A Proposal for a CLP-Based Multilayer Channel Router

Andrew Hughes
Department of Computer Science
University of Saskatchewan
57 Campus Drive
Saskatoon, SK S7N 5A9
CANADA
email : amh131@cs.usask.ca

Supervisor : Tony Kusalik

Abstract

Channel Routing is an essential part of EDA (Electronic Design Automation). Properly routed channels both minimise the area required for a VLSI circuit and increase the performance of the circuit. It is known that channel routing is a 'hard' problem in that solutions which optimise the above goals are difficult to obtain. Currently employed channel routers restrict the geometry of the channel layout to efficiently complete the routing. This technique fails to exploit the large number of layers available with new VLSI technologies.

Many 'hard' problems can be effectively solved using a Constraint Logic Programming approach. This paper proposes a method to apply this paradigm to the channel routing problem. The method effectively addresses channel optimisation issues as well as taking advantage of an arbitrary number of routing layers. Constraints specific to a technology are shown to be easily imposed.

Introduction

CLP(Constraint Logic Programming) is an effective tool for dealing with computationally difficult problems (Jaffar & Maher 1994),(Fruhwirth *et al.* 1993). Channel routing is known to be a hard problem (Szymanski 1985). Many simplified graph representations of the problem have been shown to be NP-complete. It is a well understood problem though and strongly constrained. Approximate solutions are employed on a routine basis as part of Electronic Design Automation. The first part of this paper shows an effective means of converting a subset of the general channel routing problem into a CLP problem. The solution to this problem should address many of the optimisation issues in channel routing such as channel height, via minimisation and layer exploitation. It should also benefit from CLP's easy extensibility. The second part of the paper examines an implementation of a small section of the proposed CLP problem. A brief overview of both VLSI (specifically the channel routing problem) and CLP are presented as preliminaries.

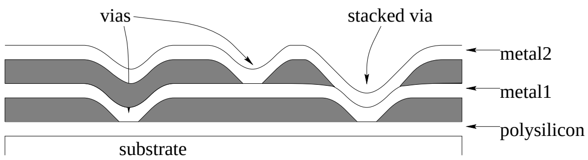

Figure 1: VLSI layers

Problem Definition

Channel Routing

A VLSI chip consists of a sequence of layers placed on a silicon substrate. Each layer has it's own purpose but this study will focus on the *routing layers* – those layers used to electrically connect the transistors in a circuit. These layers are polysilicon, metal1, metal2 and sometimes metal3. The metal layers are created using aluminium and numbered from lowest (nearest to the substrate) to highest. Each layer is separated by a layer of insulator. The best analogy is that of a sandwich made of alternating layers of aluminium foil and plastic film. Figure 1 is a cross section of such a sandwich.

Each layer is patterned into a set of rectangular regions. Where two regions touch an electrical connection between the regions is made. A single region is called a *wire*. If there is a 'hole' – a contact cut – in the insulator, a connection or *via* is made between two adjacent layers. If a connection is required between two non-adjacent layers a *stacked via* is used. See Figure 1

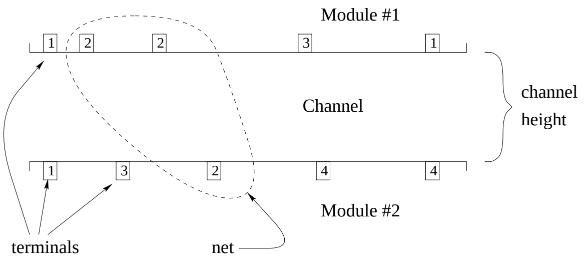

Figure 2: Channel Definition

for an illustration.

There are many technological constraints on a VLSI process, most relating to *feature size*. The minimum size of a feature defines the smallest size that a transistor or wire can have in a process. This dimension increases in the upper layers to ensure reliable connections over the increasingly irregular chip surface. The number of layers available for routing depends on the specific process. Many processes disallow stacked vias.

The large number of gates employed in current designs mandates some form of design automation. Most chips are assembled using *modules* which are themselves constructed from submodules. Modules have *terminals* defined on their boundaries for connection with other modules. The modules are placed on the chip and the regions between them are used to complete the original circuit by connecting the terminals. This process forms *channels* – rectangular regions with terminals on the upper and lower boundaries. A collection of terminals which must be connected is called a *net*. This is illustrated in Figure 2.

The primary goal of *channel routing* is to specify the layout of the channel such that all the nets are connected. Two additional objectives relate to routing area and performance. Often a significant amount of chip area is required for routing. Minimising the height of the channel reduces this 'wasted' area. The delay in transmitting signals degrades the performance of a circuit. This delay increases with both the length of the nets and the number of vias. Vias also decrease the chance that a circuit will be correctly fabricated[1]. Minimising the total wire length and the number of vias is an important routing goal.

CLP requirements

In order for a problem to be solved using a constraint-based approach it must first be expressed as a set of variables with a set of relationships among them (Jaffar & Maher 1994). There are no hard and fast rules to decide how to express a problem, but a good choice is to choose the domain of the variable to match the problem. The domain of a variable can be loosely defined as the range of values which the variable subsumes. Many domains are now available in constraint handling systems, with finite domains and rational numbers being a few examples. The constraints that can be applied to the variables are equally diverse. It is very common to have sets of equations, disequations and inequations. There are also more specialised constraints – requiring a set of variables to be disjoint for example.

There are two main portions to a constraint handling system, the consistency checker and the solver. The consistency checker maintains the set of constraints, ensuring that no constraints can be added to the set which would invalidate it. This can generally be done quite efficiently. A consistent set of constraints does not imply a solution to the problem though. A solution to a problem is obtained through use of the solver. The solver will generally produce a solution with every variable bound to specific value in it's domain. The solvers are often based on search algorithms and have exponential worst case time bounds.

The common thread among all of the systems is that the user is not required to be aware of any of the details. Ideally if the problem can be expressed as a set of constraints then the problem can be solved. In practise a little bit of extra knowledge can greatly improve efficiency.

Motivation

VLSI technology is moving towards more and more layers available for routing. Currently two metal layers are very common and three layers are also reasonably common. There is a need for a channel router designed to use an arbitrary number of layers. In designs where there are three or more layer available for routing, it is common to use cells designed for two layers and use 'over-the-cell' routing. In this style the third (and higher) layers are routed outside the channel over the top of the module. This clearly cannot be done with modules designed to use all of the available routing layers internally.

Current multi-layer routers are specific to the number of layers they are designed for. Often they are extensions of two-layer routers and utilise a 'reserved layer' model.In this model one layer is used for routing the horizontal wires and the other is used for routing vertical wires. Normally there is a single horizontal wire per net with a vertical wire at each terminal. This leads to a routing scheme with one via per terminal as shown in Figure 3. Multi-layer extensions have additional layers devoted to either horizontal or vertical wires. See (Chen & Liu 1984),(Cong, Wong, & Liu

[1] A result of misalignment between the layers

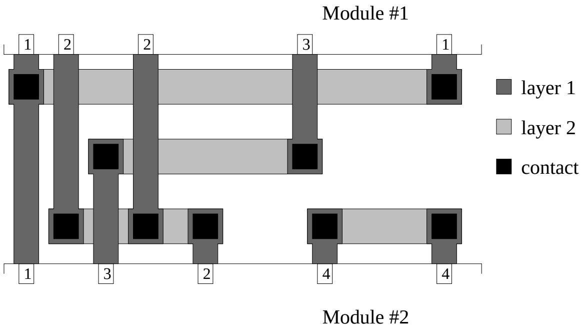

Figure 3: Reserved Layer Routing

1987).

These routing schemes can achieve excellent channel heights, but have a number of disadvantages.

- no via minimisation.
- variable width wires are problematic.
- performance driven routing is difficult.

A good multi-layer router should address all of these issues as well as minimising routing area. It should also be possible to place a tight upper bound on the wire length for a net so that a global router can estimate the wire delay without actually performing the detailed routing.

A CLP approach was chosen for a number of reasons. The problem can be expressed in very generally terms, allowing many of the previously mentioned goals to be explicitly addressed. CLP has an excellent track record for solving problems which are computationally difficult. It is expected that in terms of reusability and extensibility the virtues of a CLP approach should be very apparent.

Decomposition

Overview

The essential issue in solving a large and complex problem like channel routing is to decompose the problem into a series of simpler problems. In this situation the problem is approached in a hierarchical manner, starting with the most general problems and ending with the most specific. There is minimal interaction between the levels of the hierarchy, allowing optimisation to proceed at each level unimpeded by the constraints of the lower levels. The final solution is a solved instance of the most specific level of the hierarchy – in this case a completely routed channel. The actual problem considered is a subset of the channel routing problem – the 'Two Terminal Channel Crossing Problem'. All nets have two terminals one on each

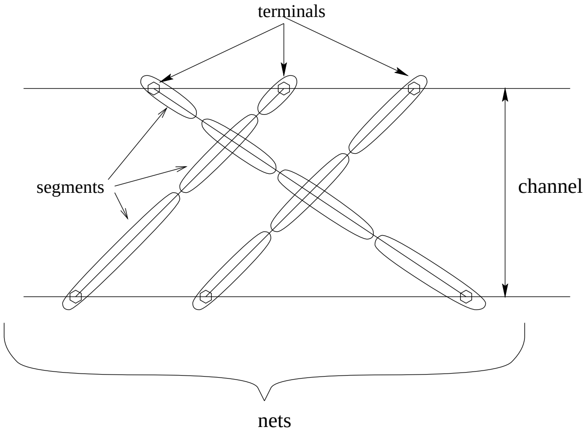

Figure 4: Topological Diagram

side of the channel. This problem is simpler to conceptualise and can be easily extended later.The final strategy chosen is :
allocate layers $\longrightarrow$ define vias $\longrightarrow$ route layers

Layer Allocation

If two nets intersect they must be assigned different layers at some point. As it is rarely possible to assign each net to a distinct layer this means that the nets must be divided into a series of *segments*. The decision on how to divide the nets into segments is defined by the topological model presented in Figure 4.

Each segment is routed entirely on a single layer. If two segments cross in the diagram, they must be routed on different layers. The order in which the segments are arranged along the net is defined from left to right. The segments at the terminal are assigned the layer of that terminal.

Technological restrictions include the number of layers available for routing and stacked vias. If stacked vias are disallowed, the maximum change between two adjacent segments on a net is one layer. Performance based constraints include the number of vias allowed in a single net and the height of the channel. The number of vias in a net is equal to the number of times that a net changes layers while the channel height is minimised by distributing the nets evenly over all of the layers available for routing.

Via Definition

The allocation of layers to the segments of a net implies the existence of vias in the net. If two adjacent segments of a net are allocated to different layers then there must be a via at some point. The physical location of the via must be defined before the channel can complete routing. The vias are placed in the horizontal spaces between the terminals. An example of a

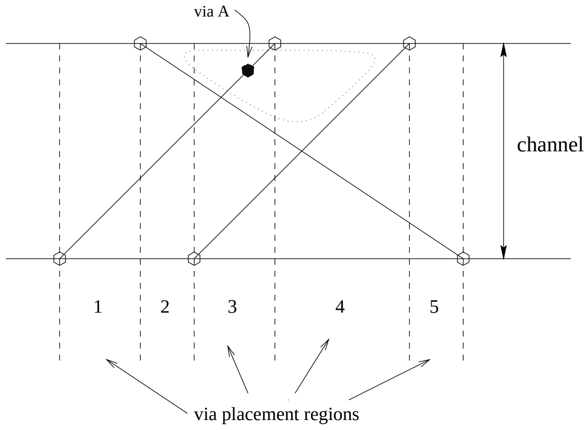

Figure 5: Via Placement

channel divided into *via placement regions* is Figure 5. Each via is assigned to one region, with the ordering of the segments along the nets defining which regions are candidates. In Figure 5 via A can be placed in regions 2-4.

Each via requires a minimum amount of horizontal space in the channel, the actual amount varying with the specific layer and technology. For this reason it is necessary to ensure that no more vias are placed in a region than the region has room for. After the vias have been placed, the number of nets crossing a vertical line drawn at a terminal is fixed. This value determines the minimum height of the channel at that point. The vias should be placed in such a way as to minimise this *congestion*.

When all the vias are assigned to a specific region, the vertical net order on each layer is *invariant*. This means that the order (bottom to top) in which the nets on a layer intersect a vertical line drawn at either a via or a terminal cannot change. This is independent of how the vias move horizontally within the placement region.

Layer Routing

Given the previous definitions and solutions, a constraint satisfaction problem can be imposed to determine the routability of a solution. Each layer can be constrained separately from the others, with only the vias as points of contact between them. Vias will have the same horizontal and vertical location in all the layers that they contact although they will not necessarily have the same physical dimensions on all layers. Stacked vias result in *floating vias* on the layers between their defining layers. These floating vias do not contact any nets but still occupy space on a layer.

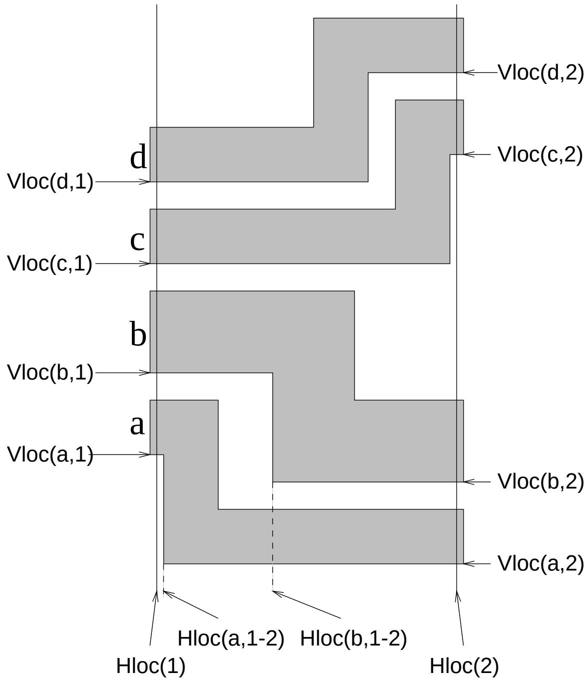

Figure 6: Vertical Routing Region

Single Layer Routing Each terminal or via on a layer has a column associated with it. These columns have a net order defined as well as a horizontal location. The horizontal location of a terminal is the physical location of the terminal while the location of a via is a variable within it's placement region. The space between adjacent columns is used to connect a net from one column to the corresponding net in an adjacent column. This space is the *vertical routing region*. The variables over which the constraints will be applied are the horizontal location of the vias, the vertical location of each net in a column and the horizontal location of the vertical transition for each vertical routing region. Figure 6 illustrates the locations of the various objects in each vertical routing region.

Vertical net order constraints apply directly to the nets in a column. The vertical location of the nets must preserve the defined order for the column. In Figure 6 the constraints for column #1 are :

$$Vloc(d,1) \geq Vloc(c,1) + Width(c)$$

$$Vloc(c,1) \geq Vloc(b,1) + Width(b)$$

$$Vloc(b,1) \geq Vloc(a,1) + Width(a)$$

Horizontal routing constraints define how close together two columns can be, as well as the details of how the nets in two columns are connected. Under the above assumptions, is possible to route the vertical routing region using at most one vertical transition per net. The horizontal constraints apply to the location of the vertical transition. If a net would overlap an adjacent net then the vertical transition must lie at a different horizontal location. The horizontal constraints for nets a and b in Figure 6 are :

$$Vloc(a,2) + Width(a) \geq Vloc(b,1) \Rightarrow$$
$$\text{Hloc(a,1-2)} \geq Hloc(b,1-2) + Width(b)$$

$$Vloc(a,1) + Width(a) \geq Vloc(b,2) \Rightarrow$$
$$\text{Hloc(a,1-2)} \leq Hloc(b,1-2) - Width(a)$$

The $\Rightarrow$ in the above equations is the binary entailment constraint, requiring that if the constraint on the left side of the arrow is true, then the constraint on the right side is true.

If the final system of constraints is consistent and the variables can be assigned to specific values then the channel is *routeable*. The height of the channel is determined by the height of the tallest column on all of the layers, and if the channel cannot be routed in this area, then it cannot be routed with the current definitions at all.

Implementation

The second part of this paper — A router for a single vertical routing region was chosen as a test case to implement. The constraint handling system chosen for the implementation was ECLIPSE, with the finite domains as the domain of choice for the variables. This system has all the usual arithmetic constraints – $\geq, \neq, <$ ... and provides for binary relations between constraints. It allow the constraints on a vertical routing region to be implemented in a very natural way.

Input

The input for a vertical routing region consists of a pair of columns. In an implementation of the entire algorithim, the columns would be derived from information about the vias and terminals on a layer. To simulate this, two columns are provided. Each column is a list of net identifiers – a single unique letter assigned to the net. One net in each column will be associated with either a terminal or a via. A terminal has a specifed horizontal location in the channel so this must also be supplied. For example, the input to create the vertical routing region in Figure 7 is : $C1 = [a, b, (via, c), d], C2 = [(term, a, 10), b, c, d]$. This will specify that column #2 (the right column) is at location 10 – an arbitrary value in this case.

Some additional information is required before any constraints can be imposed. The width of each net and the minimum size of a via are provided with two predicates – $net\_width([(Id, Width), (Id, Width), \ldots])$ and $via\_size(Via_Size)$. These dimensions are scaled to the 'fineness' of solution that the user desires. If a solution accurate to 1/10 of a μmeter is desired, the the width of a 1.5 μmeter wire is 15. The spacing required between wires is included in the width of a wire. In Figure 7 the area in a net above the dotted line is the required inter-wire space.

This notation is consistent with ECLIPSE and allows for a wide range of situations to be tested.

Discussion

The basic strategy behind most constraint programs is generate the constraint set and then search for a solution to the set. This implementation follows this scheme as follows.

```
route(Col1,Col2,(Vloc1,Hloc12,Vloc2)) :-
    vertical_constraints(Col1,Vloc1),
    vertical_constraints(Col2,Volc2),
    horiz_constraints(Vloc1,Vloc2,Hloc12),
    solve(Vloc1,Vloc2,Hloc12).
```

The vertical_constraints predicate takes the previously mentioned representation of a column and converts it into one where each net is assigned a domain variable representing the location of the associated wire. These *vertical location* variables then have constraints imposed requiring them to remain ordered. Each wire only needs to consider the wire immediately below it and must be placed at least the width of that wire above that wires location. For example in Figure 7 on column #1 :

$$Vloc(b,1) \geq Vloc(a,1) + Width(a)$$

A few special cases arise. If the net in question is a terminal, then the vertical location of the wire is fixed – either the top or bottom of the channel. If the net is a via then the width of that net must be adjusted to the maximum of the net's width or the via size. The vertical_constraints also supplies the horizontal location of the column. In the case of terminals this location is fixed, while for vias it is a variable.

The horiz_constraints predicate uses these vertical location variables to construct the vertical transistions. For each wire that crosses the vertical routing region, a vertical transition is needed. This vertical transition must go *somewhere* inside the vertical routing region and a variable is created for this location. These locations are returned by the predicate for purely descriptive purposes – it is only necessary for the constraint set on the vertical transitions to be satisfied, not that they be assigned to a known location. For the purposes of this implementation they are assigned locations, but this is merely to help visualize the solution.

Here as well, each wire only needs to consider the wire immediately adjacent to it. This is the wire that will be immediately to either the left or right of this wire. The constraints ensure that these wires will not

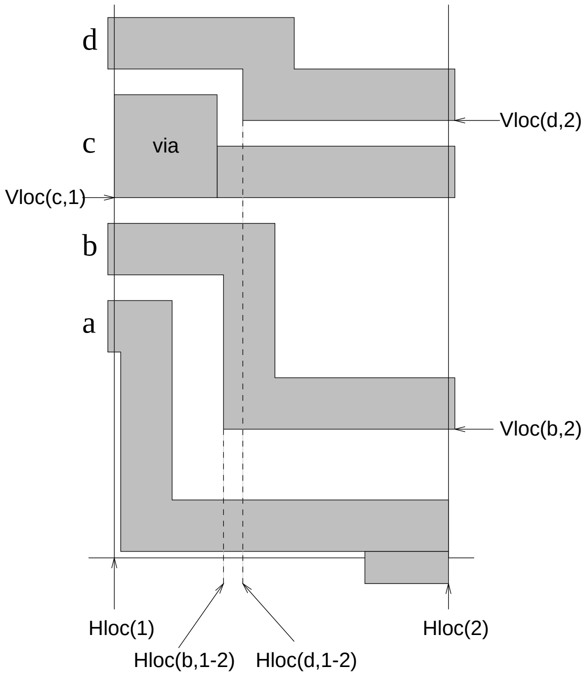

Figure 7: Detailed Constraints

overlap. In Figure 7 the appropriate constraint between nets a and b is :

$$Hloc(b, 1-2) \geq Hloc(a, 1-2) + Width(a)$$

The only special case that can arise is for vias. If a via intrudes on the vertical routing region then constraints are applied to ensure that the vertical transitions do not overlap it. It suffices to constrain the wires directly adjacent to the via. In Figure 7 the additional constraint required for net d is :

$$Hloc(d, 1-2) \geq Hloc(1) + Width(via)$$

The solve predicate attempts to instantiate the variables to a member of their domain - in this case the domain is integers. Additional solutions are produced by backtracking in the solve predicate.

Results

A number of test runs were done, using a broad spectrum of input data. Two examples are presented here.

The first one is a typical run, identical with Figure 7. For this example, the input was as follows :

```
net_width([(a,2),(b,2),(c,2),(d,2)]).
via_size(4).
max_channel_height(20).
route([a,b,(via,c),d)],
      [(term,a,12),b,c,d],
      (Vloc1,Hloc12,Vloc2)).
```

Many solutions were generated on backtracking, but the one corresponding to Figure 7 is :

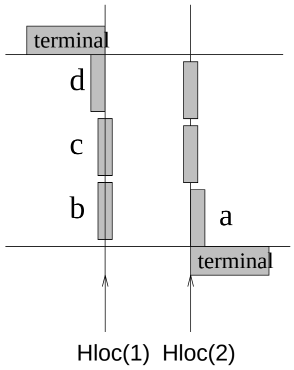

Figure 8: Detailed Constraints

```
Vloc1 = (0,[(a,7),(b,10),(c,13,via),(d,18)]
Vloc2 = (12,[(a,0,term),(b,4),(c,13),(d,16)]
Hloc12 = [(a,0),(b,4),(c,0),(d,5)]
```

The second example is an input for which no successful routing could be found.

```
net_width([(a,2),(b,2),(c,2)]).
via_size(4).
max_channel_height(6).
route([(a,b,(term,c,0)],
      [(term,a,3),b,c],
      (Vloc1,Hloc12,Vloc2)).
```

This corresponds to the situation in Figure 8. Here it is impossible to route the region because there is insufficient room between the terminals to allow for the necessary vertical transitions.

The results of this implementation require some interpretation. The implementation is strictly a test of the ideas presented. In an actual full scale implementation of the algorithim, there would be many vertical routing regions on several layers. This implies that there would be many more constraints and thus fewer solutions.

The goal of the single layer routing is to find *any* solution. If there is no solution possible then the algorithim must backtrack to via allocation or layer assignment.

Conclusions

Theoretical Conclusions

The decomposition is an effective way of dealing with an extremely complex and difficult problem. The hierarchical approach allows the problem to be dealt with at many different levels, from a topological exercise to the nitty-gritty of detailed routing. It also limits the power of the constraint based approach. Ideally a constraint based solution would represent the *entire* problem as a set of variables, impose the appropriate constraints and solve the entire problem simultaneously.

This allows all the optimisation to be done at once and in parallel. This is unfeasible for the current problem primarily because of the extremely large constraint set that would arise $O(n^4)$. It proved very easy to impose technology based constraints such as number of layers or stacked vias.

There is no requirement for the channel to be the same height on all layers. This property allows for over-the-channel-routing to be considered as a subset of the proposed scheme. The over-the-cell layer is enlarged to the size of the allowable region. The nets will be preferentially shunted to that layer to reduce channel congestion. In a very similar manner, variable width channels are also subsumed. The columns are required to fit within the channel region in which they occur. If a new column is added wherever the channel changes width then the nets will route properly.

It should be noticed that this is a true multi-layer router – to increase the number of layers requires no change in the program. The number of layers is increased and the program is run. The benefits are immediate. Variable width nets are an integral feature as well as is via minimisation. The nets are always routed horizontally between the two defining terminals so a well defined limit can be placed on the maximum wire length. The number of vias required is dependent on the geometry of the particular case at hand, rather that being fixed by fiat. If many layers are added it seems likely most nets will be routeable on a single layer, with vias used only if two terminals are on different layers. The algorithm makes no claims to optimality but offers the potential for channel routing that scales with technological advances.

Extensions

There are many significant extensions required to this router. Most importantly it needs to be generalised to handle multiterminal nets. Most nets in a practical design do *not* conform to the strictures of the 'Two Terminal Channel Crossing Problem'. This can be handled in a simple manner by extending the router to handle two terminal nets with both terminals on the same side of the channel. Then multiterminal nets are decomposed into a set of two terminal nets with a spanning tree. Floating wires on the ends of the channel also need to be accommodated. Most global routing schemes route wires into and out of a channel at the ends. Often these wires don't connect to any terminal in the channel. The basic problem here is to find a way of incorporating them into topological model in a way that make sense. The final extension is to find a way to eliminate the entire decomposition hierarchy so that the problem can be solved simultaneously on all fronts.

References

Braun, D.; Burns, J.; Devadas, S.; et al. 1986. Chameleon: A new multi-layer channel router. In *Proceedings of the 23rd Design Automation Conference*, 495–502.

Chen, H. H., and Kuh, E. 1986. Glitter: A gridless variable-width channel router. *IEEE Transactions on Computer-Aided Design* 5(4):459–465.

Chen, Y., and Liu, M. 1984. Three-layer channel routing. *IEEE Transactions on Computer-Aided Design* 3(2):156–163.

Cong, J.; Wong, D. F.; and Liu, C. L. 1987. A new approach to the three-layer channel routing problem. In *IEEE International Conference on Computer-Aided Design*, 378–381.

Enbody, R. J., and Du, H. C. 1986. Near-optimal n-layer channel routing. In *Proceedings of the 23rd Design Automation Conference*, 708–714.

Fruhwirth, T.; Herold, A.; Kuchenhoff, V.; Provost, T. L.; Lim, P.; Monfroy, E.; and Wallace, M. 1993. Constraint logic programming – an informal introduction. Technical Report 5, ECRC, 17 D-8000 Munchen 81, Germany.

Hambrusch, S. E. 1985. Channel routing algorithms for overlap models. *IEEE Transactions on Computer-Aided Design* 4(1):23–30.

Horbst, E., ed. 1986. *Logic design and simulation*, volume 2 of *Advances in CAD for VLSI*. New York, N.Y., U.S.A: Elsevier Science Pub. Co.

Hsu, C. P. 1983. General river routing algorithim. In *Proceedings of 20th Design Automation Conference*, 578–583.

Jaffar, J., and Maher, M. 1994. Constraint logic programming : A survey. *The Journal of Logic Programming* 19(20):503–581.

Leiserson, C., and Pinter, R. Y. 1983. Optimal placement for river routing. *SIAM Journal of Computing* 12(3):447–462.

Lienig, J., and Thulasiraman, K. 1993. A genetic algorithm for channel routing in vlsi circuits. *Evolutionary Computation* 1(4):293–311.

Reed, J.; Sangiovanni-Vincentelli, A.; and Santamauro, M. 1985. A new symbolic channel router: Yacr2. *IEEE Transactions on Computer-Aided Design* 4(3):208–219.

Sherwani, N. 1993. *Algorithms for VLSI Physical Design Automation.* Assinipi Park, USA: Kluwer Academic Publishers.

Szymanski, T. G. 1985. Dogleg channel routing is np-complete. *IEEE Transactions on Computer-Aided Design* (4):31–41.

Taylor, G., and Russell, G., eds. 1992. *Algorithmic and knowledge based CAD for VLSI.* London, U.K: P. Peregrinus on behalf of the Institution of Electrical Engineers.

Yoshimura, T., and Kuh, E. S. 1982. Efficient algorithms for channel routing. *IEEE Transactions on Computer-Aided Design* 1(1):25–30.