# A High Performance Bus Communication Architecture through Bus Splitting

Ruibing Lu and Cheng-Kok Koh
School of Electrical and Computer Engineering
Purdue University,West Lafayette, IN, 47907, USA
{lur, chengkok}@ecn.purdue.edu

**Abstract— A split shared-bus architecture with multiple simultaneous bus accesses is proposed. Compared to traditional bus architectures, the performance of proposed architecture is higher because of the ability to deliver multiple bus transactions in one bus cycle. We also propose an implementation of the arbiter, which not only detects and grants multiple compatible bus transactions, but also controls splitters properly to establish the communication paths for those transactions. Experimental results show that the bus architecture can have up to 2.3 times improvement in the effective bandwidth and up to 5 times reduction in the communication latency. Moreover, the arbiter implementation has reasonable area and timing cost, making it suitable for high performance SoC applications.**

## I. Introduction

Today's VLSI technology makes it both feasible and economical to integrate a complex system on a single chip. Designing such a system on a chip (SoC) usually involves stitching pre-designed IP cores together through various forms of communication links. As the delay of global interconnects becomes the dominating factor of system performance, the design of high performance global communication architectures becomes the key to successful SoC designs.

One of the most commonly used on-chip communication architectures is the shared-bus architecture. The main advantage of shared-bus architectures includes simple topology, low cost, and extensibility. Several companies have developed their own on-chip bus architectures, such as CoreConnect [1], AMBA [2], and OpenCore [3]. Modules connected to a bus are divided into two categories: masters and slaves. Masters can initiate a bus transaction, while slave modules merely respond to transactions initiated by masters. Since a bus is typically shared by multiple masters, arbitration is required and a master can access the bus only after it receives an access grant from the arbiter. Commonly used arbitration methods include priority-based arbitration [2] and time division multiplexing (TDMA) [3]. Randomized arbitration is also introduced in LOTTERYBUS [4]. As only one module can access the bus at any time, the bandwidth is limited when the number of modules attached to a bus is large. The bandwidth can be improved by hierarchical bus architectures [1], in which multiple buses are connected with each other through bridges. Studies in [5, 6] propose algorithms to perform bus hierarchy optimization based on communication profiles. However, hierarchical bus architectures may suffer long latency for inter-bus communications, and bridges usually have high area cost due to their large number of buffers.

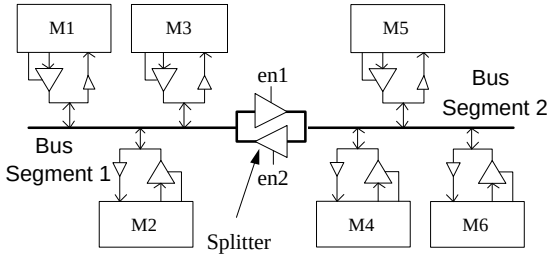


Fig. 1. Split shared-bus architecture

Another drawback of traditional bus architectures is that data is always propagated to the whole bus for any bus transaction between two modules, whether they are far apart or adjacent. Therefore, traditional bus architectures are not energy efficient. To overcome that, a split shared-bus architecture is proposed in [7] to reduce the power consumption. A split shared-bus architecture is shown in Fig. 1. Essentially, a bus is partitioned into bus segments by splitters, each composed of two tri-state buffers driving two bus segments in opposite directions. It can either transfer data from one segment to the other, or isolate the two segments. The data is propagated across a splitter only if such a transfer is necessary. Therefore, bus segments that are not necessary for data propagation can stay idle in order to reduce power consumption.

In this paper, we propose a new split shared-bus architecture that allows multiple simultaneous bus accesses. In this architecture, the arbiter may grant multiple bus accesses at the same time if they are compatible with each other. Therefore, the bandwidth and communication latency of the split bus architecture can be improved. We also propose an effective implementation of the arbiter for the split bus architecture, which can arrange and control multiple bus accesses automatically. Experimental results show that the bus architecture can have up to 2.3 times improvement in the effective bandwidth and up to 5 times reduction in the communication latency. Moreover, the proposed arbiter implementation has reasonable area and timing cost, making it suitable for high performance SoC applications.

## II. Split bus architecture with multiple bus accesses

### A. Overview

The performance of traditional bus architectures is strongly limited by the arbitration policy that only one module can access the bus at any time. Such a policy leads to unneces-

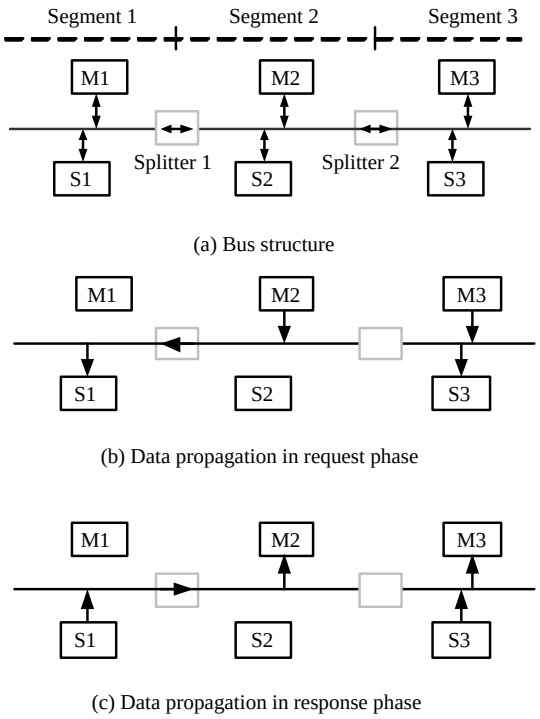


Fig. 2. Multiple simultaneous bus transactions

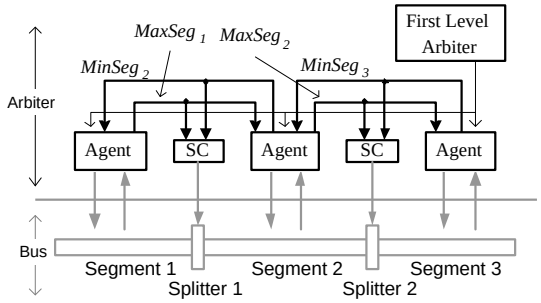


Fig. 3. Structure of a 3-segment split bus arbiter, where 'SC' refers to 'Splitter Controller', and signals $MaxSeg_i$ and $MinSeg_i$ pass segment usage information between arbitration agents of bus segments.

sary performance loss if there are pending bus communications with distinct data propagation paths. In the split bus architecture, splitters make it possible to isolate multiple compatible bus transactions. Therefore, the performance of the split bus architecture can be improved by multiple simultaneous bus accesses. Note that the split bus architecture is not meant to replace the hierarchical bus architecture. In fact, they are compatible approaches. Hierarchical bus architectures can be viewed as coarse bus segmentations. Bus splitting performs further, fine-grain segmentation to improve the performance of each bus in a hierarchical architecture.

Splitters in the proposed bus architecture have the same structure as those in [7]. They have three possible actions:

(i) Passing data from the left segment to the right, which we call forward data passing;

(ii) Passing data from the right segment to the left, which we call backward data passing;

(iii) Isolating the two segments on both sides.

We use 'F', 'B', and 'I' to denote these three actions, respectively. There are two phases in a bus cycle: request phase and response phase. In the request phase, masters send communication request to slaves; slaves respond to the received communication request in the response phase. Note that splitters always take opposite actions in the request and response phase of one bus cycle. Actions 'F' and 'B' are opposite actions for each other; the opposite action of 'I' is still 'I'. In the split bus architecture, bus transactions are viewed to be compatible if there is no common segment among their data propagation paths.

Fig. 2 shows an example of multiple simultaneous bus transactions on a split bus. The bus has three segments, three masters ($M1$, $M2$, $M3$), and three slaves ($S1$, $S2$, $S3$). Master $M2$ and $M3$ have pending transactions with slave $S1$ and $S3$, respectively. The propagation paths of these two transactions are {segment 2, segment 1} and {segment 3}, respectively. Therefore, they are compatible and can be performed simultaneously. In the request phase, splitter 1 takes action 'B' such that the communication request of master $M2$ can be sent to slave $S1$. The action of splitter 2 is 'I', therefore, master $M2$ and $M3$ can access the bus simultaneously without affecting each other. In the response phase, splitter 1 takes the opposite action, 'F', so the response data generated by slave $S1$ can be transfered to master $M2$.

In order to detect multiple compatible bus accesses and control splitters properly, it is necessary for the bus arbiter to know the destination segment of each transaction. Therefore, a master has to inform the arbiter its destination segment during the bus access request. This can be done by introducing additional interconnects between masters and the bus arbiter. Fortunately, as the number of segments in a split bus is small, the additional interconnect cost is typically low compared to the performance gain acquired through multiple simultaneous bus accesses. For example, up to 7 segments can be supported with 3-bit request signals; the first 7 combinations of a 3-bit signal are used to identify seven different destination segments and the last combination is used when there is no bus access request.

### *B. Arbitration of the split bus architecture*

The arbiter plays a crucial role in the split bus architecture. Not only does it grant bus accesses to compatible bus transactions, it also controls all splitters to establish proper data propagation paths for those granted bus transactions. The speed of bus arbitration is also an important factor of the bus performance, because pending communication requests have to wait until they obtain bus access signals. The long arbitration delay can be alleviated by pipelined arbitration; however, this may lead to an increase in communication latency.

Our arbiter implementation for a split bus with three segments is shown in the upper portion of Fig. 3. 'SC' in the figure refers to 'Splitter Controller'. The arbiter performs two levels of arbitration. The first level arbitration can be performed by any traditional arbitration method (*e.g.* priority-based, TDMA), and only one module is selected as the first level arbitration winner. The second level arbitration searches for compatible communication requests and generates the bus access grant signals and splitter control signals. As the first level arbitration is well-understood, we focus only on the im-

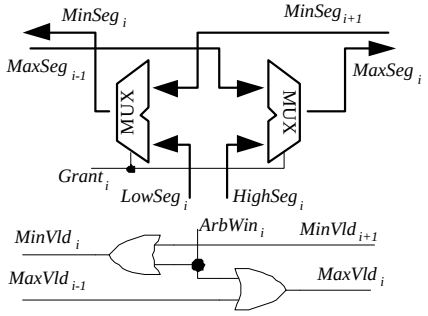


Fig. 4. Partial logic of an arbitration agent, where $arbwin_i$ is generated by the first level arbiter and indicates whether it is the winner agent of the first level arbitration. $Grant_i$ is generated by this agent following Equation 1.

plementation of the second level arbitration.

In order to simplify the detection of compatible transactions, the addresses of segments are assigned in the same order as their relative positions on the bus. The *i*-th splitter is placed between segment *i* and segment $i+1$. For each segment, there is an agent in the arbiter. For convenience, the agent for the segment, in which a module win the first level arbitration, is called the 'winner agent'. We first assume that there is only one bus access request in each segment. We will present more general cases later.

Two sets of bus segment usage information are passed between agents. The address of the highest segment occupied is passed from agents of low segments to agents of high segments; the address of the lowest segment occupied is passed in the opposite direction. These two sets of information are identified as *MaxSeg* and *MinSeg* in Fig. 3.

The structure of an arbitration agent is shown in Fig. 4. Agent *i* determines whether its pending communication can access the bus based on $MinSeg_{i+1}$ if $MinVld_{i+1}$ is asserted. The assertion of $MinVld_{i+1}$ indicates that agent *i* is before the arbitration winner. Similarly, the bus access grant decision for agent *i* is determined based on $MaxSeg_{i-1}$ when $MaxVld_{i-1}$ is asserted, which indicates that agent *i* is ordered after the arbitration winner. $ArbWin_i$ for agent *i* is asserted only if agent *i* is the winner agent. If the agent has a pending communication, $HighSeg_i$ is the address of the higher segment between segment *i* and the destination segment, and $LowSeg_i$ is the address of the lower segment. Otherwise the $HighSeg_i$ and $LowSeg_i$ are set to the highest address and lowest address of all segments, respectively, to make its bus access grant impossible. The assertion of signal $Grant_i$ indicates that agent *i* is granted for its pending communication. Signals $Grant_i$ is determined as follow:

$$\begin{aligned} Grant_i = {} & ArbWin_i \\ & \cup (MaxVld_{i-1} \cap (MaxSeg_{i-1} < LowSeg_i)) \\ & \cup (MinVld_{i+1} \cap (MinSeg_{i+1} > HighSeg_i)) \end{aligned} \quad (1)$$

Essentially, an agent acquires the bus access grant if any of the following three conditions holds:

(i) the agent is the winner agent;

(ii) the agent is at the right side of the winner agent, and its pending transaction does not conflict with other granted bus transactions from agents on its left side;

TABLE I
AN ARBITRATION EXAMPLE OF A 7-SEGMENT SPLIT BUS

| Segment/Splitter No. ($i$) | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Destination Seg. | / | 2 | 5 | 6 | 4 | 7 | 2 |
| $LowSeg_i$ | 1 | 2 | 3 | 4 | 4 | 6 | 2 |
| $HighSeg_i$ | 7 | 2 | 5 | 6 | 5 | 7 | 7 |
| $ArbWin_i$ | N | N | N | N | Y | N | N |
| $MinSeg_i$ | 2 | 2 | 4 | 4 | 4 | 6 | $M$ |
| $MaxSeg_i$ | $m$ | 2 | 2 | 2 | 5 | 7 | 7 |
| Access Grant? | N | Y | N | N | Y | Y | N |
| Splitter Action | I | I | I | B | I | F | / |

(iii) the agent is at the left side of the winner agent, and its pending transaction does not conflict with other granted bus transactions from agents on its right side.

The two multiplexers take the segment addresses from the agent *i* as their outputs if $Grant_i$ is asserted. Otherwise, the data received from other agents is propagated. Note that for agent 1 at the left bus end, when $Grant_1$ is negated, $MaxSeg_1$ is assigned a value *m* which is lower than all real segment addresses. Similarly, a value *M* higher than all real segment addresses is assigned to the *MinSeg* from the highest segment when its *Grant* signal is negated.

For splitter controller *i*, the conditions for actions 'F', 'I', and 'B' in the request phase are:

$$\begin{aligned} F: &\quad (MaxSeg_i > i), \\ I: &\quad (MaxSeg_i \le i) \cap (MinSeg_{i+1} \ge i+1)), \text{ and} \\ B: &\quad (MinSeg_{i+1} < i+1)), \text{ respectively.} \end{aligned}$$

An arbitration example for a split bus with seven segments is shown in Table I. Note that there are 6 splitters for 7-segment split bus. In this table, '/' refers to 'Not Available' or 'Not Valid'. 'Y' and 'N' refer to 'Yes' and 'No', respectively. In this example, segments 2, 3, 4, 5, 6, and 7 have pending communication requests. Segment 5 wins the first level arbitration. Through the second level arbitration, the pending communication requests of segment 2 and 6 are determined to be compatible with that of segment 5. Therefore, segment 2, 5, and 6 are granted with bus accesses finally.

The arbitration method discussed above assume that there is at most one bus access request in each segment. It is possible that a few masters in the same segment request bus access simultaneously. If that is the case, the agent selects one candidate among them for the second level arbitration. Although traditional bus arbitration techniques can be used for this selection, it is better that the selection can maximize the possible compatibility with other transactions. Therefore, we select the bus access request with the minimum number of used segments among all pending communications.

In the worst case, the critical delay path of the second level arbitration is from the winner agent at one bus end, to the agent at the other bus end for compatible transaction detection and access grant, and then back to splitter controllers near the first bus end to provide segment usage information. The delay contributed by one agent to such a path includes the delay of a

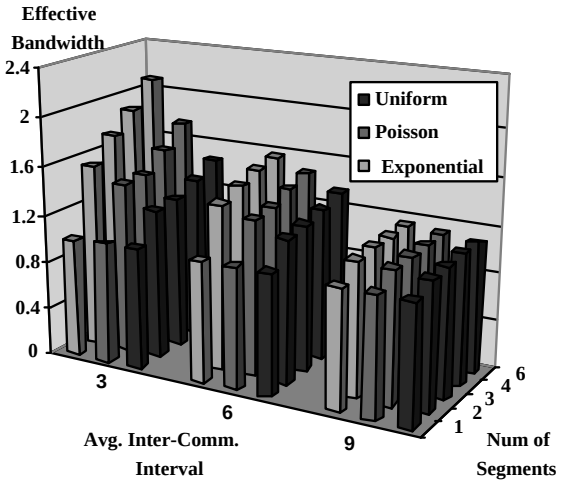


Fig. 5. Effective bandwidth of split buses with up to 6 segments under average inter-communication interval ranging from 3 to 9. A traditional single access bus can be viewed as the split bus with only one segment in this graph.

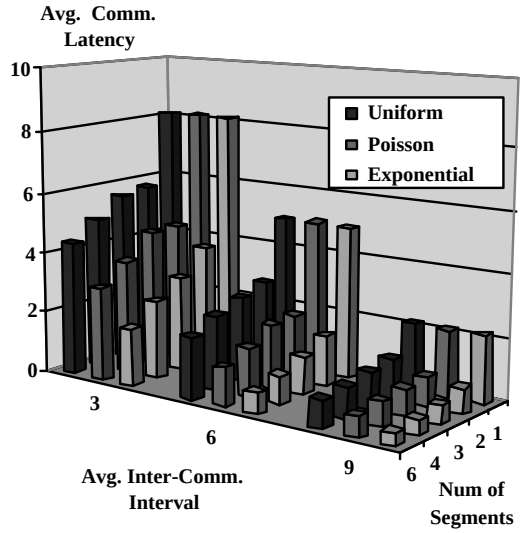


Fig. 6. Average communication latency of split buses with up to 6 segments under average inter-communication interval ranging from 3 to 9.

comparator for $Grant_i$ and the delay of two multiplexers. As the number of segments is typically very small, 2 to 4-bit comparators are usually sufficient. Therefore, the second level arbitration does not have a long delay. Note that the selection among multiple bus accesses requests inside one segment can be performed at the same time when the first level arbitration is performed; therefore, it does not affect the critical path delay of the arbitration. This is verified by our gate-level implementation of the arbiter.

## III. Experiments and results

In this section, we present our experiments and results on the performance and hardware complexity evaluation of the split bus architecture. A simulator to examine the performance of the split bus architecture is implemented through C++. We perform a gate-level implementation of split bus arbiters to evaluate the arbitration area and timing cost. We use TDMA for the first level arbitration in our experiments. Test benches for performance evaluation are generated using methods similar to those in [4, 7].

### *A. Performance evaluation*

All tested benches have 12 masters and 12 slaves. Communication traffic is randomly generated based on several parameters. The frequency of bus access requests is controlled by the inter-communication interval. For a master, the inter-communication interval is defined as the number of bus cycles after which a new bus transaction is generated since the previous bus access is granted. The inter-communication interval is randomly generated following Poisson distribution with average interval values of 1 through 11. The probability that a slave is accessed is assigned based on the distribution of communication distance. Here, the communication distance of a bus transaction refers to the number of modules between the master and the destination slave for this transaction. Three communication distance distributions are used: uniform, Poisson, and exponential. In the uniform distribution, a master has equal probabilities of accessing all slaves. In the other two distributions, average communication distance is also used as a parameter.

The experimental results first show that our arbiter implementation can always correctly detect compatible transactions and establish propagation paths for them. Two metrics are used to evaluate the bus performance: the effective bandwidth and the communication latency. The effective bandwidth is defined as the number of finished bus transactions over the total number of bus cycles used. And the communication latency refers to the number of bus cycles a master module spent to obtain the bus access grant. Fig. 5 and Fig. 6 show the effective bandwidth and average latency, respectively, for split buses with different numbers of segments and different inter-communication intervals. Note that a traditional single access bus can be viewed as a split bus with only one bus segment.

From the experimental results, we can see that splitting a bus into multiple segments can always lead to improvement in both the effective bandwidth and the average latency improvement over traditional single access buses (shown as split buses with one segment in the figures). Test cases with the exponential communication distance distribution have the highest improvement in terms of both bandwidth and latency. The bandwidth improvement can be as high as 2.3 times and the average communication latency can be reduced by up to 5 times. The reason is that there are more bus transactions among nearby modules in the exponential communication distance distribution, leading to higher possibilities of having compatible bus transactions. The implication is that the performance of the split bus architecture can be improved with proper ordering of modules on the bus. Another observation is that the bandwidth improvement is dependent on the average inter-communication interval. This is caused by the fact that there are fewer simultaneous bus access requests when inter-communication interval is large. Therefore, the multiple bus access ability of the split bus architecture is not fully exploited. However, even in

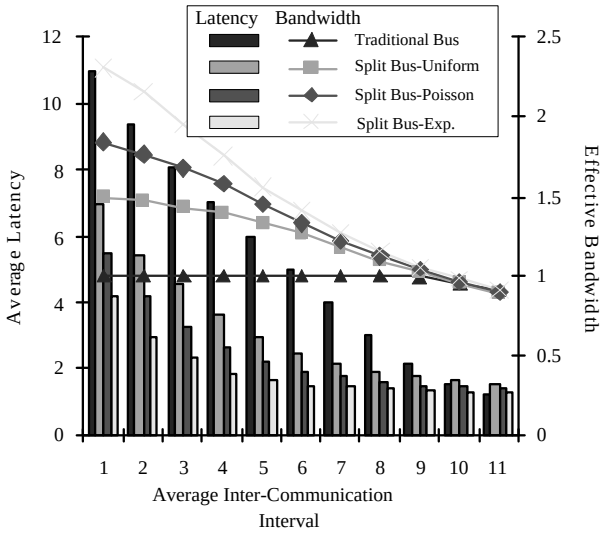


Fig. 7. Performance of 6-segment split bus with additional arbitration latency compared with traditional bus architecture with zero arbitration latency.

those situations, the improvement over communication latency is still significant. The results also show that the performance improvement is higher with more bus segments. But increasing the number of bus segments may lead to more interconnection and longer arbitration delay. Therefore, a tradeoff between the performance and hardware cost should be considered.

Another set of experiments are performed to examine the impact on the bus performance due to the delay of second level arbitration. We assume that the arbitration latency is increased by one due to the second level arbitration. Fig. 7 shows the performance of split buses with six bus segments under uniform, Poisson and exponential communication distance distribution. For comparison purpose, the performance of traditional bus architecture with zero arbitration latency are also shown. Only one set of results is shown for traditional bus architecture because the communication distance distribution does not affect the performance of traditional buses. The results show that the split buses can still have significant better performance in most cases. Only when the bus traffic is extremely low can the traditional buses outperform split buses marginally. For designs with high communication requirement, which is typical of current and future SoC, the split bus architecture would be more suitable.

### *B. Hardware implementation complexity*

In order to evaluate the hardware complexity of the proposed split bus architecture, we perform gate-level implementation of arbiters including both first and second level arbitration for two split buses. They all have 12 masters, but have 3 and 6 segments respectively. The design is mapped to TSMC 0.18 $\mu m$ technology through 'Synopsys Design Compiler'.

The delay of the 3-segment arbiter is 1.27 *ns*, and its area is 13,118 $\mu m^2$. Therefore, the arbitration can be performed in one cycle for bus speed up to 787 *MHz*. The arbitration delay of the 6-segment split bus is 2.44 *ns*, with the area of 23,005 $\mu m^2$. Hence the arbitration can be finished in one cycle for bus speed up to 409 *MHz*. These show that the proposed split bus architecture can be used for high performance SoC applications.

When the arbitration delay is a concern for some applications, pipelined arbitration can be used to improve the bus speed. With one additional cycle, the 6-segment arbiter can be used for bus speed as high as 818 *MHz*. Our experimental results in Fig. 7 clearly show that there is still significant performance improvement, which includes up to 2.2 times improvement over bandwidth and up to 2.7 times reduction over latency, even if the arbitration for a 6-segment split bus takes an additional bus cycle.

More important, as the feature size scales down to nanometer, the delay introduced by the arbiter itself can be scaled down effectively. At the same time, the global interconnect delay between masters and the arbiter would hardly be reduced; indeed, one could argue that it would even increase. As a result, such interconnect delay will become the dominating factor of the delay of the whole arbitration process. Therefore, the delay of the arbiter itself in a split bus will have only marginal effect on bus cycle time for future SoC designs.

## IV. Conclusion

We propose a high performance split shared-bus communication architecture through simultaneous multiple bus accesses. An implementation of the bus arbiter with reasonable area and timing cost is also proposed. Experimental results show that the bus architecture can have up to 2.3 times improvement in the effective bandwidth and up to 5 times reduction in the communication latency.

## References

[1] IBM. *CoreConnect Bus Architecture*, 1999.

[2] ARM, Limited. *AMBA Specification*, 1999.

[3] Sonics, Inc. *Open Core Protocol Specification*, 1999.

[4] K. Lahiri, A. Raghunathan, and G. Lakshminarayana. LOTTERYBUS: A new high-performance communication architecture for System-on-Chip designs. In *Proc. Design Automation Conf*, 2001.

[5] M. Drinic, D. Kirovski, S. Meguerdichian, and M. Potkonjak. Latency-guided on-chip bus network design. In *Proc. Int. Conf. on Computer Aided Design*, 2000.

[6] K. Lahiri, A. Raghunathan, and S. Dey. Effi cient exploration of the SoC communication architecture design space. In *Proc. Int. Conf. on Computer Aided Design*, 2000.

[7] C. Hsieh and M. Pedram. Architectural energy optimization by bus splitting. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21:408–414, April 2002.