

# 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 upto 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.

The communication architecture *topology* consists of a network of shared and dedicated communication channels, to which various SoC components are connected. These include (i) *masters*, components that can initiate a communication transaction (*e.g.*, CPUs, DSPs, DMA controllers *etc.*), and (ii) *slaves*, components that merely respond to transactions initiated by a master (*e.g.*, on-chip memories). When the topology consists of multiple channels, *bridges* are employed to interconnect the necessary channels.

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)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

Copyright 2001 ACM 1-58113-297-2/01/0006 ...\$5.00.

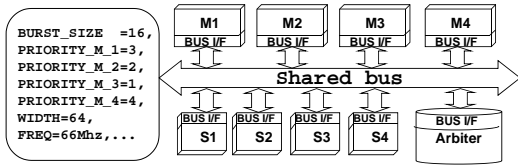


Figure 1: Static Priority based shared bus

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

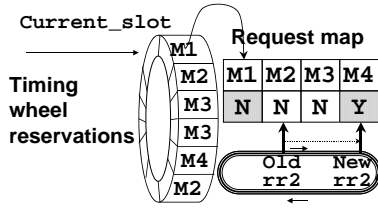


Figure 2: TDMA based shared bus

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 ( $M_4$ ).

## 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-bandwidth 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.

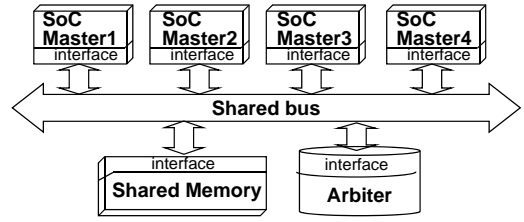


Figure 3: Example system with shared bus and 4 masters

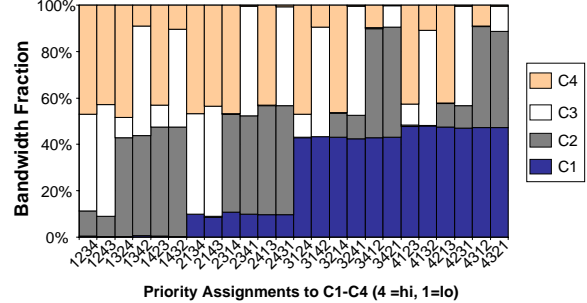


Figure 4: Bandwidth sharing under static priority based architecture

**Example 1:** Consider the example system shown in Figure 3. The system consists of a single bus with four masters, which contend with each other for access to a shared memory. The bus masters were assigned unique priority values from 4 to 1 (4 representing the highest priority level). We simulated the system by modeling the components as stochastic on-chip communication traffic generators and the bus using the PTOLEMY [17] system modeling and simulation environment as described in Section 5. The traffic generators were configured such that the bus was always kept busy, i.e., at least one pending request exists at any time. We measured the fraction of the bus bandwidth assigned to each component under the given priority assignment over a long simulation trace. The simulation was repeated for every possible priority assignment.

The x-axis in Figure 4 depicts all the possible priority combinations for the four masters that access the bus. For example, the assignment “4321” implies that component  $C_1$  has the highest priority,  $C_2$  the second highest, and so on. The y-axis denotes a percentage of the total bus bandwidth. The four regions of the graph denote the bandwidth fraction obtained by each SoC master across various priorities. For example, component  $C_1$  has increasing levels of priority from left to right, and exhibits a step-wise increase in the fraction of bus bandwidth it receives. From Figure 4, we observe that firstly, the fraction of bandwidth a component receives is extremely sensitive to the priority value it is assigned. For instance, the bandwidth received by component  $C_1$  ranges from 0.16% to 47.8%. Secondly, low priority components get a negligible fraction of the bus bandwidth as long as higher priority components have pending requests. For example, for priority combinations 1234 through 1432,  $C_1$  receives an average of 0.4% of the bus bandwidth. ■

It is fair to conclude that the static priority based architecture does not provide a means for controlling the fraction of communication bandwidth assigned to a component. Under heavy communication traffic scenarios, this leads to starvation for the low priority components.

In the next example, we consider the two-level TDMA based architecture described in Section 2.2. TDMA-based architectures can be used to provide bandwidth guarantees for each component, by appropriately assigning slots in the timing wheel. For instance, if there are two SoC masters, and bandwidth is to be allocated between them in the ratio 1 : 2, this can be achieved by assigning  $\frac{1}{3}$  of the total number of slots in the wheel to the first master, and  $\frac{2}{3}$  to the second master. While this solves the problem of proportional bandwidth allocation, it creates another potentially serious problem, as illustrated by the next example.

**Example 2:** Figure 5 shows symbolic execution traces on a TDMA based bus for two different request patterns. The system bus has

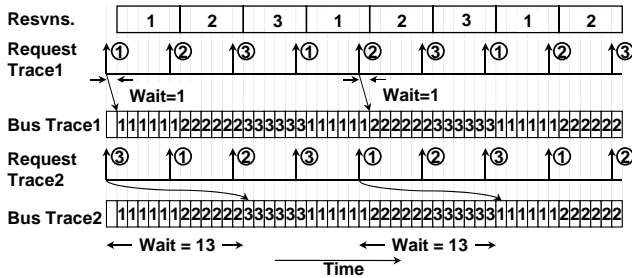


Figure 5: Large latencies under a TDMA based architecture

three masters that contend for access. Slots in the timing wheel are reserved in the manner shown in the first waveform of Figure 5, with 6 contiguous slots defining the size of a burst. The second and the fourth traces show two different patterns of communication requests generated by the components, each request marked with the associated component. The third and fifth waveforms show the actual assignment of masters to bus slots for the given reservations and request traces. From the example, we make the following observations. (1) In request trace *Trace1*, the requests from each component arrive periodically, and very well time-aligned with the slots reserved for it in the timing wheel. Consequently the time spent by a component in waiting for access to the bus is minimal, in this case only 1 slot. (2) Under request *Trace2*, we notice that the occurrence of communication requests and the reserved slots are not well synchronized. Even though the request pattern is periodic (in fact, identical to request *Trace1* except for a phase shift), the wait times have increased to 13 slots per communication transaction.

It is possible to optimize the TDMA architecture for request *Trace2* by modifying the reservations such that they are time-aligned to the request pattern of *Trace2*. However, this new assignment will provide poor performance if the request patterns exhibit any dynamic variation. For instance, if they start following the pattern of request *Trace1*, they will again suffer 13 wait slots (average) per transaction.

We conducted several experiments using the TDMA based architecture for the example system in Figure 3. We considered various types of communication traffic, and measured the latencies of each communication transaction over a long simulation trace. We obtained several cases where high priority transactions suffered large latencies. As an example, under one class of communication traffic, the highest priority component had an average latency of 18.55 cycles per word, which was more than twice as large as the per word latency of the component with next lower priority (details of these experiments are described in Section 5).

From the above example we conclude that the latency of a communication transaction in a TDMA based architecture is very sensitive to the time-alignment of communication requests and the reservations of slots in the timing wheel. Next, we consider the proposed LOTTERYBUS communication architecture, and demonstrate its ability to address the drawbacks mentioned in the previous two examples.

**Example 3:** As in Example 1, we consider again a system of four components accessing a single shared bus. We repeated the experiments described in Example 1, using the LOTTERYBUS architecture instead of the static priority architecture. Here too, the relative importance of the communications generated by a component is used to (statically) assign it “lottery tickets”, which in turn determines its communication latencies and allocated bandwidth fraction. The lottery tickets were assigned in the ratio 1 : 2 : 3 : 4. Figure 6(a) depicts the bandwidth distribution obtained by each component for 24 combinations of lottery tickets. From the figure, we observe that the fraction of bandwidth obtained by a component is directly proportional to its allocated tickets. For example, under the first 6 priority combinations, component  $C_1$  has 1 lottery ticket and receives, on the average, 11% of the total bus bandwidth. Between combinations 2134 and 2431, it has 2 lottery tickets, and therefore receives, 20.8% of the total bus bandwidth. The actual allocation of bandwidth closely matches the ratio of lottery tickets, demonstrating the ability of the LOTTERYBUS architecture to provide fine-grained control over bandwidth allocation to 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  $C_4$  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.

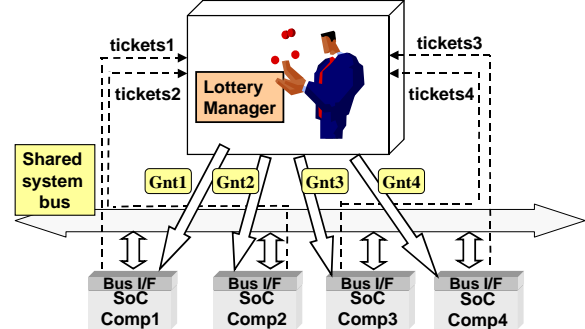


Figure 7: The LOTTERYBUS communication architecture

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  $C_1, C_2, \dots, C_n$ . Let the number of tickets held by each master be  $t_1, t_2, \dots, t_n$ . At any bus cycle, let the set of pending requests be represented by a set of boolean variables  $r_i$  ( $i = 1, 2, \dots, n$ ), where  $r_i = 1$  if component  $C_i$  has a pending request,

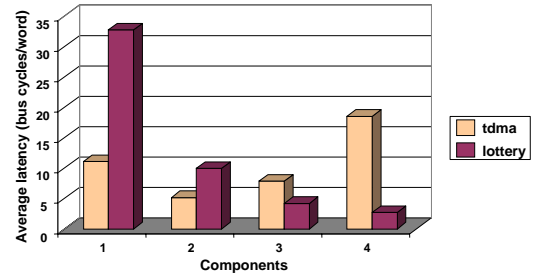
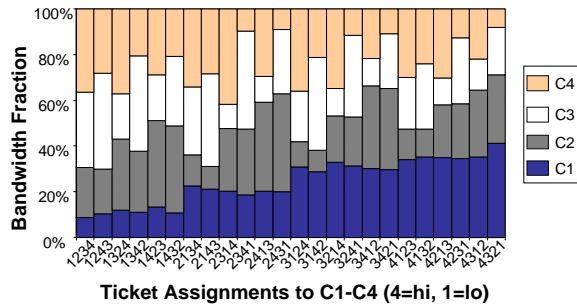


Figure 6: Advantages of the LOTTERYBUS architecture (a) bandwidth sharing, and (b) average communication latency

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:

$$P(C_i) = \frac{r_i \cdot t_i}{\sum_{j=1}^n r_j \cdot t_j}$$

To implement this probabilistic arbitration mechanism, we use the 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  $\sum_{j=1}^n r_j \cdot t_j$ . It then generates a random number (or picks a winning “ticket”) from the range  $[0, \sum_{j=1}^n r_j \cdot t_j)^1$  to determine which component to grant the bus to. If the number falls in the range  $[0, r_1 \cdot t_1)$ , the bus is granted to component  $C_1$ , if it falls in the range  $[r_1 \cdot t_1, r_1 \cdot t_1 + r_2 \cdot t_2)$ , it is granted to component  $C_2$ , and so on. In general, if it lies in the range  $[\sum_{k=1}^i r_k \cdot t_k, \sum_{k=1}^{i+1} r_k \cdot t_k)$ , it is granted to component  $C_{i+1}$ . 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  $\sum_{j=1}^n r_j \cdot t_j = 1 + 3 + 4 = 8$ . The random number, generated uniformly in the range  $[0, 8)$ , is 5, which lies between  $r_1 \cdot t_1 + r_2 \cdot t_2 + r_3 \cdot t_3 = 4$ , and  $r_1 \cdot t_1 + r_2 \cdot t_2 + r_3 \cdot t_3 + r_4 \cdot t_4 = 8$ . Therefore, the bus is granted to component  $C_4$ .

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  $1 - (1 - t/T)^n$ . 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 steps executed by the lottery manager.

<sup>1</sup>The set  $[a, b)$  includes all the integers between  $a$  and  $b$ , inclusive of  $a$  but not  $b$ .

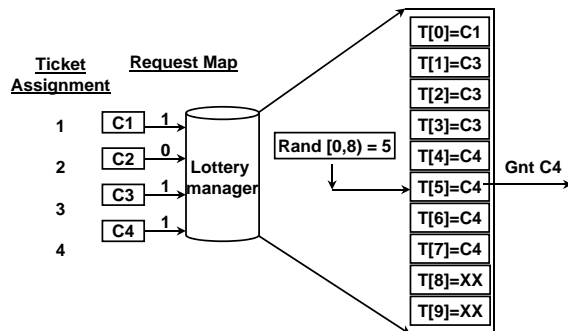


Figure 8: Example of lottery to determine bus master

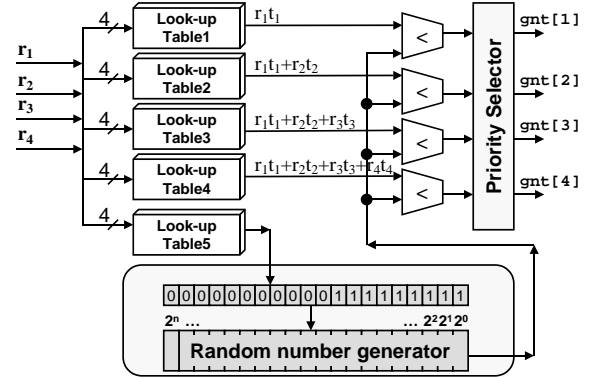


Figure 9: Lottery manager for static LOTTERYBUS architecture

**Computation of ticket ranges:** The range  $[0, \sum_{j=1}^n r_j \cdot t_j)$  is the number of tickets held by a set of contending masters. This range varies dynamically, depending on the subset of masters having simultaneously pending requests. However, in this architecture, since the number of tickets owned by a component is fixed, it is possible to precompute all potential ranges to avoid the extra cost of calculating the ranges at run time. For four masters, a 4 bit request map is used to indicate which masters have pending requests. For example,  $r_1 r_2 r_3 r_4 = 1011$  implies  $C_1, C_3$  and  $C_4$  have pending requests. For a given request map, the range of tickets owned by each component is determined statically, and stored in a look up table (Figure 9).

**Efficient random number generation:** The LOTTERYBUS architecture requires the generation of a random number, uniformly distributed in the range  $[0, T)$ , where  $T$  is the total number of tickets. If  $T$  is a power of two, random numbers can be efficiently generated using a linear feedback shift register. To take advantage of this, the ticket holdings of individual masters are modified such that their sum is a power of two. In doing this, care must be taken to ensure that the ratios of tickets held by the components are not significantly altered. For example, if the ticket holdings of three components are in the ratio 1:2:4 ( $T=7$ ), they would be scaled to 5:9:18 ( $T=32$ ).

**Comparison for grant generation:** The random number is compared in parallel against all four partial sums, as shown in Figure 9. Each comparator outputs a ‘1’ if the random number is less than the partial sum at the other input. Since for the same random number, multiple comparators may output a ‘1’, it is necessary to use a standard priority selector circuit to ensure that at the end of a lottery, exactly one grant line is asserted. For example, for the request map 1011, assuming no scaling, if the generated random number is 5, only  $C_4$ ’s associated comparator will output a ‘1’. However, if the generated random number is ‘1’, then all the comparators will output a ‘1’, but the winner is  $C_1$ .

#### 4.4 Hardware Implementation: Dynamically Assigned Tickets

We next present a hardware implementation for a dynamic LOTTERYBUS architecture. Here the steps in executing a lottery are the same as those mentioned above, but the problem is considerably harder, since the assignment of lottery tickets to components is unknown at design time.

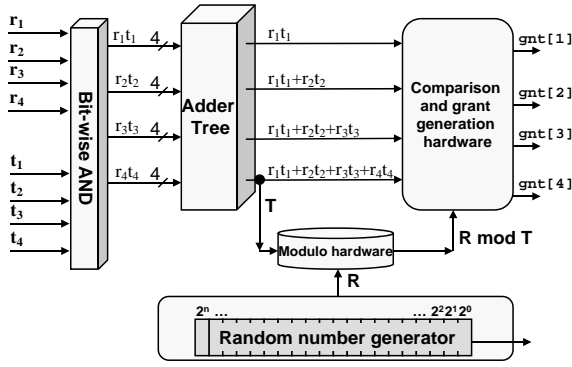


Figure 10: Lottery manager for dynamic LOTTERYBUS architecture

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  $\sum_{j=1}^i r_j \cdot t_j$  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,  $T = r_1 \cdot t_1 + r_2 \cdot t_2 + r_3 \cdot t_3 + r_4 \cdot t_4$  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 exchanging variable quantities of data and control messages during the course of their execution. Components  $M_1$  through  $M_4$  are masters, each of which is connected to a parameterized traffic generator, while components  $S_1$  through  $S_4$  are slaves. The parameters of each traffic generator can be varied to control the characteristics of the communication traffic generated by the SoC component it is connected to. Further details of the test-bed are provided in [19].

Figure 12(a) shows the results of experiments conducted to examine the ability of the LOTTERYBUS architecture to provide proportional bandwidth allocation under different classes of communication traffic. The x-axis depicts the nine different classes of communication traffic that were considered, the y-axis depicts the fraction of the total bus bandwidth allocated to various components, as well as the fraction of un-utilized bandwidth.

From Figure 12(a) we observe that for traffic classes where the bus utilization is high, the bandwidth allocated closely follows the assignment of lottery tickets. Tickets were assigned in the ratio 1 : 2 : 3 : 4, and for classes T4, T5, T7, T8, T9 the bandwidth allocated is (on the average) in the ratio 1.15 : 2.09 : 2.96 : 3.83. However, in cases where the bus is partly un-utilized, the allocation does not follow the assignment of tickets (T3, T6), but 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

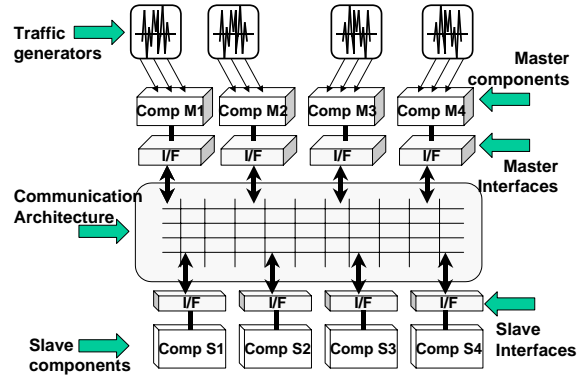


Figure 11: Test-bed for communication architecture performance evaluation [19]

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 T6. Under the LOTTERYBUS architecture, the same component with the same traffic class has an average latency of 2.7 bus cycles per word<sup>2</sup>.

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., T5, T6). The LOTTERYBUS architecture does not exhibit this phenomenon, ensuring low latencies for high priority communications.

### 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  $\mu$  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 upto 312.5MHz), 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 cell is written to an appropriate output queue. Each port polls its queue to detect presence of a cell. If it is not empty, the port issues a

<sup>2</sup>T6's data has been scaled down by 10X to fit the graph.

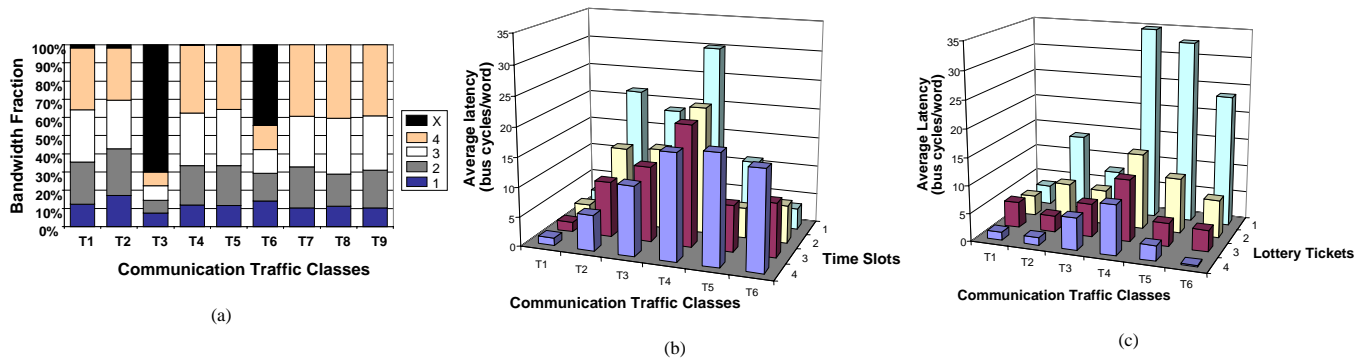


Figure 12: Performance under different communication traffic classes (a) Bandwidth allocation of LOTTERYBUS, (b) Communication latencies under TDMA, and (c) Communication latencies under LOTTERYBUS

dequeue signal to its local memory, and requests access to the shared system bus. Once it acquires the bus, it extracts the relevant cell from the shared memory, and forwards it onto the output link.

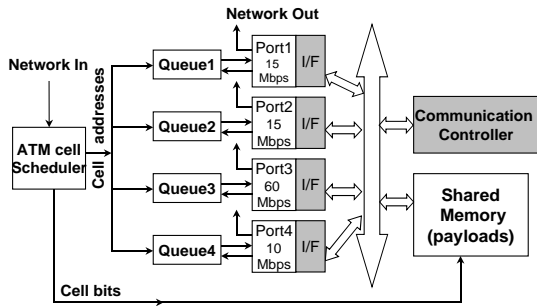


Figure 13: Cell forwarding in a 4 port ATM switch

The following quality-of-service requirements were imposed on the traffic flowing through the switch: (i) traffic through port 4 needs to pass through the switch with minimum latency, and (ii) ports 1, 2, and 3 must share the bandwidth in the ratio 1:1:4. We implemented three versions of the system, using (i) the static priority architecture, (ii) the TDMA architecture, and (iii) the LOTTERYBUS architecture. Lottery tickets, time-slots, and priorities were assigned uniformly in the ratio 1:1:4:6, for ports 1,2,3,4, respectively.

Table 1: Performance of the ATM switch

Comm. Arch.	Port 4 Latency (cycles/word)	Port 4 BW (%)	Port 3 BW (%)	Port 2 BW (%)	Port 1 BW (%)
Static priority	1.39	9.69	45.72	44.58	0.01
TDMA	9.84	10.09	47.29	21.31	21.30
Lottery	1.4	9.67	59.03	17.00	14.30

The results of the experiments are shown in Table 1. The columns denote performance metrics for each output port (bandwidth fraction and latency for Port 4, and only bandwidth fraction for Ports 1,2,3). The rows denote the performance under each alternative communication architecture. For example, Port 3 receives 59% of the total bus bandwidth under the LOTTERYBUS architecture. From the table we make the following observations. (1) Latency of high priority traffic at Port 4 is minimum under the static priority based architecture (1.39 cycles per word), while it is 7 times larger under the two-level TDMA based architecture (9.84 cycles per word). Under the LOTTERYBUS architecture, the average latency (1.4 cycles per word), is comparable to that under the static priority based architecture. (2) The bandwidth apportioned to the various ports under the static priority based architecture does not respect the reservations. Port 1 receives only 0.01% of the total bandwidth, being of the lowest priority. The same is true 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

- [1] D. D. Gajski, F. Vahid, S. Narayan and J. Gong, *Specification and Design of Embedded Systems*. Prentice Hall, 1994.
- [2] P. A. Laplante, *Real Time Systems Design and Analysis: An Engineers Handbook*. IEEE Press, Piscataway, NJ, 1993.
- [3] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, *Surviving the SOC Revolution*. Kluwer Academic Publishers, 1999.
- [4] "IBM On-chip CoreConnect Bus Architecture." <http://www.chips.ibm.com/products/coreconnect/index.html>.
- [5] D.Flynn, "AMBA: enabling reusable on-chip designs," *IEEE Micro*, vol. 17, no. 4, pp. 20–27, 1997.
- [6] "Sonics Integration Architecture, Sonics Inc." <http://www.sonicsinc.com>.
- [7] "Peripheral Interconnect Bus Architecture." <http://www.omimo.be>.
- [8] B. Cordan, "An efficient bus architecture for system-on-a-chip design," in *Proc. Custom Integrated Circuits Conf.*, pp. 623–626, 1999.
- [9] J. Turner and N. Yamanaka, "Architectural choices in large scale ATM switches," *IEICE Trans. on Communications*, vol. E-81B, Feb. 1998.
- [10] "On chip bus attributes specification 1 OCB 1 1.0, On-chip bus DWG." <http://www.vsi.org/library/specs/summary.htm>.
- [11] "Open Core Protocol Specification — v1.0." <http://www.sonics.com>, Oct. 1999.
- [12] A. S. Tanenbaum, *Computer Networks*. N.J.: Prentice Hall, 1989.
- [13] N. McKeown, M. Izzard, A. Mekkitikul, W. Ellersick and M. Horowitz, "The Tiny Tera: A packet switch core." *IEEE Micro*, vol. 17, pp. 26–33, Jan. 1997.
- [14] A. Smiljanic, "Flexible bandwidth allocation in terabit packet switches," in *Proc. of Intl. Conf. on Telecommunication*, June 2000.
- [15] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. of IEEE*, vol. 83, Oct. 1995.
- [16] A. C. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proc. Symp. on Operating Systems Design and Implementation*, pp. 1–12, 1994.
- [17] J. Buck and S. Ha and E. A. Lee and D. D. Masserchmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal on Computer Simulation, Special Issue on Simulation Software Management*, vol. 4, pp. 155–182, Apr. 1994.
- [18] F. Balarin, M. Chiodo, H. Hsieh, A. Jureska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki and B. Tabbara, *Hardware-software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, Norwell, MA, 1997.
- [19] K. Lahiri, A. Raghunathan, and S. Dey, "Evaluation of the traffic performance characteristics of system-on-chip communication architectures," in *Proc. Int. Conf. VLSI Design*, pp. 29–35, Jan. 2001.
- [20] "CBC9—VX datasheet." <http://www.necel.com>.