IEICE TRANS. FUNDAMENTALS, VOL.E89–A, NO.2 FEBRUARY 2006

PAPER

An Enhanced BSA for Floorplanning

Jyh Perng FANG[†a)], **Yang-Shan TONG**[††], *and* **Sao Jie CHEN**[††], *Nonmembers*

SUMMARY In the floorplan design of System-on-Chip (SOC), *Buffer Site Approach* (BSA) has been used to relax the buffer congestion problem. However, for a floorplan with dominant wide bus, BSA may instead worsen the congestion. Our proposed *Enhanced Buffer Site Approach* (EBSA) extends existing BSA in a way that buffers of dominant wide bus can be distributed more evenly while reserving the same fast operation speed as BSA does. Experiments have been performed to integrate our model into an iterative floorplanning algorithm, and the results reveal that buffer congestion in a floorplan with dominant wide bus can be much abated.

key words: *floorplanning, buffer insertion, routing, dominant wide bus*

1. Introduction

As modern VLSI technology advances, a significant fraction of un-routable designs are caused by congestion. Typically, congestion was estimated during the routing phase. However, congestion estimation at the routing stage frequently lengthens the design time because a design failed in routing or buffering should be returned to the preceding stage for re-design. To shorten the design time, estimating congested localities is preferred before the actual routing stage. Specifically, locating congested wires and congested buffers of a floorplan design is indispensable.

There exist two methodologies for congestion estimation, the probabilistic method [1]–[3] and the actual number method [4]–[7]. Both methodologies divide a floorplan into grids; the former evaluated congestion of each grid by accumulating probabilities that wires are predicted to pass through and buffers predicted to be inserted; the latter evaluated congestion of each grid by accumulating numbers that wires are actually passing through and buffers actually inserted. For the latter methodology, there exist two further approaches, the first is to calculate buffer locations before routing, the second is to route before buffering. Following the first approach, Cong et al. [4] proposed a *buffer block planning* (BBP) method to allocate buffers into *feasible regions* (FR's). Sarkar et al. [5] expanded FR's into *independent feasible regions* (IFR's) such that more than one buffer can be inserted on a single net. Dragan et al. [6] proposed an approximation method based on multicommodity-flow to solve the same routing problem. Fang et al. [7], [8] proposed a simultaneous routing and buffering procedure to meet the constraints of delay, wire congestion, and buffer congestion all together. In the first approach, although the delay constraint is considered while buffering, the routing sometimes is restricted because the net must go through the congested region to reach predefined buffer blocks. As to the second paradigm, Alpert et al. [9] introduced a *buffer site approach* (BSA), where a Steiner tree is built for a net, buffers are then added at optimal locations. The operation speed of BSA is fast, and its experimental results revealed that BSA is nearly as effective as BBP. However, for a floorplan with dominant wide bus, i.e., a specified region of the floorplan is occupied mainly by wide bus, while being inclined to optimize the buffer numbers, BSA may contrarily worsen the congestion of area(s) occupied by bus.

1.1 Optimized Buffering

On a given routing path, BSA [9] adopted a van Ginneken style dynamic programming algorithm [10] to insert buffers. For a net with source and target, assuming the net is routed horizontally from left to right and each grid in the routing path is denoted as $Rp[x]$, the operations of BSA can be described as follows.

1. Build an accumulated cost (aC) array for each grid from target to source according to the following formula:

$$\begin{cases} Rp[x_t].aC[i] = 0, \text{ for } 0 \le i < MGD \\ Rp[x].aC[i] = Rp[x+1].aC[i-1], \\ \quad \text{for } 1 \le i < MGD,\ x \ne x_t \\ Rp[x].aC[0] = Rp[x].buf + min(Rp[x+1].aC), \\ \quad \text{for } x \ne x_t \end{cases} \tag{1}$$

where Rp is an array with elements representing grids on the routing path, $Rp[x_t]$ is the target grid and $Rp[x]$ is any grid other than the target grid; MGD is the maximum number of grids that a buffer can drive; and min( ) is a function to return the minimum value from an array.

2. Examining aC arrays built at step (1) and starting from the grid next to source (buffer), we can decide the optimal buffer location(s) according to the index of a minimum element in the aC array, and repeat this procedure until target is reached.

Take Fig. 1 as an example. In Fig. 1(a), a routing path passing through five grids from S to T is given, where the buffer densities of those grids are respectively 3, 2, 4, 2, and

Manuscript received June 17, 2004.
Manuscript revised August 23, 2005.
Final manuscript received November 4, 2005.

[†]The author is with Department of Electrical Engineering, National Taipei University of Technology, Taipei, Taiwan, ROC.
[††]The authors are with Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, ROC.
a) E-mail: jpfang@ntut.edu.tw

DOI: 10.1093/ietfec/e89–a.2.528

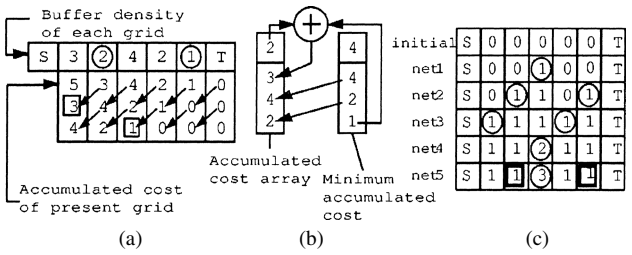


Fig. 1 Demonstration of buffer site approach.

1, and the *MGD* of this net is assumed to be 3. Firstly, we initialize the *aC* array of target as $\{0, 0, 0\}$, and build *aC* arrays of each grid from target to source according to Eq. (1). The operation of Eq. (1) is also depicted as shown in Fig. 1(b). When retracing starts from the grid next to source, as the grid next to source has $\{5, 3, 4\}$ in its *aC* array, in which the index of minimum one (framed) is 1, optimal buffer location will be the grid apart by one grid, i.e., the one with buffer density of 2 and encircled. After retracing resumes from the next grid, we examine its *aC* array and decide the next buffer location according to the index of a minimum one. As the index of minimum one in $\{4, 2, 1\}$ is 2, optimal buffer location will be the grid apart by 2 grids, i.e., the one with buffer density of 1 and encircled.

BSA can efficiently and optimally decide buffer location for a given net to minimize the buffer numbers. However, for a dominant wide bus, in which a set of nets shares the same source and target, BSA may derive an unsatisfied result. Take Fig. 1(c) as an example, in which five nets with the same terminals are buffered using BSA while their *MGD*'s are assumed to be 3 and the buffer densities of grids between source and target are initialized as 0's. After applying BSA net by net, as shown in Fig. 1(c), the buffers (encircled) of the preceding four nets are all inserted according to the mechanism demonstrated in Fig. 1(a) and Fig. 1(b), but buffer of the fifth net is inserted at a location (encircled) that will minimize the number of buffers rather than locations (framed) that will alleviate congestion. This condition is caused by a tie that exists when examining minimum value in the *aC* array, i.e., when more than two elements in the *aC* array have the same minimum value, BSA will return the one with the larger index to minimize the number of buffers.

1.2 Our Contribution

Thus, we proposed an *enhanced buffer site approach* (EBSA), in which buffers can be either optimally inserted to minimize buffer congestion or optimally inserted to minimize the buffer numbers or even optimally inserted according to the weighting between buffer numbers and degree of uniformity of distributed buffers. After integrating this approach into a floorplanner, the experimental results showed that EBSA distributes buffers of a dominant wide bus evenly, estimates buffer congestion efficiently, and generates an optimized floorplan successfully.

We did not alleviate the condition of buffer congestion of a floorplan by simply adding penalty costs for buffer-congested grids to be used, or alternately by applying non-linear (e.g., quadratic) penalty function to the number of buffers in each grid of a given routing path. The reason why EBSA not simply added penalty costs for buffer-congested grids to be used is that, EBSA implicitly adopts penalty cost because EBSA will not choose a congested grid as a location for more buffers. Besides, EBSA has properly planned the buffer locations in each routing path such that better convergence can be reached in an iterative floorplanning algorithm. As to the approach of adopting non-linear penalty function, the operation typically includes instructions of multiplication and/or division, which, in comparison to the instruction of addition/subtraction in EBSA, apparently takes more operation time to reach an acceptable result.

This paper is organized as follows: Sect. 2 formulates the problems. Section 3 describes the cost function for buffer congestion. Section 4 discusses EBSA. The experimental results are presented in Sect. 5. The last section draws a conclusion.

2. Problem Formulation

Given a net, a routing path, and the delay budget (as defined in [5]) of that net, we want to perform buffering such that

1. the number of buffers is minimized,
2. the buffer(s) is (are) inserted at the least congested region(s).

In our problem modeling, only two-pin nets are considered because a multi-pin net can be divided into two-pin nets. In such a way, a more detail evaluation procedure, such as construction of Steiner tree, can be considered during routing phase. Also, we adopt the model presented by Alpert et al. [9] to evaluate delay, in which the grids that a buffer or source can drive is expressed as *MGD*, the maximum number of grids drivable. In other words, the constraint of delay is correspondingly met if distance of buffers $\leq MGD$.

Although in most cases buffers cannot be placed into a prefixed module, some soft modules have emerged that allow buffers to be put into predefined locations. For the area that do not allow buffering, the buffering cost can be set to be ∞, and EBSA works as well.

Besides, we use standard deviation to evaluate the buffer congestion of a floorplan, the following equation is thus proposed.

$$SD = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(b_i - b_m)^2} \tag{2}$$

where b_m is mean of buffers inserted in each grid, b_i is the buffers inserted in the i^{th} grid, and n is grid numbers.

The buffer-insertion problem is thus formulated as follows:

Given : A tessellated floorplan F, a set of nets $N = \{n_1, n_2, n_3, \cdots\}$, a set of delay budgets DB_i for each net i, a buffer upper-bound matrix BUB with each element denoted

as $BUB[y][x]$, and MGD's for each net.

Goal : For the given floorplan F, buffers are optimally inserted such that

1. for all nets $i \in N$, the normalized delay value in net i (D_i) meets the delay budget, i.e., $D_i \leq DB_i$,

2. the buffer congestion in a flooplan is optimized, i.e., $SD(F)$, standard deviation for buffers inserted in each grid in a floorplan F, is minimized.

3. Congestion Evaluation Model

The buffer density of each grid in a tessellated floorplan is denoted as $Grid[y][x].buf$. Before a net is routed, buffer densities of grids $Grid[y][x].buf$ are correspondingly copied into buf members of an array $Rp[idx]$; after a net is routed and buffered, the updated buffer density in each grid of array $Rp[idx].buf$ is written back to $Grid[y][x].buf$, which will contribute to the evaluation of buffer congestion as shown below. The procedure used to evaluate buffer congestion of a floorplan includes the estimation of buffer density increased by each net based on the algorithm introduced in Sect. 4.

In our algorithm, the formulas of buffer congestion evaluation for each grid are defined as follows. For a single grid $Grid[y][x]$ on an $m \times n$ grid structure, the buffer congestion is evaluated as:

$$\begin{cases} bufCongestion = PASS, \\ \quad \text{if } Grid[y][x].buf \leq BUB[y][x] \\ bufCongestion = FAIL, \\ \quad \text{if } Grid[y][x].buf > BUB[y][x] \end{cases} \tag{3}$$

in which $Grid[y][x].buf$ is the Buffer Estimation Value and $BUB[y][x]$ is the Buffer Upper Bound (constraint of buffer congestion).

4. Evaluating Buffer Congestion

This section demonstrates the model we used to evaluate congestion and how we integrate the proposed model to an iterative floorplanning algorithm.

4.1 Deciding Routing Path

Before buffering, the routing path should be decided. To efficiently evaluate buffer congestion, nets are categorized into three types: local nets, h-v nets, and bended nets, where local net is a net with grid distance $\leq MGD$; h-v net is a net with grid distance $> MGD$, and these two terminal grids are located at the same grid row or grid column; bended net is a net with grid distance $> MGD$ and its two terminals are located at different grid rows and different grid columns. For a local net, no buffer insertion is necessary; whereas for an h-v net, its routing path will pass through grids between source and target. On the other hand, for a bended net, we use Fang's *BR* algorithm [8] to decide routing path, which is a two-phase approach. At the first phase, *BR* applies dynamic programming to accumulate routing cost from source to target; and at the second phase, *BR* retraces backwards from target to source to decide a routing path according to the accumulated routing cost. *BR* needs a time of $O(l \times w)$ for a net occupying $l \times w$ grids.

Thus, for a floorplan F with k nets, the time complexity of defining routing paths is at most $O(k \times l \times w)$.

4.2 Buffer Congestion Estimation

For each net, after its routing path is decided, the path can be treated as a one-dimensional array with each grid expressed as $Rp[x]$. Analogous to Eq. (1), in the Enhanced Buffer Site Approach (EBSA), buffers are inserted into this array according to Eq. (4).

$$\begin{cases} Rp[x_t].aC[i] = 0, \text{ for } 0 \leq i < MGD \\ Rp[x].aC[i] = Rp[x+1].aC[i-1], \\ \quad \text{for } 1 \leq i < MGD,\ x \neq x_t \\ Rp[x].aC[0] = Rp[x].buf - threshhold \\ \quad \times u\_factor + min(Rp[x+1].aC), \text{ for } x \neq x_t \end{cases} \tag{4}$$

where, Rp is an array with elements representing grids on the routing path, $Rp[x_t]$ is the target grid and $Rp[x]$ is any grid other than the target grid; MGD is the maximum number of grids that a buffer can drive; and min( ) is a function to return the minimum value from an array; each element in Rp array contains a member array (aC) for storing accumulated cost. Besides, u_factor is an adjustable factor ranging from 0 to 1, which is used for adjusting the degree of uniform; when u_factor is set to 0, the buffers are optimally inserted such that the number of buffers is minimized, and under this condition, EBSA works exactly the same as BSA does; if u_factor is set to 1, the buffers are optimally inserted such that the congested condition is minimized.

Thus, the operation of EBSA is described as shown in Fig. 2. Given a routing path from source to target, with the routing path expressed as Rp[SIZE], where SIZE is the size of routing path. Before building the aC array for each grid in the routing path, EBSA firstly searches the buffer density of each grid for a minimum one, take it as a threshold value, as shown in Step 1~Step 4 of Fig. 2. The threshold value will contribute to build the aC array for each grid, as shown in Step 10 of Fig. 2. Also, Step 5 ~ Step 12 are used to build an aC array for each grid from target to the grid next to source according to Eq. (4).

After the aC array for each grid is completed, Step 13 ~ Step 18 perform retracing and buffering. Retracing is performed in an iterative manner, which starts from the grid next to source and stops when the action reaches target. In each iteration, Step 14 examines the aC array of a grid next to a source or buffer, locates the index of a minimum one, Step 15 and Step 17 use that index as a footstep to decide which grid to be buffered.

Again, take the routing path of Fig. 1 as an example. For convenience, the routing path is redrawn in Fig. 3(a), MGD is assumed to be 3, and u_factor is set to 1. Firstly, we search for the threshold value of the routing path, which is 1, then initialize the aC array of target as $\{0, 0, 0\}$, and

Given: A routing path Rp[SIZE] from source S (not included) to target T (included), in which SIZE is the length of Rp array and each grid $Rp[x]$ has its own buffer density buf as well as associated aC array $aC[MGD]$.
Goal: decides optimal location(s) for buffer insertion.

```
Algorithm EBSA
1. threshold ← Rp[0].buf;
2. for (x ← 1; x < SIZE ; x ← x + 1)
3.   if (Rp[x].buf < threshold)
4.     threshold ← Rp[x].buf;
5. for (idx ← 0; idx < MGD; idx ← idx + 1) //Eq. (4)
6.   Rp[SIZE - 1].aC[idx] ← 0;
7. for (x ← SIZE-2;x ≥ 0; x ← x − 1) { //Eq. (4)
8.   for (idx ← 1; idx < MGD; idx ← idx + 1)
9.     Rp[x].aC[idx] ← Rp[x + 1].aC[idx − 1];
10.    Rp[x].aC[0] ← RP[x].buf − threshold∗
11.                  u_factor + min(Rp[x + 1].aC);
12. }
13. for (x ← 0; x < SIZE -1; x ← x + 1){
14.   idx ← idx_min(Rp[x].aC);
15.   x ← x + idx;
16.   if (x >SIZE-2) return;
17.   else  Rp[x].buf ← Rp[x].buf + 1;
18. }
```

Fig. 2 Algorithm EBSA.

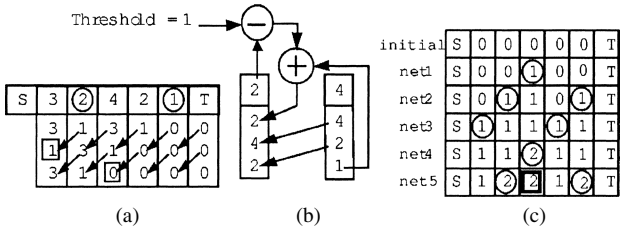


Fig. 3 Demonstration of enhanced buffer site approach.

build the aC arrays for each grid from target to source according to Eq. (4). The operation of Eq. (4) is also depicted as shown in Fig. 3(b). When retracing from the grid next to source, as the grid next to source has {3, 1, 3} in its aC array, in which the index of minimum one (framed) is 1, buffer location will be the grid apart by one grid, i.e., the one with buffer density of 2 and encircled. Then resuming from next grid, we examine its aC array and decide the next buffer location according to the index of a minimum one. As the index of a minimum one in {3, 1, 0} is 2, optimal buffer location will be the grid apart by 2 grids, i.e., the one with buffer density of 1 and encircled.

In contrast to BSA, EBSA optimally decides buffer location for nets on a dominant wide bus. Take Fig. 3(c) as an example, which is analogous to Fig. 1(c). After applying algorithm EBSA introduced in Fig. 2, the buffers of the preceding four nets are all inserted at the same locations (encircled) as done by BSA, but buffer of the fifth net is inserted at locations (encircled) that will alleviate congestion rather than framed location.

```
Procedure Floorplanning
1. fplan ← Random_floorplanning();
2. T ← initial_temperature;
3. while (T ≥ threshold) {
4.   count ← 0;
5.   while (Not_equilibrium(count, fplan, T)) {
6.     new_fplan ← Perturb(fplan);
7.     △C ← Cost(new_fplan) − Cost(fplan);
8.     if (△C < 0)
9.       if (congestionEva(new_fplan)= PASS)
10.        fplan ← new_fplan;
11.    else {
12.      prob ← Min(1, e^{−△C/T});
13.      if (Random(0, 1) ≤ prob)
14.        if(congestionEva(new_fplan)=PASS)
15.          fplan ← new_fplan;
16.    }
17.    count ← count + 1;
18.  }
19.  Update(T);
20. }
```

Fig. 4 Integrating EBSA into an SA-based floorplanner.

```
Procedure congestionEva
1. for each net k ∈ floorplan F
2.   if (BR(k) = FAIL)
3.     return FAIL; // wire congested
4.   else {
5.     if (EBSA(dupline(k), k) = FAIL)
6.       return FAIL;
7.   }
8. return PASS;
```

Fig. 5 Procedure to evaluate wire/buffer congestion.

4.3 Procedure of Floorplanning

There is a great flexibility to integrate the estimation codes in EBSA into an iterative floorplanning algorithm. The Simulated Annealing (SA) based procedure as shown in Fig. 4 is an example.

In Fig. 4, $fplan$ is the floorplan generated at each iteration. $Perturb()$ is a perturbation function used to generate the next floorplan new_fplan from the current floorplan. $Not\_equilibrium()$ is a function that decides the termination condition at a given temperature, and $Update()$ is used to cool down the temperature. After each perturbation, $Cost()$ evaluates the area/wire-length change between an old floorplan and a new floorplan, the cost change is expressed as ΔC. For each newly generated floorplan, as shown in Step 9 and Step 14 of Fig. 4, $congestionEva()$ is performed to evaluate its wire congestion and buffer congestion according to the procedure as shown in Fig. 5.

In Fig. 5, wire congestion is evaluated net by net using Fang's BR algorithm [8], and for a net k succeeding in wire congestion evaluation (i.e., $BR(k) = PASS$), if that net does not share the same source and target with other nets, $dupline(k)$ return 0, its u_factor is set to 0, $EBSA(0, k)$ is thereafter performed to evaluate buffer congestion; On the contrary, once net k shares the same source and target with

any other net, $dupline(k)$ returns 1, and EBSA$(1, k)$ is performed instead, which implies that u_factor of EBSA can be adjusted depending on the operation environment.

5. Experiment Results and Discussions

In Sect. 5.1, we take MCNC benchmarks as platforms for examining the feasibility of embedding EBSA into a floorplanner. Also, the experimental results of EBSA and BSA are compared. Then, experiments revealing the superiority of EBSA are described in Sect. 5.2.

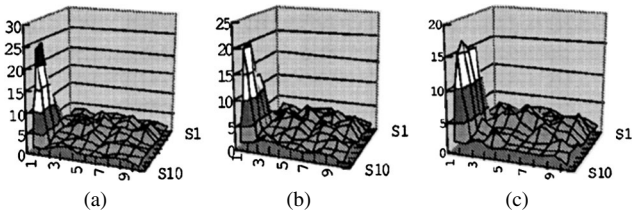
5.1 Applying EBSA to MCNC Circuits

We embedded the codes of EBSA into a SA-based B*-tree floorplanning algorithm [11] and took MCNC circuits as test benches. The experiment is performed at a PC with Intel 1.3 GHz Pentium Processor and 256 MB memory. It is assumed that each block can supply some buffers according to its area, and for each floorplan, the necessary buffer(s) can be either inserted at dead space or supplied by blocks.

With 10×10 grids on ami33 and assuming an MGD of 3, we randomly choose one net and duplicate it 32 times to simulate a 32-bits bus while all the other nets are kept unchanged, the resultant buffers inserted for final floorplan of ami33 are shown in Fig. 6, in which x-y plane represents a floorplan and z-axis denotes number of inserted buffers. In Fig. 6(a), we adopt BSA to insert buffers, the most congested grid of the final floorplan has 25 buffers while in Fig. 6(b), EBSA(0.5) decreases the number of buffers in the most congested grid to 21. Moreover, in Fig. 6(c), EBSA(1) further decreases the number of buffers in the most congested grid to 17.

Further, five MCNC circuits are floorplanned respectively using both the procedure in Fig. 4 and BSA. The floorplans are tessellated into 20×20 grids and MGD of nets are assumed to be 3. For each circuit, one net is picked and duplicated 32 times to model a dominant wide bus. Table 1 compares the resultant floorplan, in which TWL, DS, CT, SD, and BMC respectively represent total wire length, dead space, CPU time, standard deviation, and the number of buffers in the most congested grid, in which standard deviation is derived from Eq. (2).

Observed from the experiment results, the floorplan area and total wire length are interactively decided by the floorplanning algorithm and associated routing algorithm as well as strategy for buffer insertion. Besides, the constraint of buffer density is deliberately set to a higher value, the resultant floorplans generated by BSA and EBSA are identical in terms of area and total wire length. However, the BMC column in Table 1 shows that EBSA can obtain a floorplan with less congested buffers in the grids. In other words, when the buffer density in each grid in a floorplan is strictly constrained, the resultant floorplan using BSA may have larger area and longer total wire length. For example, an additional experiment shows that when buffer density is constrained to be 14, the TWL, DS, and CT of ami33 using BSA will respectively become 142.620, 8.82, and 264.



(a) (b) (c)

Fig. 6 Resultant buffers inserted for final floorplan of ami33 with different $u\_factor's$.

Table 1 Floorplanning MCNC circuits using EBSA.

Circuit	Approaches	TWL	DS(%)	CT(s.)	SD	BMC
ami33	BSA	113.259	3.94	220	6.8431	16
	EBSA	113.259	3.94	224	6.8423	14
ami49	BSA	1891.78	3.49	397	9.9545	31
	EBSA	1891.78	3.49	406	9.7930	30
xerox	BSA	807.751	6.87	4	5.6649	20
	EBSA	807.751	6.87	4	5.6551	19
hp	BSA	238.567	6.42	9	3.2049	8
	EBSA	238.567	6.42	9	3.1951	8
apte	BSA	797.362	2.51	55	4.6218	13
	EBSA	797.362	2.51	59	4.2072	11

The time spent by EBSA is slightly more than the time spent by BSA because an additional step of finding/subtracting threshold is needed by EBSA.

Also, it is interesting to observe that, the number of buffers in the most congested grid as well as the standard deviation obtained by EBSA are not dramatically enhanced in comparison to those obtained by BSA. The reason is that, once a bus is routed and buffered, for the subsequent nets, the routing processes are inclined to find a path of less cost, and the buffering processes are inclined to insert buffers into grids of less cost. Thus, those grids with less buffer densities will have higher probabilities to be chosen as buffer sites than the most congested grid do.

However, in some cases, grids are occupied mainly by wide bus, the bus connecting AND plane and OR plane of a programmable logic design is a typical example. As the buffer congestion in these cases is irremediable and BSA is inclined to cause congestion, for these cases, EBSA will be a preferred approach to avoid congestion.

5.2 The Effect of EBSA

To demonstrate the effect of EBSA, we use 1000 nets routed from the same source and reaching the same target to model the bus connecting AND plane and OR plane of a programmable logic design. The number of grids between source and target is assumed to be 10. The buffers inserted using BSA and using EBSA with different MGD's are listed in Table 2, where EBSA(0.5) uses u_factor of 0.5 while EBSA(1) uses u_factor of 1. Obviously, EBSA(1) can uniformly distribute buffers, and the penalty for uniformly distributing buffers is that, the buffer numbers inserted by EBSA(1) in comparison to those inserted by BSA are increased from 0% to 11%. When the u_factor is adjusted

Table 2 Buffers inserted into grids with different *MGD*'s using BSA and EBSA.

MGD size	Source	Approaches	Grid 1	Grid 2	Grid 3	Grid 4	Grid 5	Grid 6	Grid 7	Grid 8	Grid 9	Grid 10	Target	Total buffer numbers
MGD = 3		BSA	273	363	364	273	363	364	273	363	364	273		3273
		EBSA(0.5)	300	350	350	300	350	350	300	350	350	300		3300
		EBSA(1)	333	333	334	333	333	334	333	333	334	333		3333
MGD = 4		BSA	200	200	300	300	200	200	300	300	200	200		2400
		EBSA(0.5)	222	222	278	278	222	222	278	278	222	222		2444
		EBSA(1)	250	250	250	250	250	250	250	250	250	250		2500
MGD = 5		BSA	200	200	200	200	200	200	200	200	200	200		2000
		EBSA(0.5)	200	200	200	200	200	200	200	200	200	200		2000
		EBSA(1)	200	200	200	200	200	200	200	200	200	200		2000
MGD = 6		BSA	125	125	125	125	250	250	125	125	125	125		1500
		EBSA(0.5)	143	143	143	143	214	214	143	143	143	143		1572
		EBSA(1)	166	166	167	167	167	167	166	166	167	167		1666

Table 3 Buffers inserted into grids using different schemes.

MGD size	Source	Approaches	Grid 1	Grid 2	Grid 3	Grid 4	Grid 5	Grid 6	Grid 7	Grid 8	Grid 9	Grid 10	Target	SD
MGD = 3		BSA	23	54	77	94	75	75	71	57	37	26		22.5896
		EBSA(1)	40	64	80	85	83	80	77	59	45	33		18.4456
		BSA+EBSA(1)	38	62	79	86	83	79	76	59	45	32		18.7853

to 0.5, as shown in row EBSA(0.5), the buffers inserted are not as uniform as what done by EBSA(1), and the number of buffers inserted is less. Furthermore, when the u_factor is adjusted to 0, i.e., EBSA(0), then EBSA acts exactly the same as BSA does and has the same result as BSA.

To further demonstrate the effect of EBSA for nets which are partly overlapped, 200 nets each with randomly given start point and end point are routed on a specified path which passes through 12 grids (including grids marked Source and Target), and randomly chosen nets are deliberately duplicated. The typical number of buffers inserted using different schemes is listed in Table 3, where the first scheme uses BSA only, the second scheme uses EBSA(1) only, while the last scheme, BSA+EBSA(1), uses EBSA for bus (nets sharing the same terminals) and uses BSA otherwise. The BSA+EBSA(1) scheme is a variation of EBSA(1). The difference between these two schemes is that the former applied algorithm EBSA to bus only and applied algorithm BSA to other nets while the latter applied algorithm EBSA to all nets.

The resultant standard deviations in Table 3 implicate that both EBSA(1) and BSA+EBSA(1) can distribute buffers more uniformly than BSA only, and the congestion of dominant wide bus can be more effectively alleviated.

6. Conclusion

We proposed an enhanced buffer site approach to distribute buffers evenly while minimizing the number of inserted buffers such that congestion of a floorplan design can be effectively eliminated.

Our approach is in distinction from the existing approach especially when the floorplan contains a dominant wide bus. Our approach features (1) avoiding the tendency of congestion when multiple nets share the same routing path while (2) maintaining the fast operation speed of original buffer site approach, and (3) providing a linearly adjustable threshold switch, through which buffer insertion can be optimized according to the significance of minimizing number of buffers or lessening buffer congestion. The experimental results showed that EBSA can be integrated into an iterative floorplanner and successfully generate an optimized floorplan even if the floorplan contains a dominant wide bus.

Acknowledgments

This work was supported by the National Science Council, R.O.C., under Grant NSC91-2215-E002-042. The authors would like to thank Prof. Jason Cong for providing the source code of Buffer Block Planning [4] and benchmarks.

References

[1] J. Lou, S. Thakur, S. Krishnamoorthy, and H.S. Sheng, "Estimating routing congestion using probabilistic analysis," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.21, no.1, pp.32–41, Jan. 2002.

[2] C.W. Sham, W.C. Wong, and E.F.Y. Young, "Congestion estimation with buffer planning in floorplan design," Proc. Design Automation and Test in Europe, pp.696–701, Paris, France, March 2002.

[3] C.W. Sham and E.F.Y. Young, "Routability driven floorplanner with buffer block planning," Proc. International Symposium on Physical Design, pp.50–55, Del Mar, CA, USA, April 2002.

[4] J. Cong, T. Kong, and D.Z. Pan, "Buffer block planning for interconnect-driven floorplanning," Proc. International Conference on Computer-Aided Design, pp.358–363, San Jose, CA, USA, Nov. 1999.

[5] P. Sarkar, V. Sundararaman, and C.-K. Koh, "Routability-driven repeater block planning for interconnect-centric floorplanning," Proc. International Symposium on Physical Design, pp.186–191, San

Diego, CA, USA, April 2000.
[6] F.F. Dragan, A.B. Kahng, I.I. Mandoiu, S. Muddu, and A. Zelikovsky, "Provably good global buffering using an available buffer block plan," Proc. International Conference on Compter-Aided Design, pp.104–109, San Jose, CA, USA, Nov. 2000.
[7] J.P. Fang, Y.S. Tong, and S.J. Chen, "Simultaneous routing and buffering in floorplan design," Proc. International Symposium on VLSI Technology, Systems, and Applications, pp.188–191, Hsinchu, Taiwan, Oct. 2003.
[8] J.P. Fang, Y.S. Tong, and S.J. Chen, "Integration of an efficient routing and buffering procedure into a floorplanner," Proc. International Symposium on Nanoelectronic Circuits and Giga-scale Systems, pp.68–73, Miaoli, Taiwan, Feb. 2004.
[9] C. Alpert, J. Hu, S. Sapatnekar, and P. Villarrubia, "A practical methodology for early buffer and wire resource allocation," Proc. Design Automation Conference, pp.189–194, Las Vegas, NV, USA, June 2001.
[10] L. v. Ginneken, "Buffer placement in distributed RC-tree network for minimal elmore delay," Proc. IEEE International Symposium on Circuits and Systems, pp.865–868, New Orleans, LA, USA, May 1990.
[11] Y.C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-tree: A new representation for non-slicing floorplans," Proc. Design Automation Conference, pp.458–463, Los Angeles, CA, USA, June 2000.

Jyh Perng Fang received the B.S. degree in Electrical Engineering from Chung Yuan Christian College in 1977 and, respectively, the M.S. and Ph.D. degree in Electrical Engineering from National Taiwan University in 1986 and 2004. Currently, he is an associate professor in the Department of Electrical Engineering, National Taipei University of Technology. His current research interests include: VLSI physical design automation and Embedded System.

Yang-Shan Tong received the B.S. degree in Physics from National Taiwan University in 1997 and the M.S. degree in Electrical Engineering from National Taiwan University in 2002. Currently, he is a PhD student in the Graduate Institute of Electronics Engineering, National Taiwan University.

Sao-Jie Chen received the B.S. and M.S. degrees in electrical engineering from the National Taiwan University, Taipei, Taiwan, ROC, in 1977 and 1982 respectively, and the Ph.D. degree in electrical engineering from the Southern Methodist University, Dallas, USA, in 1988. Since 1982, he has been a member of the faculty in the Department of Electrical Engineering, National Taiwan University, where he is currently a full professor. During the fall of 1999, he was a visiting scholar in the Department of Computer Science and Engineering, University of California, San Diego, USA. During the fall of 2003, he held an academic visitor position in the Department of System Level Design, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA. His current research interests include: VLSI physical design automation, Wireless LAN and Bluetooth IC design, and SOC hardware/software co-design. Dr. Chen is a member of the Chinese Institute of Engineers, the Chinese Institute of Electrical Engineering, the Association for Computing Machinery, a senior member of the IEEE Circuits and Systems and the IEEE Computer Societies.