TCG-Based Multi-Bend Bus Driven Floorplanning

Tilen Ma

Department of CSE
The Chinese University of Hong Kong
Shatin, N.T. Hong Kong

Evangeline F.Y. Young

Department of CSE
The Chinese University of Hong Kong
Shain, N.T., Hong Kong
Tel: 852-26098401
Fax: 852-26035024
e-mail fyyoung@cse.cuhk.edu.hk

[1] **Abstract—In this paper, the problem of bus driven floorplanning is addressed. Given a set of modules and bus specifications, a floorplan solution including the bus routes will be generated with the floorplan area and total bus area minimized. Some previous works have addressed this problem with restricted bus shapes of 0-bend, 1-bend or 2-bend [1]. However, in this paper, we address this bus driven floorplanning without any limitations on the shapes of the buses. We solve this problem by a simulated annealing based floorplanner using the Transitive Closure Graph (TCG) representation [6]. Experimental results show that we can improve over [1] significantly in terms of both run time and quality, since there are more flexibilities in routing the buses and complex shape validataion steps are not needed. For data sets with buses connecting a large number of blocks, our approach can still generate high quality solutions effectively, while the approach [1] of restricting to 2-bend buses often cannot give any feasible solutions.**

I. Introduction

As technology advances, the amount of interconnections between different modules on a chip increases rapidly. Bus routing has become more and more important. Bus driven floorplanning considers bus placement. The objective of the problem is to obtain a bus-routable floorplan such that the area of the chip and the total area of the buses are minimized. In [4], the authors proposed a unified method to handle simultaneously different kinds of placement constraints, including alignment and abutment. This approach is not suitable for bus driven floorplanning neither as for a bus, the order in which the blocks are passed by the bus is not fixed.

In [3], the authors made use of the idea from [5] and designed an intact algorithm to solve the bus driven floorplanning problem based on a simulated annealing framework. Each candidate floorplanning solution is checked by an evaluation step to see if the buses are feasible, i.e., the required set of blocks can be passed through by a 0-bend bus. Chen and Chang [2] also addressed this bus driven floorplanning problem based on the B*-tree representation. One major drawback of these two approaches is that, only horizontal and vertical buses are considered and the solution quality will deteriorate when the number of blocks involved in each bus is large. Another previous work [1] improved over them by allowing 0-bend, 1-bend, and 2-bend buses.

It is still very restrictive to allow only 0-bend, 1-bend and 2-bend buses. There is no reasons to impose such restriction except for reducing the number of vias used since vias have adverse effects on delay, area and circuit reliability. However, if all bendings occur at the blocks on the bus net, no extra vias is required since a via exists anyway to connect to each block on the net. If a bending occurs somewhere rather than at the modules on the net, an extra via is required to connect the horizontal and vertical bus components.

Therefore, in this paper, we try to address this bus driven floorplanning problem under the constraint that all bendings must occur at the blocks on the bus net. There is no limitations on the bus shape and the number of bendings as long as the above requirement on the bending positions is satisfied. We solve this problem in a floorplanner based on the TCG representation. In our approach, we will first compute the shape of a bus satisfying the above constraint and minimizing the total bus length. This is done by applying a modified minimum spanning tree algorithm on a combined constraint graph (called *common graph*). After this step, each bus is decomposed into a set of horizontal and vertical bus components. We will then compute the positions of the blocks in the floorplan realization step by properly adjusting the block positions such that all the bus components can pass through their respective blocks successfully. Experimental results have shown that we can improve over [1] in terms of both run time and quality, by having more flexibilities in the shapes of the buses, and replacing the complex shape validation steps by simplier methods. For data sets with buses connecting a large number of blocks, our approach can give satisfactory results effectively, while the approach [1] of restricting to 2-bend buses often cannot give any feasible solutions.

The rest of this paper is organized as follows. A formal definition of the bus driven floorplanning problem will be given in section II. We will discuss the general placement constraints for buses and the bus ordering issue in section III and IV. Details of our algorithm will be described in section V . The experimental results will be presented in section VI.

[1]The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. 4181/06)

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

II. Problem Formulation

We assume that there are two metal layers reserved for bus-routing, one for horizontal buses and the other for vertical buses. In the bus driven floorplanning (BDF) problem, we are given the following:

1. A set of n rectangular modules $M = \{m_1, m_2 \dots m_n\}$ and each modules m_i is associated with an area a_i and an aspect ratio bound $[r_i, s_i]$ where $r_i, s_i \in R^+$, and
2. A set of k buses $B = \{b_1, b_2 \dots b_k\}$ and each bus b_j has a width t_j and a bus net N_j, where $t_j \in R^+$ and $N_j \subseteq M$.

Our goal is to decide the position of each block and the route of each bus, such that no overlapping occurs between any two blocks and between any two horizontal (vertical) components of the buses. Besides, all bendings of the buses must occur at the modules on the corresponding bus nets in order to minimize the number of vias used. The objective is to minimize the chip area and the total bus area.

In this paper, we propose a novel algorithm to solve this problem, without fixing the bus shapes nor limiting the number of bendings as long as they occur at the modules on the bus nets. With more flexibilities in the shapes of the buses, the size of the solution space is increased and a better BDF solution can be obtained. Besides, the overall efficiency can be improved since complex bus shape validation steps are not needed.

III. Placement Constraints for Bus

In this section, we will discuss how we can align blocks in a packing in order to allow buses with zero or more bends to pass through. These basic technique will be used in our floorplanner.

A. Zero-Bend Bus

There are only two types of zero-bend buses, horizontal and vertical. In the following, we only discuss the placement constraints for horizontal buses. The vertical buses can be handled similarly.

Consider a set of k modules $\{m_1, m_2 \dots m_k\}$, where module m_i has a width w_i and a height h_i for $w_i, h_i \in R^+$. If all the k modules are aligned horizontally, the corresponding horizontal closure graph G_h is shown in Fig. 1 (assuming that they align in the order of $m_1, m_2 \dots$). Due to the transitive closure property, each node is connected to all the "downstream" nodes by a horizontal edge with a weight equal to its width. On the other hand, the vertical closure graph G_v will contains only k isolated nodes without any edges between them.

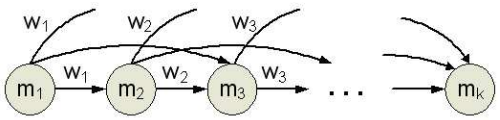

Fig. 1. G_h for horizontally aligned modules.

Suppose that we need to generate a horizontal bus b with width t and a bus net $N \subseteq M$. In order to allow the bus to pass through all its blocks in the final floorplan, we need to maintain a relative relationship between the modules in the vertical direction, i.e., the vertical overlap of the modules has to be at least the bus width t. This can be done by adding constraint edges to G_v. We will first add to G_v a dummy module m_d of height t and zero width to represent bus b. Then, we will add constraint edges between m_d and each m_i in N. In this case, the distance of m_i's lower right corner relative to m_d's lower left corner must be in the range of $[-h_i + t, 0]$, so a pair of constraint edges are added to G_v:

1. An edge from m_d to m_i with weight $t - h_i$
2. An edge from m_i to m_d with weight 0

Similarly, if we want to generate a vertical bus, a dummy module m_d of zero height and width t will be added to G_h. Then, a pair of constraint edges will be added to G_h between each m_i in N and m_d as follows.

1. An edge from m_d to m_i with weight $t - w_i$
2. An edge from m_i to m_d with weight 0

Notice that instead of adding pairs of edges between every pair of modules in a bus net, this approach of adding a zero area dummy node to represent the bus can help reducing the number of additional constraint edges from quadratic to linear, and hence to improve the efficiency of the floorplanning algorithm. Fig. 2 shows the vertical closure graph G_v after inserting the constraint edges.

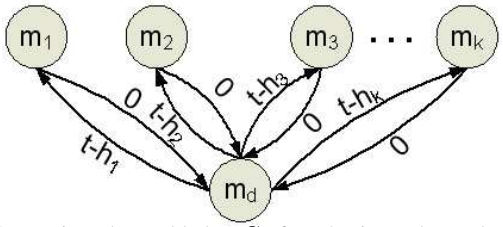

Fig. 2. Constraint edges added to G_v for a horizontal zero-bend bus.

B. Multi-Bend Bus

A multi-bend bus is formed by one or more zero-bend bus components. After decomposing a multi-bend bus into a set of 0-bend bus components, the corresponding sets of additional constraint edges for each component can be inserted into the constraint graphs as discussed in the previous section to align the blocks for the bus component to pass through. Fig. 3 shows a placement of four blocks and an L-shaped bus with two bus components, one horizontal and one vertical. The TCGs with the additional constraint edges are shown on the right.

Now we are left with the problem of how to decompose a bus into a set of horizontal and vertical bus components such that all bendings will occur at the modules of its bus net. In our approach, we will first build a graph called *common graph* for each bus. By finding a suitable spanning tree on this graph, we will be able to determine the bus components. More details will be given in section V.

IV. Bus Ordering

In a feasible BDF solution, no buses should overlap with one another on each metal layer. It means that no horizontal components should overlap with another horizontal component and similarly for the vertical components. This non-overlapping requirement can be enforced by imposing a bus ordering between

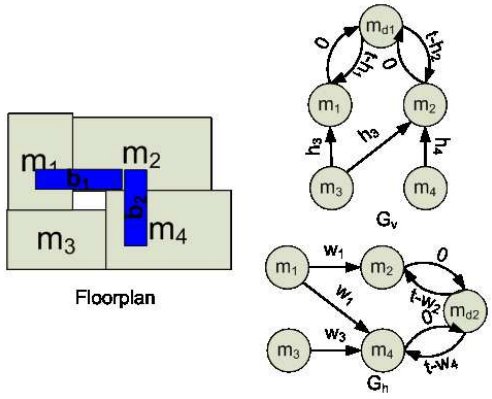

Fig. 3. An example of handling multi-bend buses.

the buses, e.g., bus i must be put on top of or on the right hand side of bus j. Given a floorplan of n modules $\{m_1, m_2 \ldots m_n\}$ with constraint graphs $G_h = (V, E_h)$ and $G_v = (V, E_v)$, the edges in E_h and E_v, representing the relative positions between the modules, may give a natural ordering between two buses b_1 and b_2 with bus nets N_1 and N_2 respectively as follows:

Case 1 b_1 is on top of b_2 when

1. b_1 and b_2 are both horizontal bus components, and
2. $\exists e_{ij} \in E_v$ such that $m_j \in N_1$ and $m_i \in N_2$.

Case 2 b_1 is on the right side of b_2 when

1. b_1 and b_2 are both vertical bus components, and
2. $\exists e_{ij} \in E_h$ such that $m_j \in N_1$ and $m_i \in N_2$.

Fig. 4 shows an example in which a natural ordering can be deduced from the vertical constraint graph. In this example, the bus going through block m_1 and m_2 must be above that going through m_4 and m_5 since m_4 is required to be placed below m_2.

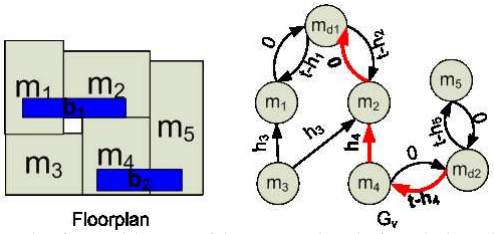

Fig. 4. Two horizontal buses with a natural ordering deduced from the constraint edges.

For those bus pairs which do not have such natural orderings, we need to assign their orderings explicitly if they may overlap. There are only two cases that two bus components b_1 and b_2 may overlap:

Case 1 $N_1 \cap N_2 \neq \emptyset$, i.e., b_1 and b_2 share at least one module.

Case 2 $N_1 \cap N_2 = \emptyset$ and $\exists m_i \in N_1$ and $m_j, m_k \in N_2$ or $\exists m_i \in N_2$ and $m_j, m_k \in N_1$ such that e_{ji} and $e_{ik} \in E_h$ (or e_{ji} and $e_{ik} \in E_v$), i.e., the modules of b_1 and b_2 interleave with each other in the x-direction (or y-direction).

In these two cases, we will impose an explicit bus ordering to prevent overlapping. Suppose t_1 and t_2 are the widths of b_1 and b_2 respectively and m_{d1} and m_{d2} are their corresponding dummy modules in the constraint graphs. An explicit bus ordering can be enforced as follows:

1. When b_1 and b_2 are both horizontal, we add an edge from m_{d1} to m_{d2} with weight t_1 or an edge from m_{d2} to m_{d1} with weight t_2 to G_v
2. When b_1 and b_2 are both vertical, we add an edge from m_{d1} to m_{d2} with weight t_1 or an edge from m_{d2} to m_{d1} with weight t_2 to G_h

Fig. 5 shows an example of how bus overlapping can be prevented by imposing an explicit bus ordering. In this example, the overlapping between the two horizontal bus components is removed by adding an edge of weight t_2 from dummy node m_{d2} to node m_{d1} in G_v.

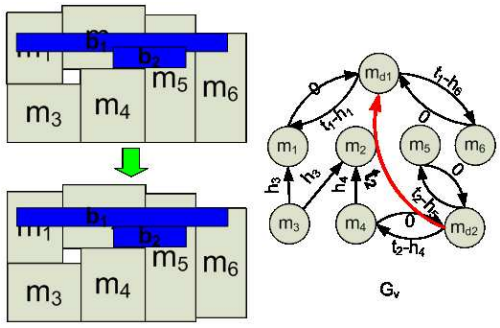

Fig. 5. Prevention of bus overlap by imposing explicit bus ordering. In this example, b_1 is connecting m_1 and m_6, and b_2 is connecting m_4 and m_5.

V. Methodology

Simulated annealing (SA) is used as the basic searching engine in our floorplanner. In each iteration of the annealing process, a floorplan, represented by a pair of transitive closure graphs (G_v and G_h), is generated. A pair of reduced constraint graphs (G'_v and G'_h) (whose structures will be described later) will then be constructed to for the efficiency of the later processes. For each bus, we will create a graph called common graph from the two reduced constraint graphs, on which we will apply a modified minimum spanning tree algorithm to determine the set of bus components. Then, we will decompose the bus into a number of horizontal and vertical components. A set of constraint edges will be added to G_v and G_h to align the blocks for the bus components to pass through according to the method in section III. Meanwhile, we will check whether the bus is feasible. If the bus is infeasible, its constraint edges will be removed and a penalty term will be added to the cost of the annealing process. After processing all the buses, some more constraint edges will be inserted to prevent bus overlapping. Finally, we will perform a single source longest path algorithm to determine the positions of the modules and the buses. At the end, we will compute the cost of the BDF solution according to the total chip area, the total bus area and the number of infeasible buses.

A. Construction of Reduced Graphs

Given the constraint graphs $G_h = (V, E_h)$ and $G_v = (V, E_v)$ of a candidate floorplan solution, we will construct a

pair of reduced graphs $G'_v = (V', E'_v)$ and $G'_h = (V', E'_h)$, where

1. $V' = \cup_j N_j$ for all $1 \leq j \leq k$,
2. $E'_h \subseteq E_h$ and $e_{ij} \in E'_h$ iff $e_{ij} \in E_h$ and $m_i, m_j \in V'$,
3. $E'_v \subseteq E_v$ and $e_{ij} \in E'_v$ iff $e_{ij} \in E_v$ and $m_i, m_j \in V'$,
4. the weight of $e_{ij} \in E'_h$ (E'_v) is the longest path distance between m_i and m_j in G_h (G_v) respectively.

The reduced graphs (G'_h and G'_v) contain the constraint modules as nodes and the weights on the edges represent the distances between the modules in G_h and G_v. The weights of the edges in E'_h and E'_v can be found by performing an all pair longest path algorithm on G_h and G_v.

B. Construction of Common Graph

For each bus b_j with width t_j and bus net N_j, we will further construct a common graph denoted by $G_{cj} = (V_j, E_j)$, where

1. $V_j = N_j$,
2. $E_j = \{e_{ik} | e_{ik} \in E'_h \cup E'_v$ and $m_i, m_k \in N_j\}$, and
3. the weight of an edge in E_j is the same as that of the corresponding edge in E'_h or E'_v, depending on where it comes from.

The common graph for bus b_j contains all the modules on the bus net N_j. Its edge set includes all the edges in G'_v or G'_h connecting any two modules in N_j. Due to the transitive closure properties of G'_v and G'_h, the resulting common graph G_{cj} is a complete graph with $|N_j|$ nodes.

C. Spanning Tree for Bus Assignment

A bus is required to pass through all the modules on its bus net. No matter what its shape is, the routing of the bus must span all the nodes in the common graph. Therefore, our aim is to find a good spanning tree denoted by $T_j(V_j, E_{Tj})$ from the common graph $G_{cj}(V_j, E_j)$.

In order to reduce the total bus area, our goal is to find a minimum spanning tree. However, the minimum spanning tree on G_{cj} does not always lead to a feasible bus. The first reason is that the number of bus components passing through a module may exceed the maximum number allowed (we call this the *capacity* of the module), e.g., a connected module can at most allow one horizontal and one vertical bus component of the same bus to pass through. The second reason is that adding the corresponding set of constraint edges for a particular bus component (as described in section III) may create positive cycles in the constraint graphs because its alignment requirements on the modules may contradict with those of some other selected bus components.

To solve the first problem, we modified the Kruskal's algorithm as follows. When constructing the spanning tree, we will update the number of vertical edges (edges from G'_v) and horizontal edges (edges from G'_h) connected to m_i for all $m_i \in N_j$. Whenever a new edge (m_i, m_k) is included in T_j, not only that we will check if T_j becomes cyclic (as in the traditional Kruskal's algorithm), we also check if the capacities of m_i and m_k are violated. We will just skip the edge if either of them is true. If no spanning trees can be constructed at the end, the bus is regarded as infeasible. To solve the second problem, we will incorporate the bus feasibility check to be described in section E.

D. Formation of Bus Components

There are two possible types of edges in the spanning tree T_j. Those coming from G'_v are vertical edges while those coming from G'_h are horizontal edges. We will group those adjacent tree edges of the same kind to form one bus component. This grouping is performed until every tree edge is contained in one and only one component. Finally, the adjacent vertical edges will form a vertical bus component and the adjacent horizontal edges will form a horizontal bus component.

E. Bus Feasiblity Check

In the modified Kruskal's algorithm discussed in section C, in fact, the positive cycle detection can be performed for each edge found during the spanning tree construction, However, the run time will be very expensive in that case. Therefore, in practice, we will perform the detection only after the whole spanning tree and all bus components of a bus is found.

For each bus with all its bus components found, a set of constraint edges will be added as discussed in section III. These edges together with the dummy modules (one for each component) will be added to the reduced graphs G'_h and G'_v. By using the Bellman-ford algorithm, positive cycles in either G'_h or G'_v can be detected. The bus is regarded as infeasible if positive cycles exist. Otherwise, we will keep the constraint edges and the dummy modules in G'_h and G'_v and also copy them to G_h and G_v. The total number of dummy modules in the constraint graphs will be equal to the total number of bus components among all the feasible buses.

F. Overlap Removal

To prevent overlapping between two bus components which do not have a natural ordering, an explicit ordering (and thus additional constraint edges) will be needed if they may overlap as discussed in section IV. In fact, there may be more than one feasible orderings for a set of bus components, and we can consider exhausting all cases to find the best one. However, as the width of a bus is relatively small compared with those of the modules, the ordering has little effect on the bus feasibility. Therefore, we will just choose one ordering arbitrarily in our floorplanner.

G. Floorplan Realization

After adding all the constraint edges for the buses, the resultant floorplan can be obtained by performing a single source longest path algorithm on the constraint graphs. Besides finding the coordinates of the modules, the y-coordinate of a horizontal bus component and the x-coordinate of a vertical bus component can be obtained from the longest path distances of the dummy nodes in G_v and G_h respectively.

H. Simulated Annealing

Simulated annealing (SA) is used as the basic searching engine in our floorplanner.

H.1 Set of Moves

There are four kinds of operations to perturb a TCG: (1) swap two nodes in both of G_h and G_v, (2) exchange a module's height and width, (3) Reverse a reduction edge in G_h or G_v, and (4) move a reduction edge from one TCG (G_h or G_v) to the other.

H.2 Cost Function

The objective of the BDF problem is (1) to minimize the area of the floorplan, (2) to minimize the total bus area, and (3) to accommodate all the buses, so the cost function is defined as follow:

$$Cost = \begin{cases} \alpha A + \beta B + \gamma I & if B > \delta \\ \alpha A + \gamma I & if B \leq \delta \end{cases}$$

where A is the chip area, B is the total bus area and I is the number of infeasible buses, and α, β, γ and δ are parameters that can be specified by the users. The parameter δ is a threshold for the bus cost which allows the floorplanner to give solutions with smaller dead space percentage. If the bus cost is smaller than this threshold, it is not added to the total cost.

H.3 Speedup of the Annealing Process

Bus assignment is the most time-consuming step in our floorplanner. In order to reduce run time, we will estimate the cost in each annealing iteration before invoking the bus assignment step. To estimate the cost, we will first compute the chip area A and compare it with the cost of the previous BDF solution (C). If $A < C$, we will continue with the bus assignment. Otherwise, we will continue with a probability $e^{-\frac{A-C}{T}}$, where T is the current temperature. By adding this simple computation, many poor BDF solutions can be pruned at an early stage. This improvement can reduce over 70% of the run time for our floorplanner and its effectiveness can be seen from the experimental results.

I. Soft Module Adjustment

We will adjust the dimensions of the soft modules in a post-processing step. This soft block adjustment step is also done by simulated annealing with the same cost function. In each iteration of the annealing process, a module lying on a critical path will be selected, and either its width or height will be changed a little bit. However, if some originally feasible buses become invalid after this adjustment, the candidate solution will be rejected.

VI. Experimental Results

We compare our results for a data set which previouly used in [1] (Details can be found in [1]). Since there is no experimental results for hard modules given in [1], we only compare the results after the soft module adjustment. Experimental results show that our approach can reduce the dead spaces by 22.62% on average (Table VI).

In order to have a better comparison including run time with the approach presented in [1] and to demonstrate the advantage of our algorithm that favors test cases with large bus nets, we have created another set of test cases based on the ami33 benchmarks. Test cases from ami33-a to ami33-e are explicitly created to have a gradual increase in the average net size. Details are shown in table II. Our proposed algorithm was implemented using the C language. All test cases are run by both our floorplanner and that in [1] on the same machine, Dell Optiplex 280 Intel P4 (3.2GHz) with 2GB memory. We run each test case for ten times and then record the average.

TABLE I
Comparison on Data Set One with [1].

	[1]		Our Work		Comparison*
	Time (s)	Dead Space (%)	Time (s)	Dead Space (%)	(%)
ami33-3	32	1.01	10	0.84	-16.83
ami33-4	92	1.90	29	0.72	-62.11
ami33-5	95	3.80	31	1.28	-66.32
ami49-4	88	0.63	32	0.87	+38.10
ami49-5	261	1.17	44	1.13	-3.42
ami49-6	140	2.19	104	1.64	-25.11
				Average:	-22.62

*It is calculated by $[(x_1 - x_0)/x_0] \times 100\%$, where x_0 and x_1 are the dead space obtained by [1] and our algorithm respectively.

TABLE II
Data Set Two.

Data	No. of Blocks	No. of Buses	Avg./Max. No. of Blocks on a Bus Net
ami33-a	33	5	3 / 4
ami33-b	33	5	4 / 5
ami33-c	33	5	5 / 6
ami33-d	33	5	6 / 7
ami33-e	33	5	7 / 8
ami33-f	33	5	8 / 9

The results are shown in table III. When the bus net size increases, experiments show that both the run time and dead space percentage of our floorplanner will increase. However, comparing with [1], our algorithm still perform better in run time by 32.07% and in dead space percentage with and without soft module adjustment by 11.17% and 21.34% respectively. More importantly, note that the approach in [1] is not able to generate any feasible solutions when the bus net size increases further. For ami33-e, only one out of the ten annealing processes generates a feasible final floorplan. For ami33-f, none of the ten resulting floorplans is feasible.

More details of the experiments for data set are reported in table VI, which may give more insights to our approach. The increase in bus flexibility (without restricting the number of bendings) has increased the percentage of feasible candidate BDF solutions in the annealing process and this is one major reason of why our algorithm can generate solutions with higher quality. For run time, there are three factors contributing to the reduction. Firstly, there is no more complex shape validation steps; Secondly, our searching can find feasible solutions with less iterations because of the relaxed restriction on bus shapes. Lastly, the speedup step as discussed in section H.3 can significantly reduce the run time by skipping 88% of the iterations on average.

Finally, we also derive a new set of test cases from ami49 with bus net sizes ranging from 10 to 49. To the best of our knowledge, no previous approaches can handle a buses with

TABLE III
COMPARISON ON DATA SET TWO WITH [1].

	[1]			Our Work			Comparison		
	Time* (s)	Dead Space (%)		Time* (s)	Dead Space (%)		Time (%)	Dead Space (%)	
		No Soft Adjust	With Soft Adjust		No Soft Adjust	With Soft Adjust		No Soft Adjust	With Soft Adjust
ami33-a	31.3 (+0)	6.40	1.80	9.7 (+1)	6.30	1.42	-69.01	-1.56	-21.08
ami33-b	32.8 (+1)	8.23	3.16	12.3 (+0)	6.59	1.84	-62.50	-19.92	-41.73
ami33-c	34.0 (+1)	8.51	2.58	16.4 (+0)	7.82	2.34	-51.76	-8.10	-9.46
ami33-d	31.6 (+1)	10.35	2.57	22.0 (+1)	8.36	2.35	-30.38	-19.24	-8.64
ami33-e	20.4 (+0)	10.35	3.13	24.5 (+2)	8.99	1.97	+20.10	-17.58	-36.83
ami33-f	26.5 (+1)	**9.52	2.21	26.8 (+1)	9.46	1.98	+1.13	-0.63	-10.29
						Average:	-32.07	-11.17	-21.34

*Figures inside brackets represent the run time for soft module adjustment.

**No feasible solutions is generated.

TABLE V
RESULTS ON DATA SET THREE.

Data	No. of Blocks	No. of Buses	Net Size	Run Time (s)	Dead Space (%)	Dead Space after Soft Adj. (%)	No. of SA Iter.	Iter. for Bus Asgmt.	Feasible Floorplan
ami49-a	49	1	10	40(+2)	5.32	1.21	131,210	5693	91.78%
ami49-b	49	1	20	40(+2)	5.97	1.55	124,252	6402	89.36%
ami49-c	49	1	30	40(+2)	6.47	1.50	127,971	6639	84.86%
ami49-d	49	1	40	49(+2)	7.86	1.74	131,270	6717	81.02%
ami49-e	49	1	49	51(+2)	9.35	2.16	148,372	8736	77.82%

TABLE IV
DETAILED RESULTS ON DATA SET TWO.

	[1]		Our Work		
Data	No. of SA iter.	Feasible Floor-plan*	No. of SA iter.	Iter. for Bus Asgmt.**	Feasible Floor-plan*
ami33-a	1,557,336	73.8%	73,971	9,131	92.1%
ami33-b	1,557,336	58.7%	81,388	8,420	84.6%
ami33-c	1,557,336	50.2%	63,881	6,936	73.7%
ami33-d	1,557,336	29.0%	84,149	9,920	71.7%
ami33-e	1,557,336	0.0%	69,557	8,888	75.6%
ami33-f	1,557,336	0.0%	81,590	11,136	69.6%

*The percentage of floorplans generated by the SA in which all the buses are feasible.
**The number of floorplans that passes the test in section H.3.

such a large net size. The results are shown in table V. Our approach can generate floorplan solutions quickly, even for the test case with largest net size (ami49-e), the total run time is still less than one minute. The resulting floorplans generated have small dead space percentage both before and after the soft module adjustment step.

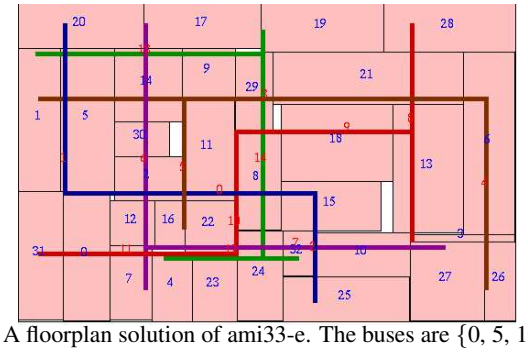

Fig. 6. A floorplan solution of ami33-e. The buses are {0, 5, 10, 15, 20, 25}, {1, 6, 11, 16, 21, 26}, {2, 7, 12, 17, 22, 27, 30}, {0, 3, 8, 13, 18, 23, 28, 31}, {1, 4, 9, 14, 19, 24, 29, 32}

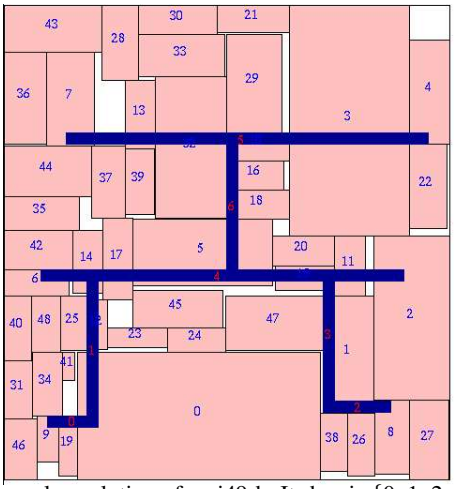

Fig. 7. A floorplan solution of ami49-b. Its bus is {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19}

REFERENCES

[1] Jill H.Y. Law and Evangeline FY Young. "Multi-Bend Bus Driven Floorplanning", ISPD, pp. 113-120, 2005.

[2] T.-C. Chen and Y.-W. Chang "Modern floorplanning based on fast simulated annealing", ISPD , pp. 104–112, 2005.

[3] Hua Xiang, Xiaoping Tang and Martin D.F.Wong. "Bus-Driven Floorplanning", ICCAD, 2003.

[4] Evangeline F.Y.Young, Chris C.N.Chu and M.L.Ho. "A Unified Method to Handle Different Kinds of Placement Constraints in Floorplan Design", ASP-DAC, 2002.

[5] X. Tang and D.F.Wong. "Floorplanning with alignment and performance constraints", DAC, pp.848-853, 2002.

[6] J.-M. Lin and Y.-W. Chang. "TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans", DAC, pp. 764-769, 2001.