Semi-Detailed Bus Routing with Variation Reduction

Fan Mo
Synplicity
600 W California Ave
Sunnyvale, CA 94086
1-408-215-6103
fanmo@synplicity.com

Robert K. Brayton
University of California
573 Cory Hall, UC Berkeley
Berkeley, CA 94720
1-510-643-9801
brayton@eecs.berkeley.edu

ABSTRACT
A bus routing algorithm is presented which not only minimizes wire length but also selects the bits in the bus to avoid twisting and conflicts. The resulting bus routes are regular, thus having strong immunity to variations. Minimization for wire length/delay differences between different bits is also implemented. The algorithm is fast, which allows for being embedded in floorplanning and other physical design algorithms.

1. INTRODUCTION
Buses are important signals in modern integrated circuits. A bus is composed of a set of nets that transfers information in parallel from one functional unit to others. Buses are widely used in computation and storage intensive designs like microprocessors [5]. The growing trend of Intellectual Property (IP) integration also involves an increasing portion of bus connections and bus interfacing.

Obviously, bus routing can be realized by routing each bit of a bus as a normal net. However, this counteracts the advantages and reasons for using a bus. Special algorithms to handle the bus routing problem are motivated by the following concerns.

Matching: The matching of driver-load length and driver-load delay need not be exact but should be within a small tolerance that can be given as a special constraint for bus routing. Given that all the bits are routed using isomorphic topologies, the matching of wire length/delay for the bits in a bus becomes easier.

Saving runtime: If all the bits of a bus use isomorphic topologies, a single representative net can be used. Once routed, this *virtual net* is copied to all the bits. The copying process is trivial except for an overhead of processing turning points and Steiner points. Thus, runtime for routing is reduced, as well as the time for estimation of parasitics, delay, and power.

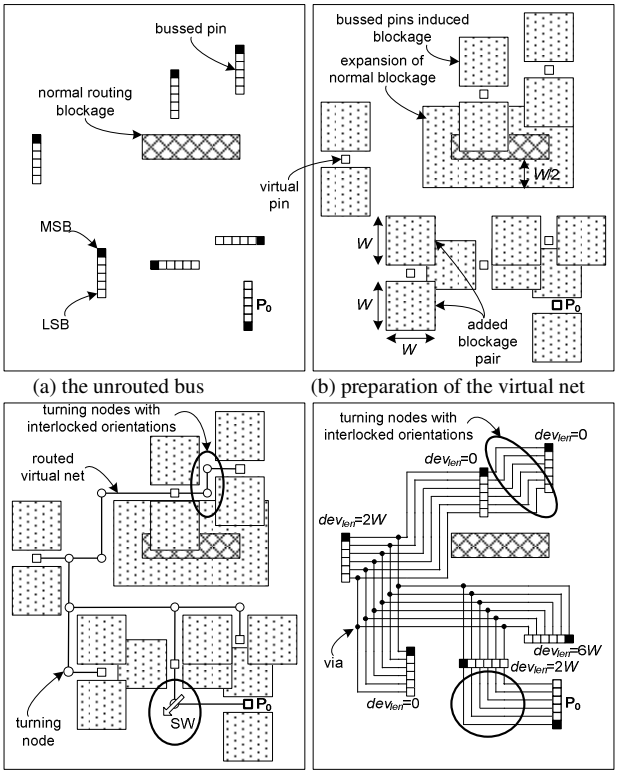
Routability: Routability problems with a bus often come from the pin region. A bused pin of an *n*-bit bus contains *n* closely located connecting points. Inappropriate connection of one pin can easily block the connections to many other pins. Also, twisted segments can cause congestion in non-pin areas. Sometimes, a bus with a seemingly simple routing solution can get a general purpose router into a quagmire.

Variation immunity: The regularity of a bus implies that all bus bits remain contiguous when routed from the driver to the load(s). Therefore, parametric variations [4], no matter how they distribute and act, affect all the bits nearly equally, and thus have little effect on the delay matching of the bus bits.

Persky and Tran [9] proposed a bus routing algorithm that identified the topological commonality of different bits of the same bus and tried to re-use a common routing topology. The idea of a common routing topology in our router ~~and~~ is called the "virtual net topology". Other existing bus routing/planning methods are mostly used in floorplanning [1-3], where buses are limited to fanout = 1 or 2 and to simple topologies like one-bend and T-shaped routes. However, all these algorithms leave the problem of organizing the bits in the bus to the detailed routing. Global routing/planning of buses as in [9] is not always enough, since a detailed router may fail to implement the planned bus topologies because of twisting of bus bits and decreased accessibilities of the bused pins. If the detailed router must re-route some of the bus bits using different topologies, the advantages of using buses are lost. The bus router we present is "semi-detailed" in that it considers the ordering of the bits in a bus, which is a key to successful bus routing.

Figure 1 illustrates our bus routing algorithm. An unrouted bus is shown in (a) with some normal routing blockage present. The first step, as illustrated in (b), is to abstract a virtual net from the bus. Each bused pin is transformed to a virtual pin located at the center point of the bused pin. A virtual pin induces a pair of blockages. North/south aligned pins have an above/below pair of blockages, representing half the width of the bus. East/west oriented pins have a left/right pair. The normal routing blockages are expanded appropriately. The routing of the virtual net, composed of the virtual pins, produces a routing topology, as illustrated in (c). Junctions on the topology with degree two or more, are called turning nodes. Their orientations are important to the final routing. Turning nodes which are too close together (related to the bus width) are considered "locked" as shown in (c). The algorithm determines the orientations of the turning nodes, and if successful, it generates the routed bus as shown in (d).

In floorplanning where detailed routing of the bus is not required, our bus routing algorithm can be simplified into a bus validity checking procedure. The algorithm also provides the opportunity to minimize driver-load wire length/delay deviations between different bits of the bus.



(a) the unrouted bus (b) preparation of the virtual net

(c) routing of the virtual net (d) generating the bus

Figure 1. Example of bus routing algorithm.

The paper is organized with the basics of the bus routing, including assumptions, definitions and terminologies, given in Section 2. The algorithm is described in Section 3. Section 4 gives experimental results and Section 5 concludes.

2. BASICS

Because long or timing critical nets are normally assigned to a pair of high metal layers, one horizontal and one vertical, we only consider the two-layer bus routing problem. For the purposes of this paper, we assume the lower layer L_H is horizontal and the upper layer L_V is vertical. The pins are located on L_H. A uniform routing pitch P is assumed for both L_H and L_V. The pins from different bits of a bus, forming the so-called "bused pins", are placed side by side using wire pitch P. Hence an n-bit bused pin, either vertically or horizontally arranged, occupies a range of $W = n \times P$. Vertically oriented bused pins can connect directly to horizontal segments since both are in the lower layer. Horizontally arranged bused pins, when connected to vertical segments, require n vias to reach L_V. Horizontal segments are not allowed to connect directly to a horizontally arranged bused pin. The reason for this restriction is that, pins (bused or not) of a macro block usually lie on its periphery, and if we allow access to a bused pin from its side, the bus may make the connections to the other pins around it very difficult. So a horizontally arranged bused pin is only accessible on L_V from the north or south. Similarly, a vertically arranged bused pin is only accessible on L_H from the east or west.

For brevity, we only consider strict and untwisted bus routing, i.e. all the bits of the bus are routed side by side in a W-wide band and the sequence of the bits in the band does not alter (monotonic sequence from LSB to MSB). The special case in which narrow routing blockages are present will be discussed briefly in Section 3.1.

Virtual net topology

The routing of a virtual net results in a tree topology and all segments are either vertical or horizontal. A node, representing a bused pin, is called a pin node. Non-pin nodes, are either points connecting one horizontal segment and one vertical segment, or Steiner points. Due to the addition of the pairs of companion blockages to the virtual pins, the degrees of the virtual pins can be only one or two. The number of nodes, denoted by U, in the resultant routing tree depends on not only the fanout of the net, denoted by F, but also the number and distribution of the blockages.

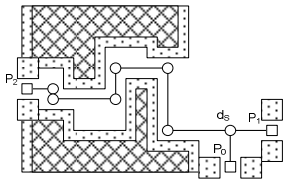


Figure 2. Virtual net topology.

An example is shown in Figure 2. Although $F = 2$, the total number of nodes in the virtual net topology is $U = 10$. In general, U is affected by the routing blockages, both the normal ones and the ones induced by the bused pins. It is not uncommon to see staircase, zigzag and serpent shaped connections, like the P_2-d_S connection in the figure.

Orientation

We define the orientation of node d, denoted by $r(d)$, as the direction from the LSB towards the MSB. It can be N(North), S(South), E(East) or W(West) for a pin node and NW(Northwest), NE(Northeast), SW(Southwest) or SE(Southeast) for a non-pin node. Figure 1(c) shows that the turning node after driver P_0 has an orientation of SW and its realization can be found in Figure 1(d). The primary aim of the algorithm is to determine each non-pin node's orientation.

Orientation set

All valid orientations of node d form the orientation set $R(d)$ of the node. Obviously, the orientation set of a virtual pin node is uniquely one of the following: {N}, {E}, {S} or {W}. A non-pin node might have more than one valid orientation in its orientation set. The possible valid orientation sets are {NW}, {NE}, {SW}, {SE}, {NW,NE}, {NE,SE}, {SE,SW}, {SW,NW}, {NE,SW}, {NW,SE} and {NW,NE,SW,SE}. For simplicity, we use the following notations:

{IL}={} or empty set,
{NH}={NW,NE}, {EV}={NE,SE},
{SH}={SE,SW}, {WV}={SW,NW},
{TP}={NE,SW}, {TN}={NW,SE}, {AL}={NW,NE,SW,SE}.

Orientation operations

The first operation is "propagate", denoted by Θ. Suppose node d_j has $R(d_j)$={NW} and it connects to node d_i through a horizontal segment. As a result, $\Theta(R(d_j),\text{dir}(d_j,d_i))$={NW,NE} becomes the orientation set that d_i has to match, because at node i, the arrangement of the bits can slant either way, NW or NE. Generally, the argument $\text{dir}(d_i,d_j)$ will be dropped for simplicity. The "propagate" operator is implemented as a lookup table:

$\text{dir}(d_i,d_j)\backslash R(d_j)$	N	W	S	E	NW	NE	SW	SE	NH	SH	WV	EV	AL	TP	TN	IL
vertical	IL	WV	IL	EV	WV	EV	WV	EV	AL	AL	WV	EV	AL	AL	AL	IL
horizontal	NH	IL	SH	IL	NH	NH	SH	SH	NH	SH	AL	AL	AL	AL	AL	IL

It is left to the reader to verify the correctness of this table.

Another operator is "join", which is the set intersection "AND" operation between two orientation sets. One node i can propagate and then join its orientation set, $R(d_i)$, into its neighboring node's orientation set, $R(d_j)$.

$$R(d_j)=\Lambda(\Theta(R(d_i)), R(d_j))$$

where $\Lambda(,)$ is the "join" operator, also implemented as a lookup table:

$\Theta(R(d_i))\backslash R(d_j)$	N	W	S	E	NW	NE	SW	SE	NH	SH	WV	EV	AL	TP	TN	IL
NW	N	W	IL	IL	NW	IL	IL	IL	NW	IL	NW	IL	NW	IL	NW	IL
NE	N	IL	IL	E	IL	NE	IL	IL	NE	IL	IL	NE	NE	NE	IL	IL
SW	IL	W	S	IL	IL	IL	SW	IL	IL	SW	SW	IL	SW	SW	IL	IL
SE	IL	IL	S	E	IL	IL	IL	SE	IL	SE	IL	SE	SE	IL	SE	IL
NH	N	W	IL	E	NW	NE	IL	IL	NH	IL	NW	NE	NH	NE	NW	IL
SH	IL	W	S	E	IL	IL	SW	SE	IL	SH	SW	SE	SH	SW	SE	IL
WV	N	W	S	IL	NW	IL	SW	IL	NW	SW	WV	IL	WV	SW	NW	IL
EV	N	IL	S	E	IL	NE	IL	SE	NE	SE	IL	EV	EV	NE	SE	IL
AL	N	W	S	E	NW	NE	SW	SE	NH	SH	WV	EV	AL	TP	TN	IL
TP	N	W	S	E	IL	NE	SW	IL	NE	SW	SW	NE	TP	TP	IL	IL
TN	N	W	S	E	NW	IL	IL	SE	NW	SE	NW	SE	TN	IL	TN	IL
IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL

Orientation interlocking

If two non-pin nodes are connected by a segment shorter than W and the two nodes both have segment(s) in the direction orthogonal to the short segment, their orientations are interlocked. Orientation propagation is not allowed when interlocking occurs. This is illustrated in Figure 1(c), where the circled segment on the right top corner is short and the two nodes of this segment must both take NW.

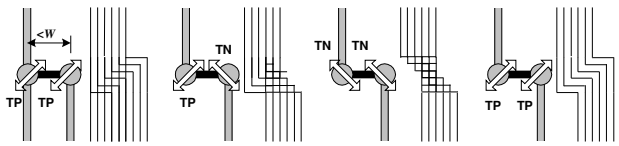


Figure 3. Orientation interlocking.

A detailed example is given in Figure 3, in which the black segment in the middle has a length shorter than W. (a), (b) and (c) are all invalid because of the overlapping routes when the bus is realized. Only (d) can produce a valid final bus routing. A necessary condition is that the two nodes each can have at most one segment in the orthogonal direction and the orthogonal segments of the two nodes must lie on opposite sides. Only the four topologies shown in the following table are valid for the short black segment in the middle. Then corrections for the interlocking orientation,

$$R(d_i)=\vartheta(R(d_i)) \text{ and } R(d_j)=\vartheta(R(d_j))$$

are applied to narrow down the orientation sets of the two nodes. The interlocking operator ϑ is implemented as the following lookup table:

topology				
$\vartheta(R(d_i))=$	$\Lambda(R(d_i),\{TP\})$	$\Lambda(R(d_i),\{TN\})$	$\Lambda(R(d_i),\{TP\})$	$\Lambda(R(d_i),\{TN\})$

Note that the short segment can extend beyond the nodes in the same direction. This does not affect the validity checking and the correction for the orientation interlocking. Orientation interlocking could possibly reduce the choices of the orientation sets of the affected nodes. Because of the added blockages besides the virtual pins, interlocking with a pin node will never occur.

Deviation

Due to the choice of the orientations at the turning nodes, even strict bus routing can result in different driver-load wire lengths between different bits. At a load, the wire length difference among all bits is characterized by wire length "deviation". In strict bus routing, we only need to focus on the wire lengths of the MSB and LSB, because the lengths of all other bits can be linearly interpolated between these lengths. Denote the MSB-LSB wire length deviation by

$$dev_{len} = |len(MSB) - len(LSB)| \cdot \quad (1)$$

The turning nodes are the only source for dev_{len}. A turning node can contribute $-\Delta$, 0 or $+\Delta$ to the MSB-LSB wire length difference, where $\Delta=2W$, depending on the turning direction and the orientation choice made at the node. Accumulating the turning node contributions along the driver to load path provides dev_{len} at the load (before taking the absolute value). The freedom in node orientations, defined as orientation sets of the nodes, can be used to minimize dev_{len}.

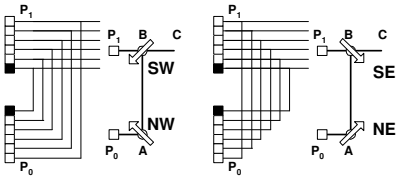


Figure 4. Wire length deviation and its relation to the node orientations.

Figure 4 shows an example of connecting two bused pins. The wire length deviation from driver P_0 to load P_1 in (a) is -2Δ, before taking the absolute value; while in (b) the deviation is 0. The example indicates that non-pin turning nodes usually provide freedom in node orientations. One may deliberately introduce serpent- and staircase-shaped node chains in the routing of the virtual net in order to obtain more freedom in controlling deviations. However this also means longer wire lengths and/or more vias. Another drawback is that too much freedom in orientations can potentially slow down the minimization algorithm. In our bus routing algorithm, we do not want to sacrifice wire length to achieve smaller deviations, because the deviations are only a secondary effect in the bus routing, compared to the entire wire length. Also in Figure 4, the path from P_0 to C, through A and B, has an accumulated deviation of $-\Delta$ in both (a) and (b).

Replacing $len(i)$ in equation (1) by delay(i) results in the delay deviation at a load, dev_{del}. Unfortunately no linear interpolation of delays exists even for strict bus routing because delay formulation is more complicated than wire length formulation and it is non-linear in general. But considering the fact that only the turning nodes could possibly lead to varied capacitances and resistances for different bits, the delay difference is still a secondary factor provided that the turning nodes are a small portion of the entire bus. Therefore, we assume that the delay deviation can be approximately minimized by concentrating on minimizing (1).

For a multiple fanout bus, the objective can be the minimization of either the total (or average) deviation or the maximum deviation among all loads.

Now we derive a bound for the maximum deviation of a bus, when the nodes in the virtual net all have non-empty orientation sets.

Definition 1: A *chain* is a series of consecutive nodes in the net topology, with terminals being pins/Steiner nodes and all nodes in the middle being degree-2 turning nodes. The deviation of a chain is the accumulated deviation of all the nodes in the chain, excluding the beginning node (because it contributes deviation to a previous chain if it is not the driver pin). A driver-load path consists of a series of chains.

Lemma 1: A chain, d_0, d_1,.., d_m, d_{m+1}, with d_0 and d_{m+1}'s orientations selected from their orientation sets, can achieve at least one of the following deviations,

-2Δ, $-\Delta$, 0, $+\Delta$ and $+2\Delta$, if d_{m+1} is Steiner, or

$-\Delta$, 0, $+\Delta$, if d_{m+1} is a pin.

Proof: Any pair of interlocked nodes in the chain can be ignored, because their Z-shape acts as a relay to the rest of the nodes in the chain and their contribution to the total deviation is always zero (refer to Section 2, *Orientation interlocking*). Suppose the simplified chain consists of turning nodes d_1, d_2, …, d_m, where $m \geq 0$, and the end nodes (pin or Steiner node) d_0 and d_{m+1}. If $m=0$ or 1, done, because each turning point can contribute $-\Delta$, 0 or $+\Delta$, while the ending node d_{m+1}, if a Steiner node, can possibly contribute another $-\Delta$, 0 or $+\Delta$. If $m \geq 2$, d_1 must have two orientations in its orientation set that are both compatible with d_0's orientation (refer to the "propagate" operation in Section 2). Similarly for d_m. All other turning nodes in the middle, d_2,..,d_{m-1}, have all four orientations in their orientation sets. Starting from d_1, one of the orientations in its orientation sets should provide 0 deviation, based on d_0's orientation, and we choose this orientation. This procedure is repeated until d_m, and the accumulated deviation is 0. d_m has to choose an orientation that matches the d_{m+1}'s orientation but the choice may lead to a $-\Delta$, 0 or $+\Delta$ deviation. If d_{m+1} is a Steiner node, it can contribute another $-\Delta$, 0 or $+\Delta$, totaling up to $\pm 2\Delta$ deviation. But if d_{m+1} is a pin, the total deviation of the chain can be controlled within $\pm\Delta$. **QED**

Suppose a driver-load path passes n_P load pins (including the end load pin) and n_S Steiner nodes. The n_P pins and the n_S Steiner nodes along the path form n_P chains ending with pins and n_S chains ending with Steiner nodes. By Lemma 1, orientation assignment exists such that the deviation for the end load pin does not exceed $(n_P+2n_S)\Delta$. The number of Steiner nodes in a net is at most $F-1$ [8], where F is the fanout of the net. There are also F load pins in the net. But if a path involves n_S Steiner nodes, at least n_S load pins are branched away from the path. Hence, $n_P \leq F-n_S$. Then we have:

Theorem 1: The maximum (absolute) deviation for any load pin can be controlled within $MaxDev = (2F-1)\Delta$.

The theorem implies that we only need to consider deviations within $\pm MaxDev$ for each node, although a long chain of turning nodes, like the P_2-d_S connection in the Figure 2, can potentially produce a much larger deviation. This also limits the size of the data structure and the call times of the core minimization procedure presented in Section 3.4.

3. THE ALGORITHM

Figure 5 outlines the bus routing algorithm. The flow starts with constructing a virtual single bit net with each pin of the virtual net representing a bused-pin. After modification and addition of routing blockages, a normal obstacle-avoiding router is called to build the topology of the virtual net. The orientation set is generated for each node in the virtual net. If an illegal or empty {IL} orientation set occurs, a fixing mechanism is applied and several iterations are carried out before admitting failure. If the orientation set generation is successful, we continue to determine orientation for each node to minimize deviations. Finally the virtual net is translated into a routing of the bus. If only checking bus validity is necessary, the algorithm can terminate immediately when an orientation set generation completes without {IL}. The steps in the flow are detailed in the following sub-sections.

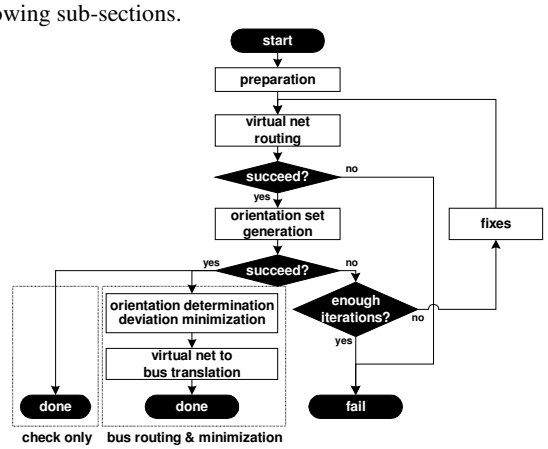


Figure 5. Bus routing flow.

3.1 Preparation

For each bused pin, the center point of all bits is used as the location of the corresponding virtual pin. Due to the accessibility restrictions discussed in Section 2.2, accompanying blockages are added for each virtual pin to force the access direction of the pin. As illustrated in Figure 1(b), a pair of blockages, both squares of W by W, are added above and below a N or S pin, or left and right of a W or E pin. This guarantees the correct direction for pin access, when the virtual net is transformed back to the routing of the bus. $W \times W$ must be used to guard the situation where the bus leaves the pin and makes an immediate turn or U-turn. Regular blockages are inflated on all sides by $W/2$. By doing so, no overlap with blockages will occur in the final bus.

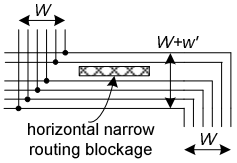


Figure 6. Avoiding narrow routing blockages.

Narrow routing blockages can appear within the routing area of the bus. A blockage on layer $L_H(L_V)$ is considered narrow if the left/right(bottom/top) edges are shorter than a certain threshold. The bused pins of other buses are typical narrow routing blockages. To accommodate these, narrow blockages are temporarily removed when the virtual net is routed. Later if a narrow blockage is within a band of W-wide bused segments, as shown in Figure 6, the segments are split by the blockage, as if a few more bits were inserted. However, this may fail because the narrow blockage can lie in a W-wide channel formed by regular blockages and its existence causes insufficient effective routing tracks within the channel. Multiple narrow blockages can make it more complicated. If failure happens, starting from the next iteration, the appropriate narrow blockage will be "upgraded" to regular and thus inflated. For simplicity, the effect of splitting bus

segments due to narrow blockages is ignored, when discussing deviations.

3.2 Virtual net routing

The routing of the virtual net is nothing special, but the router must be able to avoid routing blockages. In our experiments, we used a variant to the Steiner Arborescence algorithm [7] that works on a Hanan graph induced by the edges of the blockages and the pins [6]. Wire length minimization is the objective of the router, although timing-driven and other features can be incorporated. When the routing succeeds, a node tree is generated, rooted at the driver pin. The virtual net routing may fail, and when this happens, the bus routing fails immediately, meaning that the bus cannot be strictly routed based on our assumptions and restrictions.

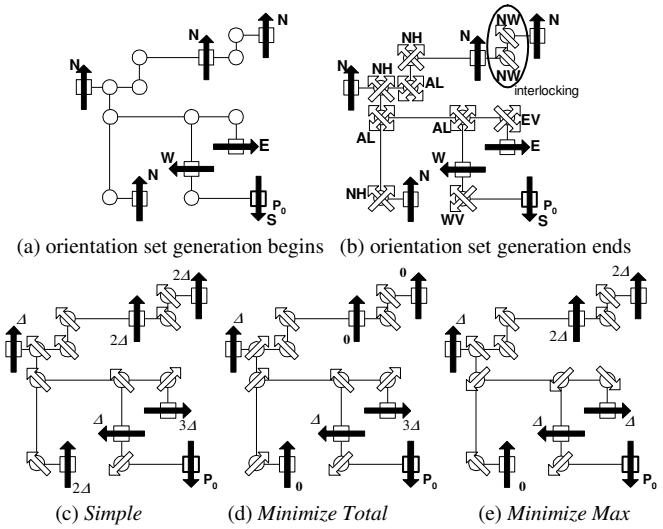
3.3 Orientation set generation

First, all the pin nodes initialize with their orientation sets and all non-pin nodes are given the set {AL}. The next step is the orientation interlocking check and correction by calling CheckAndCorrectInterlocking(d_{ROOT}), where d_{ROOT} is the root node of the virtual net, or the driver pin.

```
CheckAndCorrectInterlockingRC(node d)
{  for (each child node d_child of d)  {
       CheckAndCorrectInterlockingRC(d_child)
       if (no_interlocking(d,d_child)) then continue
       R(d)=ϑ(R(d)), R(d_child)=ϑ(R(d_child))
   }
}
```

Then, the orientation sets are backward propagated from the leaf nodes and sets are merged enroute, by calling BackwardJoinOrientationSet(d_{ROOT}).

```
BackwardJoinOrientationSetRC(node d)
{  for (each child node d_child of d) {
       BackwardJoinOrientationSetRC(d_child)
       if (no_interlocking(d,d_child)) then R(d) =Λ(R(d), Θ(R(d_child))
   }
}
```



(a) orientation set generation begins (b) orientation set generation ends

(c) *Simple* (d) *Minimize Total* (e) *Minimize Max*

Figure 7. Orientation set generation and orientation determination.

The example in Figure 7 illustrates the process of orientation set generation. As shown in Figure 7(a), the pin nodes have their fixed orientation sets (single orientation) when the process begins. The resultant orientation sets of the nodes are shown in Figure 7(b). Notice the two non-pin nodes on the top right corner. They both receive {NW}, because of the interlocking operation.

When checking bus routing validity is the only concern, for instance in the inner loop of a floorplanner, the success of this step will terminate the whole algorithm, since already there exists at least one solution.

3.4 Orientation determination

Three orientation determination methods are discussed, namely, *Simple*, *Minimize Total* and *Minimize Max*. The *Simple* method, starting from the root node, arbitrarily assigns a valid orientation from a node's orientation set and forward propagates the decision. The latter two, using a similar forward decision-propagation approach, offers minimization capability with focus respectively on the total wire length deviation at all load pins and the maximum deviation.

Continuing the example in Figure 7, the total/max deviations obtained by the three methods are $11\Delta/3\Delta$, $5\Delta/3\Delta$ and $7\Delta/2\Delta$, as illustrated in (c), (d) and (e) respectively.

Simple

After setting $r(d_{ROOT})$ to any orientation contained in $R(d_{ROOT})$, calling ForwardDetermineOrientationSimpleRC(d_{ROOT}) will determine the orientation for all other nodes. Choices are made arbitrarily.

```
ForwardDetermineOrientationSimpleRC(node d)
{  for (each child node d_child of d) {
       if (interlocking(d,d_child)) then r(d_child)=r(d)
       else R(d_child)=Λ(R(d_child), Θ({r(d)}))
       r(d_child)=any r∈R(d_child)
       ForwardDetermineOrientationSimpleRC(d_child)
   }
}
```

Minimize Total/Max

To assist the recursive total/max deviation procedure, a three dimensional array

Q[node, orientation_father, deviate_father]

is established, recording

{cost_best, orientation_best},

the best orientation and cost for a given node, its father's orientation and accumulated deviation. Since the driver node has no father, a special orientation DF is created to denote the "orientation" of the driver node's "father" node. After resetting all Q[].costbest=INFINITY, procedure MinimizeEvaluateRecur(d_{ROOT}, DF, 0) is called to start the recursive computation. When reaching a node, its father node's orientation and the accumulated deviation so far are known. Then for each possible orientation of the current node, the best downstream configuration, if not computed yet, is sought. Besides avoiding duplicate computation, the current accumulated deviation is always checked against $MaxDev=(2F-1)\Delta$, based on Theorem 1, before recursion. The minimization for the total deviation and the maximum deviation only differ in two places, as commented in the pseudo code.

Finally, MinimizeRealizeRC(d_{ROOT}, DF, 0) is called to forward propagate decisions. In these two procedures, DeviateInc(*d*,*d_child*) returns the contribution to the accumulated deviation when turning to the child node, and the possible values are $-\Delta$, 0 and Δ.

```
int MinimizeEvaluateRecur(node d, r_father, deviate_father)
{  if (r_father==DF) then r_set=AL
   else r_set = (interlocking(d_father,d)) ? {r_father} : Θ({r_father})
   r_set = Λ(R(d), r_set)
   costbest = INFINITY
   for (each r∈ r_set) {
      cost=0
      for (each child node d_child of d) {
         deviate = deviate_father
         if (not_pin(d)) then deviate += DeviateInc(d,d_child)
         if (ABS(deviate)≤MaxDev) then {
            cost_child = Q[d_child, r, deviate].cost_best
            If (cost_child==INFINITY) then
               cost_child=MinimizeEvaluateRecur(d_child, r, deviate)
            if (Minimize Total) then cost += cost_child
            else if (cost_child>cost) then cost=cost_child
         }
      }
      if (cost < costbest) then { costbest=cost, rbest=r }
   }
   if (is_pin(d)) then {
      if (Minimize Total) then costbest += ABS(deviate_father)
      else costbest = MAX(ABS(deviate_father), costbest)
   }
   set Q[d, r_father, deviate].cost_best=costbest,
       Q[d, r_father, deviate].r_best=rbest
   return costbest
}
```

```
MinimizeRealizeRecur(node d, r_father, deviate_father)
{  if (r_father==DF) then r_set=AL
   else r_set = (interlocking(d_father,d)) ? {r_father} : Θ({r_father})
   r_set = Λ(R(d), r_set)
   select rbest ∈ r_set with the minimum
      Q[d, r_father, deviate_father].costbest
   for (each child node d_child of d) {
      deviate = deviate_father
      if (not_pin(d)) then deviate += DeviateInc(d,d_child)
      MinimizeRealizeRecur(d_child, rbest, deviate)
   }
}
```

3.5 Virtual net to bus translation

This step is trivial.

3.6 Fixes

The orientation set generation step can produce two kinds of failures from the two functions in Section 3.3. One is that conflicting orientation sets meet. The other is that two interlocking nodes conflict. A simple method of fixing is employed, which cuts the edges associated with problematic nodes (by inserting a routing blockage) and thus forces a re-routing of the virtual net. Since the virtual net might have different topologies with the same minimal wire length, it is possible that re-routing does not increase wire length. We monitor the wire length increase using the result of the first iteration, wl_{REF}, as a reference. A re-route with

$$wl_{REROUTE} > 120\% \times wl_{REF} \text{ and } wl_{REROUTE} > 100\mu m$$

will trigger the message "virtual net routing failure". Ideally, the virtual-net router should be written to avoid orientation conflicts, but this was not done.

3.7 Summary

The number of times MinimizeEvaluateRecur() is called for a node *d* is dependent on the combinations of *r_father* and *deviate_father*. *r_father* can take up to 5 values (including DF), while *deviate_father* can take up to 2*MaxDev*+1 values (from $-MaxDev$ to $+MaxDev$ with step size Δ). So the call counts of the procedure for a node is O(*F*), since $MaxDev=(2F-1)\Delta$. If there is a total of *U* nodes in the virtual net, the complexity of this procedure is O(*F*×*U*). Procedure MinimizeRealizeRecur() has the same complexity. All other procedures are O(*U*). Hence, the overall complexity is O(*F*×*U*). It is worth noting that when routing blockages are plentiful, *U* can be independent of and much larger than *F*. Without *MaxDev* given by Theorem 1, the algorithm might become $O(U^2)$. The memory complexity, essentially due to the use of Q[], is O(*F*×*U*). Obviously the algorithm finds an optimal solution. The above analysis does not include the routing of the virtual net, which can dominate the overall complexity, depending on the obstacle-avoiding routing algorithm that is used.

4. EXPERIMENTAL RESULTS

4.1 Unit tests

All programs are run on a Sun Blade 1000 workstation. Four bit-widths, 8, 16, 32 and 64, were tested. The fanout range was from 1 to 16. Hence there were a total of 4×16=64 groups. For each group, 1024 testcases (one bus per testcase) were randomly generated to provide sufficient samples. Wire pitch was 1μm. Each testcase was required to be in 1000×1000μm region, and 0~2 random routing blockages were present. The random testcases were generated so that no test case was obviously unroutable, e.g., two bused pins overlap each other or pins overlap with blockages. Our bus routing algorithm, running under *Minimize Total* mode, was compared against a reference router (denoted by *nrouter*), which routes each bit separately in a non-bus routing style.

Figure 8 shows the success rates of the two routers. As the fanout increases, the success rate tends to drop for both routers. But for the bit-width = 32 testcases, *nrouter*'s success rate drops quickly to almost zero when fanout reaches 8. For bit-width = 64 testcases, the situation for nrouter is much worse. In contrast, the success rate of our bus router drops slowly for all bit-widths, and it performs well for wide buses. There are buses for which only one of the routers can complete. Thus when our router fails, possibly because of the strict bus routing requirements, a strategy would be to fall back to a non-bus routing style.

Runtime comparison is given in Figure 9. Each data point represents the total runtime of a group of 1024 testcases, regardless of the success of each individual test case. *nrouter* is configured such that it does not make too much effort on "very-likely-to-fail" testcases; otherwise the time spent on a few testcases, even if they finally fail, might become dominant. On average our bus router is about 20 times faster than nrouter. The figure also clearly shows that our bus router's runtime is insensitive to the bus bit-width, which is a natural benefit of using a virtual net. However, our runtime does depend on the fanout. In contrast, *nrouter* is sensitive to both bus bit-width and fanout.

Figure 10 compares wire length for those buses where both routers complete successfully. For bit-width = 8 and bit-width = 16 testcases, the average wire length ratios between our router and *nrouter* are both 0.99. Our router performs better when fanout < 11 and can lose up to 2% in wire length when fanout is close to 16. This trend is likely to remain for bit-width=32. There are not enough data points for bit-width = 64 to analyze its trend.

The rest of the comparisons are made for only those testcases in which both routers succeeded. For bit-width=32/fanout>9 and

bit-width=64/fanout>3, where *nrouter* had less than 20 successful runs per group, we show the solo results of our router ("both-succeed" plus "this-succeed-only"). Figure 11 compares the average and maximum driver-to-load deviations. Both average and maximum deviations tend to rise as fanout increases. But our bus router achieves, on average, 188% less average deviation and 469% less maximum deviation than *nrouter*. Similar results are obtained for average and maximum driver-to-load delay deviations, as shown in Figure 12. On average, our router results in 286% less average delay deviation and 273% less maximum delay deviation. The data also demonstrates that, in our algorithm, the minimization of the wire length deviation correlates well with the minimization of the delay deviation.

To test how the two routers are affected by variations, we assumed a simple variation: For any position x>*ChipWidth*/2, where *ChipWidth* = 1000μm for all the unit testcases, the wiring capacitance is increased by a factor of (x−*ChipWidth*/2) × 0.1%. For example, a vertical segment at x = 1000μm has 50% larger capacitance than normal. To save space, we only present the maximum driver-load delay deviations with variations. The chart in Figure 13 shows that, with such variations, the maximum delay deviation produced by *nrouter* can be as much as 25ps, while the changes by our router are all kept below 7ps.

4.2 Real designs

Five module-based designs, taken from applications with abundant buses, were tested. Their characteristics are given in Table 1. All designs have many buses. The total number of bits in all buses is given in column "#buses-nets".

Table 1. The characteristics of the designs

design	#nets	#bused-nets	#blockages
A1	668	221	6
A2	964	246	14
A3	1160	427	12
A4	1002	496	9
A5	2832	764	4

A routing flow was established by integrating the bus routing algorithm. A simple rip-up-reroute based global router is built with all the buses routed with our bus routing algorithm. The buses, however, are wider (depending on their bit-widths) compared to regular nets, and thus consume more routing resources. Rip-up-reroute has a tendency to re-route regular nets rather than buses, and re-route narrower buses rather than wider buses. Global routing ends as soon as no two buses overlap with each other and less than 1% of all routing bins have overflows. All failed buses are treated as regular nets. Then all successful buses have their detailed routing topologies implemented and written as "ROUTED" into the DEF file, while all regular nets have their topologies discarded. The DEF file, along with the necessary LEF files, are passed to *nrouter*, which runs both global and detailed routing. The routed topologies of the buses are maximally maintained by *nrouter*; but they can be slightly altered to alleviate congestion in the detailed routing.

Table 2. The comparison of results, *nrouter* versus *this+nrouter*(*this+nr*)

	total wire length(μm)		#via		runtime (min:sec)	
design	*nrouter*	*this+nr*	*nrouter*	*this+nr*	*nrouter*	*this+nr*
A1	425823	421889	4257	3621	3:25	3:27
A2	779634	781627	7213	6092	5:11	4:45
A3	1013562	1000581	8605	6669	1:25	1:10
A4	7472888	7453371	6894	4927	3:59	3:54
A5	5770986	5755639	27482	22471	3:57	4:08

Table 3. The ave/max driver-load delay deviations (ps)

	average deviation		maximum deviation		max dev w/ variation	
design	*nrouter*	*this+nr*	*nrouter*	*this+nr*	*nrouter*	*this+nr*
A1	24.1	6.57	64.1	19.3	68.0	19.3
A2	42.5	8.34	113	18.9	121	19.7
A3	33.2	9.49	79.5	22.0	84.1	28.4
A4	54.1	7.95	265	21.1	542	70.6
A5	45.9	9.16	162	22.8	266	40.9

The bus-specific routing flow (*this+nrouter*, or *this+nr* for abbreviation) is compared against *nrouter*, which routes all nets from scratch. All designs are successfully routed. The total wire lengths, number of vias, and runtimes are given in Table 2. The average and maximum driver-load delay deviations among all buses are given in Table 3. The maximum delay deviations are also compared in lieu of variation, as formulated in 4.1.

The difference in total wire length is small, with *this+nr* being about 0.5% shorter. However *this+nr* saves 25% of the vias on average, compared to *nrouter*. On average *this+nr* is 7.5% faster than *nrouter*. Since the routers are not run in timing-driven mode, comparing delays is meaningless. But the driver-load delay variations among bus bits is still of interest. The average delay deviation by *this+nr* is about 4 times smaller than for *nrouter*. Regarding the maximum delay variation, the difference can be as large as 7 times. For *nrouter*, the maximum delay deviation under variation increases more than 200ps in design A4, compared to only about 50ps for *this+nr*. Other designs also show the advantage of *this+nr* in tolerating variations.

5. CONCLUSION

In this paper, a semi-detailed bus routing algorithm was presented. A bus is transformed to a virtual net, each pin of the virtual net is supplemented with a pair of routing blockages to force the direction of the pin access, and normal routing blockages are reshaped. A single-net router is used to produce the topology of the virtual net. Then a series of orientation operations is performed to obtain the set of valid orientations for each non-pin node. An orientation determination step, which offers the option of minimizing deviations, chooses the final orientation for each node. The virtual net, routed and with orientations determined, is transformed back to a routed bus. The bus routing algorithm outperforms a reference router in terms of routability, wire length, and wire length/delay deviations. It also runs about 20 times faster for the unit testcases. It exhibits much stronger immunity to variations. The algorithm can be adopted for floorplanning, global routing and other physical design algorithms. It can also serve as a checker for designers who manually plan circuit modules and buses. In addition, it can be useful in developing a router that can automatically satisfy node orientations.

6. REFERENCES

[1] H.Xiang, X.Tang and M.D.F. Wong, "Bus-Driven Floorplanning", *Int. Conf. Computer Aided Design*, 2003, pages 66-73.

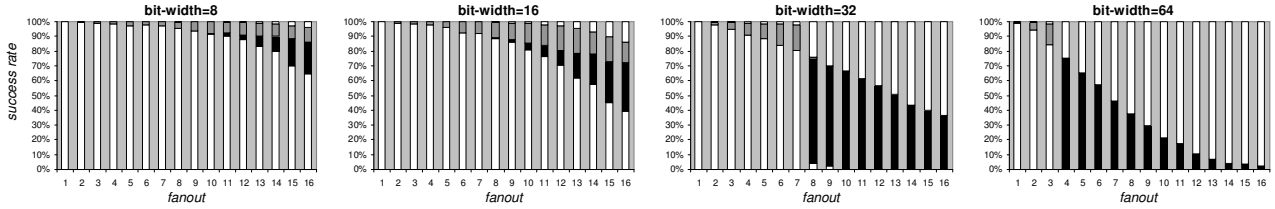
[2] F.Rafiq, M.Chrzanowska-Jeske, H.Yang and N.Sherwani, "Integrated Floorplanning with Buffer/Channel Insertion for Bus-Based Microprocessor Designs", *ISPD* 2002, pages 56-61.

[3] J.H.Y.Law and E.F.Y.Young, "Multi-Bend Bus Driven Floorplanning", *Int. Symp. Physical Design*, 2005, pages 113-120.

[4] M. Lavin and L. Liebmann, "CAD Computation for Manufacturability: Can We Save VLSI Technology from Itself?", *Int. Conf. Computer-Aided Design*, 2002, pages 424-431.

[5] S. Iwata, et al., "Performance Evaluation of a Microprocessor with On-Chip DRAM and High Bandwidth Internal Bus", IEEE Custom Integrated Circuits Conf., May 1996, pages 269-272.

[6] J.L. Ganley and J.P. Cohoon, "Routing a Multi-Terminal Critical Net: Steiner Tree Construction in the Presence of Obstacles", *Int. Symp. Circuits and Systems*, 1994, pages 113-116.
[7] J. Cong, A.B. Kahng and K.S. Leung, "Efficient Algorithms for the Minimum Shortest Path Steiner Arborescence Problem with Applications to VLSI Physical Design", *IEEE Trans. CAC of IC and Systems*, vol. 17, no. 1, Jan 1998, pages 24-39.
[8] E.N. Gilbert, H.O. Pollak, "Steiner Minimal Trees", SIAM J. Appl. Math. 16, 1968, pages 1-29.
[9] G. Persky and L.V. Tran, "Topological Routing of Multi-Bit Data Buses", *Design Automation Conf.*, 1984, pages 679-682.



The 4 sectors of a bar, from bottom up, are "both succeed"(white), "this-succeed-only"(black), "nrouter-succeed-only"(gray) and "both-fail"(white).

Figure 8. Successful rate.

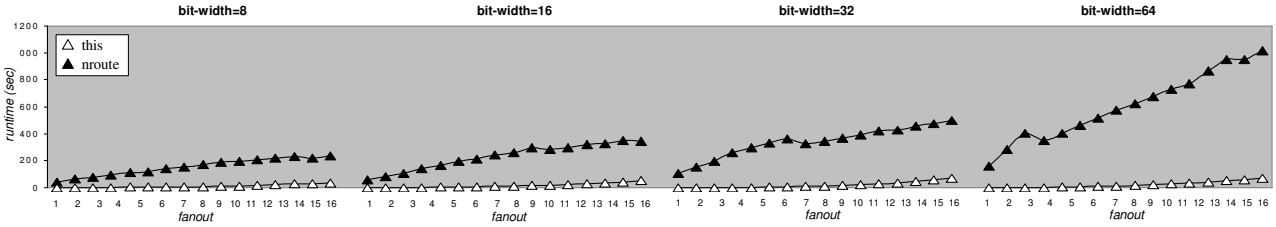


Figure 9. Runtime (for all buses).

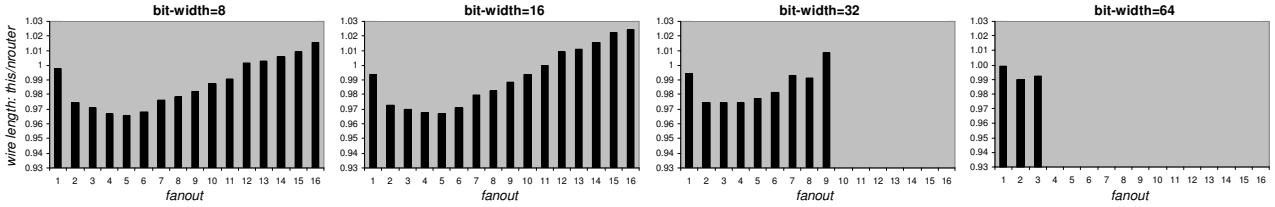


Figure 10. Wire length ratios (for buses of the "both succeed" category and the category contains at least 20 samples. Same below).

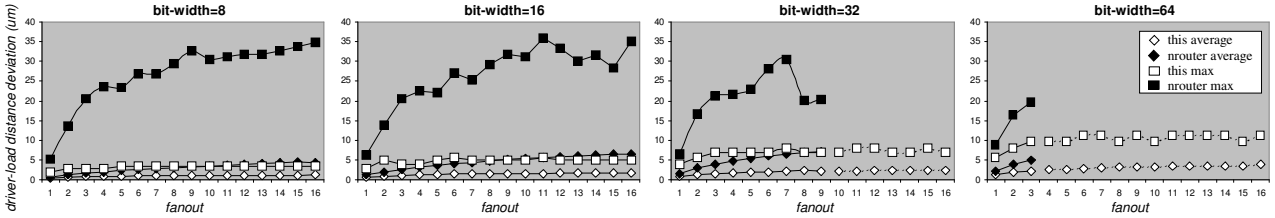


Figure 11. Driver-load wire length deviations (Dots on the dotted lines are "both-succeed" and "this-succeed-only". Same below).

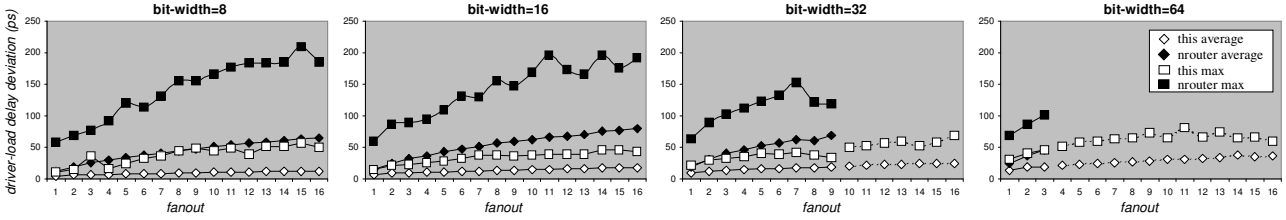


Figure 12. Driver-load delay deviations.

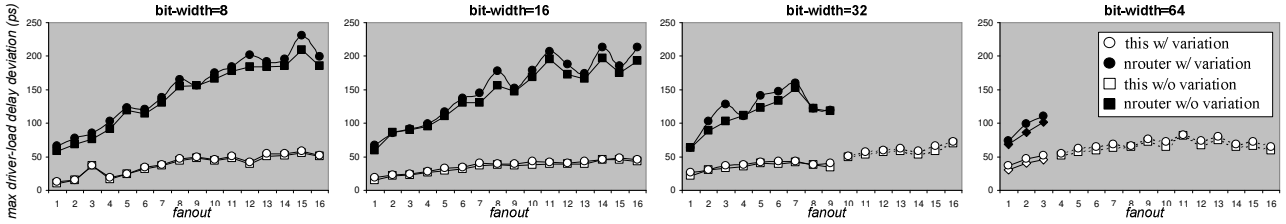


Figure 13. Driver-load delay deviations under variation.