Abstract – Circuit partitioning is the most critical step in the physical design of various circuits in VLSI design. In this paper, Genetic algorithm for circuit Bi-partitioning has been attempted. The genetic operators can easily be applied in this type of problem. In the partitioning main objective is to minimize the number of cuts. This chapter addresses the problem of partitioning and particular the use of the genetic algorithms for circuit partitioning. The objects to be partitioned in VLSI design are typically logic gates or instances of standard cell. Here the circuits are considered as graph where the nodes represent logic gates or cells and edges represent the connection between these gates or cells. Hence circuit partitioning problem becomes as graph partitioning problem. The algorithm can partition circuit into two sub-circuits. Our method calculates the fitness value and discards solution with low fitness value. The increase in number of crossover point does not necessarily increase the fitness, as the fitness achieved depends on crossover as well as mutation probability. Especially in the paper find minimum cut size.

Keywords – Partitioning, PCB, Genetic Algorithm, Net list, Crossover, Mutation, fitness function, cut size.

I. INTRODUCTION

Efficient designing of any complex system necessitates decomposition of the same into a set of smaller subsystems. Subsequently, each subsystem can be designed independently and simultaneously to speed up the design process. The process of decomposition is called partitioning [1].

A VLSI system is partitioned into several levels due to its complexity. At the highest level, a system is divided into a set of PCBs. This is called system level partitioning. The partitioning of a PCB into chips is called board level partitioning while the partitioning of a chip into smaller sub-circuits is called chip level partitioning.

Due to the limitations of memory space and computation power available, it may not be possible to layout the entire chip in the same step. Therefore, the chip is normally partitioned into sub-chips (called blocks). Each block has terminals located at the periphery that are used to connect the blocks. The connection is specified by a netlist, which is a collection of nets. The net is a set of terminals which have to be made electrically equivalent. Figure 1 (a) shows a circuit, which has been partitioned into three subcircuits. Note that the number of interconnections between any two partitions is four (as shown in Figure 1(b)).

If the circuit assigned to a PCB remains too large to be fabricated as a single unit, it is further partitioned into subcircuits such that each subcircuit can be fabricated as a VLSI chip. However, the layout process can be simplified and expedited by partitioning the circuit assigned to a chip into even smaller subcircuits. The partitioning process of a process into PCBs and a PCB into VLSI chips is physical in nature. That is, this partitioning is mandated by the limitations of fabrication process. In contrast, the partitioning of the circuit on a chip is carried out to reduce the computational complexity arising due to the sheer number of components on the chip. The partitioning is a hierarchical procedure of a computer system.
A. Problem formulation

The partitioning problem can be expressed more naturally in graph theoretic terms. A hypergraph \( G = (V, E) \) representing a partitioning problem can be constructed as follows. Let \( V \) be a set of vertices and \( E \) be a set of hyperedges. Each vertex represents a component. There is a hyperedge joining the vertices whenever the components corresponding to these vertices are to be connected. Thus, each hyperedge is a subset of the vertex set i.e., in other words, each net is represented by a hyperedge. The modeling of partitioning problem into hypergraphs allows us to represent the circuit partitioning problem completely as a hypergraph partitioning problem. The partitioning problem is to partition \( V \) into set of subsets \( V_1, V_2, \ldots, V_k \) such that

\[ V_i \cap V_j = \emptyset \quad i \neq j \]

\[ \bigcup_{i=1}^{k} V_i = V \]

B. Classification of Partitioning Algorithms

The mincut problem is NP-complete, it follows that general partitioning problem is also NP-complete. As a result, variety of heuristic algorithms for partitioning has been developed. Partitioning algorithms can be classified in three ways. The first method of classification depends on availability of initial partitioning. There are two classes of partitioning algorithms under this classification scheme:

- Constructive algorithms: The input to constructive algorithms is the circuit components and the netlist. The output is a set of partitions and the new netlist. Constructive algorithms are typically used to form some initial partitions which can be improved by using other algorithms. In that sense, constructive algorithms are used as preprocessing algorithms for partitioning. They are usually fast, but the partitions generated by these algorithms may be far from optimal.

- Iterative algorithms: Iterative algorithms, on the other hand, accept a set of partitions and the netlist as input and generate an improved set of partitions with the modified netlist. These algorithms iterate continuously until the partitions cannot be improved further.

The partitioning algorithms can also be classified based on the nature of the algorithms. There are two types of algorithms:

- Deterministic algorithms: Deterministic algorithms produce repeatable or deterministic solutions. For example, an algorithm which makes use of deterministic functions, will always generate the same solution for a given problem.

- Probabilistic algorithms: The probabilistic algorithms are capable of producing a different solution for the same problem each time they are used, as they make use of some random functions.

The partitioning algorithms can also be classified on the basis of the process used for partitioning. Thus we have the following categories:

- Group Migration algorithms: The group migration algorithms start with some partitions, usually generated randomly, and then move components between partitions to improve the partitioning. The group migration algorithms are quite efficient. However, the number of partitions has to be specified which is usually not known when the partitioning process starts. In addition, the partitioning of an entire system is a multi-level operation and the evaluation of the partitions obtained by the partitioning depends on the final integration of partitions at all levels, from the basic subcircuits to the whole system. An algorithm used to find a minimum cut at one level may sacrifice the quality of cuts for the following levels. The group migration method is a deterministic method which is often trapped at a local optimum and can not proceed further.

- Simulated Annealing and Evolution based algorithms: The simulated annealing/evolution algorithms carry out the partitioning process by using a cost function, which classifies any feasible solution, and a set of moves, which allows movement from solution to solution. Unlike deterministic algorithms, these algorithms accept moves which may adversely effect the solution. The algorithm starts with a random solution and as it progresses, the proportion of adverse moves decreases. These degenerate moves act as a safeguard against entrapment in local minima. These algorithms are computationally intensive as compared to group migration and other methods.

- Other partitioning algorithms.

Among all the partitioning algorithms, the group migration and simulated annealing or evolution have been the most successful heuristics for partitioning problems. The use of both these types of algorithms is ubiquitous and extensive research has been carried out on them. In this paper we introduce a genetic algorithm
based bi-partitioning algorithm and compare it with a well known group migration algorithm (KL algorithm).

In section II we give some previous work done in this field and in section III we discuss about Genetic Algorithm. In section IV we introduced the proposed method. Section V lists the experimental result. Finally the conclusive remarks are given in section VI.

II. RELATED WORKS

Many approaches have been proposed for the circuit partitioning problem. They include group swapping [3], [4], simulated annealing [2], network flow [5], eigenvector decomposition [6], etc.

Kernighan and Lin proposed a two-way partitioning algorithm with constraints on the subset size. This algorithm randomly starts with two subsets, and pairwise swapping is iteratively applied on all pairs of nodes. Subsequently, many improvements have been made to this method. Schweikert and Kernighan proposed the use of a net cut model so that the algorithm can handle multipin net cases [3].

Fiduccia and Mattheyses improved this algorithm by reducing time complexity to O(P) with respect to the number of pins P, and Krishnamurthy [4] further added in lookahead ability. The Kernighan-Lin based algorithm is quite efficient but it needs a predefined subset size to start with.

Simulated annealing [2] is another method based on iterative improvement. The objective function in simulated annealing is analogous to energy in a physical system, and each move is analogous to changes in the energy of the system. The Metropolis Monte Carlo method is used to decide whether a move is accepted. Simulated annealing usually produces good results at the expense of very long running time.

The maximum-flow-minimum-cut algorithm was presented by Ford and Fulkerson. They transformed the minimum cut problem into the maximum flow problem [5]. In order to separate a pair of nodes into two subsets, the minimum number of crossing edges is equal to the maximum amount of flow from one node to the other. Although this algorithm can find the optimum solution between any pair of nodes in a network, there is no constraint on the sizes of resultant subsets. In practice, the result is not useful if two very unevenly sized subsets are generated.

In eigenvector decomposition [6], connections are represented in a matrix. The eigenvectors of the matrix define the locations of all components and thus derive partitioning results. This method requires the transformation of every multipin net into several two-pin nets in real circuits before establishing the matrix.

Chandrasekhar et al. [7] described a GA for the node partitioning problem (NPP), with applications in VLSI design. This algorithm was used to solve several graph-theoretic problems, including graph colouring, clique cover, and maximal clique, which may be viewed as instances of a general NPP. Three applied problems in VLSI design, which are instances of NPP, were solved using this algorithm. The problems include test scheduling for “built-in self test” (BIST) schemes of VLSI circuits, input encoding of finite state machines or digital logic in control synthesis, and the determination of row and column folding to obtain an optimal area PLA.

Bui and Moon [8] described a hybrid GA for the graph partitioning problem. The algorithm includes a fast local improvement heuristic. One of the features of the algorithm is the schema preprocessing phase that improves the solution space searching capability of the GA.

III. GENETIC ALGORITHM

An overview of genetic algorithm is shown in Figure 2. The number of individuals N is kept constant throughout all generations.

```
Create_initial_population (P_c)
Fitness_calculation (P_c)
P_best = Best_individual (P_c)
For generation = 1 to max_generation
  P_n = P
  For offspring = 1 to max_decedent
    P_a = selection (P_c)
    P_b = selection (P_c)
    P_n = P_n U crossover (P_a, P_b)
  End for
  Fitness_calculation (P_n)
  P_c = Reduction (P_c U P_n)
  Mutation (P_c)
  Fitness_calculation (P_c)
  P_best = Best_individual (P_best U P_c)
End for
```

Figure 2: Outline of the Genetic Algorithm.

The main operators used in the Genetic algorithm are
• Initial population creation
• Evaluation
• Selection
• Crossover
• Mutation
• Replacement

The operators of the genetic algorithm are discussed in the following subsections:

A. Initial population creation

The initial population of candidate solutions is usually generated randomly across the search space. However, domain-specific knowledge or other information can be easily incorporated in the initial population.

B. Evaluation

Once the population is initialized or an offspring population is created, the fitness values of the candidate solutions are evaluated.

C. Selection

Selection allocates more copies of those solutions with higher fitness values and thus imposes the survival-of-the-fittest mechanism on the candidate solutions. The main idea of selection is to prefer better solutions to worse ones, and many selection procedures have been proposed to accomplish this idea, including roulette-wheel selection, stochastic universal selection, ranking selection and tournament selection.

D. Crossover

A crossover operator is used to recombine two strings to get a better string. In crossover operation, recombination process creates different individuals in the successive generations by combining material from two individuals of the previous generation. In the crossover operator, new strings are created by exchanging information among strings of the mating pool. The two strings participating in the crossover operation are known as parent strings and the resulting strings are known as children strings. It is intuitive from this construction that good sub-strings from parent strings can be combined to form a better child string, if an appropriate site is chosen. With a random site, the children strings produced may or may not have a combination of good sub-strings from parent strings, depending on whether or not the crossing site falls in the appropriate place. But this is not a matter of serious concern, because if good strings are created by crossover, there will be more copies of them in the next mating pool generated by crossover. It is clear from this discussion that the act of cross over may be detrimental or beneficial. Thus, in order to preserve some of the good strings that are already present in the mating pool, all strings in the mating pool are not used in crossover.

E. Mutation

Mutation adds new information in a random way to the genetic search process and ultimately helps to avoid getting trapped at local optima. It is an operator that introduces diversity in the population whenever the population tends to become homogeneous due to repeated use of reproduction and crossover operators. Mutation may cause the chromosomes of individuals to be different from those of their parent individuals. Mutation in a way is the process of randomly disturbing genetic information. On the other hand, it might produce a weak individual that will never be selected for further operations.

F. Replacement

The offspring population created by selection, recombination, and mutation replaces the original parental population. Many replacement techniques such as elitist replacement, generation-wise replacement and steady-state replacement methods are used in GAs.

IV. PROPOSED METHODOLOGY

In this paper we introduced a Genetic Algorithm for partitioning. The Group Migration algorithm [1] is, however, quite robust. The complexity of these types of algorithms is considered too high even for moderate size problems. We have done a comparative study of these algorithms and our proposed method on several moderate size problems and find a satisfactory result.

A. Initial population creation

For bi-partitioning a circuit represented as graph, we first create two partition LEFT and RIGHT by using the following algorithm Generate_Partition( ) a in Figure 3. This algorithm randomly generates N numbers between 1 to N, where N be the number of nodes in the circuit. First \( \lfloor N/2 \rfloor \) numbers are stored in LEFT and remaining numbers are stored in RIGHT. Since the LEFT and RIGHT are two disjoint set of nodes we check for repetition of number in the set. The numbers are temporarily stored in an array A [1…N].

To create initial population we call Generate_Partition() several times. Each call to Generate_Partition() randomly creates LEFT and RIGHT
partition. So after $K$ calls to the Generate_Partition( ) algorithm, we get $K$ different solutions for bi-partitioning, which are considered as initial population. For example assume that the initial population of our algorithm is as in Figure 4.

**FIGURE 3: GENERATE_PARTITION( ) ALGORITHM**

```c
Generate_Partition ( )
{
    K=1;
    While (K <= N)
    {
        Repeat = 0
        X = (random ( ) % N) + 1
        For (I =1 to K)
        {
            If (A [I] = = X)
                Repeat = 1
        }
        If (Repeat = = 0)
        {
            A [K] = X
            K = K+ 1
        }
    }
    For (I =1 to [N/2])
    {
        LEFT [I] = A [I]
    }
    J=1
    For (I = [N/2] + 1 to N)
    {
        RIGHT [J] = A [I]
        J = J + 1
    }
}
```

**FIGURE 4: INITIAL POPULATION**

<table>
<thead>
<tr>
<th>Partition</th>
<th>LEFT</th>
<th>RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 3 4</td>
<td>5 6 7 8</td>
</tr>
<tr>
<td>2</td>
<td>1 3 5 6</td>
<td>4 2 7 8</td>
</tr>
<tr>
<td>3</td>
<td>1 7 8 6</td>
<td>4 2 3 5</td>
</tr>
<tr>
<td>4</td>
<td>1 4 5 6</td>
<td>3 2 7 8</td>
</tr>
<tr>
<td>5</td>
<td>1 3 7 6</td>
<td>4 2 5 8</td>
</tr>
</tbody>
</table>

B. Evaluation

Since we want to bi-partition the circuit such a way that the interconnection between these two partitions is minimized, we fixed up fitness function of a partition as the cut size i.e. number of interconnection between two partitions. If we generate a partition with minimum fitness value we get a better partition.

Fitness value of the Partition P i.e. $F (P) = \text{cut size of the partition } P$.

The partition $P$ which is defined by two set of nodes LEFT and RIGHT, the cut size of the partition is calculated as follows:

\[
\text{Cut size} = \text{Number of edges between vertex pair } u \text{ and } v \text{ such that } u \in \text{LEFT and } v \in \text{RIGHT.}
\]

For example if the circuit represented as graph as shown in Figure 5, and the partition $P$ which is defined by two sets LEFT = $\{1, 2, 3, 4\}$ and RIGHT = $\{5, 6, 7, 8\}$ then the cut size is 9. Because there exist edges between $(1, 5), (1, 6), (2, 5), (2, 6), (3, 6), (3, 7), (3, 8), (4, 7)$ and $(4, 8)$.

**FIGURE 5: PARTITION P**

Similarly we calculate the fitness of the all partitions in the initial populations of Figure 4 in Figure 6.

**FIGURE 6: FITNESS OF INITIAL POPULATION OF FIGURE 6.**

<table>
<thead>
<tr>
<th>Partition</th>
<th>LEFT</th>
<th>RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 3 4</td>
<td>5 6 7 8</td>
</tr>
<tr>
<td>2</td>
<td>1 3 5 6</td>
<td>4 2 7 8</td>
</tr>
<tr>
<td>3</td>
<td>1 7 8 6</td>
<td>4 2 3 5</td>
</tr>
<tr>
<td>4</td>
<td>1 4 5 6</td>
<td>3 2 7 8</td>
</tr>
<tr>
<td>5</td>
<td>1 3 7 6</td>
<td>4 2 5 8</td>
</tr>
</tbody>
</table>

C. Selection

After calculating the fitness of all partitions in the initial population we chose one of the partitions randomly from the population. In our example, consider that the
randomly chosen partition is Partition$_2$ as shown in Figure 7.

<table>
<thead>
<tr>
<th>Partition$_2$</th>
<th>LEFT$_2$</th>
<th>1 3 5 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RIGHT$_2$</td>
<td>4 2 7 8</td>
</tr>
</tbody>
</table>

**FIGURE 7: SELECTED PARTITION**

**D. Crossover**

The crossover on the selected partition (P) is done by the following algorithm Crossover( ) in Figure 8.

```plaintext
Crossover( )
{
    Step 1: C = Chose a random number between 1 and \[\lfloor N/2 \rfloor\]
    Step 2: A new Partition P$_{new}$ is created by crossover the LEFT and RIGHT sets of channel P at column C, i.e. the LEFT set of new channel P$_{new}$ is created by taking column 1 to C from LEFT set of P and C+1 to rest from RIGHT set of P. Similarly, RIGHT set of P$_{new}$ is created by taking column 1 to C from RIGHT set of P and C+1 to rest from LEFT set of P.
}
```

**FIGURE 8: CROSSOVER( ) ALGORITHM**

The crossover( ) algorithm is illustrated by the following example on the selected partition of Figure 9.

Step 1: Let C = 2 is chosen a random number between 1 and 4.

Step 2: A new partition P$_{new}$ is created by crossover at column 2 on Partition$_2$ of Figure 9 as shown below:

<table>
<thead>
<tr>
<th>Partition$_2$</th>
<th>LEFT$_2$</th>
<th>1 3 5 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RIGHT$_2$</td>
<td>4 2 7 8</td>
</tr>
</tbody>
</table>

Hence the new partition P$_{new}$ is created as:

<table>
<thead>
<tr>
<th>P$_{new}$</th>
<th>LEFT$_{new}$</th>
<th>1 3 7 8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RIGHT$_{new}$</td>
<td>4 2 5 6</td>
</tr>
</tbody>
</table>

Now, the fitness of the partition P$_{mutation}$ is 1.

**E. Mutation**

In mutation operation we chose a random column between 1 and \[\lfloor N/2 \rfloor\] (\[N/2\] be the maximum column of the channel) and swap the node of LEFT and RIGHT set of this column since it may decrease the cut size.

We illustrate the procedure with the channel P$_{new}$ generated from crossover operation.

Assume that chosen random number between 1 and 4 (here \[\lfloor N/2 \rfloor = 4\]) is 1. Thus new partition (P$_{mutation}$) is generated by swapping the node of LEFT and RIGHT at column 1 of P$_{new}$.

<table>
<thead>
<tr>
<th>P$_{new}$</th>
<th>LEFT$_{new}$</th>
<th>1 3 7 8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RIGHT$_{new}$</td>
<td>4 2 5 6</td>
</tr>
</tbody>
</table>

Now, the fitness function of new solution P$_{new}$ is calculated and $F(P_{new}) = 7$.

**V. RESULT**

We apply this algorithm on different benchmark circuits for bi-partition. In all cases the algorithm is successful to provide solutions with minimum interconnection between two partitions. We compare our algorithm with a well known group migration algorithm i.e. KERNIGHAN-LIN ALGORITHM (KL algorithm) and we find that our algorithm finds solution with less time than KL algorithm. The solution for different benchmark problem is given in Table1.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Nodes</th>
<th>KL algorithm</th>
<th>Proposed algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Initial Cutsize</td>
<td>Final Cutsize</td>
</tr>
<tr>
<td>Actlow</td>
<td>18</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>Moore</td>
<td>21</td>
<td>19</td>
<td>7</td>
</tr>
<tr>
<td>Regfb</td>
<td>25</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>Mealy</td>
<td>37</td>
<td>34</td>
<td>12</td>
</tr>
<tr>
<td>Sequence</td>
<td>49</td>
<td>42</td>
<td>11</td>
</tr>
</tbody>
</table>

**TABLE 1: COMPARISON OF KL ALGORITHM WITH PROPOSED METHOD**

**VI. CONCLUSION**

In this paper we provide a genetic algorithm for circuit bi-partitioning that provide result with less interconnection between partitions in less time. This algorithm can also be extended for multi-way partitioning of the circuit.

**VII. REFERENCES**


