PSS Placer Tool for PiCoGA approach

BY:

Chaitanya B. Modi

08MCE024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2010

PSS Placer Tool for PiCoGA approach

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

BY:

Chaitanya B. Modi

08MCE024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2010

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Computer Science & Engineering at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Chaitanya Modi

Certificate

This is to certify that the Major Project entitled "PSS Placer Tool for PiCoGA approach" submitted by Mr. Chaitanya B. Modi (08MCE024), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. S.N. Pradhan
P.G. Coordinator,
Department Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Prof. D. J. Patel
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. K Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

To Whom It May Concern

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

Mrs. Jyoti Malhotra
Project Specialist
STMicroelectronics

Mr. Himanshu Srivastava
Senior Software Engineer
STMicroelectronics

Approved as to style and content by:

I certify that I have read this dissertation and that in my opinion it is fully adequate, in cope and quality, as a dissertation for the degree of Master Of Technology in Computer Science and Engineering.

Mrs. Namerita khanna
Section Manager
ECL(TR&D),STMicroelectronics

Abstract

With recent advances in silicon device technology, Field-Programmable Gate Arrays (FPGAs) have become very important implementation media for digital circuits. Their programmability has been the key to the success of FPGAs.

Embedded Field-Programmable Gate Arrays (eFPGA) are flexible circuits that can be reconfigured by the designer and embedded with devices. The efficient use of these circuits requires complex CAD tools. Placement is One of the steps of the design process for eFPGA.

PiCoGA is STMicroelectronics Specific FPGA Chip which is an acronym for Pipelined Configurable Gate Array. The PiCoGA is designed to implement a peculiar pipeline where each stage corresponds to a piece of computation, so that high throughput circuits can be mapped. In this way a sequence of PiCoGA instructions can be processed filling the pipeline in order to exploit parallelism.

Along with this configurable unit also preserved its state across instruction executions. A new PiCoGA instruction may directly use the results of previous ones, thus reducing the pressure on the register level. Moreover a tight integration in the processor core gives the opportunity to use the PiCoGA in many different computational cores. With the arrival of PiCoGA, the problem of multi-computing was solved to achieve a much faster computation.

The main goal of this project is to Design a CAD (Computer Aided Design) tool and optimize it which efficiently perform the placement of Reconfigurable Logic Blocks on eFPGA(Embedded FPGA), address the challenges occurring because of the different constraints due to the architecture of chip.

Acknowledgements

It is my pleasure to take this opportunity to thank all those who helped me directly or indirectly in researching indexed topics. Not everything that I have received can be acknowledged with a few words, not everything that I acknowledge will relieve me from my indebt ness.

I am grateful to **Dr. S. N. Pradhan** and **Mrs. Namerita Khanna** and **Mrs. Jyoti Malhotra** for giving me the opportunity of working and taking research in mentioned research area of FPGA placement and CAD tool development.

I am heartily thankful to my research guide **Dr. S. N. Pradhan** Coordinator(M.tech), **Prof. D. J. Patel** Head of Computer Science and Engineering, **Dr. K Kotecha**, Director, Institute of Technology, Nirma University, Ahmedabad. and Mrs. Jyoti malhotra, Project guide, Stmicroelectronics, Noida. for showing me way to my research area. I am indebted to them for the pristine and enlightening guidance given to me throughout my learning time.

I would like to give a special acknowledgment to **Mr. Himanshu Srivastava** and **Mrs. Jyoti Malhotra** at STMicroelectronics, TR&D Department for their help in formulating the technical aspects of this work. Their contribution of ideas and software greatly aided in the development of my research.

I express my immense gratitude to my colleague, with same research area for their encouragement, guidance, understanding and lots of support and trust without which this seminar would produce something or not.

Chaitanya B. Modi

08MCE024

Contents

Declaration ii

Certificate iv

Abstract vi

Acknowledgements vii

List of Figures xi

List of Tables xii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Research Approach and Work . . . 3
- 1.3 Scope Of Work . . . 4
- 1.4 STMicroelectronics . . . 5
- 1.5 Thesis organization . . . 6

2 BackGround and Literature Survey 8
- 2.1 FPGA Architecture . . . 9
  - 2.1.1 Island Styled FPGA . . . 11
  - 2.1.2 Architectural Issues of FPGA . . . 11
- 2.2 PiCoGA Architecture . . . 12
- 2.3 Reconfigurable Logic Block . . . 14
- 2.4 Versatile Placement & Routing By Vughan Betz . . . 15
- 2.5 Fast Place and Route Approaches for FPGAs By Russel Tessier . . . 15

3 Challenges and objectives 16
- 3.1 Challenges . . . 16
  - 3.1.1 Routing . . . 16
  - 3.1.2 Placement . . . 17
  - 3.1.3 Clustering . . . 17
- 3.2 Objectives . . . 18

4 Algorithms and Technology **20**
4.1 Proposed Algorithms . . . 20
4.1.1 FPGA Routing . . . 20
4.1.2 FPGA Placement-Simulated Annealing . . . 22
4.1.3 Simulated Annealing: . . . 26
4.1.4 Clustering Based Placement . . . 32
4.1.5 Direct Inter-connectivity supported Placement . . . 37
4.2 Tools . . . 40
4.2.1 Visual studio 6.0 & Eclipse CDT(GCC): . . . 40
4.2.2 QT Framework: . . . 40
4.2.3 IBM Rational-Rose & Clear-Case: . . . 41
4.2.4 Lex & Yacc - Compilers . . . 41

5 System Design and implementation **43**
5.1 CAD flow for eFPGAs . . . 43
5.2 Chip Architecture . . . 44
5.3 PSS Placer Tool Flow . . . 45
5.3.1 Read Input and Generate Chip View . . . 46
5.3.2 Clustering & Placement . . . 47
5.3.3 Generating Output . . . 49
5.4 Test And Validation Codes . . . 51
5.4.1 Testing . . . 52
5.4.2 Validation . . . 53
5.4.3 Test Automation & Data Collection . . . 54
5.5 Code integration and code management . . . 56
5.5.1 Code Integration . . . 56
5.5.2 Code Management . . . 57
5.6 Debugging . . . 57
5.6.1 General Bugs . . . 58
5.6.2 Linux Bugs . . . 58

6 Results and Analysis **60**
6.1 Test & Results . . . 60
6.2 PSS:Placer Tool Output . . . 62
6.3 Analysis . . . 62
6.3.1 Observation of results . . . 64
6.4 PSS placer Tool Information . . . 67

7 Conclusion and Future Work **69**
7.1 Summery . . . 69
7.2 Future Work . . . 70

A Acronyms **72**

Website References **74**

References **75**

List of Figures

2.1 A Generic eFPGA 10
2.2 An Island Style eFPGA 11
2.3 PiCoGA Architecture 13

4.1 Routing flow of P&R 21
4.2 Two block scenario of Direct Inter-connectivity 22
4.3 Block diagram of Placement problem 23
4.4 SA algorithm Comparison 25
4.5 Solution Finding approach of SA 27
4.6 Sample Cluster Model 32
4.7 Adding a BLE to a cluster. 33
4.8 It decrease No. of inputs. 33
4.9 Clustering Interpretation 36
4.10 Two block scenario of Direct Inter-connectivity 37
4.11 Direct Interconnects Utilization for Multiple Blocks scenario 39

5.1 CAD flow of eFPGA 44
5.2 PSS Tool Flow. 45
5.3 Read Input and Generate Chip View. 47
5.4 Clustering generation Flow. 48
5.5 Coarse/Detailed Placement Flow. 49
5.6 Generating Output. 50
5.7 Testing Automation Flow 55
5.8 GUI and Placer Integration (Source and Header) 56

6.1 Placement By Cluster Based SA 63
6.2 Placement By Cluster Based SA and Direct Interconnect Support . . 64
6.3 Connected wired view for complex_calc.rlc 65
6.4 50 vs. 100 Iteration- result comparison for WL Only 66
6.5 50 vs. 100 Iteration- result comparison for DI Only 66
6.6 50 vs. 100 Iteration-result comparison for Both 67
6.7 comparison of methodologies applied for placer tool 67

List of Tables

I Temperature Updation Conditions 30

I CAD tool Generated Placement based on Algorithm 61
II CAD tool Generated Placement using SA with cluster and DI support 62
III Manual Placement using GUI placement editor 62
IV Result comparison between 32 and 64 Linux platform 63
V Code size of PSS_placer tool 68

Chapter 1

Introduction

1.1 Motivation

With recent advances in silicon device technology, a new branch of computer architecture, reconfigurable computing, has emerged. While this computing domain holds the promise of exceptional fine-grained parallel performance, the amount of time required to compile a program to a reconfigurable computing platform can be prohibitive for many applications.

A large portion of this compile time is typically spent performing device layout for field-programmable gate arrays (FPGAs), the core hardware components of most reconfigurable computing systems. In this report, an new integrated routing and placement system for FPGAs, called PnR, is detailed. This system has been designed to optimize FPGA layout time at the cost of modest increases in device logic and routing resources. Experimental results are presented which demonstrate an order of magnitude speedup over traditional layout approaches for island-style FPGA architecture.

This report documents contains the development of a tightly integrated set of layout tools for island-style FPGAs that have been optimized to reduce place and route time. Much of this compile time reduction comes from the assumption that the

user design is structured as a set of register transfer level (RTL) macro blocks that have predefined placement and internal routing structure. To support this design style, a macro based FPGA placer and fast router have been developed that can be tuned to tradeoff implementation quality for layout optimization time. In the course of this dissertation, this system is used to explore a range of tradeoffs between these two competing layout goals. In practice, the interaction between these tools can be varied depending upon tolerable tradeoffs specified by a system user.

An FPGA can implement any circuit simply by being appropriately programmed. Other circuit implementation options, such as Standard Cells or Mask Programmed Gate Arrays (MPGAs), require that a different VLSI chip be fabricated for each design. Use of a standard FPGA, rather than a custom FPGA, has two key benefits: lower non-recurring engineering (NRE) costs, and faster time-to-market.

Time-to-market is the other key advantage of eFPGA, because it can be programmed in minutes, and any bugs found once the chip is tested in system can be corrected in minutes by reprogramming the FPGA. With today's short product cycles, the time-to-market advantage this provides is often compelling.

FPGA programmability carries a price, however. In FPGAs and Standard Cells circuitry is interconnected with metal wires. FPGAs, in contrast, must connect circuitry via programmable switches. These switches have higher resistance than metal wires and add significant capacitance to connections, reducing circuit speed. As well, the switches take up more area than metal wires would, so an FPGA must be considerably larger than an FPGA to implement the same circuit. Typically a circuit implemented in an FPGA [1] with support of PiCoGA is about 10 times larger and 3 times slower than the same circuit implemented via an FPGA in an equivalent process.

Industrial FPGA research tends to focus on point solutions companies make reasonable design choices and guesses to get a product to market in a timely fashion. Industrial FPGA researchers generally do not explain which design choices were carefully researched and which were educated guesses, and they usually do not investigate

(or at least do not publish) the effect radically different design choices would have had. One way academic FPGA research, such as this report , therefore adds value is by exploring an architectures. We determine the best choice for various architecture parameters, the sensitivity of FPGA performance to each parameter, and useful "rules of thumb" for designing good eFPGA.

1.2 Research Approach and Work

The project has been implemented into different major phases. The initial phases was to obtain routing for FPGA chip, that consists of Preprocessing architecture and design data. This routing part was already achieved when I joined Mrs. Jyoti malhotra in this research based project. Other big phase was to achieve GUI framework to display solution and make whole FPGA placement process User interactive as well as to give user access of modifying automated placement.

Since my joining in research and development team with Mr. Himanshu and Mrs. Jyoti, we were started working on FPGA placement's main implementation called simulated Annealing based placement algorithm with several other small supportive algorithm like clustering and Direction based support to placement.

With this CAD tool development we were also working on identifying issues of making tool platform independent in sense of window to Linux and from 32bit to 64bit Operating system. Some automated Testing environment creation was also essential part of development during project development and research.

So we adopted experimental research approach, Our work is limited towards STM eFPGA PiCoGA architecture, so we started applying routing and placement methodologies that meets constraints of PiCoGA chip.We had basic different circuits designed for PiCoGA chip and architecture file that we read and apply algorithm on that, which lead us to fast and efficient development of PSS placer tool.Ideally, one would lay out FPGA architecture based designs of interest to obtain exact area measurements and highly accurate delay values. In this thesis we will investigate some different designs,

We are developing more abstract area models and delay estimates that do not require layout of the FPGA, but which are still sufficiently accurate to allow fair comparison of different design circuits in easy to use GUI framework.

1.3 Scope Of Work

The primary goal in placement process is to place the heavily connected blocks together for,

- Decreased wire length.
- Increased Circuit speed.
- Optimize area usage.

The main objective of this thesis is to modify the existing annealing algorithms to utilize the PiCoGA architecture and to claim the performance improvement. But placement of logic blocks on eFPGA has many constraints due to the architecture of chip. We need to implement an algorithm in such a way that we can have a minimum wires and minimum size of a chip. There should be no long wires in a chip. Next challenging thing is to design an algorithm in such a way that it is independent on architecture on a chip. So, algorithm should not depend on the number of rows and number of columns of a chip.

The scope of this thesis work encompasses the architectural study and programming methodology of the PiCoGA architecture. Based on this knowledge placement algorithms selected is Simulated Annealing.Adopted VPR Tool's algorithm is an extended version of Simulated Annealing. The basic pseudo code for both these algorithm is same but there is some changes in VPR line Cost computation, Updated Temperature, Computation of a swapping region, etc. After completion of a code of placer tool we need to do manual testing. In the whole process of designing a placer tool, very important thing is to understand the SDK used to design the placer tool.

How best we can apply this SDK features in designing the placer tool. In addition we have provided the user the facility to enter into the algorithm from any point.

1.4 STMicroelectronics

ST Microelectronics is a global independent semiconductor company and is a leader in developing and delivering semiconductor solutions across the spectrum of microelectronics applications. An unrivalled combination of silicon and system expertise, manufacturing strength, Intellectual Property (IP) portfolio and strategic partners positions the Company at the forefront of System-on-Chip (SoC) technology and its products play a key role in enabling today's convergence trends.

The Company's products are manufactured and designed using a broad range of fabrication processes and proprietary design methods. To complement this depth and diversity of process and design technology, the Company also possesses a broad intellectual property portfolio that it has used to enter into cross-licensing agreements with many other leading semiconductor manufacturers.

I was assigned in ECL(Embedded Configuration Logic) team, which is handled by Mrs. Namerita Khanna. This team is part of Technology Research and Design department(TR&D). ECL works on two different aspects of VLSI, mainly Placement and routing of Chips. One part of team works on Application specific IC placement and routing, and my team, leaded by Mrs. Jyoti malhotra works on FPGA placement and routing CAD tool development for PiCoGA architecture.

Corporate Headquarters, as well as the headquarters for Europe and for Emerging Markets, are in Geneva. The Company's U.S. Headquarters are in Carrollton (Dallas, Texas); those for Asia/Pacific are based in Singapore; and Japanese operations are headquartered in Tokyo.

1.5 Thesis organization

The rest of the thesis report is organized as follows and very next **Chapter 2**, *Background and Literature Survey*, explains background and related work committed in recent years and in area of CAD and FPGA architectures.It has basic definitions of the area and background work, like FPGA architecture and inland style FPGA architecture, PiCoGA chip architecture.

Chapter 3, *Problem statements and objectives*, In this section I have stated different challenges during development. After defining our problems and challenges I mentioned different objectives we want to achieve within this research project development.

Chapter 4, *Algorithms and Technology*, describes identified problem and defined objectives. Here I have explains various techniques used for placement and routing. It also gives an insight details that which techniques is useful and why.This part also contains discrete details of different tools used in development.

Chapter 5, *System Design and implementation*, has described the CAD Tool for Placement of eFPGA PiCoGA chip, How placer tool is designed and how it works that is explained. Implementation expect of platform independent PSS placer tool shows here how to deal with certain issues of implementation.

Chapter 6, *Results and Analysis*,shows experimental development we had some designs for PiCoGA architecture, Time to time on some phase completion we tested each design and collected results. Routing time wire length, Before placement Wire length, After Placement Wire length, With clustering supported placement, different utilization factors are also noted down to compare and get analysis of Placement and to measure efficiency.

Chapter 7, *Conclusion and Future Scope*,has Some conclusion we achieved during development and result gathering is provided here, future work that can be

made in this area and extension to PSS placer tool also. Suture scope of PSS placer tool shows us market value and demand in real life.

Appendix A, *Acronyms*, contains list of acronyms and abbreviations.

References, shows references referred in thesis to get information from previously published papers, thesis, articles and so on.

Chapter 2

BackGround and Literature Survey

The first half of this chapter provides background information about FPGA architectures,PiCoGA architecture and briefly describes the prior work relevant to this thesis. The second half of the chapter describes the related work has been done in FPGA placement and routing.

FPGAs were introduced in early 90's, since then many research has been attempted, Here we are more concerned abpout FPGA routing and placement.Carl sechen and Alberto sangivanni developed TimberWolf Placement and Routing Package that was the first noticeable attempts for FPGA placement and routing. In next years FPGA placement for digital circuits were researched and developed by Mr. Jonathan Rose, and later vaughan bets developed "VPR(Versatile Placement and Routing) tool"[2] in 1998 that can identify several fpga architecture and work on many designs. This tool was landmark for the research of FPGA placement and routing. Though VPR was developed on basic idea of simulated annealing[3] which was introduced by S. Kirkpatrick, C. D. Gelatt, Jr. M. P. Vecchi in 1983. After that in 1999 Russell G. Tessier also attempted research in "Fast Place and Route Approaches for FPGAs"[4] but this has many improved methods than VPR and provided support for different architectures.

Here we are dealing with special FPGA architecture called PiCoGA, that gives many advantages over the regular FPGA architecture, PiCoGA is island style FPGA

architecture that itself have many advantages in Digital world.We are developing Placer tool for PiCoGA architecture, and we were bounded by many architectural constraints [1], So specific placement and routing methodologies we also adopted as Vaughan bets and Russel Tessier adopted from other researchers.In addition to placement algorithm simulated annealing we have designed some supportive methods that extend our research and development work to next level and brought a new enhanced PSS tool.

2.1 FPGA Architecture

Commercial FPGAs can be classified into three groups, based on their routing architecture. The FPGAs of Xilinx, Lucent and Vantis are island-style FPGAs, while Actel's FPGAs are row-based, and Altera's FPGAs are hierarchical [5]. In this report, we will almost exclusively investigate the island-style routing architecture, so we describe this style of routing architecture below.

All FPGAs are composed of three fundamental components: logic blocks, I/O blocks and programmable routing, as shown in Figure 2.1. A circuit is implemented in an eFPGA by programming each of the logic blocks to implement a small portion of the logic required by the circuit, and each of the I/O blocks to act as either an input pad or an output pad, as required by the circuit.

Commercial FPGAs can be classified into three groups, based on their routing architecture. The FPGAs of Xilinx, Lucent and Vantis are island-style FPGAs, while Actel's FPGAs are rowbased, and Altera's FPGAs are hierarchical.

Here we are developing a complete CAD tool for placement of logic blocks and the routing of the channel on a chip. This chip is named PiCoGA. It contains 24 rows and 16 columns. So, there are 16*24 tiles available on this chip. Each contains some number of tracks and an RLC.Total number of tracks available in single tile are.

- 3 Length Horizontal Lines.

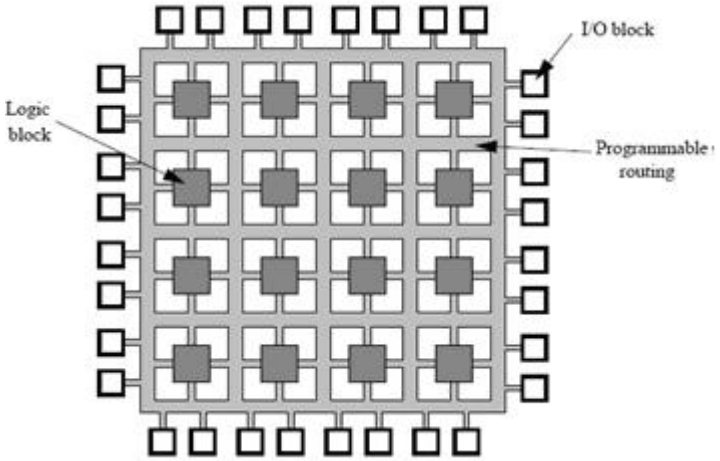

Figure 2.1: A Generic eFPGA

- 3 Length Vertical Lines.
- Direct lines.
- Long Lines
  - Global Input Lines.
  - Global Output Lines.
  - Global horizontal lines.

Also to connect all these tracks to each other, to other RLC and to input and output registers we need switching blocks to complete the flow of circuit.Total number of Switch Boxes in a single tile to connect Tracks are:

- O/P Connection Box Vertical
- O/P Connection Box Horizontal
- Switch Box connecting 3 length lines
- I/P Connection Box Vertical
- I/P Connection Box Horizontal

2.1.1 Island Styled FPGA

Island-style routing architectures figure 2.2 are generally characterized by their two dimensional symmetry and their inclusion of wire segments that span one or more logic blocks. The percentage of segments of each length (or segmentation) in each routing channel along with the grain size of the logic block in terms of look-up tables and flip-flops defines a specific island-style family.

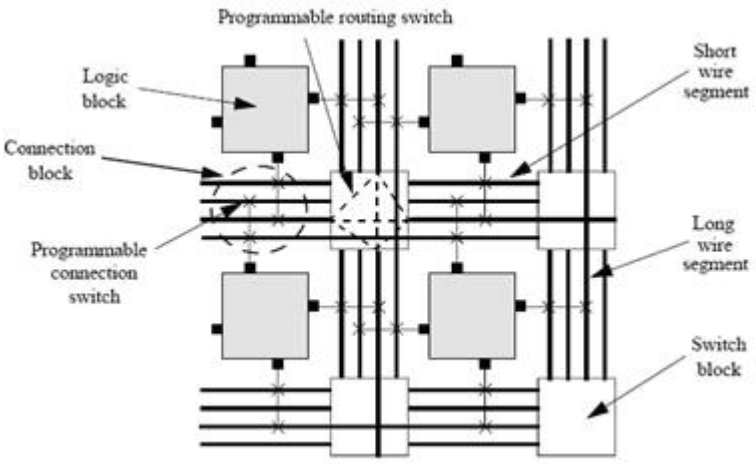

Figure 2.2: An Island Style eFPGA

The segmentation of wires allows for high-speed connectivity of signals, removing the need for signals to pass through an excessive number of routing switches.Island-style routing architectures [5] can be generalized based on connectivity between adjacent wire segments and between segments and logic blocks.

2.1.2 Architectural Issues of FPGA

The first issue we investigate is global routing architecture [5]. The global routing architecture of an eFPGA specifies the relative width of the various channels within the chip. In eFPGA and standard cell implementations, a custom chip is created for each design, so routing channels can easily be made wider in areas of a chip where the demand for routing is greater.

In FPGAs, however, all placement resources are prefabricated, so the width of all the routing channels is set by the eFPGA manufacturer. Our goal, then, is to find the distribution of routing or placement resources, or tracks, to the various channels that permits their efficient utilization by the largest class of circuits. If there are too few tracks in some area of the chip then many circuits will be unroutable, while if there are too many tracks, they may be wasted. There is no agreement amongst commercial FPGAs on the best global routing architecture, so this question has clear commercial relevance.

The final eFPGA issue we examine is that of detailed routing architecture[5]. The detailed routing architecture of an eFPGA defines how logic block inputs and outputs can be interconnected. Detailed routing architecture is the key element of an eFPGA because most of an eFPGA's area is devoted to routing, and most of a circuit's delay is due to routing delays rather than logic block delays.

2.2 PiCoGA Architecture

PiCoGA: A Pipelined Configurable Gate Array architecture is Two-dimensional array of LUT-based Reconfigurable Logic Cells and each row implements a possible stage of a customized pipeline independent and concurrent with the processor which is up to 4x32-bit input data and up to 2x32-bit output data from/to register File. PiCoGA [6] is embedded function unit for dynamic extension of the available Instruction Set which was introduced by Andrea Lodi, Andrea Cappelli, in partnership with STMicroelectronics.

The PiCoGA is an array of rows, each representing a possible stage of a customized pipeline. The width of the datapath obtained should fit the processor one, so each row is able to process 32-bit operands. As shown in Figure 2.3, each row is connected to the other ones with configurable interconnect channels and to the processor register file with six 32-bit busses

In a single cycle, four words can be received from the register file and up to two

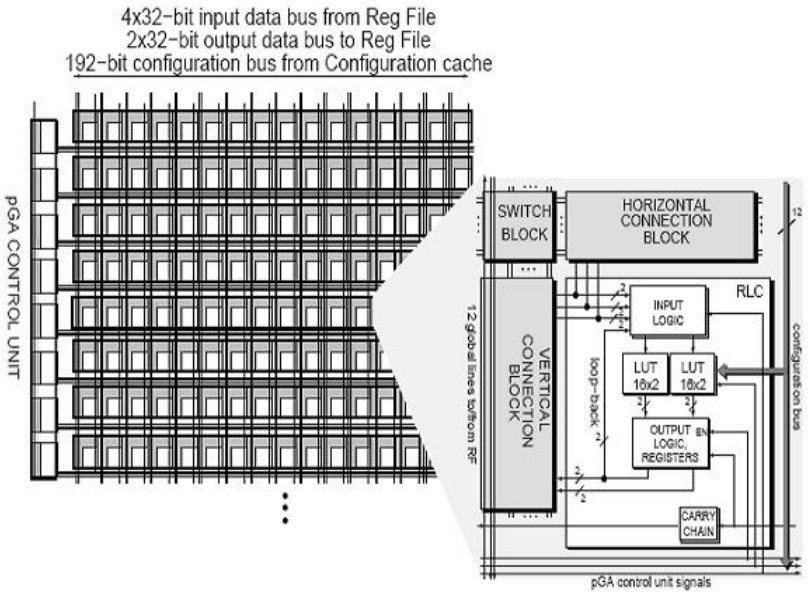

Figure 2.3: PiCoGA Architecture

words can be produced for write-back operations. The busses span the whole array, so that any row can access them, improving routability. Pipeline activity is controlled by a dedicated configurable control unit, which generates three signals for each row of the array. The first one enables the execution of the pipeline stage, allowing the registers in the row to sample new data. In every cycle, only rows having input data ready are activated. In this way, a state stored in flip-flops inside the array can be correctly held and at the same time unnecessary power dissipation is avoided. The second signal controls initialization steps of a state held inside the array, while the third enables a burst write of LUTs with data available in the processor Register File. Each row is composed of 16 Reconfigurable Logic Cells (RLC) and a configurable horizontal interconnect channel. Vertical channels have 12 pairs of wires while horizontal ones have only 8 pairs of wires.

Switch blocks adjacent to each RLC connect vertical and horizontal wires. Since most of the remaining portion of control logic not mapped in the processor standard dataflow is implemented in the configurable control unit, the array core can be data

path oriented. Therefore, the PiCoGA has a 2-bit granularity for both interconnections and LUTs, except for input connection blocks which have 1-bit granularity. This is a good compromise if we consider that bit-level operators such as bit permutation, which are frequent in cryptography algorithms, are not well supported by other functional units.

The PiCoGA reconfigurable device is integrated as a Functional Unit (FU) of the processor core, thus reducing communication overhead to and from other FUs. The PiCoGA can load up to 4 pga operations for each of its 4 configuration contexts. PiCoGA operations loaded in the same context can be executed concurrently. An embedded hardwired control logic handles conflicts on write back channels when more pga operations need to write on the processor register file. Furthermore, the PiCoGA can operate concurrently to the processor functional units and consistency is ensured by a register locking mechanism.

2.3 Reconfigurable Logic Block

A reconfigurable logic block [7], an input circuit configured to acquire reconfiguration data of logic gates for building a plurality of logic circuits; a memory array having a plurality of memory units corresponding to a number of the plurality of logic circuits, the memory units including paired first and second variable resistive memories, configured to store the reconfiguration data based on a magnitude relation of resistance between the first and second variable resistive memories in accordance wit h a write signal corresponding to the reconfiguration data.

From the architecture composition, we see that both the fixed part and the reconfigurable part are programmable, thus the programmability of the processor based architecture is retained. The reconfigurable unit can even extend the instruction set of its fixed counterpart, therefore makes the programmability of the architecture even stronger. The reconfigurable unit usually is a scalable gate array with high amount of parallel computation resource. This gives the reconfigurable unit a potential perfor-

mance advantage over the processors. Due to the nature of the field-programmable logic, the reconfigurable unit is usually not as efficient as the **ASIC** implementation in terms of power and speed, but the flexibility compensates for it.

2.4 Versatile Placement & Routing By Vughan Betz

VPR [2] is capable of targeting a broad range of FPGA architectures, It route and the associated net list translation / clustering tool VPACK [2] have already been used in a number of research projects worldwide, and should be useful in many areas of FPGA architecture research. VPR can then perform either a global route or a combined global/detailed route of the placement. VPR's output consists of the placement and routing, as well as statistics useful in assessing the utility of FPGA architecture, such as routed wire length, track count, and maximum net length. VPR is to kept the tool flexible enough to allow its use in many FPGA architectural studies.

2.5 Fast Place and Route Approaches for FPGAs By Russel Tessier

More restrictive placement and routing constraints in Xilinx FPGA designs was designed by Russel tessier in his Phd. thesis "Fast place and Route Approaches for FPGA",[4] conventional physical design tools for general placement and routing architectures usually do not work well for all FPGA designs. Moreover, to generate high quality circuits which are easy to place and route specially for xilinx products, it is important to consider the specific physical design constraints during the technology mapping process. In this thesis, he has presented a performance driven placement algorithm specifically developed for the Xilinx FPGAs [4], that is also developed based on simulated annealing algorithm and it also use clustering and two phase placement method for cell based xilinx designs.

Chapter 3

Challenges and objectives

In following chapter I am going to define challenges and objectives. By doing such we can identify problems and distill objectives that are also mentioned below. As stated before we are working on FPGA placement and routing, from which routing part is already developed, So I will concentrate on placement part.

3.1 Challenges

Following section shows challenge statements we are dealing with to build CAD tool for FPGA placement.

3.1.1 Routing

Challenge statement for routing of wire in FPGA architecture is "Routing is the process of identifying exactly which routing segments and switches should be used to create connected paths from net sources to net destinations for all nets in a circuit." For routing of FPGA devices we also need to define routing domain which is "A set of discrete wire segments in an FPGA device that can be connected together to form a routing path. If a path cannot be formed between two wire segments through switches or other segments, the segments are said to be in different domains."

3.1.2 Placement

First I would like to define placement problem. "Given an electronic circuit (realized to its synthesized form), map place-able elements (PE) onto corresponding physical locations [1], as present in the given PiCoGA architecture with the view of optimizing an objective within certain constraints. The problem statement of placer tool along with its inputs, outputs and different type of constraints are dependent on PiCoGA architecture. The format for the various inputs and outputs along with the constraints are left for succeeding phases of this project".

Placement objectives are often difficult to formulate and computationally difficult to satisfy.As a result, placement tools use approximations to objectives like (Reducing congestion, Increasing routability of design, Decrease power dissipation).The measurement most commonly used in modern tools is the half perimeter wire length (or HPWL) of the minimum rectangle enclosing all cells on a net.

3.1.3 Clustering

The motivation for using clustering with simulated annealing is to reduce the number of entities that need to be placed. This reduces the search space by reducing the degrees of freedom for simulated annealing, because we do not examine cell configurations where the cells within a cluster are apart. The resulting computational gains need not be at the expense of the quality of the solution, because the search efficiency is also improved. This follows by observing that without clustering, for two highly connected cells to move together, from one would have to be selected to move. Then, the second would have to somehow find its way close to the location of the first one. This would take much longer because several moves would have to be tried before they can come together.

3.2 Objectives

In CAD tool development of placement algorithms and techniques we face both logical and physical objectives to achieve high performable placement tool for PiCoGA as shown in figure 1.

Physical Objectives:

a. Developments of the placement problem for reconfigurable eFPGA in window 32 bit OS environment [8].

b. Porting of the same framework and project on the Linux 32 bit OS environment to get better optimized performance and minimizing time complexity of project.

c. Design of the UML diagrams and documents for the desired CAD tools development for the PiCoGA placer [9].

d. Testing and analysis of the developed placer tool in window as well as Linux environment.

Logical Objectives:

Placement problem is about to achieve following Implicit and explicit objectives and solving the eFPGA placement issues during the placement project development phase while considering architecture constrains.

- **Explicit objectives**
  a. Increase Rout-ability of design.
     - Congestion aware placer.
  b. Minimize wire-length. (Semi perimeter of placed Bounded-Box)
  c. Maximize circuit Performance. (Timing aware/ driven placer)
- **Implicit objectives**

a. In achieving rout-ability and providing ease in placement.

b. Obtain entire Placement & Router results within respectable time frame.

c. Optimizing the complexities of placer tool.

Chapter 4

Algorithms and Technology

In current chapter describes the Proposed Algorithms used to develop PSS placer Tool. Algorithm for FPGA placement we used are Simulated Annealing and Clustering. For FPGA routing we have already developed modules available of path routing using advanced A* Algorithm and some graph routing methods. Here in next part i will show tools we are using for CAD tool development of PSS.

4.1 Proposed Algorithms

In PnR tool we have routing part is already fully implemented and tested in routing we are using pathfinder algorithm that performs two step routing, and as placement is not yet fully implemented, we are studying simulated annealing algorithm and searching where we could optimization, and developing according to constraints provided for PiCoGA [1] chip.

4.1.1 FPGA Routing

Once locations for all the logic blocks in a circuit have been chosen, a router determines which programmable switches should be turned on to connect all the logic block input and output pins required by the circuit. In FPGA routing, one usually

represents the routing architecture of the FPGA as a directed graph. Each wire and each logic block pin becomes a node in this routing-resource graph and potential connections become edges. Some prior research has represented FPGAs as undirected graphs, but a directed graph representation is needed if directional switches, like tri-state buffers and multiplexors, are to be modeled correctly.

Routing a connection corresponds to finding a path in this routing-resource graph between the nodes representing the logic block pins to be connected. To avoid using up too many of the limited number of wires in an eFPGA, one wants this path to be as short as possible. As well, it is important that the routing for one net not use up routing resources another net needs, so most eFPGA routers have some kind of congestion avoidance scheme to resolve contention for routing resources. An additional optimization goal is to make nets on or near the critical path fast by routing them using short paths and fast routing resources. Routers that attempt to optimize timing in this way are called timing-driven, whereas delay-oblivious routers are purely routability driven. Since most of the delay in FPGAs is due to routing, timing-driven routing is crucial to obtain good circuit speeds.

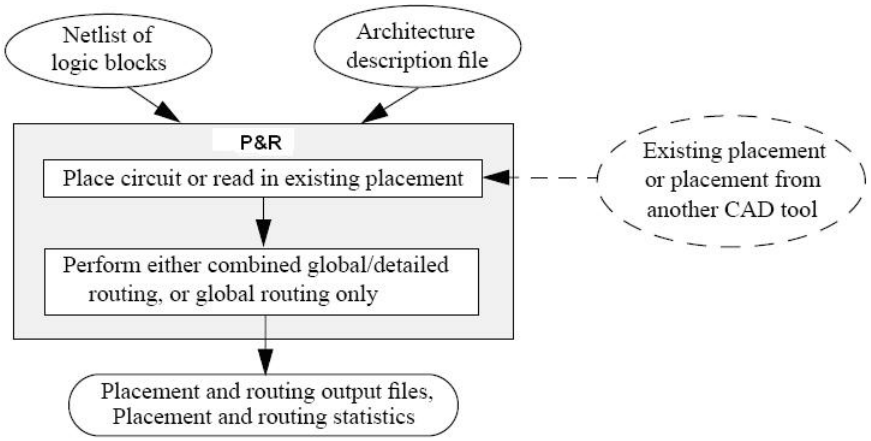

Figure 4.1: Routing flow of P&R

eFPGA routers can be divided into two groups. Combined global-detailed routers determine a complete routing path in one step, while two-step routing algorithms

as in figure 4.1 first perform global routing to determine which logic block pins and channel segments each net will use, and then perform detailed routing to determine the wire(s) each net will use within each of the specified channel segments [10]. A channel segment is the length of routing channel that spans one logic block a channel that spans M logic blocks contains M channel segments. The task of an eFPGA detailed router is often difficult or impossible because eFPGA routing has limited flexibility and the detailed router is highly constrained by the decisions the global router made about which channel segments each net must use. Combined global-detailed routers have the potential to more fully optimize the routing, since they are free of such constraints.

4.1.2 FPGA Placement-Simulated Annealing

Placement problem Statement

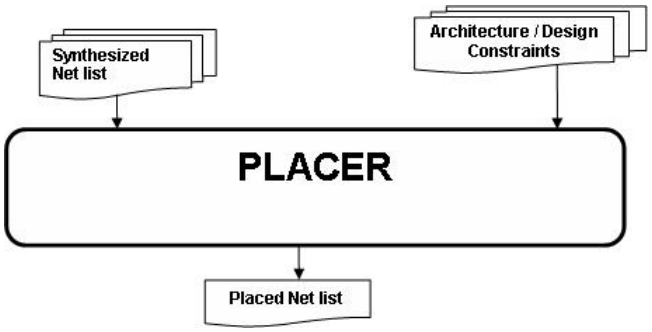

Figure 4.2: Two block scenario of Direct Inter-connectivity

Now formally we can define FPGA placement 4.2 in following words.

- Input: Hypergraph Gh(Vh:Eh)
  - Vertice set $Vh = [v1, v2, ..., vn]$ representing cells.
  - Hyperedge set $Eh = [e1, e2, ..., en]$ corresponding to signal nets.
- Output: Placement vectors X and Y

– $X = (x1, x2, ..., xn)$ and $Y = (y1, y2, ..., yn)$

a. Cells of the hypergraph are positioned in valid non overlapping locations.

b. Within placement region.

c. Maximizing the objective function (Wirelength or logical Dmax)

Simulated Annealing is a Iterative search based [3], Combinatorial Optimization problem which explores a discrete space of admissible configurations S in a deterministic fashion with the probability one for global minima convergence.

Placement objectives are often difficult to formulate and computationally difficult to satisfy.As a result, placement tools use approximations to objectives like (Reducing congestion, Increasing routability of design, Decrease power dissipation) [10].The measurement most commonly used in modern tools is the half perimeter wire length (or HPWL) of the minimum rectangle enclosing all cells on a net.

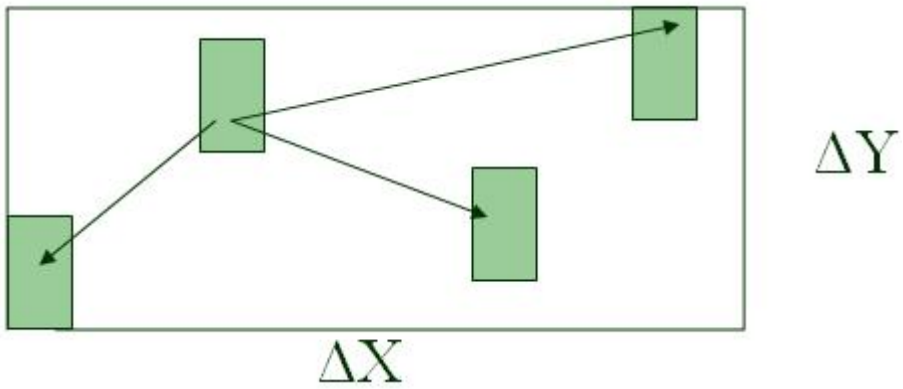

Figure 4.3: Block diagram of Placement problem

There are three main methods which perform efficient placement like,

a. **Min-cut algorithm:**This algorithm is partition based. There are predefined areas available in which it will fix the best suitable logic block.

b. **Analytic Algorithm:**This algorithm as recursive in nature. It will place the blocks in areas and the calculate the cost. If the new cost is lower than previous then this will adapt that solution. This will repeated until the best possible solution.

c. **Simulated Annealing:**This method based on some external parameters like temperature, distance between two blocks, some other constants etc.

Search-based placement methods are able to minimize HPWL directly. Partitioning- and analytic-based methods minimize approximations to HPWL, such as the minimum net cut or quadratic wire length.As a result of approximation we can process very large problem instances. and Hence used for very big designs at cost of quality where placing the design is the major challenge in the first instance with objective functions as secondary criteria.

Hence hybrid placement techniques first stage use an approximate technique to encourage quick cell placement, second stage calls direct HPWL-minimizing algorithm to improve the quality of results,i.e partition with SA.

Search Based Placers:

Search based placement method involve iterative improvement of an existing solution. Simulated annealing-based placers use stochastic search (Statistics involving randomness) like in VPR. Now a days Genetic algorithms are no longer used in modern CAD tool development. Search based method is a Most well developed, well studied method for module placement. At conclusion of its study, simulated annealing remains a widely used heuristic for placement in tightly constrained design styles

Partitioning Based Placers

Partition based placer [11] use Divide and Conquer based placement algorithms, that solves a given problem recursively.

General idea behind partition based placer is "To seeks to decompose (Divide) the given placement problem instance into smaller instances such that good solutions (Conquer) to smaller instances combine (Merge) into good solutions of the original problem" This is a Top-down recursive algorithm, that use following two steps.

- Divides the placement area and the circuit net-list into smaller pieces
- Using either bi-section or (less commonly) quadric section and a minimum-cut (or other) objective which approximates into wire-length estimate

After studying all three types of placement we concluded, and agreed to Quality Runtime tradeoff presented by Scott Hauck et.Following figure 4.4 shows Time VS. Quality tradoff. It shows that simulated annealing is best method among all shown for the FPGA placement, and in our case with PiCoGA architecture also it has given best results so far.

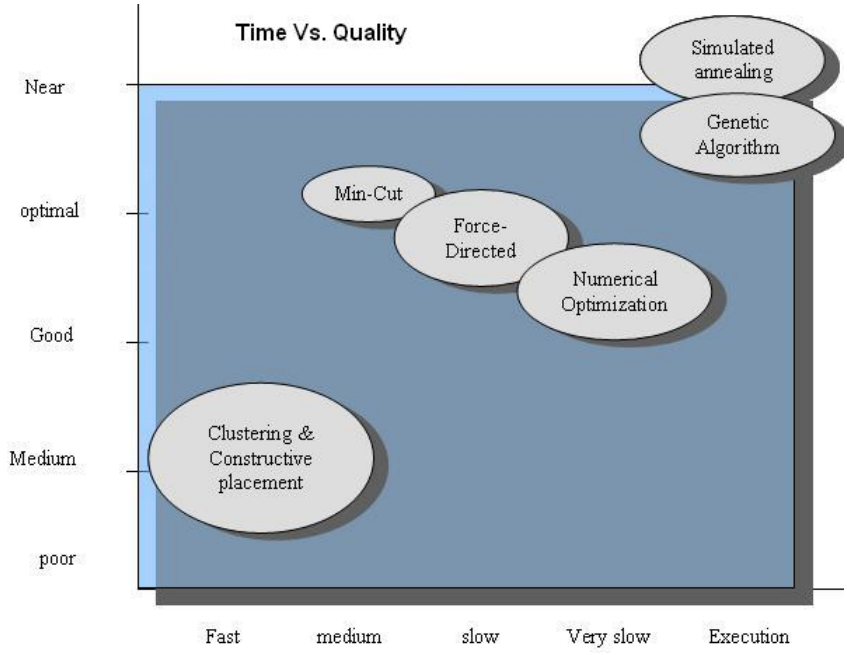

Figure 4.4: SA algorithm Comparison

4.1.3 Simulated Annealing:

The most common iterative technique for island-style eFPGAs (and for many other design problems) is simulated annealing. The simulated annealing algorithm starts with a feasible placement, created either through random assignment of design logic blocks to physical logic blocks, or through the use of constructive approaches and then repeatedly generates placement perturbations in the form of logic blocks swaps. While it clearly makes sense to greedily accept perturbations that reduce overall cost, the search aspect that makes simulated annealing unique is its treatment of swaps that increase or have no effect on overall cost.

An important aspect of the simulated annealing algorithm is the determination of how frequently cost-increasing swaps are accepted. For most algorithms, this acceptance rate is determined based on a probability, $e^{-\frac{\Delta C}{T}}$, where ΔC is the swap cost increase and T is the temperature, a probability parameter which directly controls the acceptance rate. Initially, T is set to a high value so that almost all swaps, good and bad, are accepted. During progression of the algorithm, T is repeatedly reduced and fewer higher cost permutations are accepted, thus allowing convergence to a final result. Important factors that effect the run time and quality of simulated annealing algorithms are the determination of starting temperature T, adjustment of T, number of permutations attempted at each T, and the ending criteria for the algorithm. These parameters have been the subject of a great deal of research and are reviewed in subsequent sections. In figure 4.5 we can see :

- Each state is represented by a node in the configuration set S.
- Each state has an associated cost, directed edge point to low cost states.
- A greedy heuristic after advancing to stage 5 might get stuck in local minima and not reach stage 1 because of higher cost transition stage 3.
  - Efficient heuristic allow hill climbing [3] moves.

- SA is a such a heuristic for solving Combinatorial optimization problems that has inherent hill climbing moves.

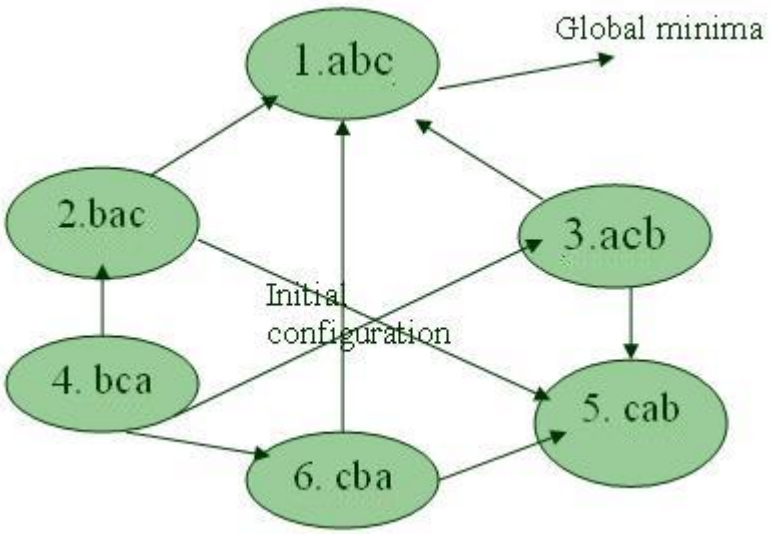

Figure 4.5: Solution Finding approach of SA

While individual implementations of simulated annealing subtasks, such as starting temperature determination and temperature adjustment, vary, the flow of a typical annealing formulation, is constant across most implementations. The italicized subtasks in the figure **??** have been the subject of extensive research with regard to placement for eFPGAs and other design problems. For the evaluation performed in this chapter, previously-tested subtask approaches are employed to create an understanding of the growth rate of annealing time versus design size [12] for the five benchmark circuits previously used to evaluate the growth rate of routing time relative to design size.

Simulated Annealing algorithm Skeleton:

Algorithm 1 Simulated Annealing

```
1: SimulatedAnnealing(solution Seed)
2: T_initial = W_armingUp(Seed)
3: Soln_curr = Seed
4: T_curr= T_initial
5: while convergence not achieved do
6:    T_curr = updateTemperature (T_curr)
7:    while equilibrium not achieved do
8:       Soln_new = generate(Soln_curr)
9:       ΔC = evaluateChangeInCost (Soln_new , Soln_curr)
10:      if accept ΔC , T_curr then
11:         Soln_curr = Soln_new
12:      end if
13:   end while
14: end while
```

Annealing Algorithm components

a. **Initial temperature** The idea is to supply external energy to the system which is high enough to dislodge even the most stressful state in given solution space.

- Normal methods: Let δ be the standard deviation costs of n random generated configuration with each configuration accepted. Appropriate initial temp T0 such that $\delta \ll$ T0
  - $T0 = K\delta$
    - K such that to accept a move with probability P whose cost is 3δ worse than the present configuration
    - $K = \frac{-3}{ln(p)}$
    - K = 25
- Adaptive methods:
  - A scheme by which no fixed parameter (20 or p) is required instead the algorithm itself tunes the temperature to the solution by running

one inner iteration of SA.

b. **Acceptance criteria** The idea is to provide mechanism to escape out of local minima. This is where SA incorporates probabilistic methods to better its search for global minima. Most of the criteria's are direct analogies from statistical mechanics which govern the acceptance (of electron/particle movement) in metal annealing.

- Boltzman based if $(\Delta C \prec 0)\, P(\Delta C) = 1$ else $P(\Delta C) = e^{\frac{-\Delta c}{T}}$ or Min /1.0, $e^{\frac{-\Delta c}{T}}$
- Fermi-Dirac based $P(\Delta C) = \frac{1}{1+e^{\frac{-\Delta c}{T}}}$
- Adaptive if $(\Delta C \prec 0)\, P(\Delta C) = 1$ else $P(\Delta C) = e^{-\Delta C/K|\Delta C|}$ where ΔC is average cost decrement per iteration K is the backword degree constant

c. **Temperature updation**

- Play a fundamental role in the annealing schedule
- Convergence to global optima depends on T that updated as per shown in table I.
  - Aggressive cooling might miss out important hill climbing moves and COP might get stuck in local optima
- In general temperature is updated in a monotonically decreasing fashion.
- Quasi equilibrium could be maintained by requiring the expected decrease in average cost to be less than δ
  - $\Delta C = \lambda\delta$ where $\lambda \leq 1$ thus
    - $T_{new} = T_{old}^{\left(\frac{\lambda_{told}}{\delta}\right)}$
    - δ is the standard deviation of moves accepted at Told
    - $T_{new} = \gamma_{told}$

d. **Equilibrium condition (inner loop criteria)**

α	γ
$\alpha \succ 0.96$	0.5
$0.8 \prec \alpha \leq 0.96$	0.9
$0.15 \prec \alpha \leq 0.8$	0.95
$\alpha \geq 0.15$	0.8

Table I: Temperature Updation Conditions

- Inner loop exit criteria
- Establishment of steady state probability distribution of accessible states
- Define Maximum tolerance limit as
- Target within count and Maximum tolerance defined based on values of
- Equilibrium achieved when

e. **Stopping or convergence criteria (Frozen state)**

- To detect when no further optimization is probable or when the solution has converged to some minimum state.
- The analogy of frozen implies that as the temperature is reduced to very low values, almost no moves are accepted.
- At the end typically the where-you-are(WAY) solution [9] is returned though sometimes Best-so-far (BSF) solution is also returned sometimes.
- Maximum and minimum costs among the accepted states at that temperature are compared with the maximum change in cost in any accepted move. If they are same, then apparently all accessible states have comparable costs and there is no need to continue SA.

f. **New state generation operator**

- Module refers to fundamental unit specified in the netlist
- Location exchange of two modules - Pair wise interchange

  - Randomly selected modules
  - Belonging to same pipeline
- Module displaced to new location - Move based
  - Randomly selected cell to randomly selected location but among a subset of row
- Orientation change of modules
  - When move based fails
  - Module translated to mirror images.
- Orientation change continued
  - Let r be ratio of module displacements to pair wise exchange
  - r has a large impact on placement and wire length calculation.

Cost Function(Application Specific parameter)

An important issue for simulated annealing is the cost function used to evaluate the quality of the global placement. In most cases, including the, the overall placement wire length, determined from net bounding boxes, is used to judge placement quality. Several other cost metrics have recently been used with simulated annealing for eFPGA with varying success. In [13], a cost function was formulated which explicitly took into account the demand versus supply of routing resources for small regions of the target device. Interestingly, it was found that this explicit evaluation of routing congestion achieved very little improvement in reducing W_{min}, the minimum channel width needed to route the design, at the cost of more than an order of magnitude additional computation time. In [4], annealed placement and path finder routing were combined into a single placement formulation. In this case, the cost function for annealed placement swaps was derived from the number of unrouted design nets and the desired minimum clock period of the circuit. While this approach created layouts with about 30% better performance than the discrete layout flow of placement

followed by routing, run times for even small designs[1] of 200 logic blocks measured over 3 hours 20 minutes, effectively making the approach non-scalable.

4.1.4 Clustering Based Placement

Clustering Problem Statement

Given a list of macros (M) and a set of macro level nets (E), form a set of K clusters such that every macro belongs to exactly one cluster.

Given H(M, E) and cluster size bounds L & U, construct $p_k = [C_1, C_2, ...C_k]$ with $L \leq= |C_i| \leq U$, that optimizes a given objective function $f\,(P_k)$.

Constraints :

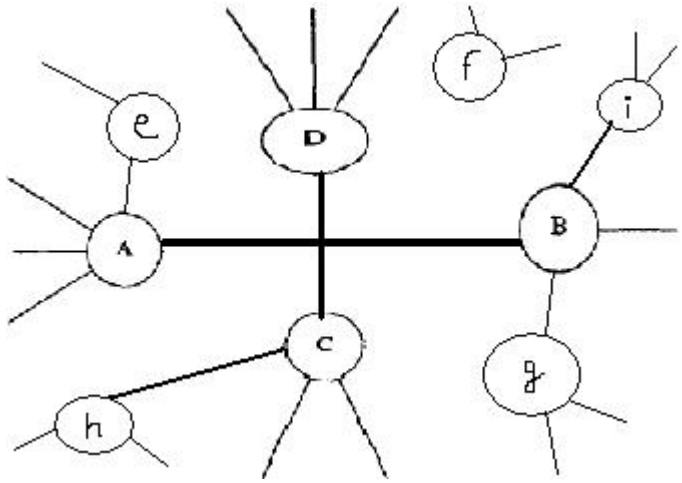

Figure 4.6: Sample Cluster Model

a. Cluster should have macros that are only belonging [10] to a same pipeline level.

b. If macros belongs to a single pipeline level and that also fits in no of physical rows perfectly, that time macros should be distributed in such a way that they must share same wire net.

c. In above case maintenance no of RLC must not increase in size that are initiated from list 0 to list n.

As above mentioned we have modified Simulated algorithm developed in VPR and Tessier, to meet PiCoGA constraints and other basic criteria what PSS tool defines.Only Simulated Annealing based algorithm is not fast enough for FPGA placement, Either we need to use portioning based top down approach or clustering, Here we have developed both approaches and after deep analysis we concluded that Clustering based support for PSS tool is suitable. Basic Clustering flow and different variants applied for clustering are stated below.

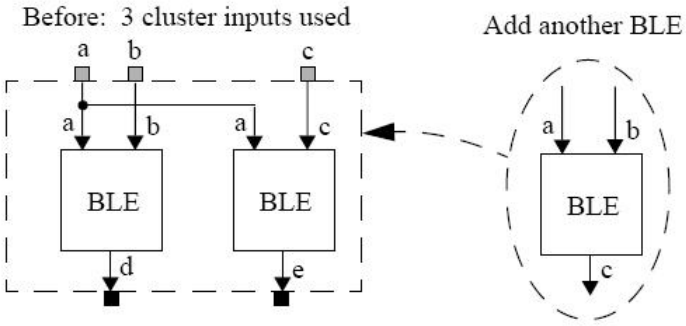

Figure 4.7: Adding a BLE to a cluster.

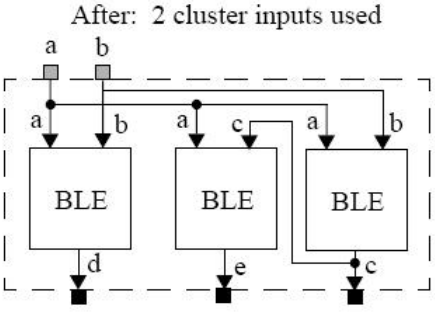

Figure 4.8: It decrease No. of inputs.

Clustering Flow

Input: Macro Level netlist with pipeline information

Output: Placement result

Flow:

a. **Clustering Phase**

   (1) Cluster macro level netlist.

b. **Placement phase:**

   (1) Invoke a placer to perform placement on the clustered netlist.

c. **Unclustering phase:**

   (1) Map clustered netlist to original netlist.

   (2) Perform detailed placement again.

Variants for Clustering methodology

Variant 1: To generate clusters we have derived some constraints that shall be followed in design and implementation of PSS tools' clustering module are explained below.

a. In order to produce a cluster, the first unclustered cell [14] from the ordered cells, referred to as u, is chosen as a seed cell and its connectivity with all of its neighbors (macro blocks belonging to same pipeline stage) is calculated.

b. This connectivity between cells u and its neighboring cell v, conn(u, v) is defined as: No of common fanins

c. After calculating the conductivities, the cell that has the maximum connectivity with u will be selected for clustering.

(1) In the case that the maximum connectivity is the same for several cells, the cell with the smallest area is chosen.

(2) If a tie still exists, that is, several cells have the same connectivity and same area, the cell that has never been visited or clustered will be chosen as the match cell for clustering.

(3) The final way to break ties is to randomly pick a cell among several candidates.

d. After a pair of cells are clustered, the net list is dynamically updated so that the generated new cluster can be visited as a normal cell and clustered again.

e. If the area of this cluster exceeds Cluster Upper Bound [14], no more cell is allowed to be added to the cluster.

(1) The area and size thresholds are used to produce clusters with balanced sizes and improve the clustering solutions.

Variant 2(value based ordering): This technique aims to alleviate routing congestion for clustered FPGAs [10] by absorbing as many small nets into clusters as possible.

a. For all macros belonging to one pipeline stage, find the c factor.

(1) An array of lists for storing all un-clustered cells is maintained [11]. Each element in the array is a list containing all unclustered cells with the same degree in ascending order of their c values.

(2) These sorted lists speed up the search process for un-clustered cells during the second clustering phase.

b. The second phase sequentially builds clusters. A cell seed with the highest degree and lowest c value will be selected to be the seed of a new cluster.

c. Once a cluster seed is chosen, un-clustered cells connected to this seed cell are assigned gain values according to their attraction to the cluster. The unclustered cell with the highest gain is absorbed into the cluster.

How seeds are clustered is shown in figure 4.9 and explained below. **Cluster For-**

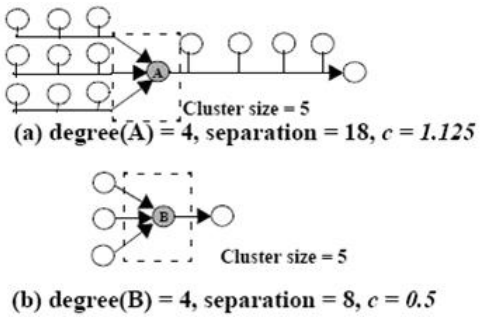

Figure 4.9: Clustering Interpretation

mation: Clusters are built sequentially

$$C = \frac{seperation}{degree^2}$$

where

separation= sum of all terminals of nets incident to the cell

degree= no of nets incident to the cell

C = connectivity factor associated with each cell [smaller c value signify that more blocks are located in a given blocks neighborhood]

Consider an Accessory candidate cell X which is being considered for inclusion into a currently open cluster C. The attraction of X to C is a function of the weight of the net x and the number of pins of x already inside cluster C. We define this gain quantitatively as:

$$G\left(X, C, x\right) = 2nw\left(x\right) * \left(1 + \alpha x\right)$$

where

- αx = number of pins of net x already inside C.
- n is the cluster size.
- w(x) is the weight of net x (w(x) = 2/r where r is the number of pins on x).

If adding X to C fully absorbs net x , then is multiplied by a constant $k(k > 10)$.

This essentially ensures that blocks attached to the currently open cluster through smaller nets are more likely to be absorbed in the cluster since they are assigned higher weights. Assigning higher weights to such cells is essential, as there can be other un-clustered cells attached to C (for example,element Y) through multiple long nets which may not be entirely absorbed in C.

4.1.5 Direct Inter-connectivity supported Placement

Consider the following 4.10 two block scenario where blocks are proposed to be swapped to gain higher routability [9] and decrease minimum length with considering PiCoGA architecture [1] constraints for placement.

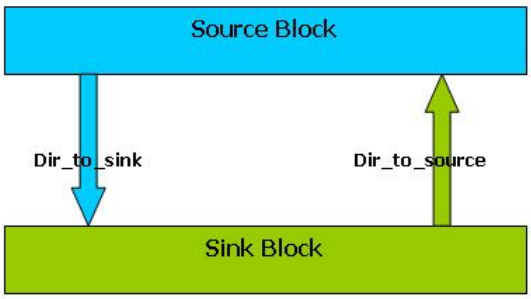

Figure 4.10: Two block scenario of Direct Inter-connectivity

In above scenario we have two interconnected blocks [9] that are placed on some coordinate SoB(X,Y) and SiB(X',Y'). Now when we apply swap function to optimize route path for source to sink block based on direction aware placement algorithm. So according to method we need to swap block on such a place so that we don't utilize

tracks to connect each of the block, when we have direct interconnects are already available tow connected block.

If we got blocks swapped [1] as per the method of direct interconnect our placement would use internal direct interconnected input wires to connect source and sink to each other as well as to the input and output pins also in efficient way.

Below are some steps that shows the way of direct interconnect method utilization to support simulated annealing placement.

a. Source block will be at some new location (X_n,Y_n) when swap function is called with direct interconnect aware placement method.

b. Sink block will be at some new location (X'_n,Y'_n) when swap function is called with direct interconnect aware placement method.

c. SoB will provide its output to SiB as input that we earlier forwarding through track of chip.

d. And in same way SiB will provide output to SoB as input that were functioned by chip track.

Here $\left[(\Delta WL)_{swap}\right]_{nm}$ is the normalized change in wire-length after swap function is called with DI placer and has the usual meaning .So now we can say that when swap will be performed and DI effect is reflected in the placement.Overall cost will reduce for such placement which has block having more number of sources placed at the top and penalize vice versa.

We need here normalization factor because at some instance we use cost function would be our function of change in wire length only (when we only deal with cluster based Simulated Annealing placement) but now it has to accommodate DI gain too to show the effect in new cost function that also utilize DI factor.

Direct Interconnects Utilization for Multiple Blocks

When there are multiple blocks are multiple interconnects are available to be swapped in placement phase we need to extend as shown in figure 4.11 scenario for multiple blocks with following methodology.

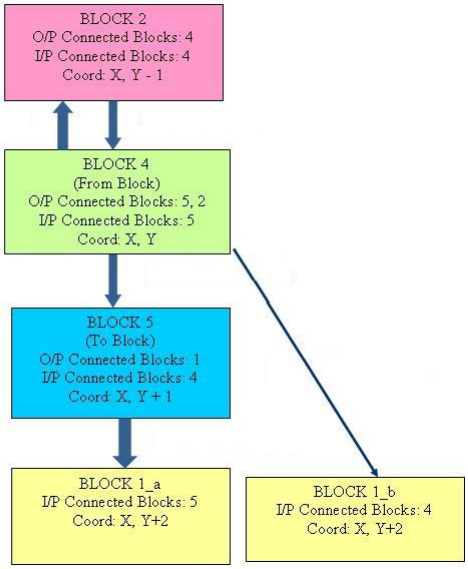

Figure 4.11: Direct Interconnects Utilization for Multiple Blocks scenario

According to architectural constrain we could get benefit of the direct interconnected blocks, so that less resources are used to pass the Input/Outpout. Here as per shown in figure 4.11 when the are no of blocks can be utilized under Direct interconnect utilization, we likely them placed in following manner so that source and sink both can get benefit from direct inter-connectivity to each other .We need to find here no of blocks to be swapped to utilize DI technique, total DI utilization amount in percentages.

4.2 Tools

As Company has planned to develop PnR tool platform Independent, And we are dealing with rapid software technologies, we have adopted ready to use compilers, IDE and as most used platform Window-XP and Linux-RedHat, considering both 32 and 64 bit environment .

4.2.1 Visual studio 6.0 & Eclipse CDT(GCC):

Microsoft Visual Studio [1] has a widely used among QT application developers. It is the leader in C and C++ Microsoft Windows application developers, as well.

The Eclipse Platform [2] is an open source tool to assist us with moving a project from the design to the test phase within a single development environment and without the need for separate tools for each stage. Ease of development is achievable because of programming model of Eclipse, which supports software building blocks called plug-ins.

In comparison Eclipse CDT is relatively new and quite popular these days in the developer's community. Eclipse CDT is designed to satisfy the developers of many OS platforms, such as Linux, and is dependent on the GCC compiler and other open source tools like QT.

4.2.2 QT Framework:

Qt [5] is a cross-platform application development framework, widely used for the development of GUI programs (widget toolkit), and also used for developing non-GUI programs such as console tools and servers and easily integrated with different IDE like Eclipse and Visual Studio.

Qt uses C++ with several non-standard extensions implemented by an additional pre-processor that generates standard C++ code before compilation. It runs on all major platforms, and has extensive internationalization support. Non-GUI features

include SQL database access, XML parsing, thread management, network support and a unified cross-platform API for file handling.

4.2.3 IBM Rational-Rose & Clear-Case:

Rational Rose [4] is an object-oriented Unified Modeling Language (UML) software design tool intended for visual modeling and component construction of enterprize-level software applications. In much the same way a theatrical director blocks out a play, a software designer uses Rational Rose to visually create (model) the framework for an application by blocking out classes with actors (stick figures), use case elements (ovals), objects (rectangles) and messages/relationships (arrows) in a sequence diagram using drag-and-drop symbols. Rational Rose documents the diagram as it is being constructed and then could also generate code c++.

Rational Clear-Case [3] is a software tool for revision control (e.g. configuration management, CVS) of source code and other software development assets. It is developed by the Rational Software division of IBM. Clear-Case forms the base of version control for many IDE like visual studio and Eclipse and mange lots of developers.

4.2.4 Lex & Yacc - Compilers

Lex and yacc are a matched pair of tools. Lex breaks down files into sets of "tokens" roughly analogous to words. Yacc takes sets of tokens and assembles them into higher-level constructs, analogous to sentences. Yacc is designed to work with the output of Lex, although you can write your own code to fill that gap. Likewise, lex's output is mostly designed to be fed into some kind of parser.

They're for reading files that have reasonably structured formats. For instance, code in many programming languages can be read using lex and yacc. Many data files have predictable enough formats to be read using them, as well. Lex and yacc can be used to parse fairly simple and regular grammars. Natural languages are beyond their scope, but most computer programming languages are within their bounds.

Lex and yacc are tools for building programs. Their output is itself code, which needs to be fed into a compiler; typically, additional user code is added to use the code generated by lex and/or yacc. Some simple programs can get by on almost no additional code; others use a parser as a tiny portion of a much larger and more complicated program.

We use lex and yacc tools to generate assembled code for architecture and design files.PiCoGA architecture file and different Design RLC files are given input to PSS placer, Both are very organized input file,

- **Lex: A lexical analyzer generator**

  A lexical analyzer [15] isn't a hand held gizmo you get on a sci-fi show. It's a program that breaks input into recognized pieces. For instance, a simple lexical analyzer might count the words in its input. Lex takes a specification file and builds a corresponding lexical analyzer, coded in C.

- **yacc: yet another compiler compiler**

  So, you've broken your input into a stream of tokens. Now you need some way to recognize higher-level patterns. This is where yacc comes in: yacc [16] lets you describe what you want to do with tokens.

Chapter 5

System Design and implementation

5.1 CAD flow for eFPGAs

Implementing a circuit in a modern FPGA requires that hundreds of thousands or even millions of programmable switches and configuration bits be set to the proper state, on or off. Clearly if a circuit designer has to specify the state of each programmable switch in an FPGA very few designs will ever be completed! Instead, users of FPGAs describe a circuit at a higher level of abstraction, typically using a hardware description language (such as VHDL) or schematic entry. Computer-Aided Design(CAD) [9] programs then convert this high-level description into a programming file specifying the state of every programmable switch in an FPGA. To keep the complexity of this procedure tractable, the problem of determining how to map a circuit into an FPGA is normally broken into a series of sequential sub problems [11], as shown in Figure 5.1. In the following three sections we will describe the synthesis, placement and routing problems and briefly outline some of the prior work in each area. Once a circuit is placed and routed, delay models and timing analysis are used to assess its speed.

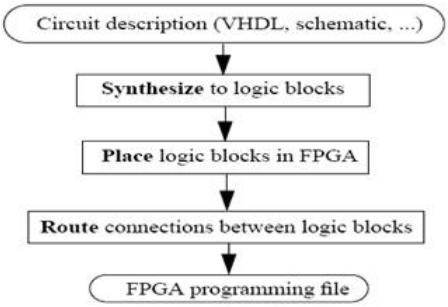

Figure 5.1: CAD flow of eFPGA

5.2 Chip Architecture

I already explained PiCoGA architecture, but here in reference to implementation point of the whole chip has been described in a particular hierarchy. There are some basic units in a chip like Input Registers, Output Register, RLCs and nets to connect these RLCs. The algorithm which we have used can directly be applied on RLCs but this process is very time consuming. So, to ease of use and to save the time plus to reduce the complexity of coding we have created a hierarchy of these units.

-Pipeline

-Cluster

-Macro

-RLC

Now start from RLC. It is a basic unit of a chip. For PiCoGA we have 24 rows and 16 columns. So, each row contains 16 RLCs. In each tile we have RLC in it because we have an Island routing architecture. According to the requirement and functioning of a chip some RLCs may be strongly connected and some RLCs may loosely connected.

Macro is a collection of RLCs. Based on the connectivity of RLCs, macro can be define. All the RLCs which are responsible for implementing the single functionality can reside in a single macro. A macro can contain two or more RLCs. At the stage

of detailed placement we do the placement at macro level.

Cluster is a collection of Macros. Based on the connectivity of Macros, cluster can be define. All the Macros which are responsible for implementing the single functionality can reside in a single cluster. A cluster can contain more than one Macros. One constraint is that cluster should not expanded more than one physical row. So, in this chip we can have minimum 24 clusters. At the stage of global placement we do the placement at cluster level.

Pipeline is a collection of clusters. Based on the connectivity of clusters, pipeline can be define. All the clusters which are responsible for implementing the single functionality can reside in a single pipeline. A pipeline can contain more than one clusters. Pipeline can be expanded more than one physical row. So, in this chip we can have minimum 1 pipeline and maximum 24 pipelines[6].

5.3 PSS Placer Tool Flow

Specific in this dissertation to design a placer tool reconfigurable logic blocks on eFPGA we have divided the whole process in subgroups. The whole process is shown in Figure5.2.

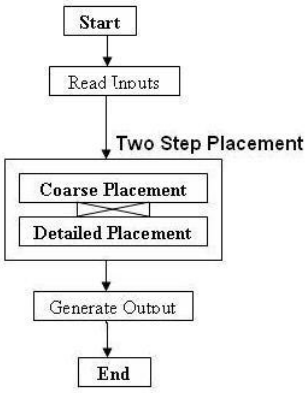

Figure 5.2: PSS Tool Flow.

5.3.1 Read Input and Generate Chip View

This is the first sub module to design a tool. In this sub module we read the input given by user, check the validity of given data, assign these values to appropriate data classes and generate the chip view 5.3. We have provided the user two types of facilities to give input to tool. We have provided a GUI as well as command line option to user to input the data. Through GUI user just need to select the particular input files. System will automatically reads the contents from those input files and design a chip view. Next is command line input option in which user enters the file names at command line and then system reads those files and generates a chip view.

In detail when we have input from user we need to check the validity of all the data. For example, first we have to check the availability of input files. Then we check the contents of all the files. Suppose we have detail of RLC [9], We check the coordinates of it, its connectivity of other RLCs, number of OPF nets, RLCs feeding output register in appropriate column.

Then we check the availability of all the files which are given by the user. We check that given path and check the existence of those files, names and path given by user. After checking this thing system will generate the list of units which can be used in placement of chip. We generate RLC list information. In which it contains all the RLC and its detail like source of RLC, sink of RLC, id, name, coordinates, number of OPF nets, parent block of that RLC, reference list index of RLC, Output register list, etc. Same as RLC, we create Cluster, Macro and Pipeline list information. Pipeline list contains the objects of pipeline. It has some other attributes like constituent block list, physical row index, etc. Cluster list contains the objects of cluster. Next is we generates the list of input and output registers. These also has some additional parameters. All these registers are 32 bits. We need total 12, 32 bit input registers to feed all input lines of RLCs.[1] Same as we need 4, 32 bit output registers to receive the output from RLCs.

After creating a chip view we generates the net information. In which we creates

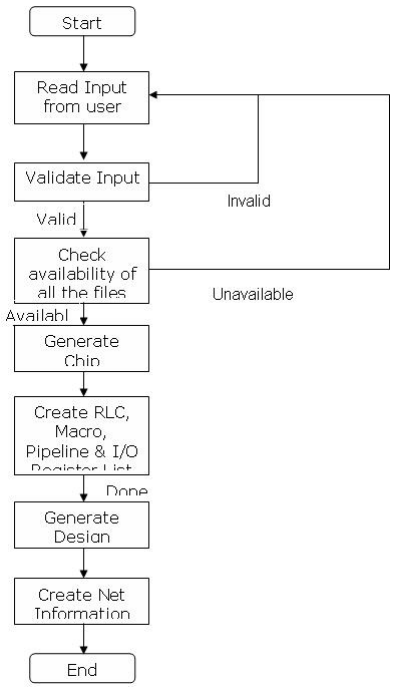

Figure 5.3: Read Input and Generate Chip View.

a RLC level, Macro level and pipeline level source and sink information. We design the complete hierarchy of units. Input registers feeding the RLCs, these RLCs are part of macro and it is part of pipeline. So, on the basis of contents in input file the whole hierarchy is defined this way.

5.3.2 Clustering & Placement

The clustering phase clusters all macros belonging to one pipeline stage with the size of cluster fixed to the no. of columns in the PiCoGA chip, so that each cluster corresponds to one physical row.

In placement phase, all clusters are placed using placer algorithm. Following two placer variants are considered for this phase:

- SA based variant
- Constructive Placer based variant

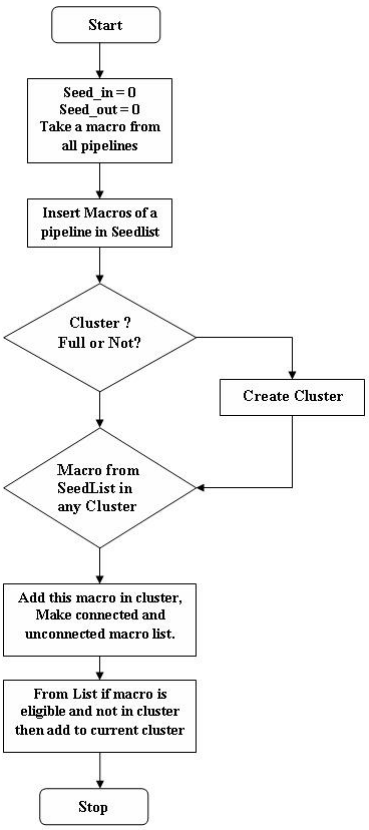

Figure 5.4: Clustering generation Flow.

Once clusters are placed [11], the congestion for each cluster is estimated and if congestion for a cluster is greater than average cutoff (as specified by the user), then a new cluster corresponding to that pipeline is added in the design to resolve the same cluster generation.

- Coarse Placement
- Detailed Placement

In first phase coarse placer[2] perform initial placement on clusters, then all clusters are un-clustered and after that again placer is applied.In next phase detailed placement all macros are verified for overlap criteria and then applied SA based placed.

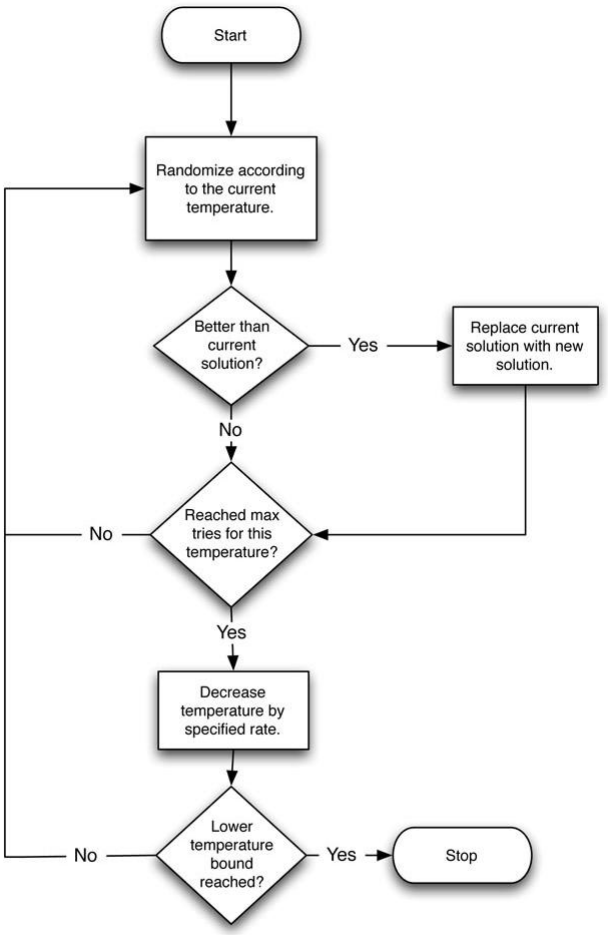

Figure 5.5: Coarse/Detailed Placement Flow.

5.3.3 Generating Output

This is a final and a very important sub module for the placement phase. In this module we generate the output as desired by the user. In this module we generate different placement output files, statistics files and log files which stores all the necessary information for next phase of this CAD tool which is routing.

Now if we look into detail mainly we have detail placement solution file which is ".rlc"[9] file which is the most important input to the next phase which is routing. In this file we dump all the details of each RLCs like name, id, source register, destination register, operation code, operands, LUTs, details of input output pins of RLCs. It

also contains some additional details like carry bit information, synchronization bit information, coordinates detail, etc. Here, a simple example is given below. In our placer tool we generate the same type of information at the end of detail placement.

For Example,

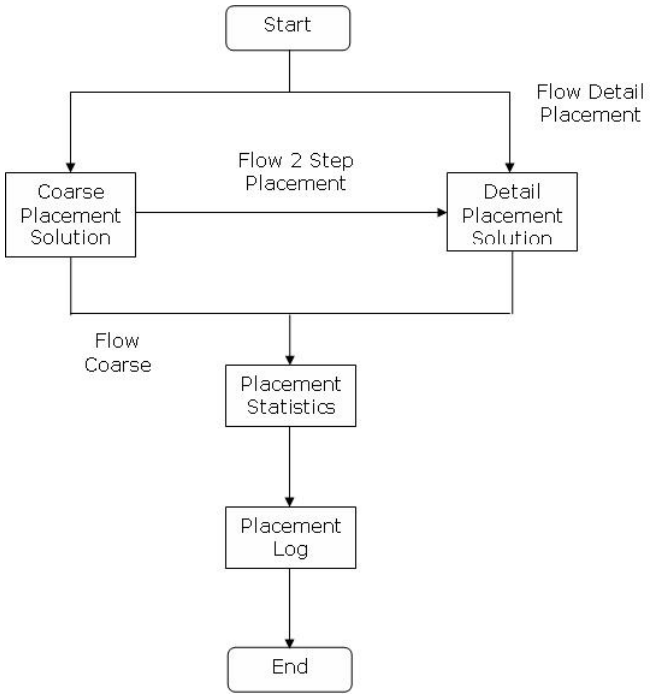

Figure 5.6: Generating Output.

```
Macro1
    RLC1
Details:
Coordinates:
    RLC2
Details:
Coordinates:
Macro2........
```

Then we have coarse placement solution file which contains input register infor-

mation, output register information, pipeline and cluster level information. For IO register this file contains register name, register ID, coordinates, etc. Now the main thing of these file is information about pipeline and cluster level information. It contains cluster level information like, cluster id, name, physical row, number of macros in it, etc for each clusters of each pipeline. Here, a simple example is given below. In our placer tool we generate the same type of information at the end of coarse placement.

For Example,

```
IO_register_information
    Register_name
    Register_Id
Pipeline_Id1
    Cluster_id
    Etc
Pipeline_id2......
```

Then we have placement statistics file which contains the information about the statistics of placement. It dumps time for coarse placement, time for detail placement, total time taken for placement. Number of clusters, number of pipelines, number physical rows all these information dumps in this placement statistics file. We also have the placement log file which keeps track all the functionalities of placement of logic blocks.

5.4 Test And Validation Codes

During first phase we completed building binaries and executable for window and linux platforms for 32 and 64 bit machines. After achieving soft milestone we are intended to make all development concrete and bug free. So challenge of testing of tool and validating the data passing as input and getting valid output and intermediate data

structure rise before we further move to optimize already developed techniques for placement using simulated annealing and clustering and direct interconnect between macros within pipeline.

5.4.1 Testing

Software Testing can also be stated as the process of checking that an executable of tool

- Meets the business and technical requirements that guided its design and development.
- Works as expected.
- Can be implemented with the same characteristics.

Tool test codes are prepared for most of the data interaction classes so that we could apply white box testing internally and can generate reports, all test outputs are generated only when related test labeled macros are enabled.

Testing Codes

As we have completed building binaries and executable for window and linux platforms for 32 and 64 bit machines. After achieving executable we are intended to make all development concrete and bug free. So challenge of testing of tool as input and getting required output and intermediate data structure rise before we further move to optimize already developed techniques for placement using simulated annealing and clustering and direct interconnect between macros within pipeline.

Testing Methodology

To test PSS cad tool, we applied unit testing to find out classes are behaving as expected and designed. With unit testing we applied functional testing for some of the classes also to check exact functional work is being carried out in that classes as

per the requirements.After unit testing within each module I applied module testing on each separate module, GUI, Placer and Router. In module testing method for every module correct output was checked by applying valid and invalid input to application. Integration testing after code integration was performed in bottom-up approach where we tested integration by braces that are being merged into the integration interface. After applying different testing methodologies to make sure whole application is working as per requirement we made Application testing through our generated automated testing scripts.

5.4.2 Validation

Data validation is the process of ensuring that a program operates on clean, correct and useful data. It uses routines, often called "validation rules" or "check routines", that check for correctness, meaningfulness, and security of data that are input to the system.

Validation of architecture and design files was the challenge and before that we also have to validate passed input data either on command prompt with executable, or internally defined parameters and combination of values constants.

Validating architecture and design files through c++ itself was a tough task and for that we developed validatory classes that have functions that could be called in regular tool and in regular flow from placer to gui.

Different Validation classes are

- Architecture_validator
- Pre_placemeent_Design_validator
- Post_placemeent_Design_validator
- Placement_database_validator
- Status_validator

- Initial_placement_validator
- Coarse_placement_validator
- Detailed_placement_validator

5.4.3 Test Automation & Data Collection

Every software development group tests its products, yet delivered software always has defects. Test engineers strive to catch them before the product is released but they always creep in and they often reappear, even with the best manual testing processes. Automated software testing is the best way to increase the effectiveness, efficiency and coverage of your software testing.

As now we have executable for window and linux are on hand and we have many design files to be tested and check that given input design files works fine with executable and gives desired placement output or not is must be tested. Either we can manually perform testing by locating each design file.

Requirement of Automated Testing:

- Locating Architecture and Design Files.
- Run tool for all design files for three optimization type.
- Directory path changing.
- Termination of binaries after the placement solution is generated.

Manual software testing is performed by a human sitting in front of a computer carefully going through application screens, trying various usage and input combinations, comparing the results to the expected behavior and recording their observations. Manual tests are repeated often during development cycles for source code changes and other situations like multiple operating environments and hardware configurations.

With automated testing5.7 we also need to collect the analysis data from the generated output files by PSS tool after placement, here we have made it as a part of automated tool so that data collection also becomes easy compared to manually data collection from every single files.

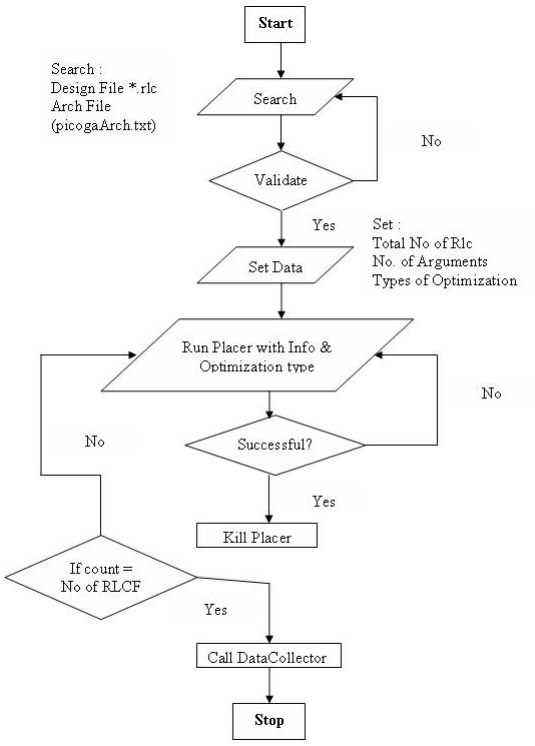

Figure 5.7: Testing Automation Flow

After generation of output reports in *.pst, *.cps and *.txt formats, I have created automated scripts to extract needed data from such output files, This scripts extracts data like design name, architecture used, placement type applied, wirelength utilization, direct-interconnect utilization, row utilization and input and output registers, No. of pipelines, No of macros, clusters generated, placement status, input processing time, clustering time, placement time, output generation time

Utilities were build to make testing automatic are test_runner, which collect data,set data and run the tests accordingly. Then File_search helps test runner to

collect file related data. binary_killer Kill the executable exactly after placement is finished and output files are generated by tracing macro generated file. Finally data_collector collect data that is need to be analyzed and put it in specific format.

5.5 Code integration and code management

Code integration involves combining codes residing in different sources and version and providing users with a unified view of these data. This process becomes significant in a variety of situations both commercial and scientific. Data integration appears with increasing frequency as the volume and the need to share existing data explodes. It has become the focus of extensive theoretical work, and numerous open problems remain unsolved.

5.5.1 Code Integration

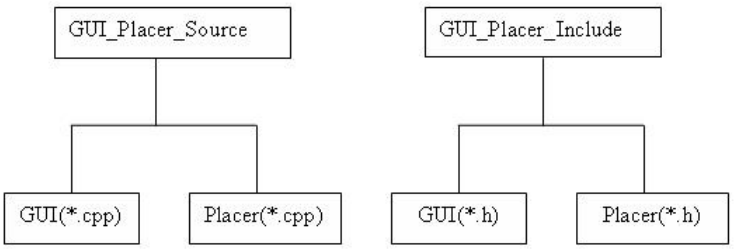

Figure 5.8: GUI and Placer Integration (Source and Header)

We have code integration [9] developed as shown in following figure in which all modules code in source and in header file is integrated with the joint interface. GUI_Placer_source interface [1] combines all cpp source files from two different module GUI and placer5.8. And same GUI_Placer_Include interface [9] combines all header file interface that combines GUI and Placer header files.

5.5.2 Code Management

Our Placement cad tool project's code is managed under IBM's rational rose's clear case tool [1], which is essential utility tool, is managed by special team. And our project team works on code by creating personal user view to work on project and later after change in code and upgrade in code views are updated on the clear-case server as a branch on main branch with meaningful label on it.

- Created View from Pss_Placer_1.0 -name : user_placer_view
- Created branch Of every Files in SRC and INCLUDE

  ```
  [mkbranch -version \main\pss\_placer1.0\0 -nc user\_placer1.0
  Filename]
  ```

- Checkout All Files and Updated with Documented Files
- labeled Every File and new branch user_placer1.0 with PSS_PLACER_PORTED
- Removed Label from All the Folders in project Directory of View
- Check-In the labeled/Documented/Updated Files on server.
- Created another View user_test_view and load Files in it with label PSS_PLACE_PORTED.

5.6 Debugging

Every time we integrate new code in either platform we make some corrections according to compiler of window and linux, but though most changes are known as I explained in earlier reviews. Else then some pre detected changes while migration, we have to go through debug to find out where the problem is coming and application is

getting break in it. We have used eclipse debugger, VC++ 6.0 debugger and DDD[6] debugger for debugging.

Debugging allow us to see what is going on inside Application runs and/or let us see what program was doing when it crashed. Tools allow us to examine a program's state (variables, stack frame, memory contents, etc.), allow you to set breakpoints to stop the program at a certain points to examine its state, and allow us to alter the value of program's state

5.6.1 General Bugs

a. **Bug:** Placement validate pointer of designer class was getting initialized to NULL after actual placement.
Solution: Create validate pointer for designer class after actual placement from the proper existing validation points from ChipInfo.

b. **Bug:** Placement status was not being updated at time of Output Generation.
Solution: Update Placement status enumeration to according status by flag status from Output Generation.

c. **Bug:** Even though the cluster contents might be logical in terms of architecture and design constraints, while assigning coordinates to the macros, not sure that constrains were met and not violated.
Solution: With updates of following information while assigning coordinates first make sure macros driving in first half of output registers are assigned coordinates in first half only. And check this information from data driving from output register and for normal macro block.

5.6.2 Linux Bugs

a. **Bug:** Placement Editor get crashed on select of macro after GUI invoke (Segmentation fault in GUI mouse Drag and Drop).

Solution: Window's code was ported to linux and after needful change while we run binaries it gave similar output, but in GUI when we tried to change place of macro application got crashed. On debug we came to know that problem was occurring due to macro's coordinate address were assigned as normal integer values where as they comes actually from Qt stack in 64 has bit integer value. So in graphical scenes of GUI module address updating was reinterpret_cast used to update type of returned Qt coordinate address.

b. **Bug:** Segmentation fault while reading architecture file.

Solution: After code porting on linux from window before starting placement and physical chip generation first input architecture file for PICOGA is read and generates according architecture on chip to run placement for. But as lex and yacc generated files were from window and using them in linux to read data from architecture file it read architecture input and throws segmentation fault, linux platform's gcc compiler was not able to read data with help of window generated lex and yacc files, So we generated needed lex and yacc files in linux using linux Bison and Flex parser.

(1) Generated arch.tab.cpp and arch.tab.h files from arch.y
using linux bison parser with following command.
$yacc -d arch.y
Rename to arch.reader.cpp and arch.reader.h

(2) Generated rlc.tab.cpp and rlc.tab.h files from rlc.y
using linux bison parser with following command.
$yacc -d arch.y
Rename to rlc.reader.cpp and rlc.reader.h

(3) Generated arch.parser.cpp and rlc.parser.cpp from arch.l and rlc.l
using flex parser with following command
$flex arch.l
$flex rlc.l

Chapter 6

Results and Analysis

Here I will describes tests results and analysis after applying various tests on PSS tool. In next section some screen shots of generated placement output of PSS tool are shown for some design. We collected results for five standard circuit designs that are made for PiCoGA architecture, based on that statistics we have some conclusion that shows positive results and optimization achieved in comparison to earlier tools for placement.

6.1 Test & Results

PSS tool deal with PiCoGA architecture that apply certain constraints for routing and placement. After applying clustering support to basic SA algorithm we show improvements in run time of placement, and achieved better placement.To perform the test we provide circuit file with architecture file of PiCoGA and can see placed macros on chip.In next section there are some output screen shots of placement results.

PSS tool performs clustering based SA placement gives faster and better results for mentioned architecture, But with support of direct interconnect aware support to placer, results are very good. We also perform Test by giving changing SA parameters like, Initial Temperature, temperature updation criteria, Iteration for each temperature, and measured results.

Placement editor is provided very easy to use interface after automated placement has been achieved, It can show different view as per shown below. Each View has different perspective of use in editor. **Pipeline view**, that shows how many pipeline are available after placement.**Macro view**, that shows highlighted macro's enlarged image and details of macro.**RLC view**, Register level view that shows wired Input and output connection within chip.**Data-Path View**, that shows input and output connection and their connection to switches.

We have developed complete automated environment to test different circuit design, and created auto script that invoke PSS and then collect data from different output files generated by PSS tool.Following table shows spasticities for five designs which supports PiCoGA architecture.

Following tableI shows placement results after development of simulated annealing and clustering supported algorithm development.

Design	% of Rows	Wirelength	Time(Sec)	Clusters
addition.rlc	83.3333	17.5588	388.795	23
Multiplication.rlc	29.1667	4.59563	93.061	7
complex calc.rlc	95.8333	22.8136	420.531	20
4x4subtract.rlc	20.8333	6.2547	367.014	5
4x4multi.rlc	33.3333	4.08853	157.738	8

Table I: CAD tool Generated Placement based on Algorithm

Following tableII shows placement results after development of simulated annealing and clustering supported with direct interconnect algorithm development. Here we can see improved results for all of the designs with but it took more time than normal placer tool.

Following table shows manual placement results after bug fixed of GUI placement editor of PSS_Placer tool. Here we can see improved results for some of the designs. This is good benefit of placement editor as we can place macros manuallyIII if design requires such change in placement.

Following are collected statisticsIV for each of optimization flag for 32 and 64 bit [9]- Linux platform (Best results are considered for Wirelength utilization)

Design	% of Rows	Wirelength	% DI factor	Time(Sec)	Clusters
addition.rlc	83.3333	16.4874	3.25	421.432	23
Multiplication.rlc	29.1667	3.99563	12.8571	102.71	7
complex calc.rlc	95.8333	19.4357	2.6087	529.351	20
4x4subtract.rlc	20.8333	5.32306	14	352.23	5
4x4multi.rlc	33.3333	3.584	17.0625	189.352	8

Table II: CAD tool Generated Placement using SA with cluster and DI support

Design	% of Rows	Wirelength	% DI factor	Time(Sec)	Clusters
addition.rlc	83.3333	14.3582	3.25	421.432	23
Multiplication.rlc	29.1667	3.2387	12.8571	102.71	7
complex calc.rlc	95.8333	15.3983	2.6087	529.351	20
4x4subtract.rlc	20.8333	2.2938	14	352.23	5
4x4multi.rlc	33.3333	4.4562	17.0625	189.352	8

Table III: Manual Placement using GUI placement editor

6.2 PSS:Placer Tool Output

This section shows placement performed for design multiplication.rlc, first when clustering based simulated annealing supported placement, in Figure 6.1, & then after introducing direct interconnect aware support into PSS placer tool, Figure 6.2.

Following figure 6.3 shows how wires are connected in placement, this data view is collected by routing algorithms in initial phase before placement.

6.3 Analysis

After comparing collected statistics with previous data and early days tools like timberwolf and VPR[2] we can conclude that PSS is taking less time for complex architecture PiCoGA, and generate cluster in less time also.After developing direct interconnect aware placer that utilize direction based data and effectively place cluster or macros in such a way so that we can get less total wirelength utilization. PiCoGA constraints help us by giving direct interconnect data in improving placement output.

As per shown in figures we are also get benefit of providing user ability to place

Rank (Wirelength Utilization)	3	1	2
Linux Platform	WL_Only	DI_Only	WL_DI
32-bit	8	11	9
64-bit	7	12	10

Table IV: Result comparison between 32 and 64 Linux platform

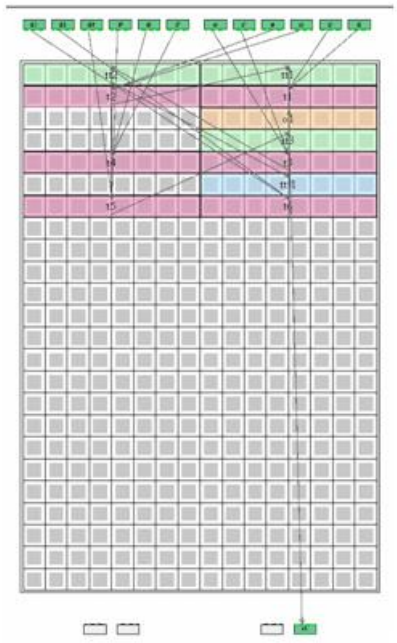

Figure 6.1: Placement By Cluster Based SA

already generated placement manually in GUI interface that we called Placement Editor.One issue is that when we introduce direct interconnect support in placement it increase run time for all design circuits by 10% to 20% more.

When we applied different SA parameters externally and changed them for some test purpose, we got some better test results for some of the design files, But we cant set that parameter permanent because on an average Chapter 4 defined SA parameters worked best for PSS tool and gives average best results.

We can also say that when we change some parameter we have to pay penalties for something to get better results, like two different sides of coins, for better wirelength utilization we attained more CPU utilization and more Run time also.

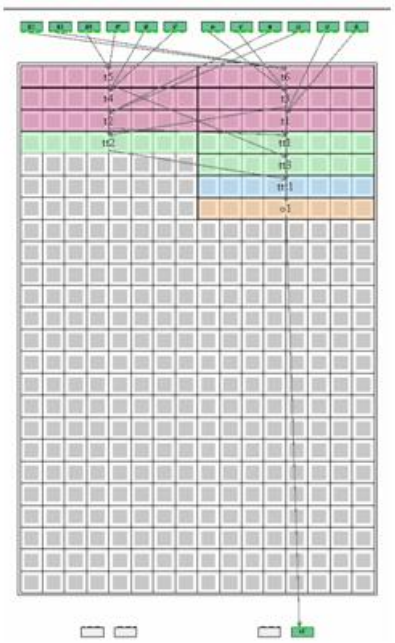

Figure 6.2: Placement By Cluster Based SA and Direct Interconnect Support

But still we are not getting up to the mark results for some design files, after placement, by manual placement we can get better results, thought that are just human inspected placement, but we are lacking some what in PSS tool.

6.3.1 Observation of results

Based on data collected during different test applied on different designed logic circuits, following are the observations that gives ideas of how tool behave and gives output.

- Design files that have row utilization 60% to 80% gives 5% different results on 32/64 bit platform of linux.
- Whenever Direct Interconnect only shows best results with respect to wirelength utilization its DI factor also come highest in optimization flag.
- Window and linux platform has almost identical result for most of all designs.

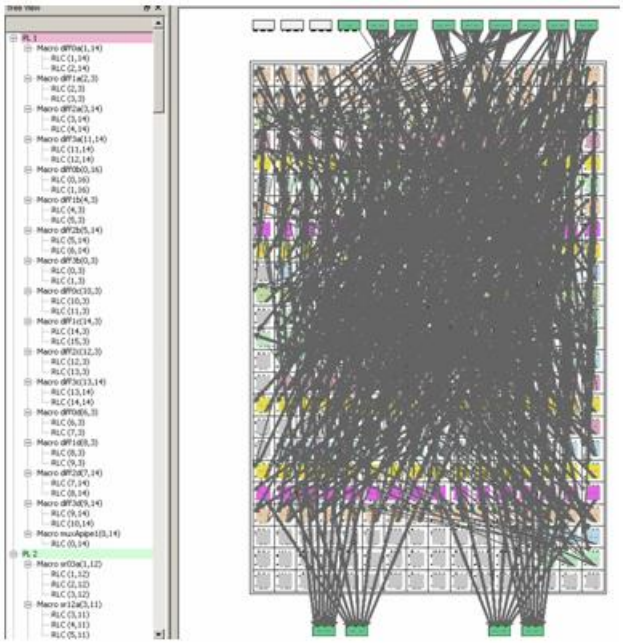

Figure 6.3: Connected wired view for complex_calc.rlc

- If we don't want to apply either direct interconnects or wirelength, then we could apply both optimization flag which gives 40% better result than Normal placement.

- DI only optimization option gives 85% better results then normal placement.

- In results 40% of designs give equal results for considered all three scenarios.

- Running tool for more number of iterations SA loop gives better result but up to certain limits only.

- As per shown in above tables we can see manual placement by editing generated placement are far better almost 10% better.

- As we can see in data collected for linux 32 and 64 bit platform, 10% of difference we gets in design, but that we think just because of different compiler.

Here following three figures shows comparison between two different executable's results for 50 and 100 iterations and in 100 iteration executable constant Information is also adjusted to the best. Fig A shows only wirelength based running mode, fig B. display direct_Interconnect only run and fig C. shows 50-50% of both's run. Here an optimization option takes solutions accordingly from he given option and come with regarding placement solution.

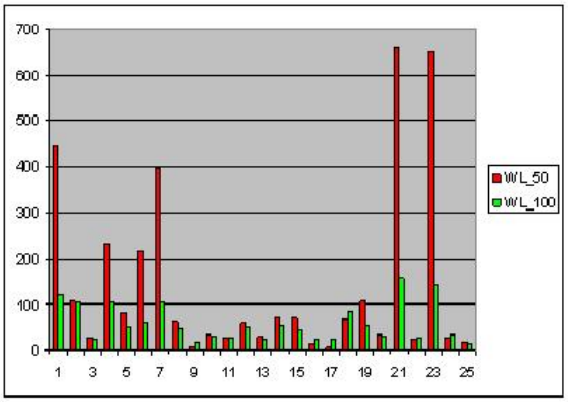

Figure 6.4: 50 vs. 100 Iteration- result comparison for WL Only

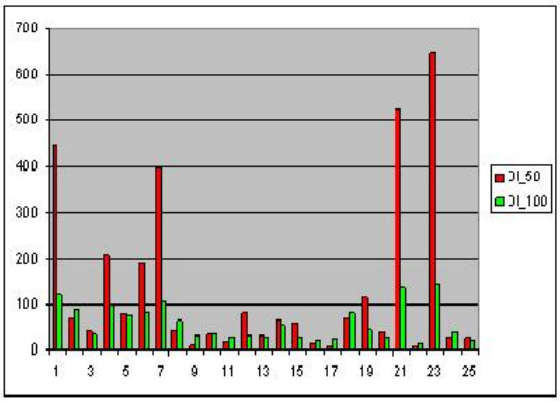

Figure 6.5: 50 vs. 100 Iteration- result comparison for DI Only

Following figure shows graphical comparison of three methodologies we applied, wirelength based only, direct interconnects based, combined methodology that takes

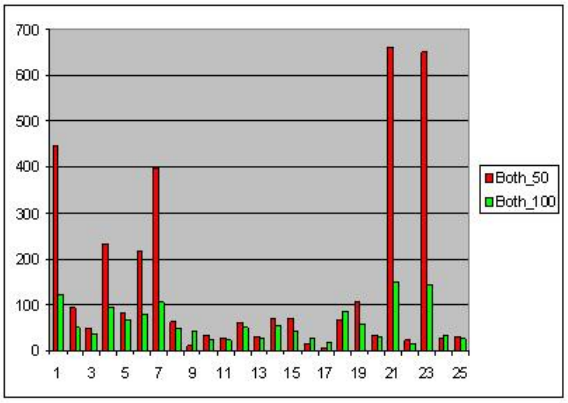

Figure 6.6: 50 vs. 100 Iteration-result comparison for Both

benefit of wirelength and DI.

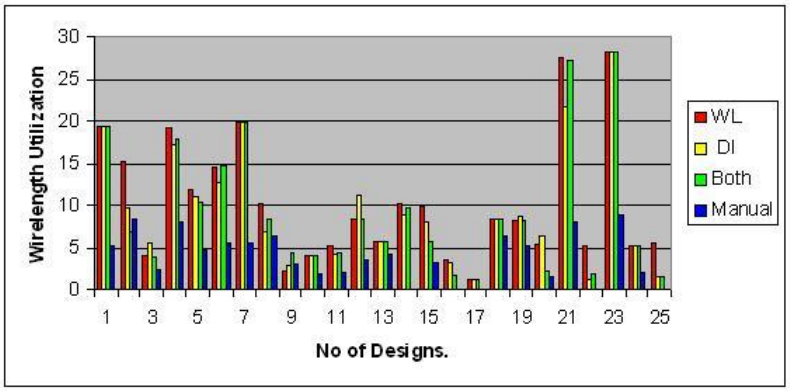

Figure 6.7: comparison of methodologies applied for placer tool

6.4 PSS placer Tool Information

Window executable: pss_placer_1.0.exe

Release Mode size: 2.93 MB

Debug Mode size: 67.8 MB

Linux executable: pss_placer_1.0

Release Mode size: 2.56 MB

Debug Mode size: 50.72 MB

Code size

Directory	Total Lines	Blank Lines	Whole	Embedded	Total Files	SLOC	Type
PSS_Tool	200228	43228	52877	2136	1464	104063	Physical
PSS_Tool	200228	43228	52877	2136	1464	64366	Logical
GUI	55923	14074	10536	1304	391	31313	Physical
GUI	55923	14074	10536	1304	391	21125	Logical
PLACER	144305	29214	42341	832	1073	72750	Physical
PLACER	144305	29214	42341	832	1073	43241	Logical

Table V: Code size of PSS_placer tool

P.S:

Physical Type of SLOC : terminated by a carriage return or EOLN character

Logical Type of SLOC : a line containing two or more source statements

Designs tested: 50

No of correct designs processed: 48

Extra utilities:

- Automated Testing scripts for window(bat) and linux(sh)
- Automated result collector for window and linux(perl)

Chapter 7

Conclusion and Future Work

7.1 Summery

This dissertation has contributed actively from STMicroelectronics in to related research areas: FPGA CAD algorithms and FPGA architecture. Developed PSS placer CAD algorithms and tools this research were described in Chapters 4 and 5, and are briefly summarized in report.

Here, we developed the first publicly-described PiCoGA Based placement tool that can work on different hierarchical level of circuit as per given parameter at command line when invoking executable. We also created a new simulated annealing based placement tool which incorporates three new enhancements over other tools.

First, we implemented a new annealing schedule that adapts automatically to different placement problems, provides good result quality and is more robust than some other tools.

Second, we developed a new, linear congestion placement cost function that enhances the routability of circuits mapped to FPGAs in which different channels have different.

Third, our newly developed direct-interconnect support to placement algorithm adds great advantage to placement and provides really improved results up to 30%

to 40%.

Now we can conclude placement tools developed and presented here that take advantage of the PiCoGA architecture. Placement is particularly difficult for this architecture because of the limited routing resources available. It also shows the modifications required to adapt these placement tools for implementing asynchronous circuits on Montage. These algorithms have implications beyond that of this single architecture.

Our three stage hierarchical placement methodology combines clustering technique with SA. First stage, Clustering technique is used to condense the input network. Produced clusters have similar sizes, which greatly aids the following annealing stage. Second stage, Condensed network is placed using SA. Third Stage uses a low temperature annealing algorithm on the flattened network to optimize the local ordering of the cells. Cells which belonged to different (nearby) clusters may be exchanged.

Overall it is clear that there is a bottleneck in FPGA design for PiCoGA architecture at the placement and routing stage. Attempts to alleviate this bottleneck havent been overly successful. Future work in parallelization of the common algorithms will prove important to accelerating the design process. Much of this parallelization may be achieved in hardware adding to the gains of the parallelization process. Other methods may still need to be developed to take full advantage of programmable logic even at the design stage. With the current state of research, it is clear that even minor improvements in established algorithms, tools and methods are cause for publication, such is the importance of this issue to the design community.

7.2 Future Work

As PiCoGA architecture has become larger and the subsequent architecture that are mapped to them grow to match to many designs, the design process needs to be improved in order to keep pace. If the design process falls behind the growth of the devices, there seems to be a risk that many of the benefits of rapid prototyping on

FPGAs may be eliminated. Parallel computation of the accepted algorithms may prove to be the savior of the design process, allowing faster placement and routing via use of several workstations or perhaps future chip-multiprocessors.

There are two different ways in which one can enhance our CAD tools: by improving the core algorithms to increase result quality, and by increasing the flexibility of the FPGA architecture generator to allow easy investigation of a wider class of FPGAs.

Implementing direction aware support for SA and clustering based placement was our further optimization supportive method we wanted to introduce in PSS placer tool. Here specific architecture of PiCoGA provides us information of direction in which input and output are connected to the relative macro or placement block.

My personal view for extension of FPGA placement for PiCoGA architecture is to introduce machine learning in placement, and in Placement editor. By adding intelligence in placement it can identify similar kind of design and place that circuit on chip very fast and with some extra effort in better way. And intelligence in placement editor helps when manually design are placed and generated less wirelength utilization, It learns and store data and in similar condition it apply such placement method and get better results.

Appendix A

Acronyms

PnR Placement and Routing

RTL Register transfer level

MPGAs Mask programmed gate arrays

eFPGA Embedded fully programmable gate arrays

SoC System on chip

ECL Embedded configuration logic

HPWL Half perimeter wirelength

UML unified modeling language

LUTs Look up Table

ASIC Application specific IC

VHDL verilog hardware description language

DI Direct Interconnects

lex Lexical analyzer

yacc Yet Another Compiler-Compiler

CVS Concurrent Versions System

SQL Structured Query Language

gcc GNU Compiler Collection

DDD Data Display Debugger

FU Functional Unit

Website References

[1] http://msdn.microsoft.com/en-us/library/ms950410.aspx

[2] http://help.eclipse.org/help33/index.jsp

[3] http://www-01.ibm.com/software/awdtools/clearcase/

[4] http://www-01.ibm.com/software/rational/

[5] http://qt.trolltech.com/graphicsview.html

[6] http://www.gnu.org/software/ddd/

References

[1] N. Khanna and E. Team, "Picoga architecture for pss:placer tool," tech. rep., 2006-2008.

[2] V. Betz and J. Rose, "Vpr: Vpack: A new packing, placement and routing tool for fpga research," *Department of Electrical and Computer Engineering, University of Toronto*, 1998-2004.

[3] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," SCIENCE, may 13 1983.

[4] R. Tessier, "Fast place and route approaches for fpgas," *Massachusetts Institute of Technology*, October 1999.

[5] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges*. Foundations and Trends in Electronic Design Automation, 2007.

[6] A. Lodi, A. Cappelli, and C. Mucci, "Reconfigurable computing,picoga architecture," September 2006.

[7] L. Sekanina and P. Mikusek, "Analysis of reconfigurable logic blocks for evolvable digital architectures," in *Brno University of Technology, Bozetechova*, 2001.

[8] A. Karpov, E. Ryzhkov, and C. Shekar, "Extend your application's reach from 32-bit to 64-bit environments,"oop-program verification systems"," *Brno University of Technology, Bozetechova*, June 2001.

[9] J. Malhotra and H. Srivastava, "Cad tool for efpga placement," tech. rep., 2007-2008.

[10] J. Malhotra, H. Srivastava, and C. Modi, "Recent directions in netlist partitioning: A survey," *STMicroelectronics-ECL-TR&D*, October 2009.

[11] S. Mallela and L. Grover, "Clustering based simuhted annealing for standard cell placement," *AT&T Bell Laboratories*, Murray Hill, NJ 07974.

[12] W. Sun and C. Sechen, "Efficient and effective placement for very large circuits," *IEEE Trans. on CAD*, pp. 349–359, March 1995.

[13] V. Betz, "Architecture and cad for speed and area optimization of fpgas," *Department of Electrical and Computer Engineering, University of Toronto*, August 1998.

[14] G. Saucier, D. Brasen, and J. Hiol, "Partitioning with cone structures and clustering for palcement," *CTechnical University of Cluj-Napoca*, January 2002.

[15] M. Lesk and E. Schmidt, *Lex - A Lexical Analyzer Generator*. Bell Laboratories, Murray Hill, NJ 07974.

[16] S. C. Johnson, *Yacc: Yet Another Compiler-Compiler*. Bell Laboratories, Murray Hill, NJ 07974.