.52	-3.01
mergesort(50)	1.46	1.46	1.95	1.61	7.98	13.39	3.77	7.79	0.08	-4.06
heapsort(20)	0	-0.03	2.05	-9.08	-3.8	-30.7	-5.86	-22.32	-2.26	1.11
heapsort(50)	0.25	0.25	3.06	2.29	12.51	10.36	13.28	10.51	-0.73	1.71
quicksort(20)	0.85	0.83	0.5	0.42	4.3	-6.45	2.06	-6.7	-3.75	-9.21
quicksort(50)	0.85	0.85	-5.27	0.37	-9.17	9.64	-8.76	5.5	0.32	-0.22
B_alg(20)	0.16	-0.6	-3.48	-6.35	-9.84	-11.1	-10.07	-19.01	33.01	33.66
B_alg(50)	1.4	1.12	-1.81	-3.92	-11.51	-53.81	-10.18	-29.6	-2.17	1.42

C represents when a restriction on the groups are passed onto Xilinx. AI = Average Interconnect Delay, WN = Worst Net Delay, W10 = Average Worst Ten Net Delays, CP = Clock Period.

Figure 106. Table showing % improvement of the physical metrics of a design when a design has been placed and routed with location constraints obtained from circuit partitioning during HLS.

Name of Bench mark (CP target in MOODS)	S10	S25	S50	S75	S100	S150
bench_1(20)	0	0.24	0.24	0.59	0.59	0.59
bench_1(50)	0	0.47	0.47	0.47	0.47	0.47
bench_2(20)	0.11	0.11	0.11	0.11	0.11	0.11
bench_2(50)	0	0	0	0	0	-0.5
bench_3(20)	1.54	1.54	1.54	1.54	1.54	1.54
bench_3(50)	-0.2	-0.2	-0.2	-0.2	-0.2	-0.2
bench_4(20)	2.27	2.27	2.27	2.27	2.27	2.27
bench_4(50)	-0.48	-0.19	-0.19	-0.19	-0.19	-0.19
bench_5(20)	1.94	0.52	2.17	2.17	-1.42	-1.42
bench_5(50)	-1.85	4.35	-1.01	-1.01	-57.99	-57.99
bench_6(20)	-2.47	1.77	-0.44	-0.44	-0.44	-0.44
bench_6(50)	-2.45	-2.06	-1.29	-1.29	-1.29	-1.29
bench_7(20)	-0.13	-1.69	-2.38	-2.95	-2.95	-2.95
bench_7(50)	0.55	-0.67	0.43	0.43	0.43	0.43
bench_8(20)	4.2	4.26	4.83	4.83	4.01	4.01
bench_8(50)	0.58	1.92	1.86	1.86	2.3	2.3
bench_9(20)	3.1	3.1	3.1	3.1	3.1	3.1
bench_9(50)	1.94	1.94	1.94	1.94	1.94	1.94
bench_10(20)	2.37	1.16	1.63	1.74	1.74	1.74
bench_10(50)	-0.05	1.55	1.55	1.55	1.55	1.55
GCD(20)	0	-1.65	-1.65	-1.65	-1.65	-1.65
GCD(50)	0	-1.65	-1.65	-1.65	-1.65	-1.65
matrix_calc_1(20)	0.67	0.67	0.67	0.67	0.67	0.67
matrix_calc_1(50)	0.67	0.67	0.67	0.67	0.67	0.67
matrix_calc_2(20)	1.83	1.95	-0.2	-0.2	0.09	1.27
matrix_calc_2(50)	-0.52	0.12	0.12	0.12	-0.31	-0.31
matrix_calc_3(20)	0.55	0.97	1.2	0.37	0.37	0.37
matrix_calc_3(50)	0.55	0.97	1.2	0.37	0.37	0.37
matrix_calc_4(20)	1.16	0.23	0	0.23	0.23	0.23
matrix_calc_4(50)	1.16	0.23	0.23	0.23	0.23	0.23
matrix_calc(20)	-0.74	-0.1	-0.1	-0.1	-0.1	-0.1
matrix_calc(50)	-0.83	-1.22	-1.22	-1.22	-0.73	-0.73
insertion(20)	0	0	0	4.11	4.11	4.11
insertion(50)	0	0	0	4.11	4.11	4.11

All criteria have equal priority. S represents the scaling factor used when calculating E in conjunction with the cost function, where * is an integer.*

Figure 107. Table showing % improvement of the Area of a design using a value that represents the Average Interconnect (AI) in the cost function against a design that does not use AI in the cost function.

Name of Bench mark (CP target in MOODS)	S10	S25	S50	S75	S100	S150
bench_1(20)	0	17.44	17.44	31.85	31.85	31.85
bench_1(50)	0	17.96	17.96	22.81	22.81	22.81
bench_2(20)	5.34	5.34	5.34	5.34	5.34	5.34
bench_2(50)	3.63	3.63	3.63	3.63	3.63	35.15
bench_3(20)	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03
bench_3(50)	0.16	0.16	0.16	0.16	0.16	0.16
bench_4(20)	42.2	42.2	42.2	42.2	42.2	42.2
bench_4(50)	21.33	32.3	32.3	32.3	32.3	32.3
bench_5(20)	73.29	76.42	72.17	72.17	-229.29	-229.29
bench_5(50)	72.49	76.17	77.86	77.86	78.81	78.81
bench_6(20)	59.6	51.32	66.49	66.49	66.49	66.49
bench_6(50)	45.2	44.47	47.5	47.5	47.5	47.5
bench_7(20)	27.5	7.97	6.62	37.78	37.78	37.78
bench_7(50)	17.86	16.03	15.01	15.01	15.01	15.01
bench_8(20)	-827.14	19.29	13.31	13.31	44.89	44.89
bench_8(50)	36.63	21.14	-4.51	-4.51	34.95	34.95
bench_9(20)	53.81	53.81	53.81	53.81	53.81	53.81
bench_9(50)	37.13	37.13	37.13	37.13	37.13	37.13
bench_10(20)	-555.23	28.23	37.63	49.85	49.85	49.85
bench_10(50)	18.42	9.37	9.37	9.37	9.37	9.37
GCD(20)	0	19.87	19.87	19.87	19.87	19.87
GCD(50)	0	19.87	19.87	19.87	19.87	19.87
matrix_calc_1(20)	2.98	2.98	2.98	2.98	2.98	2.98
matrix_calc_1(50)	2.98	2.98	2.98	2.98	2.98	2.98
matrix_calc_2(20)	3.12	4.95	2.77	2.77	2.64	2.36
matrix_calc_2(50)	-12.5	-4.66	-1.12	-7.72	-2.96	-2.96
matrix_calc_3(20)	-0.02	-3.25	-3.39	-6.5	-6.5	-6.5
matrix_calc_3(50)	-0.02	-3.25	-3.39	-6.5	-6.5	-6.5
matrix_calc_4(20)	-9.18	0.65	0	0.65	0.65	0.65
matrix_calc_4(50)	22.9	29.84	29.84	29.84	-5.6	-5.6
matrix_calc(20)	23.53	21.82	0.56	21.82	21.82	21.82
matrix_calc(50)	-0.22	-1.17	-1.17	-1.17	-0.07	-0.07
insertion(20)	0	0	0	2.86	2.86	2.86
insertion(50)	0	0	0	2.86	2.86	2.86

All criteria have equal priority. S represents the scaling factor used when calculating E in conjunction with the cost function, where * is an integer.*

Figure 108. Table showing % improvement of the CP of a design using a value that represents the Average Interconnect (AI) in the cost function against a design that does not use AI in the cost function.

Name of Bench mark (CP target in MOODS)	S25 M10(m)	S25 M10(c)	S25 M25(m)	S25 M25(c)	S25 M50(m)	S25 M50(c)
bench_1(20)	17.44	17.44	17.44	17.44	17.44	17.44
bench_1(50)	17.96	17.96	17.96	17.96	17.96	17.96
bench_2(20)	6.07	5.76	6.07	5.76	5.34	5.34
bench_2(50)	42.32	40.95	42.32	40.95	35.15	3.63
bench_3(20)	47.38	47.38	34.75	34.75	5.34	5.34
bench_3(50)	52.68	52.68	44.69	44.69	0.49	0.49
bench_4(20)	42.2	42.2	42.2	42.2	42.2	42.2
bench_4(50)	37.32	32.3	32.3	32.3	32.3	32.3
bench_5(20)	83.17	80.45	83.17	80.44	74.3	74.21
bench_5(50)	85.56	78.36	85.56	76.13	76.14	76.14
bench_6(20)	53.41	50.59	50.59	50.59	51.32	51.32
bench_6(50)	36.84	35.01	35.01	35.01	44.48	44.48
bench_7(20)	39.45	39.45	23.79	23.79	12.42	12.42
bench_7(50)	26.66	26.66	31.41	31.41	34.85	34.85
bench_8(20)	35.21	35.21	15.68	2.29	2.29	19.27
bench_8(50)	44.41	44.41	33.52	33.52	33.52	34.95
bench_9(20)	62.41	62.41	53.81	53.81	53.81	53.81
bench_9(50)	41.82	46.54	37.13	37.13	37.13	37.13
bench_10(20)	52.98	52.98	52.98	52.98	46.24	46.24
bench_10(50)	41.9	33.59	33.59	45.15	41.75	41.75

All criteria have equal priority. M represents the limiting factor used when deciding whether to merge to data path units, where * is an integer. M is at which point interconnects which are involved, are measured from. m is from the centre point and c is from the closest point between the two macros that the interconnect connects.*

Figure 109. Table showing % improvement of the CP of a design using a value that represents the Average Interconnect (AI) in the cost function against a design that does not use AI in the cost function.

Name of Bench mark (CP target in MOODS)	S25	S25 and A0.125	S25 and A0.25	S25 and A0.05	S25 and A1
bench_1(20)	0.24	0.59	0.59	0.12	0.12
bench_1(50)	0.47	0.47	0.47	0.24	0.24
bench_2(20)	0.11	0.11	0.11	0.11	0.11
bench_2(50)	0	0	0	0	0
bench_3(20)	1.54	1.54	1.54	-0.1	-0.1
bench_3(50)	-0.2	-0.2	-0.2	-2.1	-2.1
bench_4(20)	2.27	1	1	5.72	2.99
bench_4(50)	-0.19	-0.29	-0.29	-0.19	-0.1
bench_5(20)	0.52	1.49	1.87	4.78	3.36
bench_5(50)	4.35	4.35	1.67	4.47	-0.6
bench_6(20)	1.77	0.44	3.67	0.44	0.44
bench_6(50)	-2.06	1.48	2.26	-0.64	-0.64
bench_7(20)	-1.69	0.19	1.88	1.44	1.44
bench_7(50)	-0.67	0.36	1.03	1.58	0.36
bench_8(20)	4.26	5.34	5.34	4.58	4.83
bench_8(50)	1.92	3.07	3.07	0	1.6
bench_9(20)	3.1	3.1	3.1	2.77	2.77
bench_9(50)	1.94	1.94	1.94	1.6	1.6
bench_10(20)	1.16	2.21	2.21	2.42	2.48
bench_10(50)	1.55	0	0	1.71	1.71

All criteria have equal priority. A represents the scaling factor used to increase (if less than 0) the influence of AI in the Cost function, where * is an integer. S25 represents the scaling factor for AI.*

Figure 110. Table showing % improvement of the Area of a design using a value that represents the Average Interconnect (AII) in the cost function against a design that does not use interconnect prediction to aid synthesis.

Name of Bench mark (CP target in MOODS)	S25	S25 And A0.125	S25 and A0.25	S25 and A0.05	S25 and A1
bench_1(20)	17.44	31.85	31.85	13.37	13.37
bench_1(50)	17.96	22.81	22.81	19.67	19.67
bench_2(20)	5.34	5.34	5.34	5.34	5.34
bench_2(50)	3.63	3.63	3.63	3.63	3.63
bench_3(20)	-0.03	-0.03	-0.03	2.48	2.48
bench_3(50)	0.16	0.16	0.16	0.2	0.2
bench_4(20)	42.2	37.93	37.93	44.48	46.6
bench_4(50)	32.3	32.55	32.55	32.3	39.78
bench_5(20)	76.42	75.53	75.89	77.56	80.36
bench_5(50)	76.17	76.17	78.67	77.9	76.19
bench_6(20)	51.32	64.9	71.58	69.44	69.44
bench_6(50)	44.47	40.5	47.64	46.79	46.79
bench_7(20)	7.97	45.15	47.3	49.78	49.78
bench_7(50)	16.03	42.73	43.37	42.41	32.06
bench_8(20)	19.29	35.44	35.44	41.25	15.73
bench_8(50)	21.14	39.36	39.36	35.75	21.61
bench_9(20)	53.81	53.81	53.81	37.55	37.55
bench_9(50)	37.13	37.13	37.13	21.32	21.32
bench_10(20)	28.23	-0.11	-0.11	30.24	36.04
bench_10(50)	9.37	23.25	23.25	37.17	37.17

All criteria have equal priority. A represents the scaling factor used to increase (if less than 0) the influence of AI in the Cost function, where * is an integer. S25 represents the scaling factor for AI.*

Figure 111. Table showing % improvement of the CP of a design using a value that represents the Average Interconnect (AII) in the cost function against a design that does not use interconnect prediction to aid synthesis.

Name of Bench mark (CP target in MOODS)	S25	S25 M10	S25 M25	S25 M50	S25 M75
bench_1(20)	0.24	0.24	0.24	0.24	0.24
bench_1(50)	0.47	0.47	0.47	0.47	0.47
bench_2(20)	0.11	0.11	0.11	0.11	0.11
bench_2(50)	0	-27.49	-27.49	-0.5	0
bench_3(20)	1.54	-25.58	-29.81	0.67	0.67
bench_3(50)	-0.2	-29.76	-32.16	-2.51	-2.51
bench_4(20)	2.27	2.27	2.27	2.27	2.27
bench_4(50)	-0.19	-0.29	-0.19	-0.19	-0.19
bench_5(20)	0.52	-121.45	-121.45	0.6	4.33
bench_5(50)	4.35	-74.26	-74.26	1.49	2.56
bench_6(20)	1.77	0.06	3.98	1.77	1.77
bench_6(50)	-2.06	-2.13	2.13	-2.06	-2.06
bench_7(20)	-1.69	-1.38	-0.25	-0.19	-1.69
bench_7(50)	-0.67	1.34	0.67	1.82	0.43
bench_8(20)	4.26	5.22	5.28	4.96	4.26
bench_8(50)	1.92	1.6	2.69	2.69	1.92
bench_9(20)	3.1	3.17	3.1	3.1	3.1
bench_9(50)	1.94	1.67	1.94	1.94	1.94
bench_10(20)	1.16	2.95	2.95	1.58	1.58
bench_10(50)	1.55	1.12	1.02	1.23	1.55

All criteria have equal priority. M represents the limiting factor used when deciding whether to merge two data path units, where * is an integer. S25 represents the scaling factor for AI.*

Figure 112. Table showing % improvement of the Area of a design using different limits for the distance at which candidates put forward for merging need to be within, against a design that does not use interconnect prediction to aid synthesis.

Name of Bench mark (CP target in MOODS)	S25	S25 M10	S25 M25	S25 M50	S25 M75
bench_1(20)	17.44	17.44	17.44	17.44	17.44
bench_1(50)	17.96	17.96	17.96	17.96	17.96
bench_2(20)	5.34	6.07	5.76	6.07	5.76
bench_2(50)	3.63	42.32	40.95	42.32	40.95
bench_3(20)	-0.03	47.38	47.38	34.75	34.75
bench_3(50)	0.16	52.68	52.68	44.69	44.69
bench_4(20)	42.2	42.2	42.2	42.2	42.2
bench_4(50)	32.3	37.32	32.3	32.3	32.3
bench_5(20)	76.42	83.17	80.45	83.17	80.44
bench_5(50)	76.17	85.56	78.36	85.56	76.13
bench_6(20)	51.32	53.41	50.59	50.59	50.59
bench_6(50)	44.47	36.84	35.01	35.01	35.01
bench_7(20)	7.97	39.45	39.45	23.79	23.79
bench_7(50)	16.03	26.66	26.66	31.41	31.41
bench_8(20)	19.29	35.21	35.21	15.68	2.29
bench_8(50)	21.14	44.41	44.41	33.52	33.52
bench_9(20)	53.81	62.41	62.41	53.81	53.81
bench_9(50)	37.13	41.82	46.54	37.13	37.13
bench_10(20)	28.23	52.98	52.98	52.98	52.98
bench_10(50)	9.37	41.89	33.59	33.59	45.15

All criteria have equal priority. M represents the limiting factor used when deciding whether to merge two data path units, where * is an integer. S25 represents the scaling factor for AI.*

Figure 113. Table showing % Improvement of the CP of a design using different limits for the distance at which candidates put forward for merging need to be within, against a design that does not use interconnect prediction to aid synthesis.

Name of Bench mark (CP target in MOODS)	S25	S25&M50	S25&M50 & U5	S25&M50 & U10	S25&M50 & U25	S25&M50 & U50
bench_1(20)	0.24	0.24	0.24	0.59	0.59	0.59
bench_1(50)	0.47	0.47	0.47	0.47	0.47	0.47
bench_2(20)	0.11	0.11	0.11	0.11	0.11	0.11
bench_2(50)	0	-0.5	-0.5	-0.5	-0.5	-0.5
bench_3(20)	1.54	0.67	0.67	2.5	2.5	2.5
bench_3(50)	-0.2	-2.51	-2.51	-0.7	-0.7	-0.7
bench_4(20)	2.27	2.27	2.27	1	1	1
bench_4(50)	-0.19	-0.19	-0.19	-0.29	-0.29	-0.29
bench_5(20)	0.52	0.6	-96.86	-22.2	-22.2	-22.2
bench_5(50)	4.35	1.49	1.49	2.03	2.03	2.03
bench_6(20)	1.77	1.77	1.77	-1.2	-1.2	-1.2
bench_6(50)	-2.06	-2.06	-2.06	-4.84	-4.84	-4.84
bench_7(20)	-1.69	-0.19	-0.19	1.25	1.25	1.25
bench_7(50)	-0.67	1.82	1.82	0.85	0.85	0.85
bench_8(20)	4.26	4.96	4.96	5.34	5.34	3.88
bench_8(50)	1.92	2.69	2.69	3.07	3.07	1.54
bench_9(20)	3.1	3.1	3.1	3.1	3.1	3.1
bench_9(50)	1.94	1.94	1.94	1.94	1.94	1.94
bench_10(20)	1.16	1.58	1.58	1.48	1.37	1.48
bench_10(50)	1.55	1.23	1.23	0	-0.27	0.43

All criteria have equal priority. U represents the limiting factor used when deciding whether to duplicate a data path unit, where * is an integer. S25 represents the scaling factor for AI and M50 represents the limiting factor for the distance candidates for merging need to be within.*

Figure 114. Table showing % improvement of the Area of a design using different limits for the distance at which candidates put forward for duplication need to exceed, against a design that does not use interconnect prediction to aid synthesis.

Name of Bench mark (CP target in MOODS)	S25	S25&M50 & U5	S25&M50 & U10	S25&M50 & U25	S25&M50 & U50
bench_1(20)	17.44	17.44	31.85	31.85	31.85
bench_1(50)	17.96	17.96	22.81	22.81	22.81
bench_2(20)	5.34	5.34	5.34	5.34	5.34
bench_2(50)	3.63	35.15	35.15	35.15	35.15
bench_3(20)	-0.03	5.34	11.03	11.03	11.03
bench_3(50)	0.16	0.49	2.54	2.54	2.54
bench_4(20)	42.2	42.2	37.93	37.93	37.93
bench_4(50)	32.3	32.3	32.55	32.55	32.55
bench_5(20)	76.42	78.31	76.53	76.53	76.53
bench_5(50)	76.17	76.14	77.92	77.92	77.92
bench_6(20)	51.32	51.32	51.93	51.93	51.93
bench_6(50)	44.48	44.48	44.1	44.1	44.1
bench_7(20)	7.97	12.42	45.12	45.12	45.12
bench_7(50)	16.03	34.85	33.98	33.98	33.98
bench_8(20)	19.29	2.29	35.44	35.44	13.47
bench_8(50)	21.14	33.52	39.36	39.36	27.83
bench_9(20)	53.81	53.81	53.81	53.81	53.81
bench_9(50)	37.13	37.13	37.13	37.13	37.13
bench_10(20)	28.23	46.24	22.27	21.5	11.01
bench_10(50)	9.37	41.75	7.93	12.52	26.76

All criteria have equal priority. U represents the limiting factor used when deciding whether to duplicate a data path unit, where * is an integer. S25 represents the scaling factor for AI and M50 represents the limiting factor for the distance candidates for merging need to be within.*

Figure 115. Table showing % improvement of the CP of a design using different limits for the distance at which candidates put forward for duplication need to exceed, against a design that does not use interconnect prediction to aid synthesis.

The following tables are comparing different levels of interconnect prediction during HLS against using no interconnect prediction during HLS for all the benchmarks used in this thesis (except Rijndael which will be discussed by itself). The following different interconnect prediction methodologies are pursued in the following tables:

N = No interconnect Prediction is involved during Synthesis.

A = Average Interconnect Metric is used in MOODS Cost Function (Section 3.10).

B = Methodology A and restriction on merging (Section 4.11.2.1).

C = Methodology B and duplication is introduced with restriction (Section 4.11.2.2).

D = Methodology A and Individual Interconnect Aware (IIA) (Section 4.7.1).

E = Methodology B and IIA.

F = Methodology C and IIA.

These metrics Area, Average Interconnect Delay, Worst Net Delay, Average Worst Ten Net Delay, Clock Period, Total Delay, Combined Clock Period and Area, and finally Combined Total Delay and Area, will be discussed in the following figures.

Name of Bench mark (CP target in MOODS)	A	B	C	D	E	F
bench_1(20)	0.24	0.24	0.59	0.12	0.12	0.12
bench_1(50)	0.47	0.47	0.47	0.24	0.24	0.24
bench_2(20)	0.11	0.11	0.11	0.11	0.11	0.11
bench_2(50)	0	-0.5	-0.5	0	-0.84	-0.84
bench_3(20)	1.54	0.67	2.5	-0.1	0.77	0.77
bench_3(50)	-0.2	-2.51	-0.7	-2.1	-0.4	-0.4
bench_4(20)	2.27	2.27	1	5.72	2.81	2.81
bench_4(50)	-0.19	-0.19	-0.29	-0.19	-0.19	-0.19
bench_5(20)	0.52	0.6	-22.2	4.78	-20.48	-19.13
bench_5(50)	4.35	1.49	2.03	4.47	-18.42	3.52
bench_6(20)	1.77	1.77	-1.2	0.44	3.48	1.26
bench_6(50)	-2.06	-2.06	-4.84	-0.65	1.1	1.1
bench_7(20)	-1.69	-0.19	1.26	1.44	1.51	1.51
bench_7(50)	-0.67	1.82	0.85	1.58	-0.24	-0.24
bench_8(20)	4.26	4.96	5.34	4.58	4.64	4.58
bench_8(50)	1.92	2.69	3.07	0	2.75	2.43
bench_9(20)	3.1	3.1	3.1	2.77	4.1	4.1
bench_9(50)	1.94	1.94	1.94	1.6	3.54	3.54
bench_10(20)	1.16	1.58	1.37	2.42	2.11	2.11
bench_10(50)	1.55	1.23	-0.27	1.71	1.13	1.13
GCD(20)	-1.65	-1.65	-5.79	-1.65	-1.65	9.09
GCD(50)	-1.65	-1.65	-5.79	-1.65	-1.65	9.09
matrix_calc_1(20)	0.67	0.13	0.13	-1.31	0.13	-1.31
matrix_calc_1(50)	0.67	0.13	0.13	-1.31	0.13	-1.31
matrix_calc_2(20)	1.95	-5.51	-11.18	1.13	-4.78	1.75
matrix_calc_2(50)	0.12	0.04	-0.9	-0.94	-0.19	-0.61
matrix_calc_3(20)	0.97	0.53	0.76	0.65	0.3	0.65
matrix_calc_3(50)	0.97	0.53	0.76	0.65	0.3	0.65
matrix_calc_4(20)	0.7	0.7	0.7	0.43	0.7	0.27
matrix_calc_4(50)	0.7	0.7	0.7	0.43	0.7	0.27
matrix_calc(20)	-0.35	-0.62	-4.37	-24.79	-8.79	-0.56
matrix_calc(50)	-0.73	-0.66	-0.65	-0.78	-0.44	-0.47
insertionsort(20)	-1.85	4.77	-21.19	15.23	-9.01	-9.01
insertionsort(50)	-1.85	4.77	1.59	15.23	-9.01	-9.01
quadratic(20)	0.57	0.57	0.57	0.57	0.57	1.19
quadratic(50)	0.97	0.31	0.31	0.87	0.46	0.46
mergesort(20)	-1.35	-3.81	-3.53	-25.66	-16.27	-3.77
mergesort(50)	-11.37	-7.54	-1.09	-20.4	-14.55	-7.59
heapsort(20)	-14.14	-13.81	-2.72	-18.01	-13.7	-13.7
heapsort(50)	-13.25	-14.62	-10.42	-11.53	-13.95	-13.95
quicksort(20)	-5.46	-5.46	-6.69	2.62	2.74	-9.44
quicksort(50)	-8.69	-8.69	-11.76	-3.65	-3.65	-9.85
B_alg(20)	1.96	-8.2	-10.76	-14.94	-10.76	-10.76
B_alg(50)	1.96	-12.08	-9.37	-11.66	2.92	-10.82

Figure 116. Table showing % improvement of the Area of a design using different levels of interconnect prediction during synthesis within MOODS

Name of Bench mark (CP target in MOODS)	A	B	C	D	E	F
bench_1(20)	0.04	0.04	2.49	0.71	0.71	0.71
bench_1(50)	-0.63	-0.63	1.67	-0.41	-0.37	-0.41
bench_2(20)	2.52	2.52	2.52	2.52	2.52	2.52
bench_2(50)	0	-4.89	-4.89	0	-15.57	-15.57
bench_3(20)	-2.42	0.74	1.07	-4.96	5.37	5.37
bench_3(50)	-4.01	-5.9	-4.05	-11.5	-4.67	-4.67
bench_4(20)	3.44	3.44	0.36	10.52	10.41	10.41
bench_4(50)	2.79	2.79	4.74	2.79	2.79	2.79
bench_5(20)	6.51	7.73	-0.12	13.73	5.14	8.78
bench_5(50)	-1.55	-0.74	2.37	-3.19	-8.83	7.65
bench_6(20)	5.78	5.78	4.55	9.83	7.82	7.49
bench_6(50)	-1.62	-1.62	-5.91	-0.85	0.29	0.29
bench_7(20)	-2.1	-0.97	-0.38	6.41	6.17	6.17
bench_7(50)	3.88	4.14	0.69	10.03	5.62	5.62
bench_8(20)	-0.04	-8.38	2.45	-0.84	3.8	1.76
bench_8(50)	1.04	-6.46	6.42	-3.57	0.71	-0.07
bench_9(20)	-1.31	-1.31	-1.31	-6.32	-3.37	-3.37
bench_9(50)	-3.6	-3.6	-3.6	-6.32	-5.41	-5.41
bench_10(20)	4.44	2.78	0.31	2.71	1.56	1.56
bench_10(50)	13.34	11.51	10.18	12.44	6.14	6.14
GCD(20)	-1.64	-1.64	-0.79	-1.64	-1.64	-3.63
GCD(50)	-1.64	-1.64	-0.79	-1.64	-1.64	-3.63
matrix_calc_1(20)	-6.7	3.69	3.69	6.41	3.69	6.41
matrix_calc_1(50)	-6.7	3.69	3.69	6.41	3.69	6.41
matrix_calc_2(20)	-1.98	2.05	2.29	-0.63	1.6	2.36
matrix_calc_2(50)	-2.16	-2.57	-2.16	-0.07	1.3	0.72
matrix_calc_3(20)	4.73	3.88	3.1	3.17	3.62	3.17
matrix_calc_3(50)	4.73	3.88	3.1	3.17	3.62	3.2
matrix_calc_4(20)	4.58	4.58	4.58	0.53	4.58	1.8
matrix_calc_4(50)	2.1	2.1	2.1	-2.06	2.1	-0.76
matrix_calc(20)	-2.4	-1.59	-3.35	-2.77	4.02	0.34
matrix_calc(50)	1.28	-0.5	2.35	-2.92	0.64	-0.3
insertionsort(20)	4.99	2.27	0.23	16.67	-3.92	-3.92
insertionsort(50)	4.99	2.27	1.84	16.67	0.65	0.65
quadratic(20)	-1.91	-1.91	-1.91	1.95	1.58	2.56
quadratic(50)	1.67	4.18	4.18	2.07	4.5	4.5
mergesort(20)	-1.08	3.53	-2.21	-1.83	-4.63	4.63
mergesort(50)	6.97	8.26	3.51	3.22	7.59	10.44
heapsort(20)	-5.86	-10.93	0.24	-3.31	-8.7	-8.7
heapsort(50)	1.04	4.68	4.45	-2.06	5.12	5.12
quicksort(20)	2.04	2.04	6.96	0.04	5.94	7.6
quicksort(50)	-2.15	-2.15	6.17	-4.42	-4.42	8.04
B_alg(20)	-0.58	-5.42	-14.64	-4.11	-11.89	-15.42
B_alg(50)	5.75	-8.5	-2.38	-0.46	10.75	-7.47

Figure 117. Table showing % improvement of the Average Interconnect Delay of a design using different levels of interconnect prediction during synthesis within MOODS

Name of Bench mark (CP target in MOODS)	A	B	C	D	E	F
bench_1(20)	-18.57	-18.57	-5.8	-10.28	-10.28	-10.28
bench_1(50)	2.91	2.91	12.26	7.87	3.72	7.87
bench_2(20)	-2.33	-2.33	-2.33	-2.33	-2.33	-2.33
bench_2(50)	0	-39.17	-39.17	0	-11.58	-11.58
bench_3(20)	-9.49	-3.78	0.3	-9.69	-7.39	-7.39
bench_3(50)	-36.1	-15.38	-22.62	-63.03	-26.12	-26.12
bench_4(20)	11.93	11.93	18.11	9.01	25.42	25.42
bench_4(50)	-8.97	-8.97	3.25	-8.97	-8.97	-8.97
bench_5(20)	-42.68	-18.03	-26.97	-8.41	-20.16	-11.7
bench_5(50)	-7.86	-22.39	9.08	5.87	-24.91	-9.8
bench_6(20)	-6.79	-6.79	0.79	-6.09	-15.51	8.47
bench_6(50)	5.21	5.21	-1.69	-0.95	6.27	6.27
bench_7(20)	0.11	-10.33	-15.96	5.29	5.5	5.5
bench_7(50)	-14.16	10.22	-10.94	-0.51	-9.15	-9.15
bench_8(20)	-11.31	-19.77	-28.27	-12.22	-0.56	2.89
bench_8(50)	-15.48	-8.35	6.16	-7.93	-0.66	3.28
bench_9(20)	7.3	7.3	7.3	-7.89	9.56	9.56
bench_9(50)	-36.54	-36.54	-36.54	-25.38	-9.15	-9.15
bench_10(20)	14.77	7.43	7.43	1.38	9.33	9.33
bench_10(50)	17.66	-2.01	-26.16	5.56	-8.23	-8.23
GCD(20)	11.28	11.28	4.58	11.28	11.28	-1.02
GCD(50)	11.28	11.28	4.58	11.28	11.28	-1.02
matrix_calc_1(20)	-3.55	20.43	20.43	20.52	20.43	20.52
matrix_calc_1(50)	-3.55	20.43	20.43	20.52	20.43	20.52
matrix_calc_2(20)	13.39	13.8	16.96	5.46	11.48	3.47
matrix_calc_2(50)	-8.94	-6.17	5.31	-1.64	-0.68	-7.42
matrix_calc_3(20)	-17.01	1.39	-32.61	-2.12	-19.34	-2.12
matrix_calc_3(50)	-17.01	1.39	-32.61	-2.12	-19.34	1.46
matrix_calc_4(20)	6.79	6.79	6.79	-0.17	6.79	-1.24
matrix_calc_4(50)	12.14	12.14	12.14	5.58	12.14	4.58
matrix_calc(20)	-18.06	24.15	21.55	18.27	4.99	27.19
matrix_calc(50)	11.06	9.33	1.4	1.68	4.55	7.19
insertionsort(20)	8.34	8.89	4.03	24.58	-0.78	-0.78
insertionsort(50)	8.34	8.89	10.17	24.58	7.87	7.87
Quadratic(20)	44.75	44.75	44.75	18.53	42.94	33.12
Quadratic(50)	5.55	1.29	1.29	9.64	18.16	18.16
mergesort(20)	-1.5	-3.02	-19.2	8.27	8.53	4.84
mergesort(50)	16.52	5.6	9.47	22.22	14.6	26.18
Heapsort(20)	-0.38	-18.72	14.91	-11.18	-10.81	-10.81
Heapsort(50)	6.56	-2.24	-0.63	-30.51	16.85	16.85
Quicksort(20)	2.99	2.99	-5.97	-1.92	8.26	-4.74
Quicksort(50)	3.79	3.79	7.72	7.92	7.92	13.16
B_alg(20)	-29.63	-34.35	-21.34	0.58	-18.4	-22.24
B_alg(50)	-15.17	-35.17	-26.71	-24.24	8.74	-43.87

Figure 118. Table showing % improvement of the Worst Net Delay of a design using different levels of interconnect prediction during synthesis within MOODS

Name of Bench mark (CP target in MOODS)	A	B	C	D	E	F
bench_1(20)	-15.45	-15.45	-2.96	-7.11	-7.11	-7.11
bench_1(50)	2.52	2.52	8.44	5.76	3.02	5.76
bench_2(20)	-8.25	-8.25	-8.25	-8.25	-8.25	-8.25
bench_2(50)	0	-18.53	-18.53	0	-19.89	-19.89
bench_3(20)	-20.94	-22.05	-10.56	-23.02	0.46	0.46
bench_3(50)	-15.24	-2.21	-11.88	-27.42	-13.98	-13.98
bench_4(20)	-1.46	-1.46	5.11	7.94	10.9	10.9
bench_4(50)	-1.57	-1.57	-1.59	-1.57	-1.57	-1.57
bench_5(20)	-26.89	1.45	-20.09	6.61	-15.86	-9.95
bench_5(50)	-16.6	-15.74	-0.92	-5.46	-32.02	1.51
bench_6(20)	6.9	6.9	4.32	5.26	1.98	10.96
bench_6(50)	-0.4	-0.4	-1.81	2.16	7.18	7.18
bench_7(20)	-3.18	-5.59	-11.73	2	5.91	5.91
bench_7(50)	-4.62	14.11	-2.84	12.44	0.16	0.16
bench_8(20)	-7.09	-15.22	-13.81	-9.41	1.74	1.96
bench_8(50)	-2.99	-6.73	5.98	-5.7	3.67	1.92
bench_9(20)	10.9	10.9	10.9	-1.22	3.51	3.51
bench_9(50)	-11.46	-11.46	-11.46	-13.79	-6.25	-6.25
bench_10(20)	10.62	7.62	3.43	5	5.04	5.04
bench_10(50)	17.49	5.43	-0.8	9.48	-0.11	-0.11
GCD(20)	-4.98	-4.98	-7.81	-4.98	-4.98	-3.38
GCD(50)	-4.98	-4.98	-7.81	-4.98	-4.98	-3.38
matrix_calc_1(20)	-1.96	10.88	10.88	3.27	10.88	3.27
matrix_calc_1(50)	-1.96	10.88	10.88	3.27	10.88	3.27
matrix_calc_2(20)	6.96	8.1	16.41	6.93	6.8	1.31
matrix_calc_2(50)	-4.86	-1.49	1.43	1.58	-0.19	-1.91
matrix_calc_3(20)	1.05	-0.93	-4.62	-5.06	-3.12	-5.06
matrix_calc_3(50)	1.05	-0.93	-4.62	-5.06	-3.12	-2.59
matrix_calc_4(20)	-0.22	-0.22	-0.22	-3.74	-0.22	2.81
matrix_calc_4(50)	13.42	13.42	13.42	10.37	13.42	16.03
matrix_calc(20)	5.46	4.98	5.82	-3.44	8.42	13.61
matrix_calc(50)	7.62	6.72	1.83	6.21	0.83	-2.07
insertionsort(20)	0.47	7.24	2.05	21.96	-16.75	-16.75
insertionsort(50)	0.47	7.24	2.45	21.96	3.18	3.18
quadratic(20)	9.44	9.44	9.44	3.77	13.34	2.38
quadratic(50)	0.73	0.59	0.59	0.4	10.46	10.46
mergesort(20)	-9.78	-12.79	-9.99	4	1.19	11.92
mergesort(50)	8.32	6.62	11.28	9.44	9.61	20.97
heapsort(20)	-13.07	-37.36	3.59	-12.34	-26.17	-26.17
heapsort(50)	2.82	-0.71	3.06	-9.24	12.64	12.64
quicksort(20)	-5.55	-5.55	1.89	-5.61	11.11	3
quicksort(50)	-9.35	-9.35	5.98	0.15	0.15	15.99
B_alg(20)	-18.48	-27.13	-30.34	-8.21	-29.1	-34.69
B_alg(50)	-10.88	-27.09	-24.03	-17.99	15.7	-19.59

Figure 119. Table showing % improvement of the Average Worst Ten Net Delays of a design using different levels of interconnect prediction during synthesis within MOODS

Name of Bench mark (CP target in MOODS)	A	B	C	D	E	F
bench_1(20)	17.44	17.44	31.85	13.37	13.37	13.37
bench_1(50)	17.96	17.96	22.81	19.67	12.15	19.67
bench_2(20)	5.34	5.34	5.34	5.34	5.34	5.34
bench_2(50)	0	32.71	32.71	0	37.06	37.06
bench_3(20)	-0.03	5.34	11.04	2.48	5.3	5.3
bench_3(50)	0.16	0.49	2.54	0.2	7.7	7.7
bench_4(20)	42.2	42.2	37.93	44.48	35.79	35.79
bench_4(50)	32.3	32.3	32.55	32.3	32.3	32.3
bench_5(20)	76.42	74.3	76.53	77.57	77.55	77.15
bench_5(50)	76.17	76.14	77.92	77.9	77.39	76.97
bench_6(20)	51.32	51.32	51.93	69.44	67.55	69.14
bench_6(50)	44.48	44.48	44.1	46.79	45.34	45.34
bench_7(20)	7.97	12.42	45.12	49.78	32.71	32.71
bench_7(50)	16.03	34.86	33.99	42.41	19.49	19.49
bench_8(20)	17.44	0.05	33.96	39.91	6.8	44.18
bench_8(50)	21.14	33.52	39.36	35.75	31.05	33.35
bench_9(20)	50.2	50.2	50.2	32.67	32.32	32.32
bench_9(50)	44.59	44.59	44.59	30.65	34.71	34.71
bench_10(20)	28.23	46.24	21.5	30.24	36.07	36.07
bench_10(50)	9.37	41.75	12.52	37.17	28.71	28.71
GCD(20)	19.87	19.87	20.44	19.87	19.87	33.77
GCD(50)	19.87	19.87	20.44	19.87	19.87	33.77
matrix_calc_1(20)	2.98	3.17	3.17	-0.2	3.17	-0.2
matrix_calc_1(50)	2.98	3.17	3.17	-0.2	3.17	-0.2
matrix_calc_2(20)	4.95	7.42	7.48	5.41	5.24	5.08
matrix_calc_2(50)	-1.12	-1.35	-4.59	0.16	-1.52	-2.82
matrix_calc_3(20)	-3.25	-3.16	-6.46	-0.04	-3.3	-0.04
matrix_calc_3(50)	-3.25	-3.16	-6.46	-0.04	-3.3	-3.31
matrix_calc_4(20)	0.3	0.3	0.3	1.18	0.3	0.36
matrix_calc_4(50)	29.59	29.59	29.59	30.22	29.59	29.64
matrix_calc(20)	2.45	-0.17	2.74	5.56	2.33	2.65
matrix_calc(50)	-1.76	0.01	-1.38	-3.37	-1.46	-2.81
insertionsort(20)	13.28	15.06	24.17	-54.56	18.02	18.02
insertionsort(50)	13.28	15.06	4.88	-54.56	15.22	15.22
quadratic(20)	11.22	11.22	11.22	8.57	14.05	13.81
quadratic(50)	1.42	2.07	2.07	2.86	-0.41	-0.41
mergesort(20)	8.16	11.71	14.29	5.55	14.23	13.26
mergesort(50)	9.96	7.91	9.9	3.16	14.66	12.8
heapsort(20)	16.3	22.15	15.84	13.45	16.47	16.47
heapsort(50)	6.48	20.66	23.69	1.94	17.73	17.73
quicksort(20)	17.38	17.38	8.42	15.17	17.13	14.21
quicksort(50)	21.39	21.39	16.94	24.11	24.11	21.64
B_alg(20)	54.44	53.16	49.53	50.61	49.92	54.68
B_alg(50)	22.27	19.3	22.27	27.78	36.81	33.4

Figure 120. Table showing % improvement of the Clock Period of a design using different levels of interconnect prediction during synthesis within MOODS.

Name of Bench mark (CP target in MOODS)	A	B	C	D	E	F
bench_1(20)	17.44	17.44	31.85	13.37	13.37	13.37
bench_1(50)	17.96	17.96	22.81	19.67	12.15	19.67
bench_2(20)	5.34	5.34	5.34	5.34	5.34	5.34
bench_2(50)	0	32.71	32.71	0	37.06	37.06
bench_3(20)	-0.03	5.34	11.04	2.48	5.3	5.3
bench_3(50)	0.16	0.49	2.54	0.2	7.7	7.7
bench_4(20)	42.2	42.2	37.93	44.48	35.79	35.79
bench_4(50)	54.87	54.87	55.03	54.87	54.87	54.87
bench_5(20)	76.42	74.3	76.53	77.57	77.55	77.15
bench_5(50)	84.12	84.09	81.6	81.59	81.16	76.97
bench_6(20)	51.32	51.32	51.93	69.44	67.55	69.14
bench_6(50)	44.48	44.48	44.1	46.79	45.34	45.34
bench_7(20)	34.27	37.45	60.8	64.13	51.93	42.32
bench_7(50)	40.02	53.47	52.85	58.86	42.49	30.99
bench_8(20)	44.96	33.37	55.97	59.94	37.86	62.79
bench_8(50)	21.14	33.52	39.36	35.75	31.05	33.35
bench_9(20)	60.16	60.16	60.16	46.13	45.85	45.85
bench_9(50)	44.59	44.59	44.59	30.65	34.71	34.71
bench_10(20)	42.59	56.99	37.2	44.2	48.86	48.86
bench_10(50)	9.37	41.75	12.52	37.17	28.71	28.71
GCD(20)	19.87	19.87	20.44	19.87	19.87	33.77
GCD(50)	19.87	19.87	20.44	19.87	19.87	33.77
matrix_calc_1(20)	2.98	3.17	3.17	-0.2	3.17	-0.2
matrix_calc_1(50)	2.98	3.17	3.17	-0.2	3.17	-0.2
matrix_calc_2(20)	4.95	7.42	7.48	5.41	5.24	5.08
matrix_calc_2(50)	-1.12	-1.35	-4.59	0.16	-1.52	-2.82
matrix_calc_3(20)	-3.25	-3.16	-6.46	-0.04	-3.3	-0.04
matrix_calc_3(50)	-3.25	-3.16	-6.46	-0.04	-3.3	-3.31
matrix_calc_4(20)	0.3	0.3	0.3	1.18	0.3	0.36
matrix_calc_4(50)	29.59	29.59	29.59	30.22	29.59	29.64
matrix_calc(20)	2.45	-0.17	2.74	5.56	2.33	2.65
matrix_calc(50)	-1.76	0.01	-1.38	-3.37	-1.46	-2.81
insertionsort(20)	13.28	15.06	24.17	-54.56	18.02	18.02
insertionsort(50)	13.28	15.06	4.88	-54.56	15.22	15.22
quadratic(20)	11.22	11.22	11.22	8.57	14.05	13.81
quadratic(50)	1.42	2.07	2.07	2.86	-0.41	-0.41
mergesort(20)	8.16	11.71	14.29	5.55	14.23	13.26
mergesort(50)	9.96	7.91	9.9	3.16	14.66	12.8
heapsort(20)	16.3	22.15	15.84	13.45	16.47	16.47
heapsort(50)	6.48	20.66	23.69	1.94	17.73	17.73
quicksort(20)	17.38	17.38	8.42	15.17	17.13	14.21
quicksort(50)	21.39	21.39	16.94	24.11	24.11	21.64
B_alg(20)	54.44	53.16	49.53	50.61	49.92	54.68
B_alg(50)	22.27	19.3	22.27	27.78	36.81	33.4

Figure 121. Table showing % improvement of the Total Delay (Number of CPs * CP Delay) of a design using different levels of interconnect prediction during synthesis within MOODS

Name of Bench mark (CP target in MOODS)	AREA G	AREA B	AI G	AI B	WN G	WN B	W10 G	W10 B	CP G	CP B
bench_1(20)	0.24	0.24	0.04	1.82	-18.57	6.27	-15.45	-0.36	17.44	13.21
bench_1(50)	0.47	0.47	-0.63	1.67	2.91	12.26	2.52	8.44	17.96	22.81
bench_2(20)	0.11	0.11	2.52	2.52	-2.33	-2.33	-8.25	-8.25	5.34	5.34
bench_2(50)	-0.5	-0.42	-4.89	-8.78	-39.17	-17.12	-18.53	-16.17	32.71	39.04
bench_3(20)	0.67	0.48	0.74	2.17	-3.78	5.2	-22.05	-5.3	5.34	11
bench_3(50)	-2.51	-0.6	-5.9	-6.34	-15.38	-22.94	-2.21	-13.22	0.49	7.86
bench_4(20)	2.27	1.91	3.44	5.73	11.93	5.71	-1.46	2	42.2	31.54
bench_4(50)	-0.19	1.14	2.79	2.72	-8.97	-1.96	-1.57	0.71	32.3	23.13
bench_5(20)	0.6	-23.02	7.73	2.2	-18.03	-24.82	1.45	-14.46	74.3	75.89
bench_5(50)	1.49	-40.94	-0.74	-8.59	-22.39	-3.62	-15.74	-19.18	76.14	70.99
bench_6(20)	1.77	-2.21	5.78	8.36	-6.79	2.97	6.9	12.3	51.32	55.4
bench_6(50)	-2.06	-5.55	-1.62	-4.99	5.21	4.03	-0.4	-2.12	44.48	33.53
bench_7(20)	-0.19	-1.76	-0.97	-4.34	-10.33	-13.11	-5.59	-8.26	12.42	0.79
bench_7(50)	1.82	-0.18	4.14	6.18	10.22	5.61	14.11	10.52	34.86	14.79
bench_8(20)	4.96	3.56	-8.38	-7.28	-19.77	-21.03	-15.22	-17.69	0.05	41.85
bench_8(50)	2.69	2.11	-6.46	3.07	-8.35	-0.17	-6.73	6.57	33.52	21.14
bench_9(20)	3.1	3.1	-1.31	-1.31	7.3	7.3	10.9	10.9	50.2	50.2
bench_9(50)	1.94	1.6	-3.6	-6.32	-36.54	-25.38	-11.46	-13.79	44.59	30.65
bench_10(20)	1.58	1.53	2.78	11.52	7.43	27.93	7.62	22.67	46.24	40.79
bench_10(50)	1.23	1.02	11.51	10.46	-2.01	-4.44	5.43	7.5	41.75	38.77
GCD(20)	-1.65	9.92	-1.64	-2.38	11.28	7.99	-4.98	-4.08	19.87	6.14
GCD(50)	-1.65	9.92	-1.64	-2.38	11.28	7.99	-4.98	-4.08	19.87	6.14
matrix_calc_1(20)	0.13	0.44	3.69	-1.04	20.43	-0.11	10.88	3.93	3.17	0.67
matrix_calc_1(50)	0.13	0.44	3.69	-1.04	20.43	-0.11	10.88	3.93	3.17	0.67
matrix_calc_2(20)	-5.51	-9.59	2.05	-1.53	13.8	-8.54	8.1	-3.8	7.42	7.5
matrix_calc_2(50)	0.04	0.04	-2.57	-1.75	-6.17	-1.8	-1.49	-1.84	-1.35	-1.39
matrix_calc_3(20)	0.53	0.76	3.88	4.05	1.39	-0.66	-0.93	-1.27	-3.16	0.09
matrix_calc_3(50)	0.53	0.76	3.88	4.05	1.39	-0.66	-0.93	-1.27	-3.16	0.09
matrix_calc_4(20)	0.7	0.7	4.58	4.58	6.79	6.79	-0.22	-0.22	0.3	0.3
matrix_calc_4(50)	0.7	0.7	2.1	2.1	12.14	12.14	13.42	13.42	29.59	29.59
matrix_calc(20)	-0.62	-8.68	-1.59	1.18	24.15	20.66	4.98	2.13	-0.17	7.64
matrix_calc(50)	-0.66	-1.14	-0.5	1.21	9.33	-6.51	6.72	3.38	0.01	-4.32
insertionsort(20)	4.77	3.84	2.27	6.49	8.89	18.85	7.24	14.72	15.06	18.12
insertionsort(50)	4.77	3.84	2.27	6.49	8.89	18.85	7.24	14.72	15.06	18.12
quadratic(20)	0.57	0.1	-1.91	-0.74	44.75	43.04	9.44	11.21	11.22	13.88
quadratic(50)	0.31	0.31	4.18	5.69	1.29	-9.08	0.59	4.1	2.07	-0.23
mergesort(20)	-3.81	-1.63	3.53	-1.72	-3.02	0.31	-12.79	-4.83	11.71	-0.7
mergesort(50)	-7.54	-1	8.26	6.31	5.6	22.21	6.62	17.59	7.91	-4.87
heapsort(20)	-13.81	-0.74	-10.93	-4.15	-18.72	-8.83	-37.36	-27.29	22.15	24.66
heapsort(50)	-14.62	0.75	4.68	5.91	-2.24	16.08	-0.71	19.73	20.66	4.39
quicksort(20)	-5.46	-6.87	2.04	3.32	2.99	-6.3	-5.55	-11.46	17.38	12.05
quicksort(50)	-8.69	-10.2	-2.15	5.1	3.79	16.07	-9.35	13.68	21.39	13.45
B_alg(20)	-8.2	-0.74	-5.42	-8.51	-34.35	-29.8	-27.13	-26.52	53.16	50.56
B_alg(50)	-12.08	-4.98	-8.5	-3.07	-35.17	-21	-27.09	-28.95	19.3	20.78

AI = Average Interconnect Delay, WN = Worst Net Delay, W10 = Average Worst Ten Net Delays, CP = Clock Period.

Figure 122. Table showing % improvement of the physical metrics of a design using interconnect prediction during synthesis within MOODS, derived from the Greedy Algorithm (G) and secondly the Greedy algorithm and the Modified Kernighan Lin Algorithm combined(B).

Name of Bench mark (CP target in MOODS)	AREA	AI	WN	W10	CP
bench_1(20)	0	1.63	11.67	3.27	22.69
bench_1 (50)	0	0	0	0	0
bench_2(20)	0	0	0	0	0
bench_2(50)	0	0	0	0	0
bench_3(20)	0	0	0	0	0
bench_3(50)	0	0	0	0	0
bench_4(20)	0.86	3.41	7.85	2.72	15.95
bench_4(50)	0.86	3.41	7.85	2.72	15.95
bench_5(20)	0	0	0	0	0
bench_5(50)	0	0	0	0	0
bench_6(20)	-3.44	0.78	-8.8	-4.41	25.9
bench_6(50)	0	0	0	0	0
bench_7(20)	3.49	3.38	3.5	4.15	90.97
bench_7(50)	-1.83	-1.04	2.06	9.25	14.84
bench_8(20)	0	0	0	0	0
bench_8(50)	-0.64	0.43	-18.57	-5.25	0.11
bench_9(20)	0	0	0	0	0
bench_9(50)	0	0	0	0	0
bench_10(20)	-0.35	-4.01	-26.27	-4.61	-0.11
bench_1 0(50)	0	0	0	0	0
GCD(20)	0	3.51	0.79	-0.99	6.43
GCD(50)	0	3.51	0.79	-0.99	6.43
matrix_calc_1(20)	-0.27	0.72	-3.26	-9.22	0.33
matrix_calc_1 (50)	0	0	0	0	0
matrix_calc_2(20)	0	0	0	0	0
matrix_calc_2(50)	0	0	0	0	0
matrix_calc_3(20)	-0.17	4.14	11	7.79	-6.27
matrix_calc_3(50)	0	0	0	0	0
matrix_calc_4(20)	0.07	0.7	-13.84	-3.64	-5.66
matrix_calc_4(50)	0.07	0.7	-13.84	-3.64	-5.66
matrix_calc(20)	0	0	0	0	0
insertionsort(20)	0	0	0	0	0
insertionsort(50)	0	0	0	0	0
quadratic(20)	0	0	0	0	0
quadratic(50)	0.64	-3.38	-1.15	-9.96	-0.06
mergesort(20)	3.73	-10.4	4.19	-8.07	34.47
mergesort(50)	0.14	-3.24	2.08	4.89	-18.79
heapsort(20)	0.07	8.52	14.19	17.79	11.36
heapsort(50)	12.53	-3.95	10.14	4.87	-4.33
quicksort(20)	-1.69	4	4.95	9.88	-4.71
quicksort(50)	1.8	-2.29	7.16	7.92	-6.04
B_alg(20)	1.17	-2.81	4.85	2.89	4.62
B_alg(50)	4.58	18.52	8.57	24.83	2.94

Matrix_calc has been left out as the design did not fit on a Virtex chip. AI = Average Interconnect Delay, WN = Worst Net Delay, W10 = Average Worst Ten Net Delays, CP = Clock Period

Figure 123. Table showing % improvement of the physical metrics of a design using interconnect prediction during synthesis within MOODS, but this time using the Simulated Annealing Heuristic.