

Hierarchical Partitioning of High-level VHDL Structures *

Tanay Karnik and Sung-Mo Kang
Coordinated Science Lab
University of Illinois
Urbana, IL 61801.

Abstract

A systematic hierarchical partitioning algorithm for high-level VHDL descriptions is proposed. The partitioning algorithm is based on quadrisection. Major contributions of this paper include exploiting VHDL hierarchy for local expansion of VHDL entities and extension of regular quadrisection to three-stage hierarchical operation. At each hierarchical level, the complexity of our algorithm is linear in the number of nets. The results show that our hierarchical algorithm provides solutions better than min-cut and compatible with the flat quadrisection.

Keywords: VHDL, High-level Design, Partitioning, Quadrisection.

1 Introduction

VLSI design automation can be viewed as a process for automatic synthesis of physical layouts from high level specifications. It can be categorized into three levels, high level, logic level and physical level. A behavioral description is accepted at the high level and converted to structural RTL description. Logic circuit is synthesized with primitive gates from this RTL description. At the physical design level, this synthesized circuit is mapped onto the targeted IC chip so as to minimize area, time delay and power consumption. As the complexity of VLSI circuits is increasing, it has become more difficult to handcraft the physical or gate level designs. Thus, numerous methods for the design at physical and logic level have been proposed and established.

The focus of current research in VLSI design automation is geared towards high-level design. The VLSI systems are described using hardware description languages in a top-down fashion. These descriptions can be synthesized and simulated to verify the

circuit functionality. High-level simulation helps the designers take important design decisions earlier in the design process. It reduces the experimentation, tradeoff evaluation and verification time and thus effectively reduces the design cycle time.

VHDL is a standard high-level description language with semantics that support structural as well as behavioral design details. There are several research efforts attempting to understand synthesis, modeling and simulation of VHDL descriptions [1].

We present a method for module partitioning at high level given a VHDL structural circuit description. Partitioning of a high-level circuit description decomposes a large circuit into a set of smaller circuits. Each of the partitioned circuits can be synthesized or simulated very efficiently because of the smaller size. Interconnection delay often causes a bottleneck in VLSI designs. The minimization of interconnection length reduces the interconnect delay and also power consumption. Thus, the important criteria for partitioning are the minimization of the number of I/O terminals of partitioned circuits and the minimization of total interconnection among I/O terminals.

We describe the motivation behind partitioning at the high level. Physical layout parameters such as area and interconnection length can be easily estimated if the initial large circuit is properly partitioned. Various high-level design problems require reliable information on physical layout. Control synthesis process schedules operations in a control data flow graph (CDFG) of the behavioral specification in order to minimize the completion time. An accurate information of time delays in various components in the system will be helpful. Operations assigned to the same control step in control step scheduling can be partitioned effectively if the area estimates of the functional units and their interconnect distance from each other is readily available. Datapath synthesis allocates hardware to operations in the CDFG and an accurate information of physical area of the hardware

*This research was supported in part by the Joint Services Electronics Program (JSEP) under contract N00014-92-J-1270.

units will make the procedure more realistic. The accurate knowledge of the layout parameters will help in balancing the tradeoff between the completion time and the number of functional units. A solution can be to carry out the whole design process up to physical layout and feed the information back to high level for modification. Although this closed loop approach can be very accurate, it is extremely time consuming. In order to make good design decisions at high level, a fast interconnection length estimation technique is much needed.

Block level average interconnection length estimation methods were proposed by various researchers [2, 3, 4, 5, 6, 7]. However, very few methods have been proposed for physical parameter estimation at high level [8, 9, 10]. All previous approaches require some geometrical information of the modules. Probabilistic or statistical approaches assume distributions and estimate the interconnection length as an average for the whole circuit.

Our system accepts a VHDL structural description as an input. It does not require any information on the geometry of the module. We define the size of an entity as the number of transistors needed to implement it in CMOS technology. Our algorithm tries to minimize the size disparity among the blocks (set of entities) at the same level. Hence we assume that all the partitioned blocks have unit aspect ratios and they are mapped onto equal size physical area. As a VHDL structure is described hierarchically in a top-down manner, we exploit the available hierarchy and do not flatten the circuit. The hierarchical partitioning method reduces the CPU time and provides better results especially in case of large circuits. Our approach is based on a well-known partitioning method, viz., quadrisection [11, 2], which does not assume any distributions. The goal is to hierarchically partition a given VHDL structure.

As we are operating at a high level, we expect our system to perform partitioning with very short turnaround time. At each hierarchical level, the complexity of our partitioning method is linear in the number of signals/ports. The hierarchical quadrisection compares well with block-level partitioning methods. In this paper we concentrate on VHDL descriptions and do not provide extensive review of other partitioning approaches.

The major contribution of our work lies in the extension of quadrisection method to hierarchical operation on VHDL structures. Hierarchical partitioning approaches have been proposed before [12, 13, 14, 15, 16], but they operate at block-level circuit specifica-

tion. The ports/signals¹ which are related to entities at lower hierarchical levels are expected to be defined even before the first quadrisection at the root level. In VHDL description, functional and interconnection details of the entities at lower level may not be accessible to the higher superentities enclosing them. New signals/nodes can be added as the partitioning traverses from top to bottom. We use this important information to make our method hierarchical.

Rest of the paper is organized as follows. Section 2 describes our VHDL compiler and conversion of the structural VHDL hierarchy to our quadrisection data structures, referred to as *quad* structures. Section 3 consists of the description of our hierarchical quadrisection approach. Total semiperimeter length estimation is described in section 4. Total interconnection length is estimated by assuming standard cell placement with routing area equal to the cell area. This is followed by results in section 5. Section 6 concludes this paper.

2 VHDL hierarchical structures

Our system accepts VHDL structural description as input. The goal is to partition the whole circuit into smaller sets of entities such that the size variance in all the entity sets and the number of crossing signal/ports² between different entity sets are minimized.

To establish the flexibility of limiting the VHDL scope to our needs and defining intermediate circuit structures, we developed our own VHDL compiler based on IEEE 1076 VHDL grammar and a public domain parser yacc skeleton. Any VHDL description, including behavioral, is parsed and converted to intermediate data structures which we call *VHDL* structures. The conversion leads to a large hierarchical tree structure for each entity and architecture. There is no restriction on the input file. Whenever the system reads an unsupported VHDL code, it prompts the user about it, ignores the error and proceeds with the succeeding code. The *VHDL* structures are read from top-level starting entity to the bottom-level primitive gates and converted to *quad* structures acceptable to our hierarchical quadrisection.

The reader may question the necessity of these VHDL structures. Instead one may wish to parse the VHDL file and create *quad* structures directly. However, in a hierarchical design, the entities at a

¹Signals and ports are referred to as *nets*.

²Henceforth referred to as a *Net*

lower level may not be defined before the higher level parent entity. Whenever a parent cell needs details of the lower hierarchy, it has to wait or delay the connections until the corresponding lower hierarchy is defined. That means the parser conversion needs two passes through the VHDL file. Creation of *VHDL* structures corresponds to the first pass and creation of *quad* structures automatically plays the role of the second pass. Hence the two stage conversion is not an overhead.

For partitioning we can restrict our VHDL scope to structural VHDL descriptions only. Structural description consists of the top-down VHDL entity and architectural declarations describing the interconnections. No timing or control details are involved.

Our *VHDL* structures are exact replica of VHDL statements in data structures. *quad* structures consist of all entities with associated ports/signals and physical size information. An entity may have multiple component instantiations and a different *quad* structure needs to be created for each instantiation. A net may have multiple names at different hierarchical levels. Thus, all the names corresponding to the same net need to be changed to a single net identifier. New nets may be added as we go down the hierarchy. Each new net must have a unique identifier to avoid conflicts in the global circuit visualization. The size of an entity is the number of transistors needed to implement it in CMOS technology. The transistor count is computed in a bottom-up fashion from the bottom primitive gates. A more efficient method utilizing a component library [17] will be implemented in future.

Hierarchical VHDL structural description is now converted to hierarchical *quad* structures consisting of individual data structure for each cell (entity). This data structure includes the information about nets (signal/port) present in that cell and its size (transistor count). We also store the hierarchical level of each cell to simplify interconnection length computation (explained in section 4).

3 Hierarchical quadrisection

In this section we explain hierarchical quadrisection of VHDL structures. As the name suggests, quadrisection means partitioning a given set of cells into four parts. The quadrisection process can be visualized as in Figure 1. The goal of our quadrisection is minimizing the size disparity among four partitions and minimizing the overall interconnections required among different cells at different hierarchical levels. Hierarchical quadrisection involves recursively

quadrisectioning a given set of cells by adding details at each subsequent hierarchical level. In other words, quadrisection the subcells in the root level cell into four parts. Expand the cells in each of the four parts into their subcells and quadrisection each of the four parts into four parts each. Continue the process until the bottom level is reached.

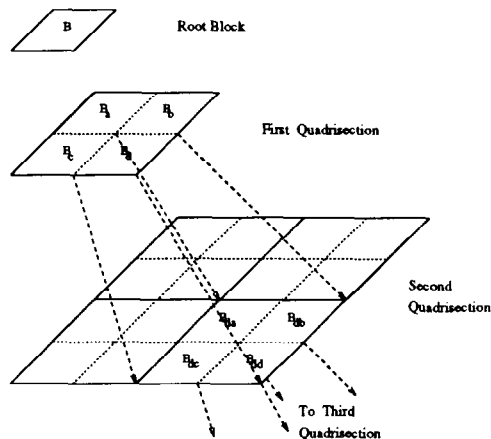


Figure 1: Hierarchical quadrisection

As we have mentioned before, our partitioning procedure takes place at high level and it needs to be a fast procedure. Our hierarchical quadrisection is a simple and reasonably accurate procedure. At each level of quadrisection, the algorithmic complexity is linear in the number of nets.

VHDL structural descriptions have two added benefits for quadrisection. The connectivity information of all the cells is readily available. This information can be expanded to block level partitioning. This minimizes the estimation error incurred by approximate connectivity measures, e.g. the ones based on Rent's rule [18]. As described in section 1, we exploit the hierarchical nature of VHDL instead of flattening the circuit. In VHDL description, functional and interconnection details of the entities at lower level may not be accessible to the higher superentities enclosing them. New signals/nodes can be added as the partitioning traverses from top to bottom. We use this important information and do not burden the higher level quadrisections by the information of all the nets in the circuit. The quadrisection proceeds from a simplified global view to the interconnection intensive primitive bottom level by locally expanding each cell into its subcells. Local expansion and quadrisection reduces the time overhead and provides better results. Structure declaration constructs in VHDL language help realize the local expansion process. We

have described the complexity in terms of the growing size of a block at each hierarchical level pictorially in Figure 1.

The VHDL structural description is converted to *quad* structures, which are directly accepted by any block level partitioning technique. The remaining discussion in the paper is more related to block level hierarchy than VHDL hierarchy. These two descriptions are interchangeable. We first explain some notation used in our hierarchical quadrisection.

Cell: An RTL entity is a cell C . Each cell is characterized by its number of nets n_C and size S_C in the input to the program.

Net: A Net N is a node in RTL. It may have different logical names, but in the input to this program we assume that the same distinct physical name is used for each net. A Net N is characterized by its number of cells α_N and a linked list of pointers to its cells $C_i, i = 1..n_N$.

Block: A block is a group of instantiated entities. A block B is characterized by its size S^B , number of cells n_C^B , number of nets n_N^B , linked list of cell $C_i^B, i = 1..n_C^B$ and linked list of nets $N_i^B, i = 1..n_N^B$.

Block hierarchy: The input hierarchical VHDL file is converted to a hierarchy of blocks. The starting top hierarchical *Cell* is the *rootblock* and it is recursively quadrisectioned up to the bottom level primitive cells. For hierarchical information, each block B maintains its pointers to its parent P^B , and subblocks A^B, B^B, C^B, D^B .

Block quadrisection: Each block is quadrisectioned into four sets of cells. Size of max cell S_{max}^B and individual sizes of subblocks $S_A^B, S_B^B, S_C^B, S_D^B$ are maintained for each block.

Additionally, number of cells connected to each net in each subblock are stored with each net $(\alpha_N^A, \alpha_N^B, \alpha_N^C, \alpha_N^D)$ and index of subblock is stored with each cell.

3.1 Init_partitioning

This is an extremely simple heuristic partitioning stage. Instead of starting with a random initial partition for quadrisection, we quadrisection the block so as to minimize the size disparity among the subblocks.

No interblock relationship is considered at this stage, i.e. each block is partitioned separately. Each

block B is considered with its own C_i^B and N_i^B only. All cells in C_i^B are sorted in a descending order by size. Four top cells in the sorted list are assigned to A^B, B^B, C^B and D^B in order. Then the next four cells are assigned in the reverse order, i.e. to D^B, C^B, B^B and A^B in order. The forward and reverse ordering is alternated until all cells are assigned to the four subblocks.

3.2 Intra_partitioning

In [2], a block consisting of all cells and nets in the design is recursively quadrisectioned until each subblock at the end accommodates only one cell. Initial assignment of cells to quadrisections is random. The algorithm iteratively improves the quadrisection so as to minimize the number of crossing nets. A possible reduction in the crossing nets caused by the movement of a cell from one quadrisection to another is calculated for each cell. There are twelve possible movements of each cell and the reductions are stored with each cell. The best move is implemented, the reductions are updated and the method is repeated. A heap of movements sorted in the order of net reductions is maintained for linear complexity update after each accepted move.

Intra_partitioning is almost the same method as [2]. Our contributions include the extension to hierarchical operation, more general cost function and consideration of size in the procedure. Hierarchical operation does not necessarily reduce the algorithmic complexity, but we have to consider a small set of cells and nets for each hierarchical block and there is an effective time gain because of the reduced size of the problem.

Each block is partitioned separately in this stage also. Each block B is considered with its own C_i^B and N_i^B only.

As this is a known algorithm, we just list our contributions:

- Cost W_N for a net is defined as:

$$W_N^B = f_\alpha \beta_N^B + f_S \sigma^B$$

where

β_N^B is the number of crossing nets in B

σ^B is the standard deviation of the sizes of four subblocks

f_α and f_S coefficients of the linear equation. Variable coefficients will be implemented in future [19].

- Global cost of a partition W^B is:

$$W^B = \sum_{N \in N, B} W_N^B$$

- A move is allowed only if maximum size disparity is less than S_{max}^B .
- Both cost reducing and increasing moves are accepted to avoid local minima. Final sequence of cost increasing moves are undone at the end of the procedure.

3.3 Inter-partitioning

This stage takes into consideration the interblock connectivities. Please refer to Figure 2 to understand the hierarchical relationship.

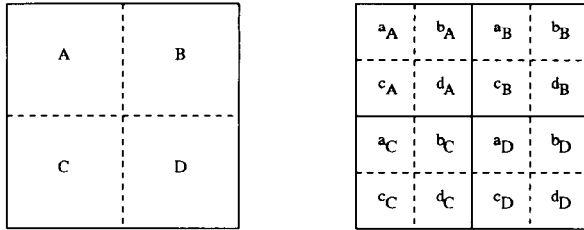


Figure 2: Block hierarchy

Let $P_{N \times K}$ be an array such that:

$$P_{ij} = \begin{cases} 1 & \text{if } i \text{ is present in } j \\ 0 & \text{otherwise} \end{cases}$$

where, N is number of nets, K is cardinality (4) of $\{A, B, C, D\}$, $i = 1..N$ and $j = 1..K$.

Let $B_{N \times k}$ be an array such that:

$$B_{ij} = \begin{cases} 1 & \text{if } i \text{ is present in } j \\ 0 & \text{otherwise} \end{cases}$$

where, N is number of nets, k is cardinality (4) of $\{a, b, c, d\}$, $i = 1..N$ and $j = 1..k$.

Correlation $R_{k \times K}$ is defined as:

$$R_{ij} = \text{Transpose}(B_{li}) \times P_{lj}$$

where, $i = 1..k$, $j = 1..K$ and $l = 1..N$.

Inter-partition cost $T_{k \times p}$ is defined as:

$$T_{ij} = \sum f_{ijl} R_{il}$$

where, p is the number of subblock positions available ($\{a, b, c, d\}$); f_{ijl} is a constant cost multiplier for i^{th}

subblock placed at j^{th} subblock position with respect to l^{th} block sibling; $i = 1..k$, $j = 1..k$ and $l = 1..K$.

T is a $k \times p$ cost matrix, where k corresponds to the block index ($\{a, b, c, d\}$) and p corresponds to a subblock position of the block, which is also one of ($\{a, b, c, d\}$). At the starting, each block is placed at its respective position. So the cost of the partition is just the sum of all diagonal entries in T .

The goal of this partitioning step is to place the blocks at appropriate distinct positions so as to minimize the overall cost of interconnections. Finding such a block-position relation from the cost matrix can be defined as the following problem.

Block-Position Matching Problem: Choose n cost entries from the cost matrix ($n \times n$) such that

- No two entries share a row or a column. For distinct block index and its position.
- The sum of all the chosen entries is minimum in all sets of valid n -tuples. Valid n -tuples are the cost matrix entries satisfying the first restriction.

This problem can be solved using the well known bipartite graph matching or network flow algorithms for an optimal solution. As we are working at a higher level of design abstraction, we do not have the time necessary to find out an optimal solution.

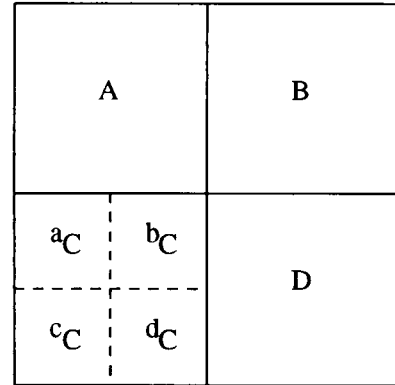


Figure 3: Block hierarchy with partitioned C block

The block hierarchy can be associated with partitioned C -block in Figure 3. As per the definition of inter-partitioning cost, we have the following observations.

To quadrisection the block ($n = 4$), let us assume that the block being inter-partitioned is C . Its subblocks are $\{a_C, b_C, c_C, d_C\}$ and siblings are $\{A, B, D\}$.

This makes b_C an inner subblock, c_C an outer subblock, a_C an edge1 subblock and d_C an edge2 subblock. Cost T_{ij} essentially represents the routing cost

of connections from a subblock to siblings for a subblock i placed at j .

For any block, cost of placing it at inner subblock is less than placing it at edge1 or edge2 subblocks. Also the cost of placing it at outer subblock is greater than placing it at edge1 or edge2 subblocks. For example, for subblock a_C ,

$$T_{aCbC} < T_{aCaC}, T_{aCbC} < T_{aCdC}$$

$$T_{aCcC} > T_{aCaC}, T_{aCcC} > T_{aCdC}$$

This implies,

$$T_{aCbC} < T_{aCcC}$$

Hence placement at inner subblock is the minimum and at the outer subblock is the maximum for all four subblocks for all four positions for any block position. We propose a constant complexity algorithm as follows:

- Choose the maximum outer cost among the outer position row of T . Place the corresponding block at outer location and delete its corresponding row and column from T .
- Choose the minimum inner cost among the inner position row of T . Place the corresponding block at inner location and delete its corresponding row and column from T .
- Now T has been reduced to (2×2) . Just select the cost saving subblocks for edge1 and edge2 positions.

The procedure described above is faster than bipartite graph matching or network flow algorithms, but it may not yield an optimal solution. To counter this suboptimality, we do not change the placement at the end of inter-partitioning if there is no cost saving.

3.4 Stopping criterion

The quadrisection continues from top to bottom VHDL entities. Top is the starting VHDL entity, but bottom is not well defined. Earlier partitioning approaches partition the circuit until every block consists of only one cell. In our approach we consider that *size* of a block is an important issue in partitioning. In VHDL descriptions, the bottom level cells may not be comparable in size. If the partitioning is continued until each block consists of a single cell, e.g. an *inverter* with only 2 transistors, there may be a significant size mismatch, which may lead to wrong semiperimeter wire length estimate (explained later).

To reduce the complexity of the hierarchical algorithm and size discrepancy among the bottom level quadrisections, we stop our algorithm when each subblock consists of less than n_{tr} transistors. We have chosen n_{tr} to be 22, since in our exercise the largest primitive entity is *DFE* which consists of 22 transistors in CMOS implementation.

The overall quadrisection algorithm is as follows:

input: *quad* structures of VHDL entities
output: quadrisectioned block structure

```

Main()
{
  rootblock = top entity
  H_quadrisection(rootblock)
  Return(rootblock)
}

H_quadrisection(block)
{
  Quadrisection(block)
  if (block→leftup)
    Quadrisection(block→leftup)
  if (block→rightup)
    Quadrisection(block→rightup)
  if (block→leftdn)
    Quadrisection(block→leftdn)
  if (block→rightdn)
    Quadrisection(block→rightdn)

  if (block→leftup)
    H_quadrisection(block→leftup)
  if (block→rightup)
    H_quadrisection(block→rightup)
  if (block→leftdn)
    H_quadrisection(block→leftdn)
  if (block→rightdn)
    H_quadrisection(block→rightdn)
  return
}

Quadrisection(block)
{
  Locally expand cells into subcells
  Init_partition(block)
  Intra_partition(block)
  if (block→parent→parent)
    Inter_partition(block)
  return
}

```

3.5 Algorithmic complexity

Init_partitioning is a linear time procedure, linear in the number of cells which is usually very small com-

pared to the number of nets. *Intra-partitioning* algorithm has the same complexity (linear in the number of nets) as regular quadrisection [11]. It can be easily observed that the complexity of the matrix multiplication step in *Inter-partitioning* is linear in the number of nets. All other steps, including our alternate procedure for bipartite graph matching, are performed in constant time. Hence we conclude that the complexity of quadrisection is linear in the number of nets. Linear time procedure conforms the requirement of a fast high-level procedure.

The linear quadrisection procedure is called for each block in the block hierarchy. In practice the number hierarchical blocks is very small as compared to the number of nets in the circuit. The hierarchical partitioning reduces the circuit size in each step. Hence the hierarchical quadrisection is a fast and reasonably accurate procedure to be applicable at the high-level VLSI design.

The final quadrisectioned VHDL hierarchy is now converted to a quad-tree as shown in figure 4. Each node in the tree corresponds to a block, which may be a group of cells or a single cell (bottom level).³ Each node, other than the bottom level, in the tree has four children corresponding to the four quadrisections.

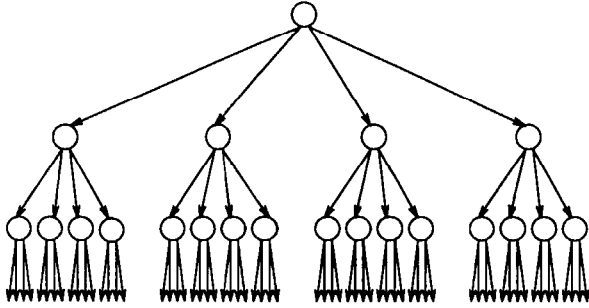


Figure 4: Quad-tree

4 Wire length estimation

The hierarchical design has been hierarchically quadrisectioned. Each cell has the information of its parent, children and subrow-column position in the parent's quadrisection. The position of the cell in the global view can be computed from this information. As we discussed earlier, the hierarchical nature of the

³It should be noted that in VHDL structural description, a cell (entity) may have less than 4 subcells (component instantiations) and the partition[s] may be empty.

partition saves computation time and hence we are not going to flatten the hierarchy.

The VHDL structure was parsed, converted to *quad* structures and then hierarchically quadrisectioned. Block level partitioning approaches estimate the total interconnection length in terms of *total semiperimeter length*. Semiperimeter length for each net is the addition of maximum row and column span in the final partitioned circuit. Total length is the summation of semiperimeter length over all nets. Block level partitioning algorithms aim to minimize *total semiperimeter length*. We discuss the algorithm to estimate *total semiperimeter length* in a hierarchy of quadrisectioned blocks. We repeat that the hierarchical operation reduces the runtime of our system and we do not want to flatten the hierarchy for semiperimeter length estimation.

4.1 Semiperimeter length estimation

As discussed in sections 1 and 2 we assume that all partitioned blocks at the same hierarchical level occupy equal area in the physical layout. A more realistic method utilizing a component library [17] will be implemented in future.

Let L_b be the level of a subblock b in the hierarchy. The topmost 4 quadrisections are at level $L_b = 1$ and the levels increase as the hierarchical quadrisection proceeds towards the bottom. The level is actually a depth measure in this case. We are trying to extract global information from the subblock's local information. The aim is to calculate the global row of the subblock, if the hierarchy were flattened, without actually flattening the hierarchy.

Each quadrisection has a subrow associated with it. Let the subrow be Row_b . Please note that Row_b is either 0 or 1. Let us define:

$$R'_b = 1 + Row_b + \sum_{i=1}^{L_b-1} Row_{Parent^i(b)} 2^i$$

where,

$$Parent^1(b) = b \rightarrow parent$$

$$Parent^2(b) = b \rightarrow parent \rightarrow parent$$

etc.

Now the global row of a subblock can be calculated as:

$$GRow_b = \begin{cases} R'_b & \text{if } L_b = L_{max} \\ (L_{max} - L_b)(2R'_b - 1) + \frac{1}{2} & \text{if } L_b \neq L_{max} \end{cases}$$

The algorithm for total semiperimeter wire length calculation is as follows:

input: Quadrisected block hierarchy
output: Total semiperimeter wire length

```

Total_wire_length(rootblock)
{
  horizontal = 0
  vertical = 0
  for each net in the hierarchy
    initialize mazrow, mazcol
    initialize minrow, mincol
    Wire_length(rootblock)
    horizontal += (mazcol - mincol)
    vertical += (mazrow - minrow)
    if(horizontal < vertical)
      Swap(horizontal, vertical)
    totallength = 2×vertical + horizontal
  return(totallength)
}

Wire_length(block)
{
  if(block→leftup)
    Wire_length(block→leftup)
  if(block→rightup)
    Wire_length(block→rightup)
  if(block→leftdn)
    Wire_length(block→leftdn)
  if(block→rightdn)
    Wire_length(block→rightdn)

  grow = Global_row(block)
  gcol = Global_col(block)

  update mazrow, mazcol, minrow, mincol
  return
}

```

The total semiperimeter wire length calculation algorithm described above is simple to understand except the step where we swap horizontal and vertical lengths. For each net, we compute the horizontal and vertical global span. These spans are added for all the nets to yield total *horizontal* and *vertical* measures. As the input is a square hierarchy of blocks, it can be transposed, i.e. horizontal and vertical dimensions can be swapped. In order to realistically estimate the interconnection length, we must allot some area for interblock routing. We map the quadrisected block hierarchy onto standard cell placement as shown in the figure 5 with routing area height equal to the cell height.

As the routing layer is added between each pair of vertical interrow connections, we multiply the *vertical* measure by 2 and add the result to *horizontal* measure to estimate the total semiperimeter interconnection

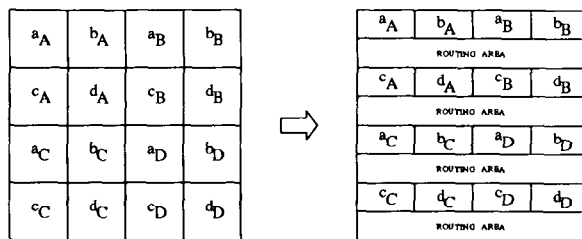


Figure 5: Standard cell placement of quadrisections

length for the VHDL hierarchy.

5 Experimental results

Automatic estimation of transistor count of VHDL structures is based on the assumption that the circuit is described top-down up to the primitive gate level. We assume that the entities corresponding to the gates in Table 1 are primitive and all other entities must be specified in terms of these primitive entities.

Primitive Entity	Nodes	CMOS Trans. Count
NAND	x	$2 \times (x - 1)$
NOR	x	$2 \times (x - 1)$
AND	x	$2 \times (x - 1) + 2$
OR	x	$2 \times (x - 1) + 2$
NOT	2	2
XOR	3	8
XNOR	3	10
DFF	5	22
BUFF	2	4

Table 1: Set of Primitive Gates

As the same entity is instantiated multiply in a single VHDL structure, a new cell must be created for each instantiation. A net at any hierarchical level is expected to maintain a unique name. Multiple instantiations of the same entity lead to multiple logical names for the same physical net. The restriction of unique name needs to be followed in local expansion during quadrisection. The creation of a new *quad* structure for each instantiation of an entity and local expansion to maintain unique node names are performed manually at present. Hence we cannot compare the speed of our algorithm with others.

We provide our partitioning results of two VHDL files. One circuit is the shifter circuit in AMD2901

(*shifter*) and the other is flag unit in GL8085 (*flag*). Both the circuits are quite large and consist of most of the statements provided in VHDL to describe high-level circuit structures. Table 2 provides the details of the circuits. *Entities* are the total component instantiations in the description, *Primitives* are the instantiations of primitive gates and *Levels* is the number of inherent hierarchical levels in the VHDL circuit.

Circuit	Entities	Primitives	Nets	Level
shifter	28	24	36	3
flag	96	94	129	4

Table 2: VHDL Circuits

Table 3 compares the total semiperimeter length, maximum size disparity among the blocks and standard deviation of block sizes estimated by our hierarchical quadrisection (*H-quad*) with regular quadrisection (*Quad*) [2].

These results show that *H-quad* is superior to *Quad*⁴, but it is not true. *H-quad* method tries to minimize size disparity and interconnection length, while *Quad* method only minimizes interconnection length. *H-quad* stops when the size of the blocks fall under 22 while *Quad* stops when each block contains one cell. *Quad* algorithm also considers source and sinks of the signals.

Although no claim can be made on whether our method is provides better results, *H-quad* is a fast partitioning procedure operating hierarchically on hierarchical VHDL structures. It exploits the interconnection and hierarchical information present in VHDL structures and successfully estimates total semiperimeter length in accordance with regular quadrisection. Quadrisection is proven to outperform min-cut partitioning method [11, 2]. Hence we claim that “*quad* provides better results than min-cut partitioning.

6 Conclusions

In this paper we proposed a systematic hierarchical partitioning algorithm for high-level VHDL descriptions. The partitioning algorithm is based on quadrisection. Given VHDL circuit is parsed and converted to hierarchical partitioning data structures.

⁴Standard Deviation for *H-quad* is more than *Quad* for *flag* circuit, because the sizes of primitive gates in the circuit do not have significant variance.

At each level of the hierarchy, VHDL entities are locally expanded and partitioned. The partitioning takes place in three stages, viz. *Init_partitioning*, *Intra_partitioning* and *Inter_partitioning*. The end product of the system is hierarchical quadrisectioned VHDL circuit. We do not expect geometry specification or assume any interconnection length distributions. At each hierarchical level, the complexity of our algorithm is linear in the number of nets. We also provide an algorithm for estimation of semiperimeter length of partitioned blocks. Our results are compared with regular quadrisection. The results show that our hierarchical algorithm provides solutions compatible with the flat quadrisection in much shorter time. The method provides better results than regular min-cut partitioning. Major contributions of this paper include exploiting VHDL hierarchy for local expansion and extension of regular quadrisection to three-stage hierarchical operation.

The cost functions used are heuristic in nature. Coefficients of different variables will be tuned to perfection in future. Size estimation of an entity will be linked to a component library. Hierarchical quadrisection reduces a large VHDL circuit into a set of smaller circuits. The effect of circuit size reduction will be shown to assist high-level synthesis and simulation. The method will also be extended to accept behavioral VHDL descriptions.

References

- [1] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.
- [2] M. Sriram and S. M. Kang, *Physical Design for Multichip Modules*. Kluwer Academic Publishers, 1994.
- [3] W. R. Heller, W. F. Mikhail, and W. E. Donath, “Prediction of Wiring Space Requirements for LSI,” *Design Automation and Fault-Tolerant Computing*, vol. 2, pp. 117–144, May 1978.
- [4] W. E. Donath, “Placement and Average Interconnection Lengths of Computer Logic,” *IEEE Transactions on Circuits and Systems*, vol. CAS-26, pp. 57–61, April 1979.
- [5] A. A. E. Gamal, “Two-Dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits,” *IEEE Transactions on Circuits*

Circuit	Semiperimeter Length		Max Size Disparity		Size Std. Dev.	
	H-quad	Quad	H-quad	Quad	H-quad	Quad
shifter	70	93	2	6	1.0	2.83
flag	659	714	18	20	4.71	4.28

Table 3: Total Semiperimeter Length

- and Systems*, vol. CAS-28, pp. 696–702, February 1981.
- [6] F. Kurdahi and A. C. Parker, “PLEST: A Program for Area Estimation of VLSI Integrated Circuits,” *ACM/IEEE Design Automation Conference*, pp. 467–473, 1986.
- [7] X. Chen and M. L. Bushnell, “A Module Area Estimator for VLSI Layout,” *ACM/IEEE Design Automation Conference*, pp. 54–59, 1988.
- [8] A. C.-H. Wu, V. Chaiyakul, and D. D. Gajski, “Layout-Area Models for High-Level Synthesis,” *International Conference on Computer Aided Design*, pp. 34–37, 1991.
- [9] F. J. Kurdahi and C. Ramachandran, “Evaluating Layout Area Tradeoffs for High Level Applications,” *IEEE Transactions on VLSI Systems*, vol. 1, pp. 46–55, March 1993.
- [10] P. K. Jha and N. D. Dutt, “Rapid Estimation for Parametrized Components in High-Level Synthesis,” *IEEE Transactions on VLSI Systems*, vol. 1, September 1993.
- [11] P. R. Suaris and G. Kedem, “An Algorithm for Quadrisection and Its Application to Standard Cell Placement,” *IEEE Transactions on Circuits and Systems*, vol. 35, pp. 294–303, March 1982.
- [12] K. Ueda, H. Kitazawa, and I. Harada, “CHAMP: Chip Floor Plan for Hierarchical VLSI Layout Design,” *IEEE Transactions on Computer-Aided Design*, vol. CAD-4, pp. 12–22, January 1985.
- [13] Y. Sugai and H. Hirata, “Hierarchical Algorithm for a Partition Problem Using Simulated Annealing: Application to Placement in VLSI Layout,” *International Journal of System Sciences*, vol. 22, pp. 2471–2487, December 1992.
- [14] R. Kling and P. Banerjee, “Empirical and Theoretical Studies of the Simulated Evolution Method Applied to Standard Cell Placement,” *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 1303–1315, October 1991.
- [15] Y.-Y. Yang and C.-M. Kyung, “HALO: An Efficient Global Placement Strategy for Standard Cells,” *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 1024–1031, August 1992.
- [16] T. Hamada, C.-K. Cheng, and P. M. Chau, “An Efficient MultiLevel Placement Technique Using Hierarchical Partitioning,” *IEEE Transactions on Circuits and Systems*, vol. 39, pp. 432–439, June 1992.
- [17] P. K. Jha, N. D. Dutt, and D. D. Gajski, “An Evaluative Study of RT Component Libraries,” Tech. Rep. ECE-93-11, University of California, Irvine, March 1993.
- [18] B. S. Landman and R. L. Russo, “On a Pin Versus Block Relationship For Partitions of Logic Graphs,” *IEEE Transaction on Computers*, vol. C-20, pp. 1469–1479, December 1971.
- [19] D. C. Yeh, S. M. Kang, and V. B. Rao, “CMOS Logic Circuit Partitioning for Equal Chip Complexity,” *International Conference on Computer Design*, pp. 358–360, 1987.