INTELLIGENT OPTIMIZATION OF CIRCUIT PLACEMENT ON FPGA

A thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology
In
Electronics & Instrumentation Engineering

Submitted by:

ASHISH KUMAR RAM
Roll No.:107EI016
Electronics & Instrumentation Engg.

SANDIP SAHOO
Roll No.:107EI025
Electronics & Instrumentation Engg.

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING,
NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

INTELLIGENT OPTIMIZATION OF CIRCUIT PLACEMENT ON FPGA

A thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology
In
Electronics & Instrumentation Engineering

Submitted by:

ASHISH KUMAR RAM
Roll No.:107EI016
Electronics & Instrumentation Engg.

SANDIP SAHOO
Roll No.:107EI025
Electronics & Instrumentation Engg.

Under the Guidance of
Prof. D.P.Acharya

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING,
NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

CERTIFICATE

This is to certify that the thesis entitled ―**Intelligent optimization of Circuit placement on FPGA** submitted by **ASHISH KUMAR RAM,** Final year student of Electronics & Instrumentation Engineering, Roll No: 107EI016 and **SANDIP SAHOO**, Final year student of Electronics & Instrumentation Engineering, Roll No: 107EI025 in partial fulfillment of the requirements for the award of B.Tech degree at NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university/institute for the award of any degree or diploma.

Prof. D.P Acharya

Associate Professor
Dept of Electronics and Communication Engg.,
National Institute of Technology Rourkela,
Odisha.

ACKNOWLEDGEMENT

First and foremost, we would like to express my gratitude and indebtedness to **Dr. D.P. Acharya, Associate Professor, Department of Electronics & Communication Engineering, NIT Rourkela** for his kindness in allowing us to work on the particular topic and for his inspiring guidance, constructive criticism and valuable suggestions throughout this project work. We are also sincerely thankful to **Mr. Prakash Rout**, **VLSI LAB, NIT Rourkela** for his continuous support during the course of the project work.

An assemblage of this nature could never have been attempted without reference to and inspiration from the works of others whose details are mentioned in reference section. We acknowledge our indebtedness to all of them. And our sincere thanks to all our friends who have patiently extended all sorts of help for accomplishing this undertaking.

Last but not the least, we wish to reaffirm our abiding affection for our respective parents and loved ones, without whose encouragement this thesis wouldn't be currently in your hands!

ABSTRACT

Field programmable gate arrays (FPGAs) have revolutionized the way digital systems are designed and built over the past decade. With architectures capable of holding tens of millions of logic gates on the horizon and planned integration of configurable logic into system-on-chip platforms, the versatility of programmable devices expected to increase dramatically. Placement is one of the vital steps in mapping a design into FPGA in order to take best advantage of the resources and flexibility provided by it. Here, we propose to test techniques of Placement Optimization on MCNC Benchmark circuits. PSO (Particle Swarm Optimization) has been implemented on circuit netlist with bounding box as cost function. Alternate cost functions were also employed to verify efficiency of optimization. Furthermore, lazy descent was introduced into the algorithm to impede premature convergence. Different values of acceleration and weighing factors were used in the implementation and corresponding convergence results were analyzed.

***Keywords*-** *FPGA Placement; Particle Swarm Optimization; MCNC Benchmarks Circuits; Bounding Box driven Placement.*

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Motivation

In the mid-1980's, field-programmable gate arrays were introduced in the commercial market and thereon, have been produced en masse. The design of digital hardware has undergone rapid changes in the meanwhile. They have become indispensable system parts because of their versatility to execute different logic functions and easy reconfigurablity in step with changing hardware requirements.

The easy availability of logic, routing resources has allowed for the existence of hardware with hundreds of FPGA devices. This has been useful for verifying prototype logic designs and parallel running computing platforms. But a matter of concern remains that the leaps and bounds in hardware prowess are not reflected in the marketed software for mapping designs automatically.

Most users agree that the greatest restriction frequently experienced in the use of contemporary FPGA systems is the average time taken to place and route circuits inside a single board. Typically in FPGAs, the above operations can take hours and in some cases, days compared to tens of minutes on other options. On board implementation isn't necessarily a simple task as conceptual reality is impeded by the above restriction. If the developer is using resources to develop one digital design over several weeks/months, then compilation period in the order of hours is acceptable but if only computing is required of the resources, it is otherwise. In

FPGAs, a bargain has to be reached when compared with ASIC platforms. When set against semi custom platforms, FPGAs can offer lower prices because of mass production. Design mapping is also a moot point. For a fixed size device, placement and routing based on the picked algorithm depends on the efficiency of optimization. Optimality in VLSI is measured in terms of chip size, wire length, delay minimization etc. which have a direct impact on the manufacturing cost, the IC performance and its power consumption.

1.2 FPGA Architecture Basics

Most commercial SRAM-based FPGA architectures have the same basic structure, a two-dimensional array of programmable logic blocks, that can implement a variety of bit-wise logic functions, surrounded by channels of wire segments to interconnect logic block I/O. [1] In most cases, FPGA logic blocks contain one or more programmable lookup tables that can be programmed to perform any Boolean logic function of a small number of inputs (typically 4-5), a small number of simple Boolean logic gates, and one or more flip-flops. User-programmable switches control interconnection between adjacent wire segments and wire segments and logic blocks.

Three main classes of SRAM-based FPGA architecture have evolved over the past decade: *cell-based, hierarchical*, and *island-style*. Each architecture is defined by the amount of logic that can be implemented in an array logic block and the length and interconnection pattern of its channel wire segments.

Cell-based FPGA architectures, such as those available commercially from Atmel Corporation and National Semiconductor, consist of a two-dimensional array of simple logic blocks which typically contain two or three two-input logic structures such as XOR, AND, and NAND gates. Inter-logic block communication is primarily made through direct-wired connections from block outputs to inputs on adjacent logic blocks. Small numbers of wire segments that span multiple logic blocks offer a minimal amount of global communication but typically not enough to implement circuits with randomized communication patterns. These routing restrictions frequently limit the application domain of these devices to circuits with primarily nearest-neighbor connectivity such as bit-serial arithmetic units and regular 2-D filter arrays.

Devices with a *hierarchical* architecture, like those available from Altera Corporation [2], contain a 2-D array of complex logic blocks with many lookup tables and flip-flops (typically 8 or more) per block. Inter-logic block signals are carried on wire segments that span the entire device providing numerous high-speed paths between device I/O and internal logic. This architectural choice leads to an ideal implementation setting for designs with many high-fanout signals. These devices can effectively be used to implement many types of logic circuits exhibiting a variety of interconnection patterns.

Island-style devices provide an architectural compromise between cell-based and hierarchical architectures. As detailed in the next section, island-style devices are characterized by logic blocks of moderate complexity generally containing a small number of lookup tables (typically 2-4) per block. Routing channels with a range of wire segment lengths are available to support both local and global device routing.

1.3 Island-style FPGA Architectures

Perhaps the best known of all FPGA architectures is the Logic Cell Array architecture available from Xilinx Corporation [3]. This island-style architecture contains a square array of logic blocks embedded in a *uniform* mesh of routing resources.

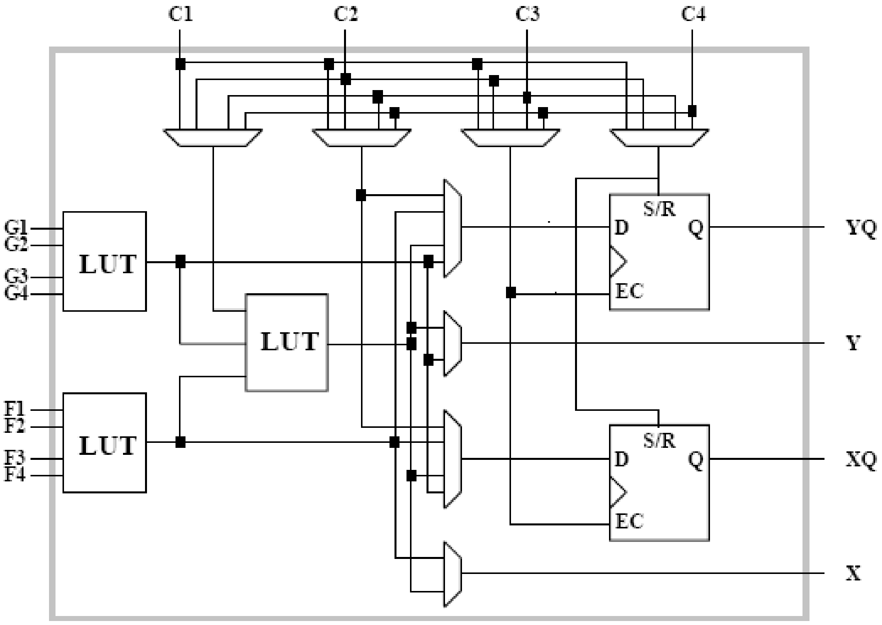


Figure 1-1 *Xilinx XC400 Logic Block*

The logic block of the XC4000 Xilinx device, shown in Figure 1-1, contains three lookup-tables (LUTs), two programmable flip-flops and multiple programmable multiplexers. With this logic block structure, any function of five inputs (with **F** and **G** inputs identical), any two functions of four inputs (**F** and **G** inputs different), and some functions of up to nine inputs can be evaluated. The multiplexers can be used to route combinational results to either **X** or **Y** outputs or to flip-flops. The C inputs provide either a ninth data input for the 3-input LUT or direct inputs to the flip-flops.

As mentioned earlier, island-style routing architectures 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. 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.

Each logic block and adjacent routing segments is considered a routing *cell*. This single cell can be highly optimized in VLSI layout and then replicated both horizontally and vertically to form a uniform array, reducing the design time needed to create a new device family or facilitating the expansion of an existing family to larger logic array sizes. An illustration of the XC4000 routing cell is shown in Figure 1-2.

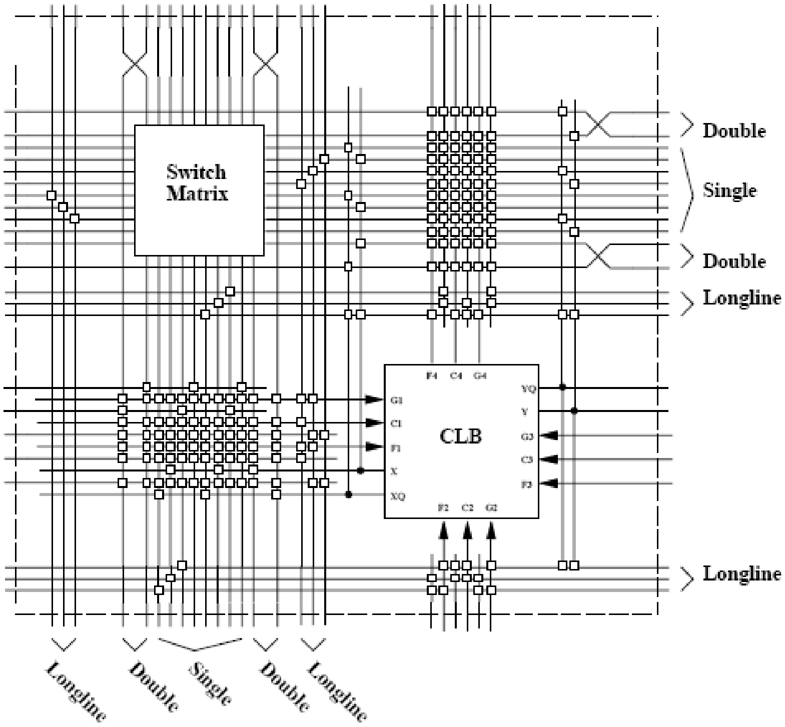


Figure 1-2 *Xilinx XC400 Placement and Routing cell*

Most interconnect in this family is in the form of single-length lines with additional connectivity provided by double-length lines and long lines which span the entire array. The small transparent squares in the figure represent programmable connections to allow for connectivity between intersecting segments or segments and logic blocks. In the next section the interconnection philosophy and physical implementation of segment to segment connectivity and segment to logic connectivity is discussed.

Other commercial segmented devices contain additional interconnect segments that spans four logic blocks (XC4000X [4]) and five logic blocks (Orca 3C [3]) while fitting within the limitation of a single routing cell.

A user algorithm is typically specified in a high-level language (such as C or C++) or in a behavioral hardware description language (VHDL or Verilog).This synthesis step typically requires the *allocation* of datapath hardware resources in the form of high-level blocks such as ALUs, multipliers, and memory components, and the *scheduling* of communication between these components.

2. CAD DESIGN FLOW

2.1 FPGA System CAD Flow

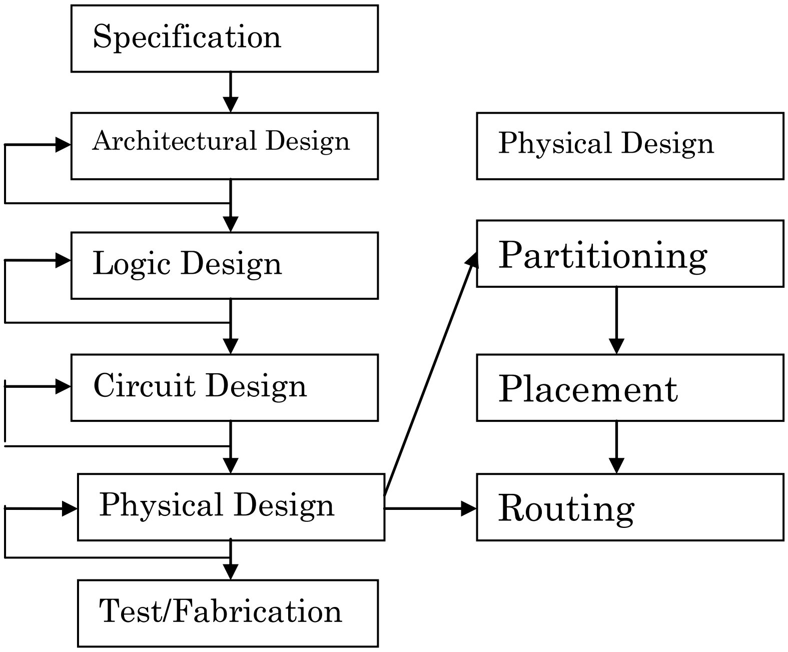


Figure 2-1 *FPGA CAD Design Flow*

The steps by which a specific reconfigurable computing software system performs this translation process, vary somewhat from system to system, but is essentially the following:

2.1.1 Partitioning:

The macro-based netlist created by high-level synthesis must be *partitioned* into smaller netlists for each FPGA device and inter-FPGA signals must be globally *routed* using system-level routing resources. In the partitioning step, the netlist generated by logic optimization is subdivided into pieces of circuitry small enough to meet the logic and inter-chip communication capacities of the target FPGA devices. As part of the partitioning process, each cluster may be assigned to a specific FPGA to guarantee that specific system-level bandwidth requirements are met.

2.1.2 Placement:

After technology mapping, all design logic has been mapped into logic blocks at the quantization level of the basic block of the island-style device. The next step in the translation process is to assign the packed blocks of logic to specific logic block locations in the prefabricated two-dimensional array. The goal of placement for island-style FPGAs is to create a placed configuration of logic blocks that can be successfully interconnected in a subsequent routing step given the routing resources available.

2.1.3 Routing:

The routing phase interconnects specified sets of terminals, i.e., the signal nets of the design, by wiring within routing regions that lie between or over the functional units. (A signal net is a set of the module output terminals and the corresponding module input terminals which need to be connected to each other using routing)

2.1.4 Compaction:

Compaction is usually the final step in the physical layout design of VLSI circuits. It is performed to reduce integrated circuit area while eliminating design rule violations. If a design has initial design rule violations, spacing could actually increase the size of the chip. Spacing minimizes the area of an integrated circuit without changing its topology. It is important to keep the topology constant to preserve the timing and performance optimization of the previous phases. Thus, the symbolic layout is transformed to a mask layout with the goal of minimizing the size of the resulting layout.

We are interested in the optimization of placement as it is a pivotal stage with respect to subsequent routing of components.

2.2 The Placement Problem

Ideally, while allocating placement, it would be desirable to estimate localized routability in each subsection of the target device since failure at any specific point in a subsequent routing step leads to an overall mapping failure. In practice, given the distributed nature of interconnect and the dependencies created by segmentation, this becomes infeasible and the total design wire length of all design nets is used as an evaluation metric for quality of placement and routability. Other widely used metrics are

- Timing
- Congestion
- Power

Almost all island-style FPGA architectures use a variant of the iterative simulated annealing algorithm for placement.

3. PARTICLE SWARM OPTIMIZATION OF FPGA PLACEMENT:

3.1 Particle Swarm Optimization

The optimization problem in hand is attempted using Particle swarm optimization, which is one of the very recent tools introduced by Kennedy (a social psychologist) and Eberhart (an electrical engineer). The initial concept of the swarm intelligence imitates the swarm of birds. It tries to locate the optima based upon the social interaction as well as the individual cognition.

In PSO, a particle is defined to be all any possible solution of the problem in a solution space and we use a group of particles at different locations to find the optimum solution. The movement of the particle is decided by two things: Individual cognition and social interaction.

1. Each particle guides itself to the best possible solution in its neighboring space, also known as *pbest*. *Pbest(previous best)* can be related to the particle's cognition of its own history in finding different results when it moves through the space.

2. Also before taking the next move each particle consider the location of the particle which achieved the best fitness, that's known as *gbest (global best)*

Both the above two factors and the present velocity of the particle affects the velocity In the next iteration. The velocity is added to the present location of the particle to get the next location which will help it move towards the best location(*gbest*), achieved by the swarm, while still looking for an even better location(improving *pbest*).

A comprehensive algorithm of PSO based computer program is given below,

1. Initialize an array of particles with random position x_i and velocity v_i in the solution space.
2. PSO_Search loop starts here:
3. For each particle in the array, evaluate the fitness function, D_i
4. Compare the particle's fitness with its own $pbest_i$. If it is better than the $pbest_i$, replace the $pbest_i$, with the present particle position, else keep the $pbest_i$ unchanged.
5. Identify the particle with the best value of fitness among all the *pbests* and assign it as *gbest*
6. Evaluate the velocity and update the particle position according to the following equations,

   $v_{i+1} = v_i + c_1 * rand_1(\,) * (pbest_i - x_i) + c_2 * rand_2(\,) * (gbest - x_i)$

   $x_{i+1} = x_i + v_i$

where rand ( ) = any random in a range of (0,1) generated each time when the function is evaluated.

c_1 and c_2 are two constants known as acceleration constants.

Velocity is kept in a range of [-v_{max} , v_{max}]

7. Check for the condition for leaving the loop i.e if a sufficiently good fitness is achieved or limited number of iterations are done?
8. End loop

In a modified form of PSO, Shi and Eberhart introduced inertia weights to the velocity equation in order to control the scope of search in an efficient way and to reduce the importance v_{max}. In this new form the modified velocity equation is given by,

$$v_{i+1} = w_i * v_i + c_1 * rand * (pbest_i - x_i) + c_2 * rand * (gbest - x_i)$$

Where w_i is the inertia weight which can be changed in each iteration to control the scope of the search. When w_i>1 the search step is higher and it can go inspect the behavior of different locations in the search. When w_i <1, it performs a rather intensive search operation with a small step size. So it is better to keep the value of w_i initially higher and then gradually reducing it to a small value to perform a rigorous search. We have used the variation of the weight governed by the following equation,

$$w^t = w_{min} + \frac{wt_{max} - t}{wt_{max}} (w_{max} - w_{min})$$

Where t is the iteration number and w_{min} and w_{max} are constants which are start and the end value of the weights.

3.2 Placement, an NP-complete problem

NP class of problems consists of the all the decision problems whose positive solutions can be verified using a a polynomial time on a non-deterministic machine. NP-complete problem means there is no known polynomial time algorithm exists. Approximate solutions for NP-complete problems can be found using heuristic methods.

Implementation of a digital circuit on FPGA involves steps such as placement of the CLBs and routing the interconnections in between the CLBs and I/O pads. As the number of CLBs in an FPGA is very large, and they can be assigned to different parts of the circuits in many possible ways, there is no way to prove that a particular solution is the best solution out of many. This is a NP-complete class of problem and different heuristic methods have been used to solve this problem by different researchers and manufacturer such as Xilinx and Altera. VPR is one of these placement tools, which uses Simulated Annealing technique and has very much become an standard in academia in this field. In our project, we have tried to optimize the MCNC circuit placement using Particle Swarm Technique.

3.3 MCNC benchmark circuits

Microelectronics Center of North Carolina (MCNC) benchmark suite was published for MCNC International Workshop on Logic Synthesis (IWLS), 1991. It included logic synthesis and optimization benchmark sets from ISCAS.85 and ISCAS.89 in addition to some other benchmarks collected from industry and academia. The benchmark suite has standardized libraries with representative circuit designs ranging from simple circuits to advanced circuits obtained from industry. Some of the circuits from list are *e-64, alu4, apex2, apex4* among others.

These benchmark circuits are widely used across industry and academia for comparing different implementation results in various deferring architecture and languages. these circuits are available in many different formats such as *'.net','.blif','.netD','.are'*. The numbers of CLBs vary from circuit to circuit and range from tens to tens of thousands. In this project **.net** formal has been used to extract the information into a matrix format in MATLAB.

3.4 Modeling of the problem

From here onward all the literature has been written for e64 circuit but they are equally valid for any circuit given in '.net' format. This circuit contains 274 CLBs , 65 inputs and 65 outputs (total 130 I/O pads). As it has 274 CLBs, it can be mapped to an 2D array of 17x17 CLBs with I/O pads present along its perimeter. The diagrammatic representation of the model with Cartesian co-ordinate system is shown in Fig. 4.1

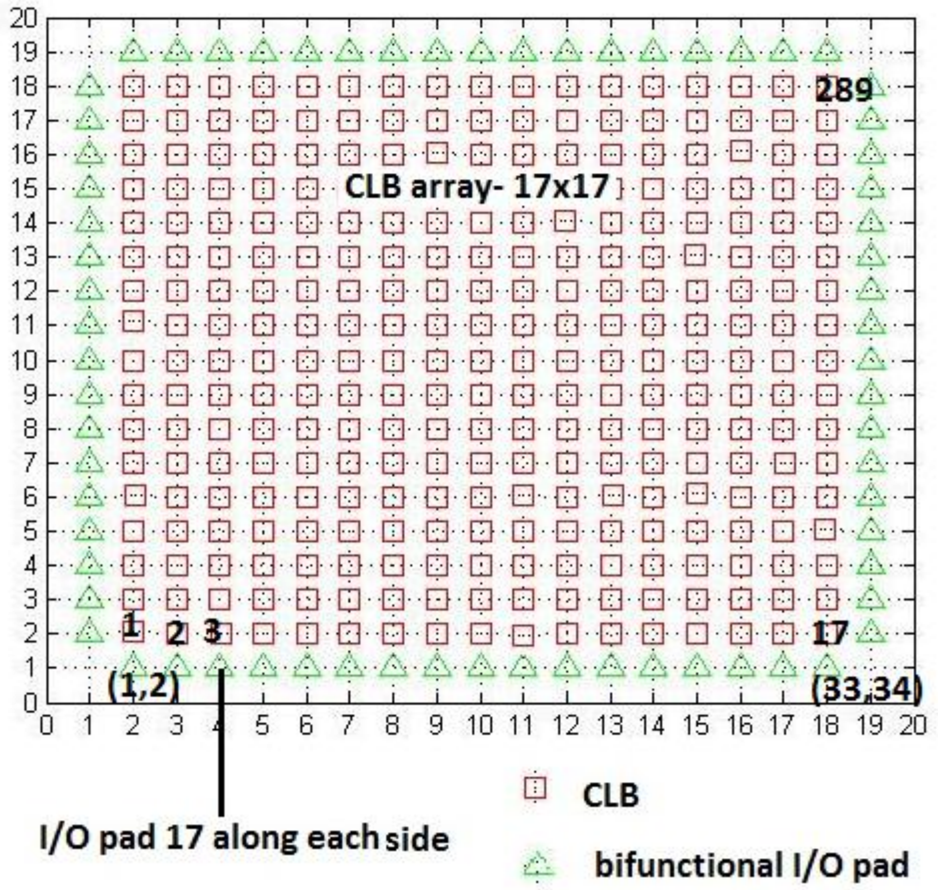


Figure 3-1 *Diagrammatic representation of model in Cartesian system*

CLBs locations were numbered from the bottom-most left CLB which goes to 289 at the top-most right side. Each I/O pad is bi-functional and can be programmed to act as either input or an output pin. To place the e64 circuit, we need to use 274 CLBs out of 289 available and 130 I/O pads out of 136 available I/O locations.

3.5 Population and Particle

In the placement problem we have to deal with placement of CLBs and I/O pads. As the location of both are not interchangeable because they cannot be physically assigned together. Therefore we have the following choices for defining swarm and applying PSO to them for better fitness.

1. *Three separate PSO loops for CLBs , I-pads and O-pads*
2. *Two PSO loops, one for CLBs and other for I/O-pads together*

According to our modeling, we can initialize the particle by assigning array of integers of suitable length. For a particle of CLB swarm, the particle looks like [1 2 6 7 99 64 56…200 288, 15] whose size is 274 for e64 circuit. Similarly for a particle of I/O pad the array assignment might be [1 99 87 40 25 4 125...2 36] of size 130. If it was for I-pads alone, the arrays size would have been 65.

3.6 Cost function

VLSI circuits are generally optimized depending upon the requirement. for instance if speed is of prime importance, then the critical delay path should be shortest and the placement should be optimized accordingly compromising some different parameters such as area, congestion, or wire-length.

For our project we have taken congestion and wire-length as cost function, so the optimized circuits will have minimum congestions and wire-length. Before going for implementation of PSO , the cost functions need to be explained in detail.

3.6.1 Bounding Box[vpr]

The wider routing channels of a FPGA should support those portions of the circuit that have more demands on the wiring resources. The relevant cost function should model the difficulty, albeit relative of routing connections in areas with different channel widths.

$$Cost_{bb} = \sum_{i=1}^{N_{nets}} q(i)\left[\frac{bb_x(i)}{C_{av,x}(i)^{\beta}} + \frac{bb_y(i)}{C_{av,y}(i)^{\beta}}\right]$$

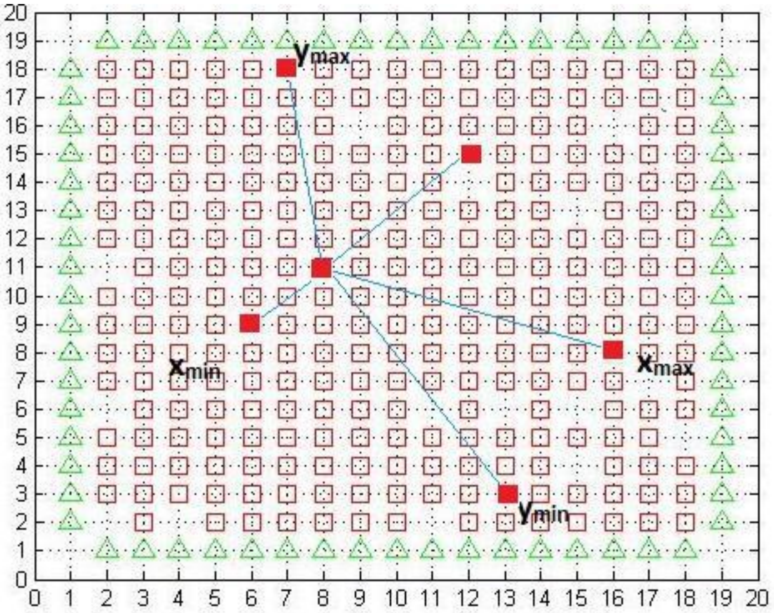


Figure 3-2 *Practical Implementation of Bounding Box function on MATLAB*

The summation is done over $N_{nets,}$ the total number of nets. For each net i, **bb(i)** and **bby(i)** denote the horizontal and vertical spans of its bounding box, respectively. When the number of terminals for a net i exceed one, q(i) makes amends for the underestimation that bounding box model undertakes for connecting tnree or more terminals. The q(i) factor compensates for the fact that the bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals. Its value depends on the number of terminals of net i.

This cost function penalizes placements which require more routing in areas of the FPGA that have narrower channels. The exponent, **ß,** in the cost function allows the relative cost of using narrow and wide channels to be adjusted. **ß =** 1 results in the highest quality placements[vpr].

For a bounding-box function, Cav is a constant, which we have taken as C_{av}= 100.

3.6.2 Wirelength

The straight-blocks distance between the CLBs or between I/O pads or between either of them is also known as the *Manhattan Wirelength.*

Wirelength$= \sum_{I,j}(|x_i-x_j| + |y_i-y_j|)$

3.7 PSO Implementation-In focus

PSO technique was implemented to get an optimized placement of *e64*. Both type of particle definitions, which is already mentioned in the problem modeling, were used to know which can give the best result.

3.7.1 Three separate PSO loops for CLBs, I and O-pads

In this type of particle definition CLBs, I-pads and O-pads have different swarms and their positions were optimized in three separate but consecutive PSO loops. The graphical modeling of the implementation can be understood from the following block diagram in Fig. 5-1..

Swarms of all the three types can be initialized and assigned to array of numbers of suitable length. In *e64*, the arrays assigned to all these three type particles will be like the following,

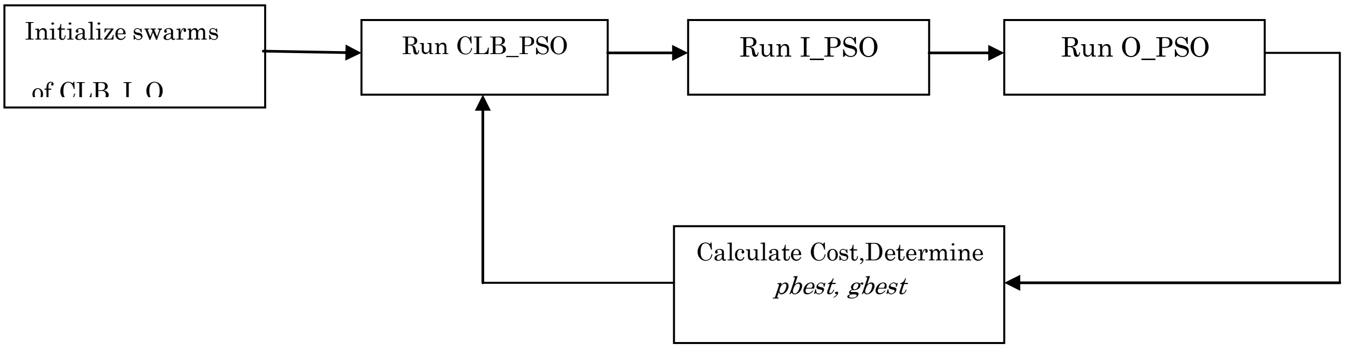


Figure 3-3 *Block Diagram detailing the process flow for the first PSO Implementation*

A particle of CLB population= $[1\ \ 6\ \ 5\ \ 78\ \ 12\ \ 125\ \ 200\ \ldots\ldots145\ \ 56\ ]_{1x274}$ (as there are 274 CLBs in *e64*). For initialization numbers are generated randomly from a set of [1,289] because the size of the array of CLBs for implementing the circuit is taken as 17x17 as mentioned before.

Similarly a particle of I-pad= $[1\ 26\ 34\ 78\ 120\ldots\ldots\ldots11\ 39]_{1x65}$ (65 I-pads in *e64*)

And for an O-pad =$[4\ 5\ 68\ 25\ 23\ \ldots\ldots10\ 121\ 130]_{1x65}$ (65 I-pads in *e64*). For both I and O-pads initial numbers are generated from a set of numbers ranging from [1,136]. This is because all the four sides of the array of CLB has 17 bi-functional I/O pads which gives a total pool of 136 numbers.

Velocity generation:

The velocity equation used in this case is given by,

v1=(w1*v(k,j)+c1*(pbest(k,j)-x1(k,j))*rand+c2*(gbest(k)-x1(k,j))*rand);

In this case velocity is an array of numbers with array length equal to the length of the particle. So, for the particle of CLB, the velocity will be an array of length 274 integers that means each element of the array of the particle will be changed after each iteration.

$$\underbrace{[1\ 65\ 23\ 20\ldots123\ 200\ 6\ 281]}_{\boxed{\text{CLB\_old}}} + \underbrace{[1\ 6\ 3\ 1\ldots1\ 2\ 6\ 9]}_{\boxed{\text{velocity}}} = \underbrace{[2\ 71\ 26\ 21\ldots124\ 202\ 12\ 290]}_{\boxed{\text{CLB\_new}}}$$

Drawbacks:

This method of optimization is erroneous. The sources of error are discussed below:

It is evident from the velocity equation is that the velocity will come as a fraction. Also when the velocity is added to the CLB swarm, there is always a chance that the updated particle will not be a set of unique numbers. This condition in the PSO loop generates an array which might give a lower cost but is impractical because two or more same numbers in the array refers to the same location in the board which is physically impossible.

Another problem arising because of separate treatment of the I and o-pads is that more than 2 I/O were assigned to a single position. In the model taken each I/O location can take maximum of two I/O pads. But as we were taking I and O in separate loops, same number can occur for I and O for more than 2 times. Which is physically not possible. This error can be seen in the simulation results, as there are lots of the I/O pads are remaining unused. This implies that many of the I/O pads are overlapped to a single location.

3.7.2 Two separate PSO loops for CLB and I/O pads

To avoid the problems in the previous method, we considered I/O together and assigned them with an array of size 130. Maximum two numbers in the array can be assigned physically to a single I/O slot on the FPGA board.

IO swarm = [1 2 3 8 9 23 124 …45](1x130)

This modification avoids the allocation of more than 2 I/O pads to a single slot on the board.

The further modification required is to avoid CLB overlapping. This is achieved by making the velocity same for all the individual elements in a single particle.

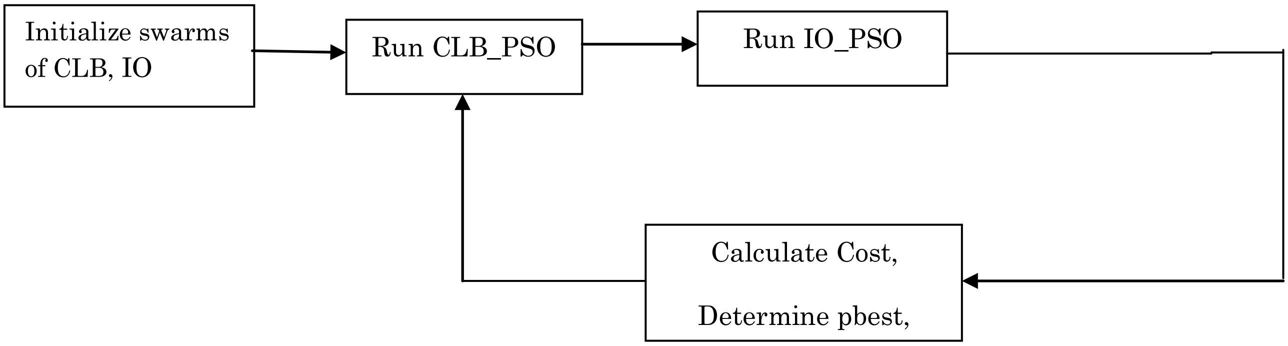


Figure 3-4 *Block Diagram detailing the process flow for the second PSO Implementation*

The further modification required is to avoid CLB overlapping. This is achieved by making the velocity same for all the individual elements in a single particle. The modified velocity equation is given by,

v_c(j)=round(w(i)*v_c(j)+c1*rand*sum(pbest(n_in+n_out+1:n_in+n_out+n_clb,j)-
x3(:,j))/n_clb+c2*sum(gbest(n_in+n_out+1:n_in+n_out+n_clb)-x3(:,j))*rand/n_clb);

But as the velocity is now calculated for a whole particle at the same time, a new problem of overshooting of numbers inside the particle arises. This problem is solved by introducing an overshooting and reallocation scheme which will be dealt in later section.

3.8 Overshooting and Reallocation

When the board positions for CLBs and I/O pads are optimized by iterative positioning through separate PSO loops, a discrepancy is observed- Each swarm particle, i.e. all the CLB and I/O positions as a whole after velocity update consequent of each iteration *overshoots*. A certain number of X positions in excess of the fifteen unoccupied for “e64.net” are indicated as ‘ideal optimized positions’. The same number of positions are rejected from the front of the grid. The movement of the particle elements can be construed to be similar to a cyclic movement whereby the vacant positions ‘move’ through the grid positions, i.e. stored in an array.

CLB € [min, max]

(CLB + V) € [min + V, max + V]

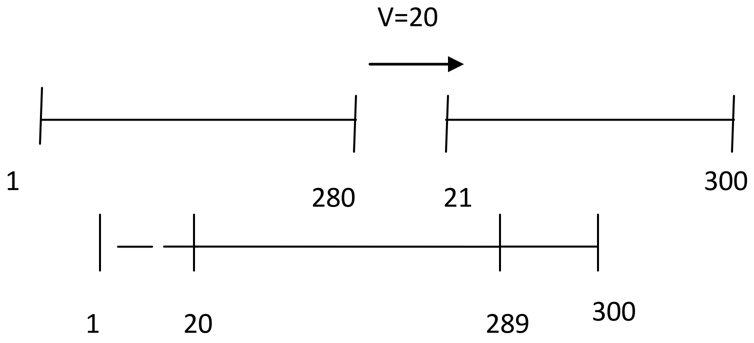
No. of overshoot < V AND No. of rejections > V

For *e64.net* ,

$$X= [\overbrace{\underbrace{1, 8, 9, 14, 15. 16 \ldots 280}_{\text{274 CLB positions}}}]$$

After a particular iteration, V=20 and therefore $X_{new,}$

X_{new} = [21, 28, 29, 34, 35, 36…300]



Solution: 290=289 + 1 x(j) > 289

x(j)= min – (x(j)- 289)

For example, CLB = [1, 8, 9…280]

V= -20;

CLB_{new} = [-19, -12…260]

[-19…-12] → [261,…289] (reallocation)

Now, x(j)= max + 1 – x(j)

If x(j) > 289,

x(j) = min –(x(j) – 289)

If x(j) < 1,

x(j) = max +1 - x(j)

3.9 local minima avoidance

If for a particularly large number of iterations, the cost function remains the same, it is said to have *converged*. If this happens for an unusually small number of iterations, it may be assumed to have prematurely converged. This could be in the case of insufficient number of iterations or in some cases, due to randomness of the program, the optimization terminates at some local minima.

All particles move towards local minima. For avoidance of local minima, slow fall method is used. In this technique, a random velocity is given to each *pbest* and iterated till the value of *pbest* is reduced.

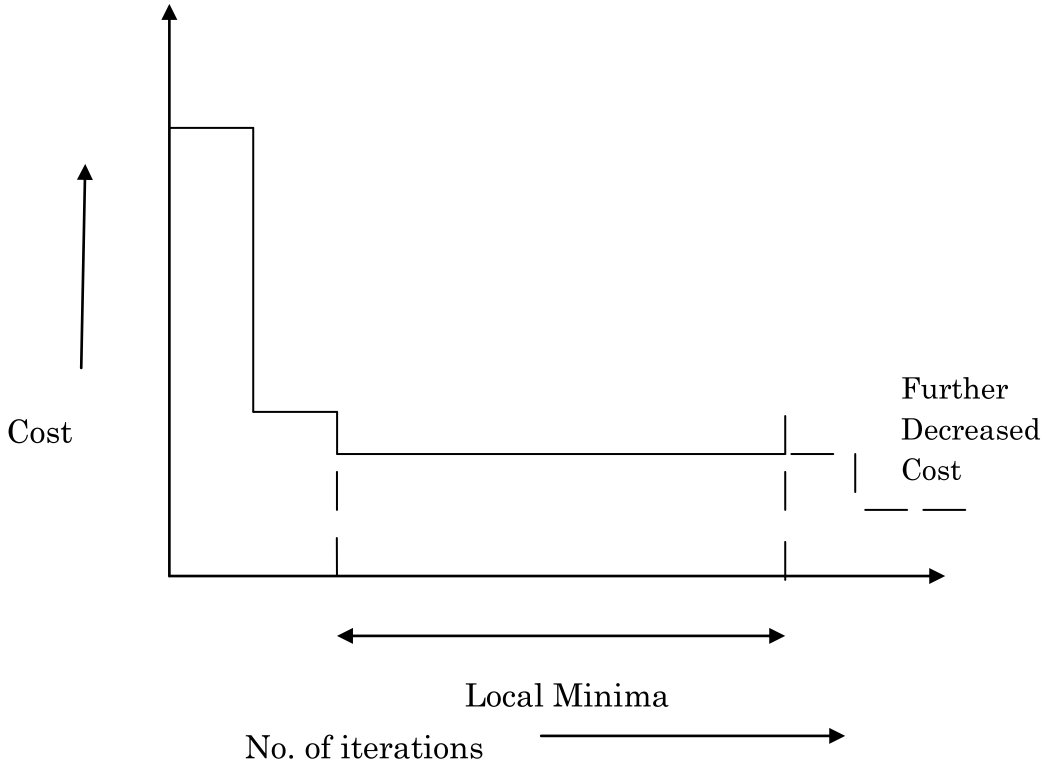


Figure 3-5 *Convergence graph with local minima avoidance*

After we get a reduced pBest, we can replace the previous pBest with the newer one. The random direction of velocity helps the PSO program to avoid the local pBest and explore a lower one to replace it.

The randomness of the *slow fall* velocity comes into picture because the method doesn't generate the velocity depending on the particle's velocity or the swarm's overall performance. Hence, its nature is different from the regular PSO.

After we get a better pBest, we replace it and we also update gBest if required. The number of iterations after which particles go into the slow fall method depends on the total number of iterations in the overall program.

Slow_fall = iter X f

(f) is a fraction € [0, 1]

3.10 Effect of acceleration constants

C_1 and C_2 in the velocity equation are known as acceleration factor. They have a huge effect on rate of convergence. C_1 is one of the co-efficient of the difference between present particle location and its *pbest.* Hence increasing its value will lessen the friction for the movement of the particle in its neighboring space which leads towards randomness to some extent. Decreasing its value ensured a very rigorous exploration of the neighboring space. C_2 is the co-efficient of the difference between the particle position and *gbest.* Increasing or decreasing its value will affect the friction for the particle to move in the whole solution space w.r.t the *gbest*. if the C_2 value is high, particle will try to move faster towards the *gbest.*

There should be a balance between the both constants for a faster convergence. It is also seen that the values should not go beyond 2.5, after which PSO becomes unstable due to very high velocity. Simulations have been done varying the values of C_1 and C_2 in a range of 1 to 2.5 while both increase at the same time and while one increases but the other decreases. And it's found that $C_1 = C_2 = 2.5$ gives the best result.

4. SIMULATION RESULTS AND DISCUSSION

Generating usable Netlist:

MATLAB was used as the platform for optimization of the placement and routing using PSO in FPGA .For implementing the optimization the netlists were needed to be in a matrix format. The netlist available in .net format has the following syntax:

element_type_keyword *blockname*

pinlist: *net_a net_b net_c ...*

subblock: *subblock_name pin_num1 pin_num2 ... # Only needed if a clb*

example:

```
.input i_9_
pinlist: i_9_

.output out:o_3_
pinlist: o_3_

.clb n_n860  # Only LUT used.
pinlist: [1867] [1869] [6474] [6475] n_n860 open
subblock: n_n860 0 1 2 3 4 open
```

A circuit element is created by specifying a keyword at the start of a line, followed by the name to be used to identify this block. The line immediately below this keyword line starts with the identifier *pinlist:* and then lists the names of the nets connected to each pin of the logic block or pad. Input and output pads (*.inputs* and *.outputs*) have only one pin, while logic blocks (*.clbs*) have as many pins as the architecture file used for this run of VPR specifies. The first net listed in the pinlist connects to pin 0 of a clb, and so on. If some pin of a clb is to be left unconnected, the corresponding entry in the pinlist should specify the reserved word *open* instead of a net name.

Subblocks are not necessary for gathering information regarding the interconnection between CLBs, input-pads and output-pads. So, these lines in the original *.net* file can be neglected for generating *.mat* matrix netlist.

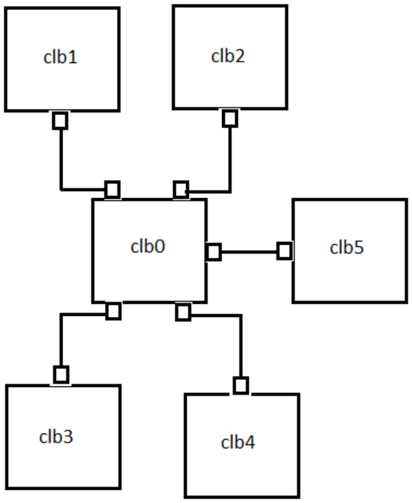


Figure 4-1: An individual CLB and its connections

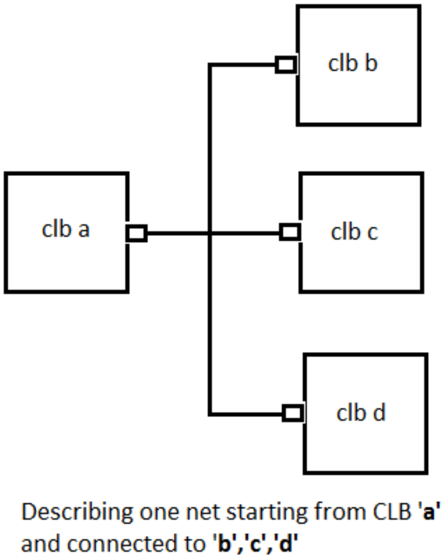

clb b
clb a
clb c
clb d
Describing one net starting from CLB **'a'** and connected to **'b','c','d'**

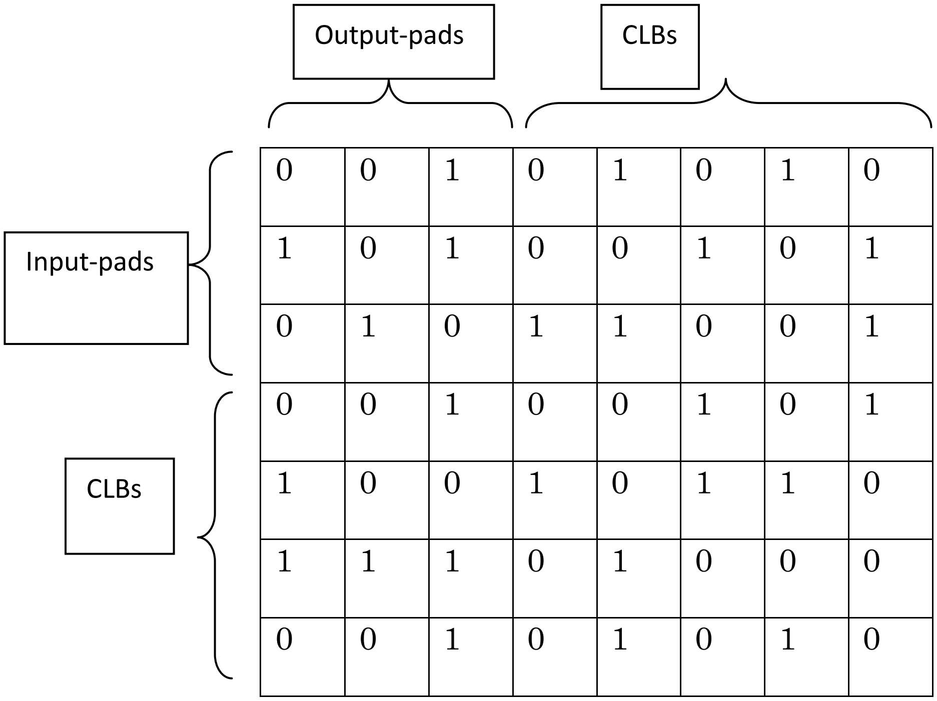
Figure 4-2: An individual net

For example in the following case:

.clb clb0

Pinlist: clb1 clb2 clb3 clb4 clb5 open

A net can be defined by all the connections madefrom a particular pin to the other pins of differentCLBs or output pads which are required to complete the circuit. The net is named after the CLB to whichthe source pin belongs to. Therefore a net can start either from the output pin of a CLB or any input pad and similarly a net can terminate at any of the input pin of a CLB or output pads.The last two connections under the *pinlist* head specify the output pin name and the global clock connection. For the purposes of placement optimization, this information was irrelevant. Therefore in the final matrix, each row represents a net and the columns represent all the possible termination points for a net. An existing connection is marked by ‘1’ otherwise a ‘0’ is written. The following table shows the orientation of the final matrix:



0	0	1	0	1	0	1	0
1	0	1	0	0	1	0	1
0	1	0	1	1	0	0	1
0	0	1	0	0	1	0	1
1	0	0	1	0	1	1	0
1	1	1	0	1	0	0	0
0	0	1	0	1	0	1	0

Figure 4-3: Connection matrix or .mat format extracted from netlist via MATLAB

Results with reallocation technique:

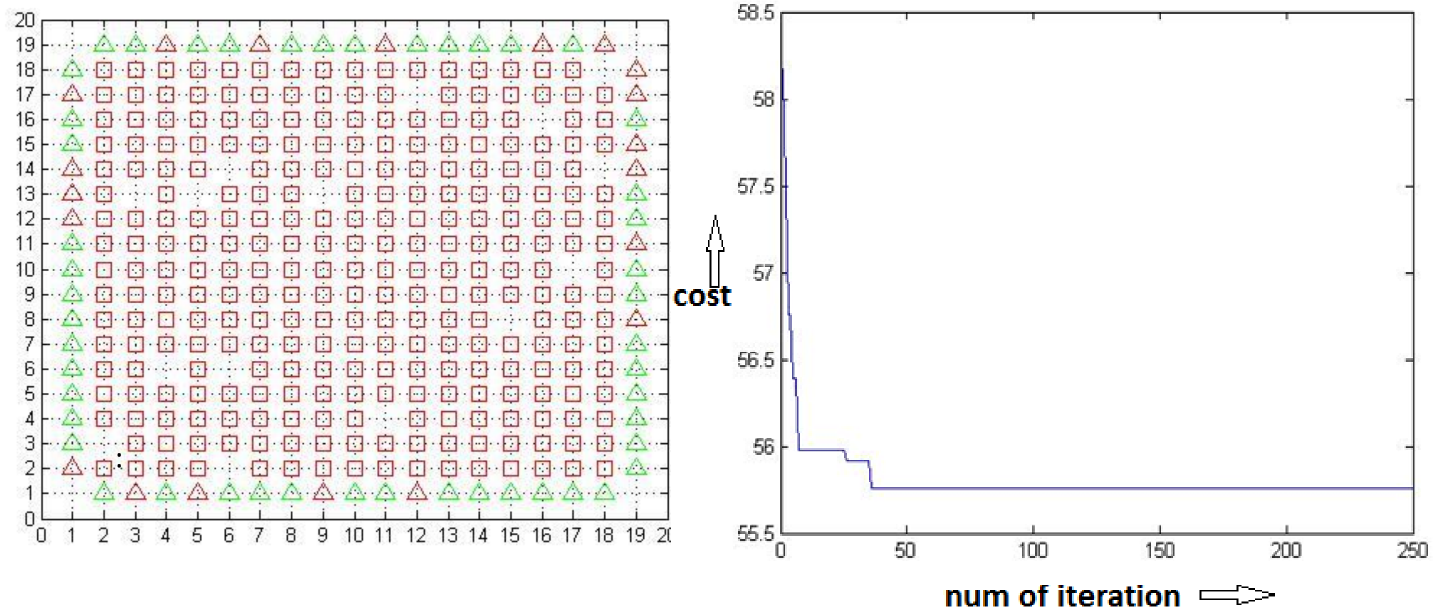


Figure 4-4 Simulation results without minima avoidance

Initial Bounding Box Cost value- 58.5

Converged Bounding Box Cost value - 55.6

Results with local minima avoidance:

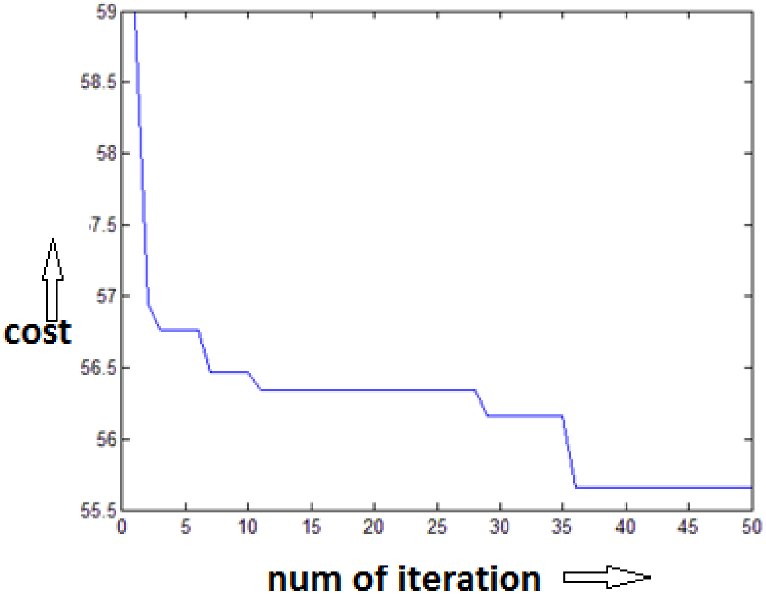


Figure 4-5 *Simulation Results with minima avoidance*

It's observed that with this technique, the convergence is faster and its achieving almost similar result with lesser number of iterations.

Effect of C_1 and C_2:

C1	C2	No. of iteration	cost
1	2.5	50	55.57
1.2	2.3	50	55.6400
1.4	2.1	50	55.2100
1.6	1.9	50	55.4300
1.8	1.7	50	55.4300
2.0	1.5	50	55.38
2.2	1.3	50	55.38
1.1	1.1	50	55.6900
1.3	1.3	50	56.1300
1.5	1.5	50	55.6900
1.7	1.7	50	55.4300
1.9	1.9	50	55.4300
2.1	2.1	50	55.9300
2.3	2.3	50	56.1200
2.5	2.5	50	54.83

We are getting the best result at c1=c2=2.5

5. CONCLUSION & SCOPE FOR FUTURE WORK

- With proper choice of acceleration constants MCNC circuit placements can be optimized using PSO. The use of local minima avoidance increases the rate of convergence.
- Till now only the placement problem has been dealt, after this Routing the interconnection in between the CLBs and I/Os to achieve the desired functionality should be done. Routing is also an NP-Complete class of problem and need to be optimized.
- After the optimization of both placement and routing, the circuit can be implemented on to a read FPGA and tested for the desired result.
- Generally circuits are optimized depending upon the requirement for a specific type of application. But there can also be a trade off between different parameters in the optimization process so that the circuits performance can be further improved.

REFERENCES:

[1] S. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, Boston, Ma, 1992.

[2] *Altera Data Book*. Altera Corporation, 1996.

[3] *The Programmable Logic Data Book*. Xilinx Corporation, 1996.

[4] S. Trimberger. *Field-Programmable Gate Array Technology*. Kluwer Academic Publishers, Boston,Ma, 1994.

[5] Vaughn Betz," Architecture and CAD for Speed and Area Optimization of FPGAs" PhD thesis, Department of Electrical and Computer Engineering University of Toronto

[6]M. Yang, A.E.A. Almaini, L. Wang, "FPGA Placement by using Genetic Algorithm," *EngineerIT*, June 2007.

[7] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proc. IEEE Int. Conf. Neural Network*, 1995, pp. 1942-1948.

[8] Y. Shi and R. Eberhart, "A Modified Particle Swarm Optimizer," in *Proc.IEEE Int. Conf. Evol. Comput.*, 1998, pp. 69-73.

[9] Riccardo Poli,James Kennedy, Tim Blackwell, "Particle Swarm Optimization;An overview" in Springer Science + Business Media, LLC 2007 .10 May 2007

[10] P.K.Rout1, D.P.Acharya , G.Panda3, "Digital Circuit Placement in FPGA based on Efficient Particle Swarm Optimization Techniques", 2010 IEEE

[11] *Yue Zhuo, Hao Li and Saraju P. Mohanty,"* A Congestion Driven Placement Algorithm For FPGA Synthesis*",* 2006 IEEE

[12] Jafar Ememipour, M.Mehdi Seyed Nejad, M. Mehdi Ebadzadeh, Javad Rezanejad, "Introduce a new inertia weight for particle swarm optimization", on 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology

[13] Ednaldo Mariano Vasconcelos de Lima, Dr. Antônio Carlos Cavalcanti and Dr. Lucídio dos Anjos Formiga Cabral, "A New Approach to VPR Tool's FPGA Placement" , Proceedings of the World Congress on Engineering and Computer Science 2007