PERFORMANCE OPTIMIZATION OF LATENCY INSENSITIVE SYSTEMS THROUGH BUFFER QUEUE SIZING OF COMMUNICATION CHANNELS*

Ruibing Lu and Cheng-Kok Koh

School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285
{lur,chengkok}@ecn.purdue.edu

ABSTRACT

This paper proposes for latency insensitive systems a performance optimization technique called channel buffer queue sizing, which is performed after relay station insertion in the physical design stage. It can be shown that proper queue sizing can reduce or even completely avoid the performance loss due to imbalanced relay stations insertion in reconvergent paths. Moreover, the problem of queue sizing and placement of the additional buffers for maximum performance is formulated and studied to properly allocate available chip areas in the layout to communication channels. An algorithm based on mixed integer linear programming is proposed. Experimental results show that queue sizing is effective in improving the performance of latency insensitive systems even under tight area constraints. Moreover, the proposed algorithm is sufficiently efficient in obtaining the optimal solution for systems of practical sizes.

1. INTRODUCTION

As the system complexity increases, effective reuse of existing designs or IP cores enables designing complex systems on a chip (SoC) in reasonable time [1]. Reliable and high performance communication links are typically required to stitch existing designs or IP cores together. However, the delay of global interconnects becomes the dominating factor of the system performance as the device feature size continues to scale down to nanometer dimensions. Due to the difficulty in accurately estimating the global interconnect delay in early design stages, large numbers of timing violations typically surface in the later physical design stages; design iterations are inevitable. In addition, with the increase of chip size and the clock frequency, global interconnects may have delay larger than clock period. In fact, the wire delay can be as long as about five to ten clock cycles [2] in the near future. As a result, it necessary to pipeline the signal transmission on global interconnects. This further complicates the delay estimation of global interconnects.

Interconnect planning [3, 4, 5] is proposed to better estimate the interconnect delay by performing routing resource allocation in early design stages. While these approaches can certainly improve the accuracy of the interconnect delay estimation at early design stages, they also pose tight constraints for the later logic and physical designs. The failure to meet those constraints still lead to design iterations.

Recently, a latency insensitive design methodology [6, 7], which can tolerate the change of communication latencies in late design stages, has been proposed. A latency insensitive system (LIS) is composed of a collection of circuit blocks that exchange data on communication channels. [8] has presented an implementation of such latency insensitive systems, in which relay stations can be used to pipeline the communication channels with timing violations discovered in the physical design stage. Although the functionality of a LIS is robust with respect of communication latencies, the same is not necessarily true for the performance. The performance analysis of LISs in [9] suggests that the system throughput is only affected by relay stations in feedback cycles. However, their analysis ignore the 'back-pressure' [10, 8], which is necessary to establish fully reliable communications by informing the source module to stop when the sink module cannot receive more data. Clearly, LISs without back-pressure is much less robust than those with back-pressure. The performance of LISs with back-pressure is studied in [11]. The result shows that imbalanced relay station insertion in re-convergent paths may also affect the performance of LISs.

In this paper, we propose a new performance optimization technique for LISs called "*channel buffer queue sizing*" , or simply "*queue sizing*", which is performed after relay station insertion in the physical stages. We found that proper queue sizing can reduce or even completely avoid the performance loss due to imbalanced relay stations insertion in reconvergent paths. In other words, the performance of LISs with back-pressure after proper queue sizing can reach that of LISs without back-pressure. Moreover, we formulate and study the problem of optimizing the performance of LISs by allocating available chip areas in the layout to communication channels for buffer queues. An algorithm is proposed to solve the problem by converting it to a mixed integer linear programming (mixed ILP) problem. Experimental results show the proposed algorithm is effective in obtaining the optimal solution for systems of practical sizes.

2. BACKGROUND: LATENCY INSENSITIVE SYSTEMS

In the section, we briefly introduce latency insensitive design methodology and the performance analysis of LISs in [11].

In LISs, relay stations are used to partition and "pipeline" the communication channels that have long interconnect delay. This introduces different latencies to communication channels. To synchronize the data, a circuit unit stalls when any of its input data is not available, and in such a case, *non-informative data* is put on the output ports of the stalled unit. After all non-informative data are screened, the function of an LIS is equivalent to the original one.

*This work was supported in part by NSF CCR-9984553.

ICCAD'03, November 11-13, 2003, San Jose, California, USA.

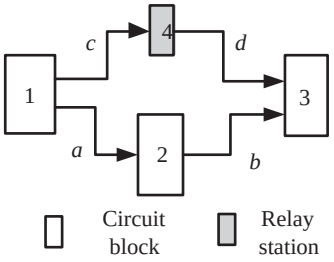

Figure 1: A simple LIS.

Table 1: The behavior of a simple LIS

Cycle No.	Circuit blocks/Relay stations				Channel buffer queues			
	1	2	3	4	a	b	c	d
1	1	1	1	τ	τ	τ	τ	τ
2	2	2	τ	1	τ	1	τ	τ
3	3	τ	2	2	2	2	τ	τ
4	τ	3	3	3	3	τ	τ	τ
5	4	4	4	τ	τ	τ	τ	τ

We define a channel as the interconnection between any two consecutive circuit units (relay stations or circuit blocks). Therefore, a communication link between two circuit blocks will become $n+1$ channels after inserting n relay stations. Each channel in LISs has a buffer queue, which is used to store the data already generated by the channel source unit, but not ready to be consumed by the channel sink unit. In the LISs suggested in [11], a circuit unit stalls if and only if either at least one of its input channels cannot provide the required data, or at least one of its output channels has a full buffer queue. For an LIS, the queue size of any channel should be at least 1 in order to guarantee the correct system behavior. A channel queue, whose size is 1, is called a "minimum queue".

Figure 1 shows a simple LIS with three circuit blocks and one relay station. All channels in this LIS use minimum queues. The behavior of this simple LIS in the first five clock cycles is shown in Table 1. In this table, a positive integer 'i' denotes the i-th informative data generated by the unit or the source unit of the channel. 'τ' indicates a non-informative data. Note that when circuit blocks take (i-1)-th informative data from their input channels, they output their i-th informative data to the output channels; relay stations simply pass to its output channels what they take from their input channels. The sequence of informative data and non-informative data of produced by a unit is called a "progressive trace" [9].

In the first clock cycle, all circuit blocks produce their first informative data, while relay stations can only stall. In clock cycle 2, the input channel d of block 3 cannot provide any informative data, so block 3 stalls. Therefore, the first informative data generated by block 2 in channel b is not processed, and it is stored in the buffer queue of channel b. As a result, channel b becomes full in the clock cycle 2. In order to avoid the informative data loss due to queue overflow, it requests the source block 2 to stall. Therefore, block 2 stalls in the third clock cycles. Similarly, the behavior of all units can be acquired. From this example, we can see that the throughput a LIS can be obtained through the simulation of progressive traces.

In order to model the structure of LISs, two kinds of graphs, *lis-graph* and *extended lis-graph*, are defined as follows:

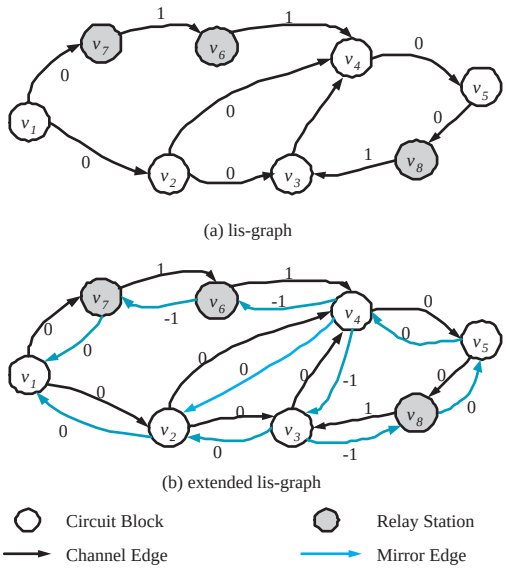

Figure 2: An example lis-graph and extended lis-graph.

Definition 1 *A lis-graph $G_l(V, E, w)$ is a weighted connected directed graph, where V is the set of all circuit units including original circuit blocks and relay stations, $(v_i, v_j) \in E$ refers to the communication channel from unit v_i to unit v_j, $w(v_i, v_j) \in \{0, 1\}$, and $w(v_i, v_j)$ is 1 if and only if the unit corresponding to v_i is a relay station.*

Definition 2 *The extended lis-graph $G_e(V, E_e, w_e)$ of an LIS is a weighted connected directed graph acquired by adding into its lis-graph $G_l(V, E, w)$ mirror edges (v_j, v_i) with weight $1-Q(v_i, v_j)-w(v_i, v_j)$ for each edge $(v_i, v_j) \in E(G)$. $Q(v_i, v_j)$ is the queue size of the communication channel (v_i, v_j).*

Figure 2 shows an lis-graph and the corresponding extended lis-graph. Note that for a minimum-queue channel, the channel edge and the mirror edge have opposite weights in the extended lis-graph. In the extended lis-graph of Figure 2, the queue size of channel (v_3, v_4) is 2, and all other channels have minimum queues.

It has been shown in [11] that the throughput of a LIS is at most:

$$1 - max_{C_i \in SC}(\frac{W(C_i)}{|C_i|}). \quad (1)$$

where SC is the set of all cycles in the extended lis-graph G_e, $W(C_i)$ is the sum of edge weights of cycle C_i, and $|C_i|$ is the number of edges in cycle C_i.

$\frac{W(C_i)}{|C_i|}$ is called the *mean weight* of cycle C_i, or simply the *cycle mean*, which is denoted as $\lambda(C_i)$. The second part of Eqn. (1) is the *maximum cycle mean*, denoted as $\lambda^\star(G_e)$, of the extended lis-graph G_e. The maximum cycle mean can be efficiently computed through several algorithms [12], of which a widely used one is Karp's algorithm [13].

This result is consistent with that in [9], which is for LISs without back-pressure, whose throughput is equal to $\lambda^\star(G_l)$. A LIS without back-pressure can be viewed as a LIS with unlimited queue size. For such a LIS, the weight of all mirror edges in the

extended lis-graph G_e is $-\infty$. Therefore, $\lambda^\star(G_l)$ and $\lambda^\star(G_e)$ are equal.

More important, experiments in [11] show that the throughput upper bound is very 'tight'. Simulation results of hundreds of thousands of randomly generated LISs with different sizes indicate that their actual throughputs are always equal to the corresponding performance upper bounds. Hence, the throughput upper bound in Eqn. (1) can be used to evaluate the performance of LISs.

3. CHANNEL BUFFER QUEUE SIZING

As the performance of an LIS is determined by the maximum cycle mean of its extended lis-graph, the reduction of the maximum cycle mean can lead to performance improvement. The weight of a mirror edge decreases by m if the buffer queue size of corresponding channel increases by m. Therefore, the mean weight of a cycle can always be reduced as long as the cycle includes at least one mirror edge. If, after queue sizing, the mean weight of any cycle with mirror edges is not bigger than the maximum cycle mean, $\lambda^\star(G_l)$, of the lis-graph, $\lambda^\star(G_l)$ and $\lambda^\star(G_e)$ are equal. In other words, the throughput of a LIS with limited queue sizes is also affected only by relay station insertion in feedback cycles of the system.

Consider the LIS shown in Figure 2. The following cycles have positive mean weights in G_e:

- $C_1 = (v_1, v_7, v_6, v_4, v_2, v_1)$, $\lambda(C_1) = 2/5 = 0.4$
- $C_2 = (v_1, v_7, v_6, v_4, v_5, v_8, v_3, v_2, v_1)$, $\lambda(C_2) = 3/8 = 0.375$
- $C_3 = (v_4, v_5, v_8, v_3, v_4)$, $\lambda(C_3) = 1/4 = 0.25$
- $C_4 = (v_4, v_5, v_8, v_3, v_2, v_4)$, $\lambda(C_4) = 1/5 = 0.20$
- $C_5 = (v_1, v_7, v_6, v_4, v_3, v_2, v_1)$, $\lambda(C_5) = 1/6 = 0.167$

Among them, C_3 is the only feedback cycle in the system, and any of the other cycles includes at least one mirror edge. Therefore, the throughput of the LIS can be improved from 0.6 to 0.75, with proper queue sizing. For this purpose, the mean weight of all these cycles must be less than or equal to 0.25. Therefore, the related channels of mirror edges in cycle C_1 and C_2 should increase their buffer queue size. Let us first consider C_1, which contains mirror edges (v_4, v_2) and (v_2, v_1). $\lambda(C_1)$ becomes 0.2 if the queue size of either channel (v_1, v_2) or (v_2, v_4) increases from 1 to 2. Assume queue size of channel (v_2, v_4) increases to 2. Now $\lambda(C_1)$ is 0.2. And C_2 becomes the cycle with the maximum mean weight. $\lambda(C_2)$ can be reduced to 0.25 if $Q(v_2, v_3)$ increases to 2. Hence, the throughput increases from 0.6 to 0.75 by adding two additional buffers, one along (v_2, v_4) and one along (v_2, v_3). However, if we increase the queue size of channel (v_1, v_2) by 1, $\lambda(C_1)$ and $\lambda(C_2)$ can be simultaneously reduced to 0.2 and 0.25, respectively. In other words, by increasing only $Q(v_1, v_2)$ by 1 while keeping other channels unchanged, we obtain the same performance but with a lower cost. Clearly, a proper queue sizing algorithm may reduce the additional area cost, and achieve better performance at the same time.

The performance improvement of LISs through queue sizing is limited by two factors: First, the maximum cycle mean of the extended lis-graph can never be smaller than that of the corresponding lis-graph. Therefore, the maximum cycle mean of the lis-graph gives a performance limit that buffer queue sizing can reach. Secondly, queue sizing is performed after the relay station insertion in the physical design stage. Therefore, the system layout is almost fixed at that time. The available chip areas for additional channel buffers may pose stringent constraints for buffer queue sizing. A candidate buffer region is typically shared by several channels; the proper allocation of the available buffer areas to channels to maximize the performance is desirable. The formalization of and the solutions to the buffer queue sizing problems for LISs are given in the next section.

4. MAXIMUM PERFORMANCE BUFFER QUEUE SIZING AND PLACEMENT

We study the following buffer queue sizing problem:
Given:

- a latency insensitive system $G_l(V, E, w)$,
- a set of available candidate buffer regions CB,
- the capacity $C(cb_t)$ of each candidate buffer region $cb_t \in CB$,
- the mapping of channels to candidate buffer regions, $m : E \rightarrow CB$; *i.e.*, $m(e) = cb_t$, for some $e \in E$ and $cb_t \in CB$.

Find the number of additional buffers $q(e)$ for each channel $e \in E$, such that

1. $\lambda^\star(G_e(V, E_e, w_e))$ is minimized;
2. $\sum_{m(e)=cb_t} q(e) \leq C(cb_t), \forall cb_t \in CB$.

Note that the queue sizes of all channels are equal to 1 before the queue sizing; therefore, the queue size $Q(e)$ is equal to $q(e) + 1$ for any channel e after queue sizing.

4.1. Mixed ILP formulation

We first study the properties of directed graphs whose maximum cycle mean is not larger than certain value $\lambda^\star$.

Theorem 1 *The mean weight of any cycle in a direct graph $G(V, E, w)$ is not larger than $\lambda^\star$ if and only if there is a function $r : V \rightarrow R$, such that $w(e) - \lambda^\star \leq r(v_j) - r(v_i), \forall e(v_i, v_j) \in E$, where R is the set of all real numbers.*

Proof: The proof for the sufficient condition is trivial. The necessary condition is proved by providing an algorithm to find such a function $r(v)$. Construct a graph $G'(V', E', w')$ by augmenting G with a new vertex s and new edges $(s, v)|\forall v \in V$, *i.e.*, $V' = V \cup \{s\}$, $E' = E \cup \{(s, v)|\forall v \in V\}$. Moreover, $w'(e) = w(e) - \lambda^\star$ if $e \in E$, otherwise $w'(e) = 0$. Clearly each cycle in G' corresponds to exactly a cycle in G, because the new vertex s in G' does not have any incoming edge. Note that the mean weight of a cycle in G' is decreased by $\lambda^\star$ compared to that of the corresponding cycle in G. Since the mean weight of any cycle in G is less than or equal to $\lambda^\star$, the mean weight of any cycle in G' is not larger than 0, in other words, graph G' does not has any positive cycle. Therefore, we can perform a single source longest path algorithm for graph G' with source vertex being s. Let $r(v)$ be equal to the longest distance from s to vertex v for each $v \in V$. For any edge $e(v_i, v_j) \in E$, $r(v_i)$ and $r(v_j)$ are the lengths of the longest paths in graph G' from s to v_i and v_j, respectively. The path from v_s to v_i, then to v_j through edge $e(v_i, v_j)$ is also a path from v_s to v_j, whose length is less than or equal to the longest path length from v_s to v_j. Therefore, $r(v_i) + w'(e) \leq r(v_j)$, or $w(e) - \lambda^\star \leq r(v_j) - r(v_i), \forall e(v_i, v_j) \in E$. □

Now we return to the problem of buffer queue sizing. Any extended lis-graph is a directed graph, for which Theorem 1 holds. Before the problem of maximum performance queue sizing/placement is studied, we first consider a related problem, in which the performance constraint is given, and the objective function is to minimize the total number of additional buffers. Such a problem is called the '*Minimum Cost Buffer Queue Sizing/Placement Problem*'. The min-cost queue sizing/placement problem can be formulated as following:

Minimize : $\sum_{e \in E} q(e)$,

subject to:

$$r(v_j) - r(v_i) \geq w(e) - \lambda^\star_{tgt}, \quad (2)$$

$$r(v_i) - r(v_j) + q(e) \geq -w(e) - \lambda^\star_{tgt}, \quad (3)$$
$$\forall e(v_i, v_j) \in E,$$

$$\sum_{m(e)=cb_t} q(e) \leq C(cb_t), \forall cb_t, \in CB, \quad (4)$$

where $G_l(V, E, w)$ is a lis-graph, $r(v_x) \in R, \forall v_x \in V$, $q(e) \in Z^+, \forall e(v_i, v_j) \in E$.

Note that Z^+ is the set of non-negative integers. $q(e)$ refers to the additional number of buffers of channel e. Constraints (2) and (3) can be viewed as the constraints for the channel edge and mirror edge in the extended lis-graph corresponding to edge e in lis-graph $G_l(V, E, w)$, respectively. Based on Theorem 1, the maximum cycle mean of the extended lis-graph is not larger than $\lambda^\star_{tgt}$ if and only these two sets of constraints can be satisfied. The target throughput of the LIS is $1 - \lambda^\star_{tgt}$. Constraint 5 guarantees that the placement of the additional channel buffers does not cause overflow violations. Therefore, the mixed-ILP problem is equivalent to the min-cost queue sizing/placement problem.

The mixed-ILP formulation for min-area queue sizing is very efficient because it has only $|E|$ integer variables, $|V|$ real variables, and $(2 * |E| + |CB|)$ constraints for an LIS $G_l(V, E, w)$.

The mixed-ILP formulation for maximum performance queue sizing/placement can be obtained by a slight change to the previous formulation:

Minimize : $\lambda^\star$,

subject to:

$$r(v_j) - r(v_i) + \lambda^\star \geq w(e),$$
$$r(v_i) - r(v_j) + q(e) + \lambda^\star \geq -w(e),$$
$$\forall e(v_i, v_j) \in E,$$
$$\sum_{m(e)=cb_t} q(e) \leq C(cb_t), \forall cb_t \in CB,$$

where $G_l(V, E, w)$ is a lis-graph, $r(v_x) \in R, \forall v_x \in V$, $q(e) \in Z^+, \forall e(v_i, v_j) \in E$.

Note that $\lambda^\star$ is a real variable in the mixed-ILP formulation. Moreover, it is also the objective function to be minimized because the throughput is $1 - \lambda^\star$.

4.2. Binary Searching for maximum performance queue sizing/placement

The mixed-ILP formulations for min-cost queue sizing/placement and maximum performance queue sizing/placement have the same number of constraints ($2 * |E| + |CB|$) and almost same number of variables (($|V| + |E|$) v.s. ($|V| + |E| + 1$)). However, our experiments show that the maximum performance queue sizing is much slower. One possible reason is that, for the maximum performance queue sizing/placement, the objective function is real, not integer. The ILP solver may spend a lot of time for very marginal improvement of the objective function, which may not be important in practice.

In order to improve speed of the maximum performance queue sizing, we propose an algorithm based on binary search for the lowest feasible value of the maximum cycle mean of the extended lis-graph. For each target maximum cycle mean $\lambda^\star_{tgt}$, min-area queue sizing is performed to check if the target is feasible. The structure of the proposed algorithm is as follows:

INPUT: lis-graph $G_l(V, E, w)$),
Set of candidate buffer regions CB,
Capacity function $C(cb_t), \forall cb_t \in CB$,
Channel-to-buffer-region function $m(e), \forall e \in E$,
Performance precision $PREC$.
OUTPUT: $\lambda^\star$.
$\lambda^\star_{max} = \lambda^\star(G_{e,M}(V, E_e, w_e))$;
$\lambda^\star_{min} = \lambda^\star(G_l(V, E, w))$;
$\lambda^\star_{tgt} = \lambda^\star_{min}$;
while $(\lambda^\star_{max} - \lambda^\star_{min} > PREC)$
Perform min-cost queue sizing/placement with target maximum cycle mean $\lambda^\star_{tgt}$;
if $\lambda^\star_{tgt}$ is feasible, $\lambda^\star_{max} = \lambda^\star_{tgt}$;
else $\lambda^\star_{min} = \lambda^\star_{tgt}$;
$\lambda^\star_{tgt} = (\lambda^\star_{max} + \lambda^\star_{min})/2$;
end while
$\lambda^\star = \lambda^\star_{max}$;
END

$G_{e,M}(V, E_e, w_e)$ is the extended lis-graph acquired by setting the size of all channel queues to 1. $PREC$ is a pre-set value to control the of the computed $\lambda^\star$.

5. EXPERIMENTAL RESULTS

We evaluate our formulation and algorithm for the maximum performance buffer queue sizing on a set of LISs with various numbers of circuit blocks from 20 to 200, which we believe are representative of the problem sizes of current and near future SoC designs. The communication channels among units are randomly generated. For each circuit block, we also assume that there is a candidate buffer region with a capacity of 2 buffers in the layout. A candidate buffer region is shared by all the fan-in channels of the block corresponding to this region, because the buffer queues are placed near the sink blocks of channels as suggested in [11]. The performance precision $PREC$ is set to be 0.003 for all experiments. We use the non-commercial LP/ILP solver *lp_solve* [14] to solve the mixed ILP problems. After buffer queue sizing/placement is performed, the performance of each system is verified through the simulation of the system with sized buffer queues. All the experiments are performed on a computer with a 1.8GHz PENTENIUM 4 processor and 256M memory.

The correctness of the formulation and algorithm are verified by our experiments because the performance obtained from simulation is always equal to the performance computed by the queue sizing algorithm. Part of the experimental results are shown in Table 2. In the table, T_{MQ} refers to the throughput of LISs with minimum buffer queues, *i.e.*, $1 - \lambda^\star(G_{e,M})$; T_{UQ} is the perfor-

Table 2: Experimental Results

Name	Num. of Blocks	Num. of Channels	Num. of Relay Stations	T_{MQ}	T_{UQ}	T_{CQ}	N_{buf}	T_{exec}(s)
LIS01	20	72	21	0.36	0.50	0.50	5	0.1
LIS02	40	188	33	0.38	0.50	0.46	5	0.5
LIS03	60	242	59	0.33	0.46	0.44	7	1.1
LIS04	80	307	59	0.40	0.53	0.52	16	2.3
LIS05	100	286	56	0.46	0.67	0.57	17	26.5
LIS06	120	413	74	0.44	0.55	0.55	18	13.1
LIS07	150	721	152	0.33	0.45	0.45	19	17.4
LIS08	170	755	95	0.33	0.44	0.44	35	72.6
LIS09	190	759	81	0.50	0.60	0.60	18	1067
LIS10	200	797	79	0.44	0.59	0.59	32	7427

mance of LISs with unlimited queue size, *i.e.*, $1 - \lambda^*(G_l)$. Here, G_l and $G_{e,M}$ are the lis-graph and the extended lis-graph, respectively. T_{CQ} is the maximal throughput computed by the queue sizing algorithm as well as the throughput obtained from the simulations. N_{buf} is the total number of additional buffers to achieve the maximum throughput. T_{exec} is the time spent by the algorithm.

The results show that the buffer queue sizing is effective in improving the throughput of LISs. For the majority of the tested cases, the performance achieved is near that of the system with unlimited queue size. In addition, the additional area cost to achieve such a performance improvement is low. More important, the improvement is achieved under stringent constraints, which requires the number of additional buffers in all fan-in channels of a block to be at most 2. Therefore, we believe that our proposed formulation and algorithm can be effective in the physical design stages when the area constraints are tight. Moreover, the run time of the proposed algorithm is fast for LISs with practical sizes even though it is not a polynomial-time algorithm. The results also show that the run time may increase dramatically as the problem size increases. We believe that approximate ILP algorithms, instead of an exact ILP solver for the optimal solutions, should be used for even larger problems.

6. CONCLUSION

We propose a performance optimization technique, channel buffer queue sizing, for latency insensitive systems. The problems of queue sizing and placement of additional buffers are formulated as mixed ILP problems. An mixed ILP-based algorithm is proposed for solving the maximum performance queue sizing/placement problem. Experimental results show that queue sizing is an effective method to improve the system performance, and the ILP-based algorithm is adequately fast for latency insensitive systems of practical sizes.

7. REFERENCES

[1] J. Cong. Challenges and opportunities for design innovations in nanometer technologies. In *Frontiers in Semiconductor Research: A Collection of SRC Working Papers*. Semiconductor Research Corporation, http://www.src.org/prg_mgmt/frontier.dgw, 1997.

[2] D. Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, 8:37–39, September 1997.

[3] Ralph H. J. M. Otten and Robert K. Brayton. Performance planning. *Integration, the VLSI Journal*, 29:1–24, 2000.

[4] J. Cong. An interconnect-centric design flow for nanometer technologies. *Proc. of the IEEE*, 89:505–528, April 2001.

[5] Ruibing Lu and Cheng-Kok Koh. Flip-flop and repeater insertion for early interconnect planning. In *Design, Automation and Test in Europe Conference*, page 690, 2002.

[6] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pages 1059–1076, September 2001.

[7] T. Chelcea and S. M. Nowick. Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. In *Proc. Design Automation Conf*, 2001.

[8] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli. A methodology for correct-by-construction latency insensitive design. In *Proc. Int. Conf. on Computer Aided Design*, pages 309–315, 1999.

[9] L. P. Carloni and A. L. Sangiovanni-Vincentelli. Performance analysis and optimization of latency insensitive systems. In *Proc. Design Automation Conf*, pages 361–367, 2000.

[10] L. P. Carloni and A. L. Sangiovanni-Vincentelli. Coping with latency in SoC design. *IEEE Micro, Special Issue on Systems on Chip*, 22(5), Sep/Oct 2002.

[11] Ruibing Lu and Cheng-Kok Koh. Performance analysis and efficient implementation of latency insensitive systems. Technical Report TR-ECE03-06, School of Electrical & Computer Engineering, Purdue University, March 2003.

[12] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 17, 1998.

[13] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Math.*, 23:309–311, 1978.

[14] ftp://ftp.es.ele.tue.nl/pub/lp_solve/.