LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs

Kanishka Lahiri, Anand Raghunathan, Ganesh Lakshminarayana
Dept. of Electrical and Computer Engg., UC San Diego, La Jolla CA
C & C Research Labs, NEC USA, Princeton, NJ

Abstract

This paper presents LOTTERYBUS, a novel high-performance communication architecture for system-on-chip (SoC) designs. The LOTTERYBUS architecture was designed to address the following limitations of current communication architectures: (i) lack of control over the allocation of communication bandwidth to different system components or data flows (e.g., in static priority based shared buses), leading to starvation of lower priority components in some situations, and (ii) significant latencies resulting from variations in the time-profile of the communication requests (e.g., in time division multiplexed access (TDMA) based architectures), sometimes leading to larger latencies for high-priority communications.

We present two variations of LOTTERYBUS: the first is a low overhead architecture with statically configured parameters, while the second variant is a more sophisticated architecture, in which values of the architectural parameters are allowed to vary dynamically. Our experiments investigate the performance of the LOTTERYBUS architecture across a wide range of communication traffic characteristics. In addition, we also analyze its performance in a 4x4 ATM switch sub-system design. The results demonstrate that the LOTTERYBUS architecture is (i) capable of providing the designer with fine-grained control over the bandwidth allocated to each SoC component or data flow, and (ii) well suited to provide high priority communication traffic with low latencies (we observed up to 85.4% reduction in communication latencies over conventional on-chip communication architectures).

1 Introduction

The communication architecture plays a key role in SoC design by enabling efficient integration of heterogeneous system components (e.g., CPUs, DSPs, application specific cores, memories, custom logic, etc). In addition, the communication architecture also significantly influences the system performance and power consumption (i) directly, since the delay and power in global interconnect is known to be an increasing bottleneck with shrinking feature sizes, and (ii) through its significant indirect impact on the computation time and power consumption in the system components [1, 2, 3].

In this work, we propose LOTTERYBUS—a novel high-performance on-chip communication architecture for complex SoC designs. The LOTTERYBUS architecture improves over the current state-of-the-art communication architectures through innovations in the communication protocol it employs, resulting in the following key advantages: (i) it provides the designer with fine-grained control over the fraction of communication bandwidth that each system component or data flow receives, and (ii) it provides fast execution (low latencies) for high priority communications.

Recognizing the importance of high-performance communication as a key to successful system design, recent work has addressed several issues pertaining to on-chip communication architectures. While several embedded system design houses and semiconductor vendors employ proprietary on-chip bus architectures [4, 5], recently, independent companies and consortia have been established to develop and license system-level integration and communication architectures [6, 7, 8]. Communication protocols commonly used in these architectures include priority based arbitration [7], time division multiplexing [6], and token-ring mechanisms [9]. We later demonstrate that some of the on-chip communication architectures mentioned above are not capable of providing control over the allocation of communication bandwidth to SoC components, while others cannot provide low latencies for high-priority communications. To our knowledge, ours is the first approach for SoC communication architectures that attempts to address these issues.

Another body of work is aimed at facilitating a plug-and-play design methodology for HW/SW SoC components and communication architectures by promoting the use of a consistent communication interface, so that predesigned components or cores can be easily integrated with other system components [10, 11]. The adoption of such standards will make it easier for system designers to exploit innovations in SoC communication architectures (such as LOTTERYBUS), without being concerned about low-level interfacing requirements.

It bears mentioning that the performance issues addressed in this work have been studied in the networking literature, in the context of shared media access control in local area networks [12], and traffic scheduling algorithms for high-speed switches [13, 14, 15]. However, previous research in the above area cannot be directly applied to system-on-chip design since (i) the protocols used are complex, leading to communication latencies and hardware costs that are infeasible for on-chip communication, and (ii) considerations such as dynamic scalability and fault tolerance apply in the design of distributed networks, leading to significantly different design decisions (e.g., distributed vs. centralized arbitration). Finally, probabilistic techniques have been used in scheduling multiple threads of computation in a multi-threaded operating system [16]. However, in that domain, while hardware implementation considerations are irrelevant, the software architecture needs to ensure security and insulation between competing applications.

2 System-on-Chip Communication Architectures: Background

In this section, we introduce concepts and terminology associated with on-chip communication architectures and describe some popular communication architectures used in commercial SoC designs. Since buses are often shared by several SoC masters, bus architectures require protocols to manage access to the bus, which are implemented in (centralized or distributed) bus arbiters. Currently used communication architecture protocols include round-robin access, priority based selection, and time-division multiplexing. In addition to arbitration, the communication protocol handles other communication functions. For example, it may limit the maximum number of bus cycles for which a master can use the bus, by setting a maximum burst transfer size. Another factor that affects the performance of a communication channel is its clock frequency, which (for a given process technology) depends on the complexity of the interface logic, the placement of the various components, and the routing of the wires.

2.1 Static Priority Based Shared Bus

The static priority based shared system bus is one of the commonly used on-chip bus architectures (e.g., [7]). The bus (Figure 1)...
is a set of address, data, and control lines shared among a set of masters that contend among themselves for access to one or more slaves. The bus arbiter periodically examines accumulated requests from the master interfaces, and grants bus access to the master of highest priority among the requesting masters. The bus supports a burst mode of data transfer, where the master negotiates with the arbiter to send or receive multiple words of data over the bus without incurring the overhead of handshaking for each word.

2.2 TDMA Based Shared Bus

In the TDMA based architecture, components are provided access to the shared bus in an interleaved manner, using a two level arbitration protocol (e.g., [6]). The first level uses a timing wheel where each slot is statically reserved for a unique master (Figure 2). If the master associated with the current slot has an pending request, a single word transfer is granted, and the wheel is rotated by one slot. To alleviate the problem of wasted slots (inherent in TDMA based approaches), a second level of arbitration identifies slots for which the assigned master does not have a pending communication request, and issues a grant to the next requesting master in a round-robin fashion. For example, in Figure 2, the current slot is reserved for \( M_1 \), which has no pending request. As a result, the second level arbitration pointer \( rr2 \) is incremented from its earlier position (\( M_2 \)) to the next pending request (\( M_3 \)).

2.3 Other Communication Architectures

In addition to the above, there are several other on-chip communication architectures. Notable among them is a hierarchical bus architecture [4], in which multiple buses are arranged in a hierarchy, with bridges permitting cross-hierarchy communications. Another common architecture is based on token rings; their high clock rate makes them an attractive alternative for high-baudrate applications such as ATM switches [9]. Note, each of the architectures described above, as well as the proposed LOTTERYBUS architecture, can be implemented with additional features such as pre-emption, multi-threaded transactions, and dynamic bus splitting. In addition, in order to reduce arbitration overhead, the arbitration operations may be pipelined with the data transfer cycles.

3 Limitations of Conventional Communication Architectures

In this section we illustrate through examples, the limitations of the static priority based bus architecture and the two-level TDMA based architecture presented earlier. We demonstrate the shortcomings in their ability to provide (i) proportional allocation of communication bandwidth among various SoC components, and (ii) low latency communications for high priority data transfers, and discuss the reasons they occur. We go on to demonstrate the potential benefits of the LOTTERYBUS communication architecture, and show that it is capable of effectively meeting both the above goals.

In the first example, we study the static priority based architecture described in Section 2.1, focusing on the manner in which it allocates the bus bandwidth to the various SoC components.
In the next example, we illustrate the ability of the LOTTERYBUS architecture to improve communication latency, and provide low latency communications to high priority burst data transfers.

**Example 4:** The experiments described in Example 2 were repeated using the LOTTERYBUS architecture for the example system of Figure 3. The lottery tickets were assigned in the same ratio as the time-slots were in the TDMA-based architecture (i.e., component $C_i$ receives the highest number of lottery tickets, and so on). Figure 6(b) compares the average communication latencies under the two communication architectures, for an illustrative class of communication traffic. The x-axis denotes different SoC components, while the y-axis denotes the average number of bus cycles spent in transferring a bus word including both waiting time and data transfer time. We observe that the latency of the highest priority component is substantially lower under the LOTTERYBUS architecture (2.7 cycles per word) than under the TDMA based architecture (18.55 cycles per word), a 7X improvement.

Having established the motivation for our work, we next present the details of the LOTTERYBUS communication architecture.

4 The LOTTERYBUS Communication Architecture

In this section, we first present an overview of the LOTTERYBUS architecture. Next, we introduce its principle of operation, and then consider two alternative embodiments, and present hardware implementations of each.

4.1 Overview

The LOTTERYBUS architecture consists of a randomized arbitration algorithm implemented in a centralized “lottery manager” for each shared channel (bus) in the system-on-chip. The proposed architecture does not presume any fixed topology of communication channels. Hence, the SoC components may be interconnected by an arbitrary network of shared channels or by a flat system-wide bus.

![Figure 7: The LOTTERYBUS communication architecture](image-url)

The lottery manager accumulates requests for ownership of the bus from one or more masters, each of which is (statically or dynamically) assigned a number of “lottery tickets”, as shown in Figure 7. The manager probabilistically chooses one of the contending masters to be the winner of the lottery, and grants access to the winner for one or more bus cycles. We allow multiple word requests, in order to avoid incurring control overhead for each word. However, to prevent a master from monopolizing the bus (in case it has a large amount of data to send), a maximum transfer size limits the number of bus cycles for which the granted master can utilize the bus (similar to the static priority based architecture). Also, the architecture pipelines lottery manager operations with actual data transfers, to minimize idle bus cycles.

4.2 Principle of Operation

Let the set of bus masters be $C_1, C_2, ..., C_n$. Let the number of tickets held by each master be $t_1, t_2, ..., t_n$. At any bus cycle, let the set of pending requests be represented by a set of boolean variables $r_i, i = 1, 2, ..., n$, where $r_i = 1$ if component $C_i$ has a pending request,
and \( r_i = 0 \) otherwise. The master to be granted is chosen in a randomized way, with the probability of granting component \( C_i \) given by:

\[
P(C_i) = \frac{r_i}{\sum_j r_j}
\]

To implement this probabilistic arbitration mechanism, we use the

![Diagram of lottery manager](image)

Figure 8: Example of lottery to determine bus master notion of a lottery [16]. To make an arbitration decision, the lottery manager examines the total number of tickets possessed by the contending components, given by \( \sum_{i=1}^{n} r_i \). It then generates a random number (or picks a winning “ticket”) from the range \( [0, \sum_{i=1}^{n} r_i] \) to determine which component to grant the bus to. If the number falls in the range \( [0, r_1 + r_2] \), the bus is granted to component \( C_1 \), if it falls in the range \( [r_1 + r_2, r_1 + r_2 + r_3] \), it is granted to component \( C_2 \), and so on. In general, if it lies in the range \( \sum_{k=1}^{j} r_k, \sum_{k=1}^{j+1} r_k \), it is granted to component \( C_{j+1} \). For example, in Figure 8, components \( C_1, C_2, C_3 \) and \( C_4 \) are assigned 1, 2, 3 and 4 tickets respectively. However, at the instant shown, only \( C_1, C_3 \) and \( C_4 \) have pending requests. Hence the number of current tickets \( \sum_{k=1}^{j} r_k \) is 1 + 3 + 4 = 8. The random number, generated uniformly in the range \( [0, 8] \), is 5, which lies between \( r_1 = 1, r_2 + r_3 = 4, \) and \( r_1 + r_2 + r_3 + r_4 = 8 \). Therefore, the bus is granted to component \( C_2 \).

One of the main concerns while designing a communication architecture is starvation, i.e., the problem of a low-priority component not being able to obtain access to the bus for extended periods of time. For the LOTTERYBUS architecture, the probability, \( p \), that a component with \( t \) tickets is able to access the bus within \( n \) lottery drawings is given by the expression \( 1 - (1 - p)^n \). The expression indicates that the probability of obtaining access to the bus converges rapidly to one, thereby ensuring that no component is starved.

Within the overall strategy outlined above, we propose two possible architectures. In the first, the number of tickets assigned to a component is statically determined. In the second, the number of tickets a component possesses varies dynamically, and is periodically communicated by the component to the lottery manager.

### 4.3 Hardware Implementation: Statically Assigned Tickets

A hardware implementation of the lottery manager with statically assigned tickets is presented in Figure 9. We next describe the various

1. The set \([a, b]\) includes all the integers between \( a \) and \( b \), inclusive of \( a \) but not \( b \).

![Figure 9: Lottery manager for static LOTTERYBUS architecture](image)
In (this architecture, the inputs to the lottery manager are a set of request lines \((r_1, r_2, r_3, r_4)\) and the number of tickets currently possessed by each corresponding master. Therefore, under this architecture, not only can the range of current tickets vary dynamically, it can take on any arbitrary value (unlike the static case, where it was confined to remain among a predetermined set of values). Consequently, at each lottery, for each component \(C_i\), the partial sum \(\sum_{j=1}^{i} r_j t_j\) needs to be calculated. For \(C_4\), this yields the total range, or the sum of the number of tickets held by all pending requests. This is implemented using a bitwise AND operation and a tree of adders, as shown in Figure 10. The final result, \(T = r_1 t_1 + r_2 t_2 + r_3 t_3 + r_4 t_4\) defines the range in which the random number must lie. The random number is generated in the range \([0, T]\) using modulo arithmetic hardware. The rest of the architecture consists of comparison and grant generation hardware, and follows directly from the static lottery manager design.

5 Experimental Results

In this section, we present results of experiments that we carried out to evaluate the performance of the LOTTERYBUS architecture. We performed experiments using the POLIS [18] and PTOLEMY [17] system design environment. All system components were specified in Esterel and C, from which PTOLEMY simulation models were generated using POLIS. PTOLEMY was used for schematic capture and HW/SW co-simulation.

5.1 Performance of the LOTTERYBUS Architecture Across the Communication Traffic Space

We conducted several experiments to examine the performance of the LOTTERYBUS architecture under widely varying characteristics of on-chip communication traffic. To perform these experiments, we made use of the system level test-bed for performance evaluation that is shown in Figure 11. The test-bed consists of 8 components each of which is connected to a parameterized traffic generator, while the remaining components were specified in Esterel and C, from which PTOLEMY simulation models were generated using POLIS. PTOLEMY was used for schematic capture and HW/SW co-simulation.

5.2 Hardware Complexity of the LOTTERYBUS Architecture

In the LOTTERYBUS architecture, the physical interconnect for the address, data, and control lines on the bus remain unchanged. The improved communication protocol is implemented by modifying the component’s bus interfaces, and the bus controller/arbitrator. In order to obtain an idea of issues involved in obtaining a practical realization, we implemented the LOTTERYBUS architecture for the four-component system described in Section 3, and mapped it to NEC’s 0.35 \(\mu\)m cell based array technology [20]. The look-up table was implemented using a register file, and the comparators and the random number generator were pipelined to maximize performance. The area of the LOTTERYBUS controller implementation was found to be 10,518 cell grids, and the arbitration time was found to be 3.2ns (i.e., arbitration can be performed in 1 cycle for bus speeds up to 312 MHz), making our implementation suitable for high-performance applications.

5.3 Example System: Output-queued ATM switch

We used the LOTTERYBUS architecture in the design of the cell forwarding unit of an output-queued ATM switch (Figure 13). The system consists of 4 output ports, each with a dedicated local memory that stores queued cell addresses. Arriving cell payloads are written to a dual-ported shared memory, while the starting address of each ticketing traffic that were considered, the y-axis depicts the fraction of the traffic. The x-axis depicts the nine different classes of communication bandwidth allocation under different classes of communication. The \(X\) value is (on the average) in the ratio \(1/9\), and for classes \(T_4, T_5, T_7, T_8, T_9\) the bandwidth allocated is roughly the same for different components. This is because the sparse nature of communications in these classes results in immediate grants being issued to most of the communication requests. We conclude that the LOTTERYBUS architecture is capable of providing efficient and fine-grained control over allocation of the bus bandwidth over a variety of on-chip communication traffic classes, at varying levels of bus utilization.

Figures 12(b) and (c) compare the latency of the TDMA and LOTTERYBUS architecture across 6 classes of traffic. The x-axis in each figure denotes different classes of on-chip communication traffic, while the y-axis denotes time-slots (Figure 12(b)) and lottery tickets (Figure 12(c)) assigned to different components. The z-axis measures the average per word communication latency. For example, in Figure 12(b), a component assigned 4 time-slots, has an average latency of 18.55 bus cycles per word under traffic class \(T_6\). Under the LOTTERYBUS architecture, the same component with the same traffic class has an average latency of 2.7 bus cycles per word.

Clearly, the LOTTERYBUS architecture exhibits better latency behavior than the TDMA architecture for a wide range of traffic conditions. In addition, the following points are worth noting. The communication latency for high-priority components varies significantly for the TDMA architecture (1.65 to 20.5 cycles/word). This is because under the TDMA scheme, the latency of a communication is highly sensitive to the position of the timing wheel when the request arrived. Moreover, under the TDMA-based architecture, components with higher priorities could experience higher latencies than those with lower priorities (e.g., \(T_5, T_6\)). The LOTTERYBUS architecture does not exhibit this phenomenon, ensuring low latencies for high priority communications.

\[ r_1 t_1, r_2 t_2, r_3 t_3, r_4 t_4 \]
Figure 12: Performance under different communication traffic classes (a) Bandwidth allocation of LOTTERYBUS, (b) Communication latencies under TDMA, and (c) Communication latencies under LOTTERYBUS

For the TDMA architecture, in row 2 we observe that Port 3 receives only 47% of the total bandwidth, while it had originally reserved 60%. This occurs because when Port 4 has no cells to send, its slots are made available to the other ports in a round robin manner. However, we observe from row 3 that the bandwidth assignments in the case of the LOTTERYBUS architecture closely match the reservations.

The results demonstrate that the LOTTERYBUS architecture offers an attractive alternative to conventional communication architectures by (a) providing low latencies for bursty traffic with real time latency constraints, and (b) at the same time, providing effective bandwidth guarantees for traffic generated by each system component.

References