

國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

應用在硬式即時系統的頻寬仲裁演算法

Bandwidth-Guaranteed Arbitration Scheme
for Hard Real-Time Applications

研究生： 林步青

指導教授： 周景揚 博士

中華民國 九十五年 六月

應用在硬式即時系統的頻寬仲裁演算法


研究生：林步青

指導教授：周景揚 博士

國立交通大學

電子工程學系暨電子研究所碩士班

摘要



在共用匯流排的單晶片系統中，當不同的矽智產同時提出匯流排存取的需求時，便會發生匯流排衝突的情形。因此，必需設計仲裁器來強制處理匯流排衝突的情形。在不同的應用中，矽智產可能有即時或是頻寬的需求(或是兩者都有)。設計一個仲裁器演算法同時滿足即時與頻寬的兩種不同的需求是一件很困難的事。在這篇論文中，我們提出了一個名為 RB_lottery 的仲裁器演算法，用來同時解決即時通訊與頻寬分配的需求。除了滿足即時通訊的需求，新提出的演算法提供更佳的頻寬分配能力。在我們的實驗結果中也可以發現，相較於其它已經提出的系統單晶片仲裁器演算法，在滿足即時與頻寬的需求上，我們所提出的演算法有更好的表現。

BANDWIDTH-GUARANTEED ARBITRATION SCHEME FOR HARD REAL-TIME APPLICATIONS

Student: Bu-Ching Lin Advisor: Dr. Jing-Yang Jou

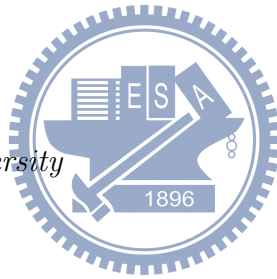
Department of Electronics Engineering
Institute of Electornics
National Chiao Tung University

Abstract

On an SoC bus, contentions occur while different IP cores request the bus access at the same time. Hence an arbiter is mandatory to deal with the contention issue on a shared bus system. In different applications, IPs may have real-time and/or bandwidth requirements. It is very difficult to design an arbitration algorithm to simultaneously meet these two requirements. In this thesis, we propose an innovative arbitration algorithm, RB_lottery, to meet both of the requirements. It can provide not only the hard real-time guarantee but also the precise bandwidth controllability. The experimental results show that RB_lottery outperforms several well-known existing arbitration algorithms.

Acknowledgements

I would like to express my gratitude to all those who gave me a great help to complete the thesis. My sincerest appreciation goes to my advisers, Dr. Jing-Yang Jou and Dr. Juinn-Dar Huang, for their guidance in this thesis. As well as the helpful suggestions come from Geeng-Wei Lee and Cheng-Yeh Wang. Without their help I could not get this work well. Last but not least, I wish to thank my parents, family and friends for their invaluable support, advise and encouragement throughout my study years.



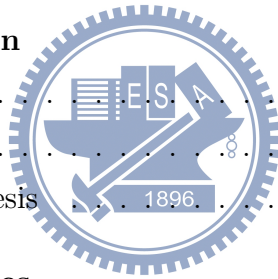
National Chiao Tung University

June 2006

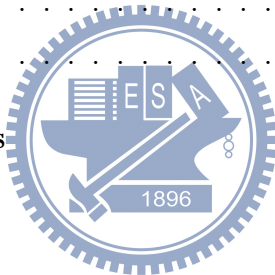
Bu-Ching Lin

Table of Contents

Abstract (Chinese)	i
Abstract	ii
Acknowledgements	iii
List of Tables	vi
List of Figures	vii
Chapter 1. Introduction	1
1.1 Introduction	1
1.2 Our contribution	3
1.3 Organization of the thesis	3
Chapter 2. Preliminaries	4
2.1 The purpose of arbiter	4
2.2 Previous works	5
2.2.1 Fixed priority algorithm	5
2.2.2 TDMA algorithm	7
2.2.3 Lottery algorithm	8
2.2.4 RT_lottery algorithm	15
2.3 Motivations	18
Chapter 3. Proposed Algorithm	20
3.1 Real-time bandwidth-regulating lottery(RB_lottery)	20
3.2 The details of bandwidth regulator	23

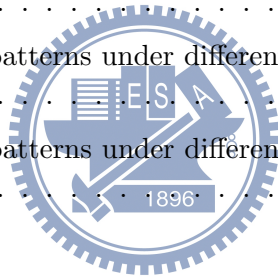


3.2.1	Observation window	23
3.2.2	Bandwidth registers	24
3.3	Our approaches	24
3.3.1	Fixed real-time bandwidth-regulating lottery (FRB)	25
3.3.2	Adaptive real-time bandwidth-regulating lottery (ARB)	26
Chapter 4. Experimental Results		29
4.1	Experiment environment	29
4.1.1	SystemC model	29
4.1.2	Traffic types	30
4.1.3	Traffic behavior	32
4.2	Experiment 1	33
4.3	Experiment 2	38
4.4	Experiment 3	40
Chapter 5. Conclusions		41
Bibliography		42



List of Tables

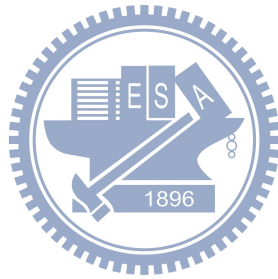
2.1	The allocated bandwidth of each master under equal ticket assignment . .	13
2.2	Weight tuning fails to meet the requirement	17
2.3	The of existing arbitration algorithms	18
4.1	The behavior of each master in the experiments	33
4.2	The number of fail patterns under different arbitration algorithms	35
4.3	The summaries of experiment 1	37
4.4	The number of fail patterns under different size of observation window in FRB	38
4.5	The number of fail patterns under different size of observation window in ARB	39
4.6	The number of fail patterns under different size of bandwidth variance in ARB	40



List of Figures

1.1	An example of shared bus architecture	2
2.1	An example of fixed priority based shared bus architecture	5
2.2	Bandwidth allocation under a fixed priority based architecture	6
2.3	An example of two-level TDMA shared bus architecture	7
2.4	An example of the maximum response latency in TDMA based architecture	8
2.5	The Lottery communication architecture	9
2.6	An example of Lottery shared bus architecture	11
2.7	Average response latencies under different tickets assignment ratio	14
2.8	Bandwidth allocation under different tickets assignment ratio	14
2.9	The architecture of RT lottery	15
2.10	The weight tuning mechanism which redistributes the ticket assignment. It moves part of tickets from S_{more} to S_{less}	16
3.1	The architecture of proposed arbitration algorithm	21
3.2	An example of proposed arbitration algorithm architecture	23
3.3	A sequence of fixed size windows	24
3.4	The algorithm flow of FRB	26
3.5	The bandwidth variance architecture	27
3.6	The algorithm flow of ARB	28
4.1	An example system using transaction level model in SystemC	30
4.2	D type master (beat number = 5; interval time = 15)	31
4.3	D_R type master (beat number = 5; interval time = 15; $R_{cycle} = 10$)	31
4.4	ND_R type master (beat number = 5; interval time = 15; $R_{cycle} = 10$)	32
4.5	Figure of Table 4.2	36

4.6 Figures of Table 4.5 and Table 4.4 39



Chapter 1

Introduction

1.1 Introduction

In SoC systems, intellectual properties (IPs) often communicate with each other to complete the functions. For example, the CPU accesses data from the memory and other I/O devices. Such communication is commonly built through the same media called bus. The shared buses act as the communication channels between those IPs. There are two major components, masters and slaves, in this shared bus architecture. Those IPs who initiate requests to access the shared buses are called the masters. These requests could be either read or write transactions. Unlike the masters, slaves have no controllability over the shared buses. The CPUs and the memories, for instance, are masters and slaves, respectively. An example of shared bus architecture is shown in Figure 1.1. There are three masters and two slaves in the system. These masters can initiate requests to the bus and the corresponding slave responses them with the proper data transferred through the shared bus [1–8].

Since masters on the same shared bus may initiate requests at the same time, an arbiter is required to decide which master is the current owner without bus conflicts. The arbiter could be implemented in a centralized or a distributed fashion. Besides, distinct arbitration algorithms may lead to different system performance. Therefore, the arbiter should be designed carefully in high performance systems [7,9].

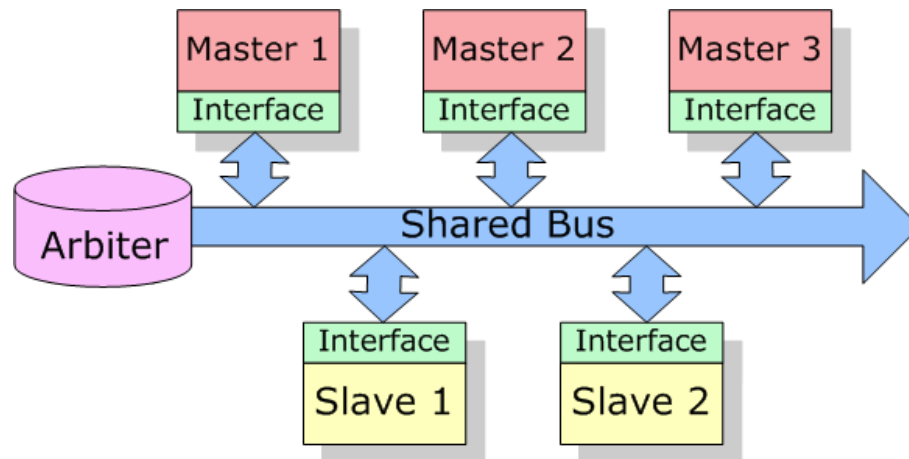


Figure 1.1: An example of shared bus architecture

The two major advantages of the shared bus architecture are high scalability and low hardware cost. In the configuration, a new IP with the same interface can be easily added into an SoC platform. Since the communication channel is shared, the hardware cost can be relatively low [9,10].

However, a shared bus architecture may limit the communication bandwidth. The maximum communication bandwidth of a bus is defined as the bus width multiplied by the bus clock frequency. It restricts the communication data amount within a fixed period. The major difficulty to design an arbiter is to meet different requirements simultaneously. A master has the bandwidth requirement if it requires at least a certain amount of bus bandwidth. In addition, some masters with the real-time requirement demand their requests accomplished within a fixed number of clock cycles. Most arbitration algorithms target at either real-time requirements or bandwidth requirements, but few of them deal with both of requirements well. It is a tough challenge to design an arbiter for a real-time system with bandwidth requirements.

1.2 Our contribution

In this thesis, we first explore the difficulty of meeting the real-time and bandwidth requirements simultaneously and then propose an innovative arbitration algorithm, called RB_lottery, to provide the hard real-time guarantee and the fine-grained bandwidth control. RB_lottery is a three-level arbitration algorithm. It handles the real-time requirements in the first level and allocates the bandwidth according to the requirements with precision less than 2% error in the following levels. Our experimental results also demonstrate that it outperforms other existing arbitration algorithms.

1.3 Organization of the thesis

The remainder of this thesis is organized as follows. Several existing arbitration algorithms are briefly introduced in Chapter 2. Chapter 3 presents the detail of the proposed arbitration algorithm, RB_lottery. Experimental results are reported in Chapter 4. Finally, we conclude this thesis in Chapter 5.

Chapter 2

Preliminaries

In this section, we briefly introduce the purpose of arbiter in Section 2.1. And then several existing arbitration algorithms [11–13] are presented in Section 2.2. The characteristics of each arbitration algorithm are summarized in Section 2.3.

2.1 The purpose of arbiter

The most important coordination circuit in SoC shared bus systems is the arbiter. An arbiter acts as a traffic officer standing at an intersection who decides which car may pass through next. When a request is initiated, the arbiter promptly grants the corresponding action, delaying any other requests until the current one is completed. While two requests are initiated at the same time, the arbiter decides which one should be granted first and then the other. The arbiter guarantees that there are no two actions under way contemporaries, just like the traffic officer prevents accidents by ensuring no two cars passing through the intersection on a collision course [14–17].

An arbiter not only needs to ensure that only one master accesses the shared bus, but also schedules the requests with requirements satisfyingly and efficiently. There are various requirements in different applications. For example, in real-time systems each transactions should be accomplished within a fixed number of clock cy-

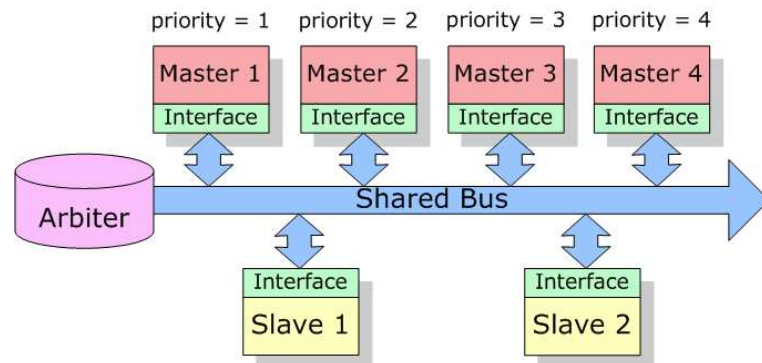


Figure 2.1: An example of fixed priority based shared bus architecture

cles which is the maximum response latency constraints. And in multimedia systems some masters with bandwidth constraints desire a fraction of total bandwidth. It is a challenge to design an arbiter for a real-time system with bandwidth requirements.

2.2 Previous works

2.2.1 Fixed priority algorithm

The fixed priority algorithm is one of the most commonly used on-chip communication architecture in modern SoC systems [1]. Each master is statically assigned an unique priority [18]. Whenever the contending requests come from different masters, the arbiter makes a decision of permitting the buses controllability to the highest-priority master.

As shows in Figure 2.1, there are four masters in a shred bus architecture. The priority of Master 1 is statically assigned to 1, Master 2 is 2, and so on¹. While bus contentions, the arbiter simply grants the master with the highest priority among the contending requests.

¹We use a convention in which lower numeric value indicate higher priority. The master with priority 1 is of the highest importance. The master of priority 2 is of next highest importance as well.

Because of its simplicity and low hardware cost, the fixed priority algorithm is still widely used in today's design. However, the masters with lower priority could be completely starved by higher priority ones.

An example system shows the relationship between bandwidth allocation and priority assignment in Figure 2.2. The four masters with the same traffic behavior are assigned a unique value as the priority from 1 to 4. All of the possible bandwidth combinations are experimented to measure the fraction of bus bandwidth allocation under different priority assignments. The notation "1:2:3:4" means that master 1 has the highest priority, master 2 is the second, and master 4 is the lowest. For example, as the priority of master 1 decreases from 1 to 4, the fraction of bandwidth is step-wise decreased from left to right in Figure 2.2. The results show that the fraction of bandwidth is extremely sensitive to the assigned priority and the low priority traffic starves as pending requests in higher priority traffic.

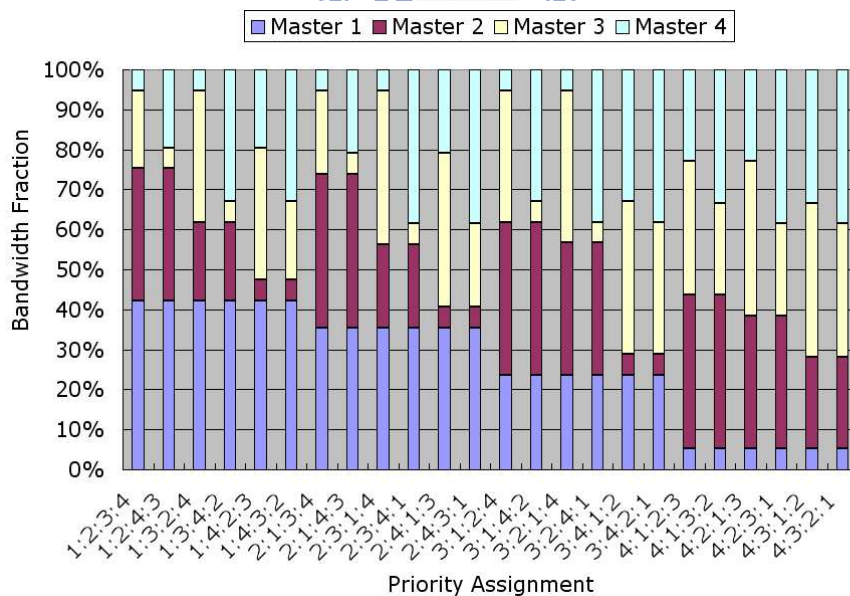


Figure 2.2: Bandwidth allocation under a fixed priority based architecture

2.2.2 TDMA algorithm

In the time division multiplexed access (TDMA) algorithm, the execution time is divided into equal-sized time slots, such as clock cycles, and each slot is statically assigned to a particular master [6]. If the master associated with the current time slot has a pending request, the arbiter permits the transaction immediately, and the time wheel is rotated by one slot. While there is no pending requests in the current slot owner, the slot will be wasted. In order to alleviate the wasted slots, a second level arbitration algorithm is usually adopted to permit bus granted to the other requesting masters.

It is an example of two-level TDMA scheme in Figure 2.3. The time wheel is divided into eight time slots which statically assign to an unique master. For example, if the current time slot is reserved to M1, the pending request of M1 is granted immediately. Otherwise, the second level arbitration algorithm is used to eliminate the waste slots. The second level arbitration algorithm can adopt any other algorithms depending on the applications.

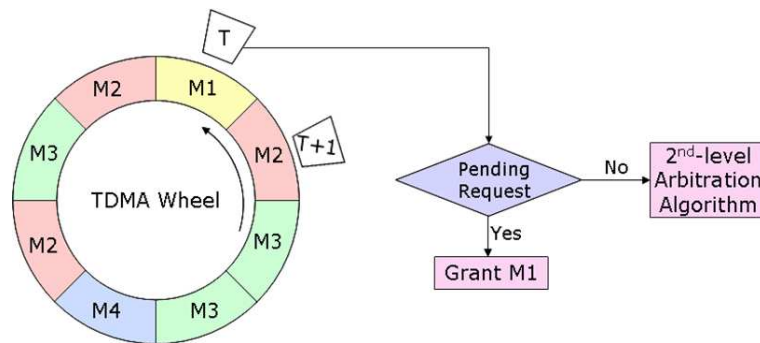


Figure 2.3: An example of two-level TDMA shared bus architecture

Because of every master is allocated a certain amount of time slots, TDMA guarantees not only a minimum bandwidth allocation but also the worst-case response latency. As shown in Figure 2.4, the reservation time slots are designed in

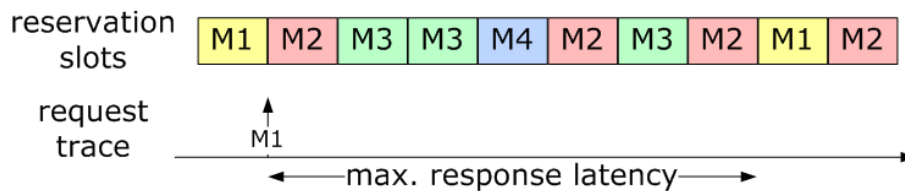


Figure 2.4: An example of the maximum response latency in TDMA based architecture

the time wheel at the first line. The second line shows the requesting pattern. If M1 initiates a request just after the owned time slot, the request remains pending until the next owned time slot. It is the worst case of the response latency for M1.

The time slots are designed depending on the requirements either real-time or bandwidth. However, it is difficult to design the time slot sequence for a non-periodic system. Moreover, the masters allocated more time slots also got more bandwidth allocation. The more allocated time slots mean the higher possibility to get granted which results in allocating more bandwidth. If the communication behaviors are complex and irregular, the analysis of time slots design is very difficult in order to get a high performance system. In addition, TDMA hardly handles the real-time and bandwidth requirements simultaneously. A master with more time slots results in short response latencies and more bandwidth allocation. It is difficult to separately control latency and bandwidth. For example, how to design a master with low bandwidth requirement and short response latency. Similarly, a master which needs high bandwidth without short response latency is also difficult to design.

2.2.3 Lottery algorithm

Another communication architecture, LOTTERYBUS, is proposed in [19]. It stochastically grants one of the contending masters according to the ticket assigned to them either statically or dynamically [20–22]. The lottery tickets acted as the

weight are accumulated through the lottery manager while bus contentions occur. A master is stochastically selected to get the bus access. In other words, it is a weighted random arbitration mechanism to grant a master while contention.

As shown in Figure 2.5, there are three masters on the bus and each of them has a number of “lottery tickets” as the probability of bus granted. First, the lottery manager accumulates the possessive tickets from one or more requesting masters in the shared bus. And then a master granted to access system bus is probabilistically chosen by the lottery manager from all contending masters.

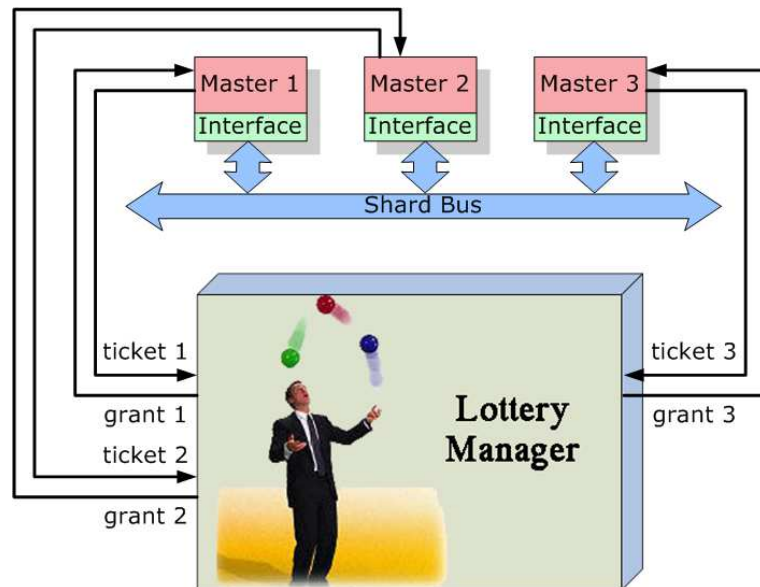


Figure 2.5: The Lottery communication architecture

While there are a set of bus masters, M_1, M_2, \dots, M_n , and each of them hold t_1, t_2, \dots, t_n tickets. The corresponding requests are represented by a set of boolean variables, r_1, r_2, \dots, r_n . If there are pending requests in master M_i , the corresponding boolean variable r_i is 1, otherwise it is 0. Equation 2.1 shows the probability for each master in a shared bus.

$$P(M_i) = \frac{r_i t_i}{\sum_{j=1}^n r_j t_j} = \frac{r_i t_i}{T} \quad (2.1)$$

In the first step, the lottery manager accumulates the total number of tickets possessed by the contending masters, given by $T = \sum_{j=1}^n r_j t_j$. In the second step, a random number is generated from the range $[0, T)$ ² to determine which master is granted. For example, while the random number falls in the range $[0, r_1 t_1)$, the bus is granted to master M_1 . In other words, master M_2 is granted when the random number is in the range $[r_1 t_1, r_2 t_2)$. Generally, the bus is granted to master M_{i+1} , if the random number lies in the range $[\sum_{k=1}^i r_k t_k, \sum_{k=1}^{i+1} r_k t_k)$.

For example, as shown in Figure 2.6, there are four masters Master 1, Master 2, Master 3 and Master 4 in the system. Each of them is assigned 1, 2, 3, and 4 tickets, respectively. If Master 1, Master 3 and Master 4 have pending requests, the lottery manager sums up the current tickets $\sum_{j=1}^n r_j t_j = 1 + 3 + 4 = 8$. And it generates a random number, *e.g.* 5, from the range $[0, 8)$ uniformly. As a result, the number lies between $r_1 t_1 + r_2 t_2 + r_3 t_3 = 4$ and $r_1 t_1 + r_2 t_2 + r_3 t_3 + r_4 t_4 = 8$, the bus is granted to Master 4.

One of the main concerns is the worst case of response latencies while the master is starved by the higher ones during designing a communication architecture. For the Lottery architecture, the probability, P , that a component with t tickets is able to access the bus within n drawings is given by the expression $1 - (1 - \frac{t}{T})^n$. It indicates the probability of obtaining access the bus converges rapidly to one. In other words, no master is starved even when it has few ticket assignment.

Next, we analyze the ability of bandwidth allocation for each master in Lottery. We define the maximum bandwidth as the greatest possible communication

²The set $[a, b)$ includes all the integers between a and b, inclusive of a but not b.

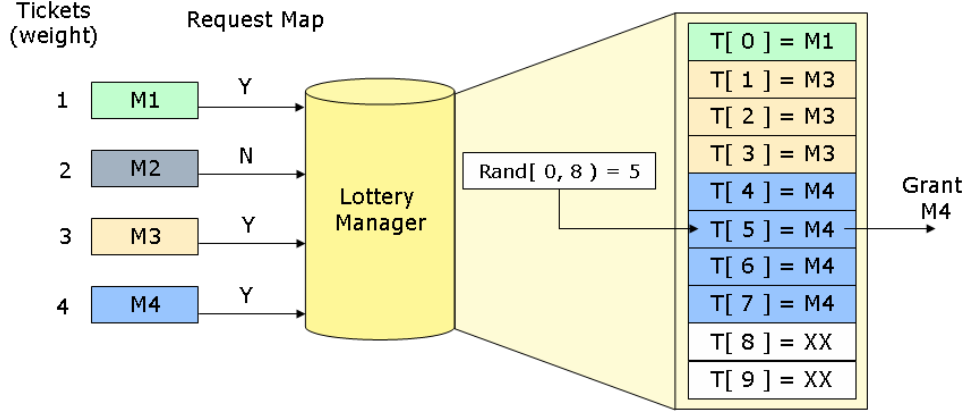


Figure 2.6: An example of Lottery shared bus architecture

amount of a master and the allocated bandwidth as the bandwidth allocation of a master after simulation. If all masters have the same maximum bandwidth in a shared bus, Equation 2.2 shows the percentage of bandwidth allocated to master M_i .

$$P'(M_i) = \frac{t_i}{\sum_{j=1}^n t_j} = \frac{t_i}{T'} \quad (2.2)$$

Clearly, Lottery ensures a fraction of bandwidth allocation for each master under the assumption of similar maximum bandwidth of each master. Because of the unrealistic assumption, it could not be directly used in SoC communication architecture.

In addition, since master M_i initiates a request, it has the probability $\frac{t_i}{T'}$ to be granted and probability $1 - \frac{t_i}{T'}$ to be pended. Let N be the number of drawings for master M_i to be able to access the bus. The expected time³ of drawings is equal to $\frac{T'}{t_i}$. It means that the average response latencies for master M_i with t_i tickets

³The required number of trials with a probability p of being a success are performed until a success occurs is a geometric random variable. And the expected value of a geometric random variable is $\frac{1}{p}$.

is $\frac{T'}{t_i}$ [23]. We find that the ticket assignment influence not only the bandwidth allocation but also the average response latencies. The more lottery tickets are assigned to a master, the higher granted probability and the fraction of bandwidth it has. Besides, masters with more tickets also have shorter average response latencies.

However, the analysis is based on the assumption that all masters have equal or similar maximum bandwidth in the system. In other words, it does not take the traffic behavior of masters into consideration during ticket assignment. While there are masters of greatly diverse traffic behaviors in a SoC system, Lottery fails to control the bandwidth allocation, which is one of the announced main advantages.

For example, if there are three masters, Master 1, Master 2 and Master 3, in a system, and each them requires at least 30% of total bandwidth. According to the bandwidth requirements, the ratio of ticket assignment is 1:1:1 by intuition which means equal granted probability while contention. As shown in Table 2.1, the sixth column is the allocated bandwidth when the maximum bandwidth of each master is 40%. All of the master meet the bandwidth requirement since they all get 33% allocated bandwidth. As the maximum bandwidth of Master 1 increases to 60% in the fourth column, the allocated bandwidth for each master is still very close to the bandwidth requirements. However, when the maximum bandwidth of master 1 increases to 80% in the second column, the allocated bandwidth of Master 2 and Master 3 misses the bandwidth requirements. As a result, the master's maximum bandwidth as well as the bandwidth requirement should be taken into consideration simultaneously during ticket assignments.

As shown in Equation 2.2, the bus granted probability is $\frac{t}{T'}$ and the average response latencies is $\frac{T'}{t}$. The more lottery tickets are assigned to a master, the higher granted probability and bandwidth allocation it has. In addition, masters with more tickets will also have shorter average response latencies. The ticket assignment

Table 2.1: The allocated bandwidth of each master under equal ticket assignment

	the max. bandwidth of master 1, master 2 and master 3				
	80%:40%:40%	70%:40%:40%	60%:40%:40%	50%:40%:40%	40%:40%:40%
Master 1	49%	46%	42%	38%	33%
Master 2	24%	26%	28%	30%	33%
Master 3	24%	26%	28%	30%	33%

not only influences the bandwidth allocation but also change the average response latencies.

As a result, how to assign the tickets assignment to a master without high bandwidth requirement needed the low response latencies? We take three masters with 60% maximum bandwidth into consideration in the following example. The system works when all of them gets at least 30% of total bandwidth. However, if Master 1 also requires the average response latencies less than 2 cycles. We demonstrate the resultant latencies and bandwidth allocation under different ticket assignment ratio from 1:1:1 to 10:1:1. The notation “10:1:1” means that there are 12 tickets in total. Master 1 has 10 tickets. Master 2 and Master 3 share out the other two tickets.

As shown in Figure 2.7, we find that the average response latencies of Master 1 is 4 cycles when ticket assignment is 1:1:1. When the tickets of Master 1 become 10 times larger than others, the average latency is less then 2 cycles which meets the latency constraint.

However, it leads Master 2 and Master 3 to violate the bandwidth requirements. As shown in Figure 2.8, all masters get about 30% bandwidth under equal ticket assignment. When increasing the tickets of Master 1 to 10 times larger than others, it receives more than 45% of total bandwidth. The bandwidth allocation misses the bandwidth requirement when Master 1 meets the average response la-

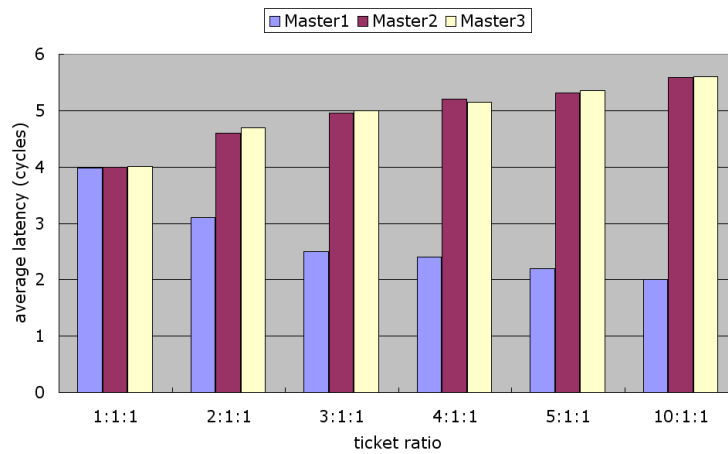


Figure 2.7: Average response latencies under different tickets assignment ratio

tencies requirements. In a word, the ticket assignment influences both response latencies and bandwidth allocation.

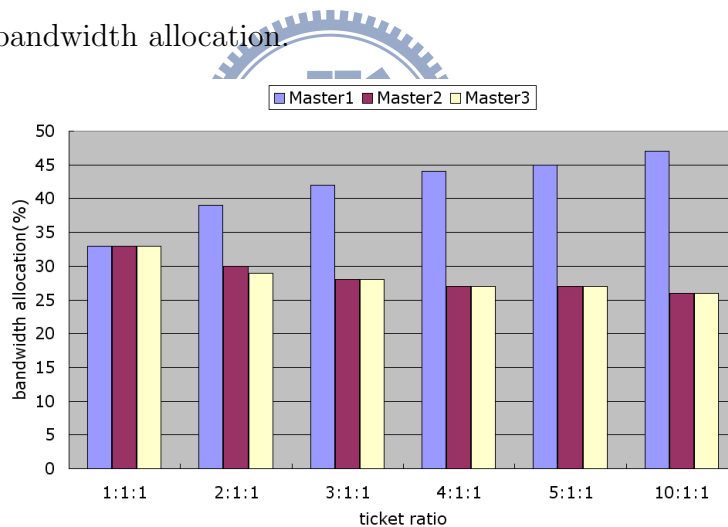


Figure 2.8: Bandwidth allocation under different tickets assignment ratio

Lottery shows better performance in response latencies and bandwidth allocation than the other conventional arbitration algorithms like fixed priority and TDMA. It has the advantages of higher controllability over bandwidth allocation and lower response latency for the high priority communication. However, Lottery guarantees the quality-of-service under the assumption of similar or equal masters's

maximum bandwidth. If the behavior of each master varies a lot in a system, the ticket assignment need to consider both traffic behavior and bandwidth requirement simultaneously.

2.2.4 RT_lottery algorithm

A Lottery-based arbitration algorithm, RT_lottery, is proposed in [21]. As shown in Figure 2.9, it is a two-level arbitration algorithm including a real-time handler and a Lottery with tuned weight. The real-time handler providing hard real-time guarantees governs all requests with real-time requirements first. Then the Lottery with tuned weight allocates the bandwidth according to the fine tuned tickets to meet the bandwidth requirements.

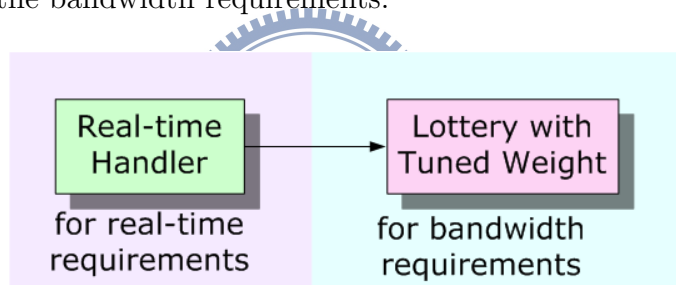


Figure 2.9: The architecture of RT_lottery

A set of real-time counters in the real-time handler record the interval from the current request to corresponding deadline for each master with real-time requirements in real-time handler. It decreases by one every cycle until the current request is served. If there are still pending requests while the real-time counter is zero, it is the real-time violation. The real-time handler also uses a warning line mechanism which considers the worst contending case to provide hard real-time guarantee [24–26]. If the request of a master is still pending and the corresponding real-time counter is less than the warning line, the real-time handler assigns the highest priority to the master and grants the bus access immediately.

Ticket assignment which plays an important role in Lottery-based algorithms should consider bandwidth requirement and the traffic behavior simultaneously. An simulation-based tickets redistribution mechanism, weight tuning, is used to decide the proper ticket assignment. It analyzes the simulation results and divides the masters into three groups, S_{more} , S_{less} , and S_{meet} . S_{more} means that the simulate bandwidth larger than the required bandwidth. While the masters's simulate bandwidth violate the minimum required bandwidth, it groups into S_{less} . And the others just meet the bandwidth requirement belong to S_{meet} . As shown in Figure 2.10, in order to meet all bandwidth requirement, weight tuning mechanism increases the granted probability of S_{less} .

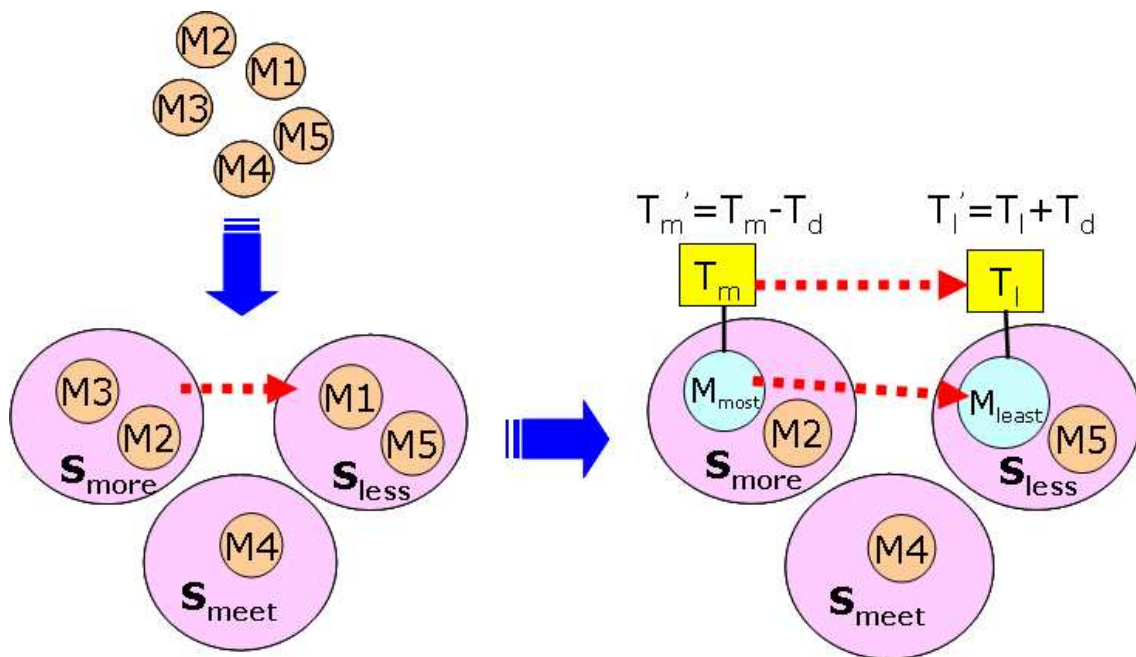


Figure 2.10: The weight tuning mechanism which redistributes the ticket assignment. It moves part of tickets from S_{more} to S_{less} .

However, the weight tuning mechanism does not always successfully fulfill the bandwidth requirements. While it is a highly loaded bus, the bandwidth allocation still violates the requirements. For example, there are three masters, Master

1, Master 2 and Master 3, which requires at least 30% of total bandwidth. The maximum bandwidth of Master 1, Master 2 and Master 3 are 80%, 30% and 30%, respectively. As shown in Table 2.2, each master has 100 tickets at the beginning in the second column. After several iterations of weight tuning, Master 1 takes 53% of total bandwidth even when it has only two tickets. Master 2 and Master 3 still violate the bandwidth requirements since they have most of tickets.

Table 2.2: Weight tuning fails to meet the requirement
the ticket assignment of three masters

	100:100:100	60:120:120	34:133:133	16:142:142	8:146:146	2:149:149
Master 1	60%	58%	56%	55%	54%	53%
Master 2	19%	20%	21%	22%	22%	23%
Master 3	19%	20%	21%	22%	22%	23%

RB_lottery takes the advantages of Lottery but has better performance. It provides hard real-time guarantee and better bandwidth control than Lottery. However, in the highly loaded bus, RB_lottery fails to meet the requirements. Hence, a more powerful arbitration scheme is surely demanded.

2.3 Motivations

When there are real-time and bandwidth requirements in a system, the existing arbitration algorithms can not provide good solutions. Table 2.3 lists the characteristics of the existing arbitration algorithms.

Table 2.3: The of existing arbitration algorithms

	Pros.	Cons.
fixed priority	simplicity area efficiency	starvation for low latency traffic no real-time consideration no means for bandwidth control
TDMA	deterministic behaviors time slot reserved bandwidth allocation	hard to design time slot sequences no real-time guarantee time slots influence latency and bandwidth
Lottery	average latencies guarantee tickets reserved bandwidth allocation	similar requesting bandwidth assumption no real-time consideration tickets influence latency and bandwidth
RT_lottery	hard real-time guarantee	fail in diverse masters' requesting bandwidth

We have the following summaries:

- fixed priority algorithm:

Because of the simplicity and low hardware cost, the fixed priority algorithm is still widely used in bus systems. However, a master with lower priority could be completely starved by higher priority ones.

- TDMA algorithm:

The allocated time slots of each master in TDMA provides fixed and predictable bandwidth. It also guarantees the worse case of response latencies

due to the time slot design. However, the design of time slot sequences is very difficult in an unpredictable system. Moreover, TDMA could not separately consider bandwidth allocation and real-time requirements.

- Lottery algorithm:

Lottery algorithm provides better control over bandwidth and guarantees the average response latencies. But it assumes that each master has equal or similar traffic behaviors. Moreover, it does not take real-time system into consideration.

- RT_lottery algorithm:

RT_lottery improves the ability to handle the real-time requirements and gives a systematic analysis flow during ticket assignments. Nevertheless, RT lottery still fails in the highly loaded bus.

As a result, we design an arbitration algorithm dealing with both bandwidth and real-time requirements. It takes the advantages of lottery based algorithms, but provides better bandwidth controllability over ticket assignments.

Chapter 3

Proposed Algorithm

The proposed real-time bandwidth-regulating lottery, called RB_lottery briefly, is introduced in the following sections. First, we present the architecture details of RB_lottery and then give an example. Second, we presents our approaches, FRB and ARB.

3.1 Real-time bandwidth-regulating lottery(RB_lottery)

The proposed arbitration algorithm, RB_lottery, is based on our previous work, RT_lottery [21]. An extra component, the bandwidth regulator, is added into RT_lottery which provides precise controllability over the bandwidth allocation. As shown in Figure 3.1, it is a three-level arbitration algorithm to solve real-time and bandwidth requirements simultaneously.

As shown in Figure 3.1, RB_lottery is a three-level arbitration algorithm. The first level, the real-time handler, addresses the requests with the real-time requirements in advance. The following levels, bandwidth regulator and Lottery with tuned weight, work together to deal with the bandwidth requirements.

- Real-Time Handler:

It is proposed in RT_lottery [21] and is described elaborately in Section 2.2.4.

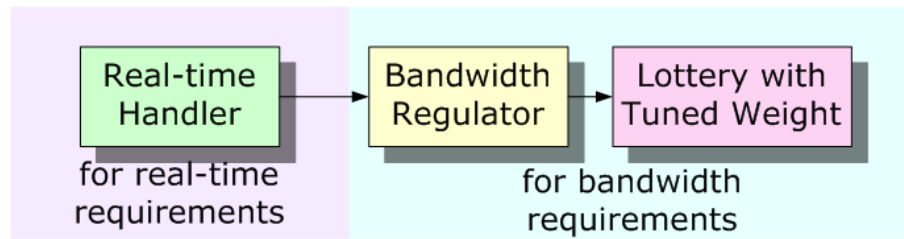


Figure 3.1: The architecture of proposed arbitration algorithm

The real-time handler governs the requests with real-time requirements and provides the guarantees of hard real-time.

- **Bandwidth Regulator:**

A traffic monitor with the controllability over bandwidth allocation records the bus communication behaviors. It includes a set of bandwidth registers which records the transaction cycles of each master in a fixed period called window. When a master gets granted for bus access, the corresponding bandwidth register records the transaction cycles. The total transaction amounts recorded in the bandwidth register could be regarded as the allocated bandwidth. Next, the bandwidth regulator checks the allocated bandwidth of each master and blocks the requests of masters with more allocated bandwidth than required. Unless the blocking signal is released, there are no chances for the blocked masters to get granted. Because the over-served masters are blocked, the masters with less allocated bandwidth than required have higher possibility to be granted.

- **Lottery with Tuned Weight:**

It is briefly introduced in Section 2.2.3 and Section 2.2.4. The contending masters are granted by the lottery manager and the ticket assignment are decided by a simulation-based mechanism named weight tuning.

As shown in Figure 3.2, a simple example explains how RB_lottery works. An addition circuitry called the bandwidth regulator is added into the RT_lottery. It monitors the communication behaviors of each master and blocks the requests of masters met the bandwidth requirement temporarily. The allocated bandwidth is recorded in the bandwidth register. Whenever a master gets granted, the corresponding bandwidth register adds one every cycle until the master completes the communication transaction. The master meets the bandwidth requirements unless and until the allocated bandwidth larger than the required bandwidth. For example, if the allocated bandwidth of M4 recorded in the bandwidth register is larger the required bandwidth, the bandwidth regulator blocks the request signal temporarily. As the instant shown, the four masters, Master 1, Master 2, Master 3 and Master 4, are static assigned 1, 2, 3 and 4 tickets, respectively. If there are pending requests in Master 1, Master 3 and Master 4, the real-time handler checks the corresponding real-time counters at the first level. If the real-time counter of a master is smaller than the warning line, it get the highest priority and is granted the bus access immediately. The real-time handler provides the hard real-time guarantee for masters with real-time requirements. Otherwise, if all real-time counters are larger than the warning line, the bandwidth regulator checks the bandwidth registers at the second level. The allocated bandwidth of each master recorded in the corresponding bandwidth register is compared with the required bandwidth. The master meets the bandwidth requirements unless and until the allocated bandwidth larger than the required bandwidth. The bandwidth regulator temporarily blocks the requests of masters who meets the bandwidth requirement in a fixed period of time.

In a word, RB_lottery is a kind of three-level arbitration algorithm. It separately deals with the real-time and bandwidth requirements at different levels. First, the real-time handler solves the real-time requirements at the first level. The second

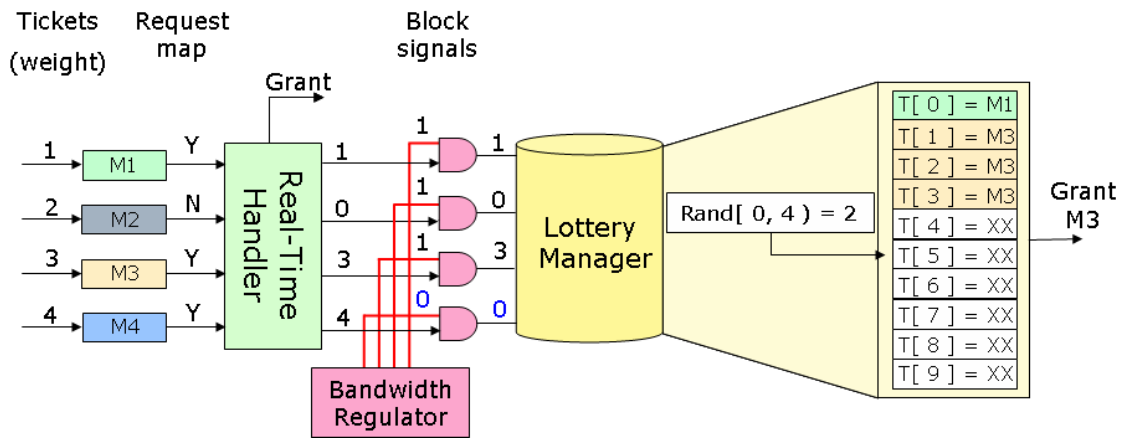


Figure 3.2: An example of proposed arbitration algorithm architecture

and third levels target at the bandwidth requirements. It records the communication behaviors and dynamically changes the granted probability.

3.2 The details of bandwidth regulator

In the following section, we present the details of the bandwidth regulator. First, we introduce the bandwidth registers recorded the communication behaviors during execution. Second, the period of time called the observation window is presented.

3.2.1 Observation window

As shown in Figure 3.3, the execution time is divided into a sequence of fixed size windows for observation. The size of an observation window is configurable. In each observation window, all the transactions are monitored to obtain the information of bandwidth allocation. Once a master gets its required bandwidth, the following requests from this master are temporarily blocked until next window starts.

The design of observation window is compromised of the hardware cost. Despite the hardware cost, we can use the unlimited size of observation window. However, it means that we also need to design a large set of bandwidth registers to record all transaction amounts for each master. It is unrealistic in hardware implementation.

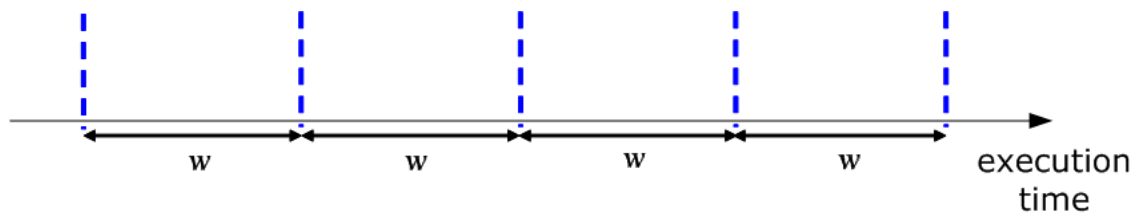


Figure 3.3: A sequence of fixed size windows

3.2.2 Bandwidth registers

A set of bandwidth registers record the communication behaviors of each master. The bandwidth registers reflect the bandwidth allocated to masters in an observation window. When a master get granted to the bus access, the corresponding bandwidth register records the amount of transactions. For example, a granted master initiates an 8-beat transactions, the corresponding bandwidth register increases its value by eight cycles when the transaction is completed.

3.3 Our approaches

The following sections describe the approaches of how does the RB_lottery precisely control the bandwidth allocation.

3.3.1 Fixed real-time bandwidth-regulating lottery (FRB)

One of the approaches of the RB_lottery is called FRB. FRB statically compares the allocated bandwidth with the required bandwidth and masks the requests from masters who has already met bandwidth requirements in an observation window. The allocated bandwidth of a master is larger than the required bandwidth, which is called over-served. Similar, the master with the allocated bandwidth less than the required bandwidth is under-served. Not until the next window comes or the pending request hits the warning line does the masking signal get released.

As shown in Figure 3.4, the algorithm flow of RB_lottery is divided into three blocks to explain. In block 1, the observation window is checked to see if it is ready to proceed to the next window. When the next window starts, all of the blocking signals are released. In block 2, the real-time handler detects the emergent real-time masters from the contending requests and grants the most emergent master immediately to avoid real-time violation. If no master is granted by the real-time handler, the lottery manager stochastically grants a master from the contending requests. The transaction amount of the granted master is recorded in its corresponding bandwidth register. In block 3, the allocated bandwidth of the granted master is calculated from the bandwidth register. If the granted master meets the bandwidth requirement, its next request is temporarily blocked in the current observation window.

FRB records the communication behavior in an observation window. The size of observation window compromises on the hardware cost. If we record all the communication behaviors, it needs a lot of registers to store the information. As a result, FRB divides the run-time into a sequence of windows and only considers the communication behaviors in the same window. The adjacent windows are independent in FRB.

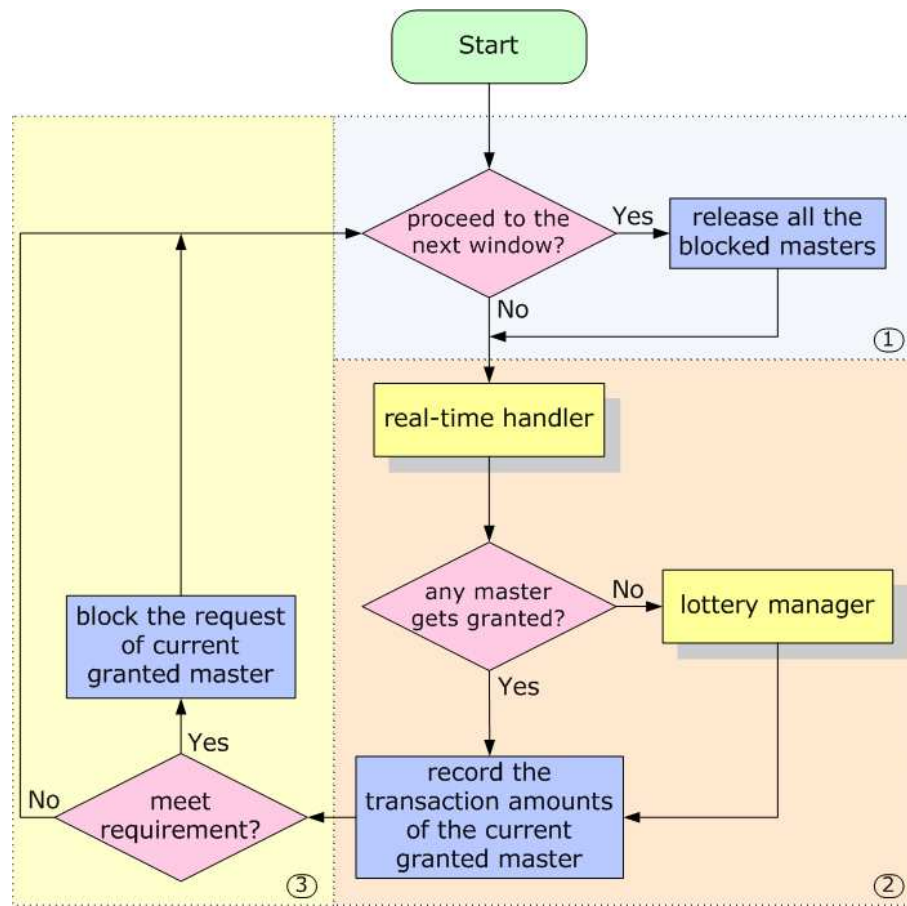


Figure 3.4: The algorithm flow of FRB

3.3.2 Adaptive real-time bandwidth-regulating lottery (ARB)

The other approach of the RB lottery is called ARB. There are a set of bandwidth variance which helps to dynamically change the boundary of blocking the requests of a master. ARB records the transaction amounts of each master and dynamically compares the allocated bandwidth with the required bandwidth and the bandwidth variance. If the allocated bandwidth is larger than the value of the required bandwidth plus the bandwidth variance, the bandwidth regulator masks the request of corresponding master. Similar to FRB, the over-served master could not get granted until the next window comes or the pending request hits the warning

line.

As shown in Figure 3.5, the bandwidth variance of each master starts at zero and is periodically increment or decrement according to the allocated bandwidth in the previous window [27]. If the allocated bandwidth is larger than the required bandwidth in the current window, the bandwidth variance will be decreased in the next window. Otherwise, it will be increased. The bandwidth variance has both an upper bound and lower bound. If a master gets more bandwidth than required frequently, the bandwidth decreases until it hits the lower bound. Similar, if the allocated bandwidth of a master is less than the required bandwidth frequently, the corresponding bandwidth boundary increases until it hits the upper bound. The bandwidth variance reflects the relationship between the adjacent windows. The larger size of bandwidth variance means the more adjacent windows took into consideration during tuning the bandwidth allocation.

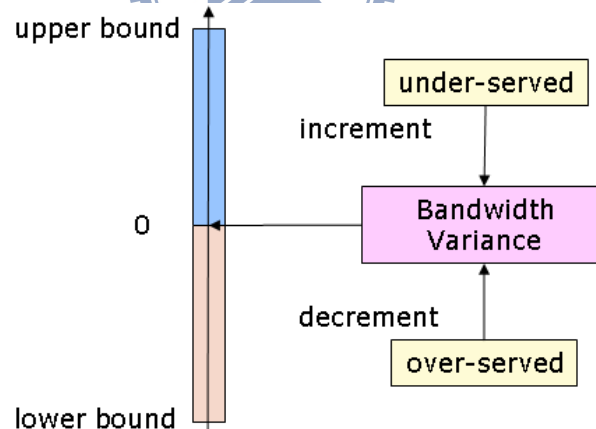


Figure 3.5: The bandwidth variance architecture

As shown in Figure 3.6, the major different between FRB and ARB is the block 1. As what FRB does, ARB checks the running window first in block 1. In the current window, ARB records the transaction amounts and temporarily blocks the requests of over-served master. However, if it is a new window, ARB not only

releases all the masked requests but also records the allocated resultant into the bandwidth boundary.

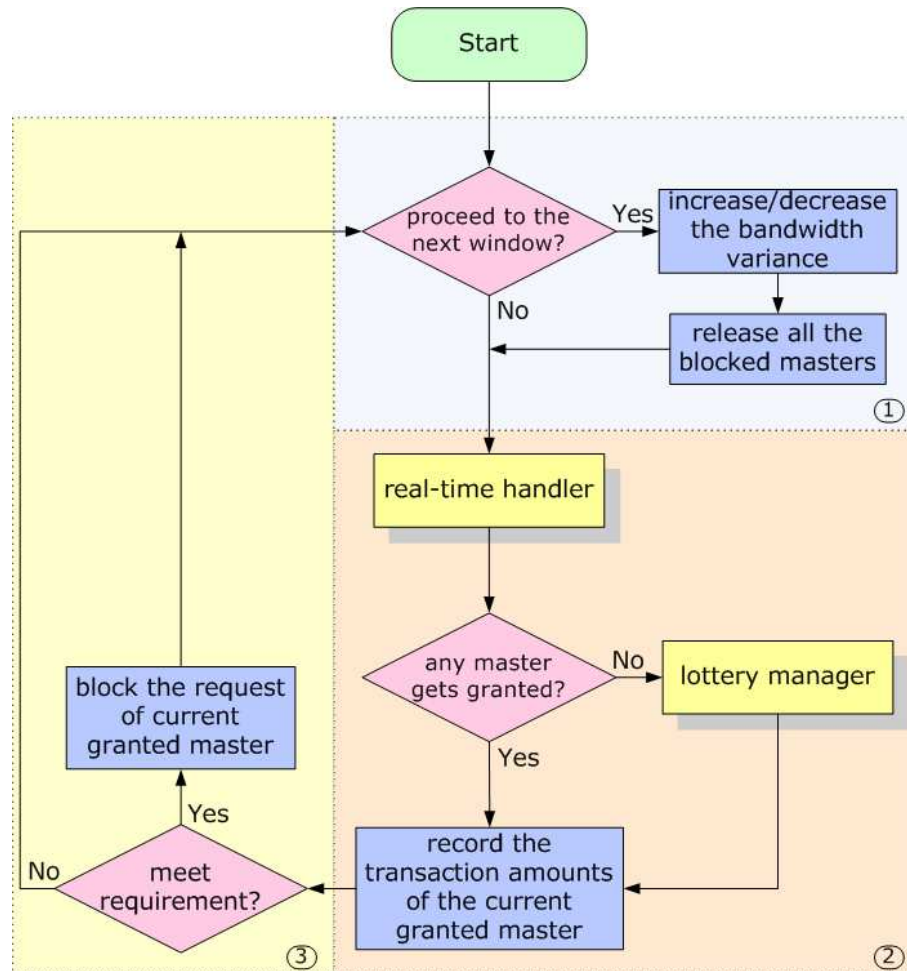


Figure 3.6: The algorithm flow of ARB

Unlike what FRB does, ARB makes a relationship between the adjacent windows. If a master gets more bandwidth allocation than required, ARB decreases the corresponding bandwidth boundary and the master gets less bandwidth allocation in the next window. Similarly, an under-served master in the current window gets more bandwidth allocation than required. The compensation mechanism more precisely controls the overall bandwidth allocation like the bandwidth requirements.

Chapter 4

Experimental Results

4.1 Experiment environment

4.1.1 SystemC model

The following experiment environment is developed under SystemC v2.1 with the transaction level model (TLM) library [28, 29]. In Figure 4.1, a simple system with N masters and one slave is shown as an example. Each master has a thread to generate bus traffic and puts the requests into the respective `tlm_transport_channel`. After a master initiates the request to the `tlm_transport_channel`, it waits for the corresponding response from the slave. The thread in the arbiter polls all of the request fifos. It decides which is the most important request and then forwards the request from the master to the slave at the same time. After the slave responds, the arbiter puts the response into the response fifo through the relevant `tlm_transport_channel`. The master then picks up the corresponding response up and completes the transaction. It is a high abstraction-level model to evaluate the system performance under different arbitration algorithms.

The arbiter is able to adopt any available algorithms to make arbitration decisions. In the following experiments, five arbitration algorithms, static priority, lottery, TDMA + Lottery (the first level arbitration is TDMA and the second level arbitration is Lottery), RT_lottery and RB_lottery are evaluated for comprehensive comparisons.

The high abstraction-level model is capable of providing a fast simulation speed that is up to one million cycles per second. Therefore, we can efficiently estimate the system performance and explore the proper system design parameters in this environment.

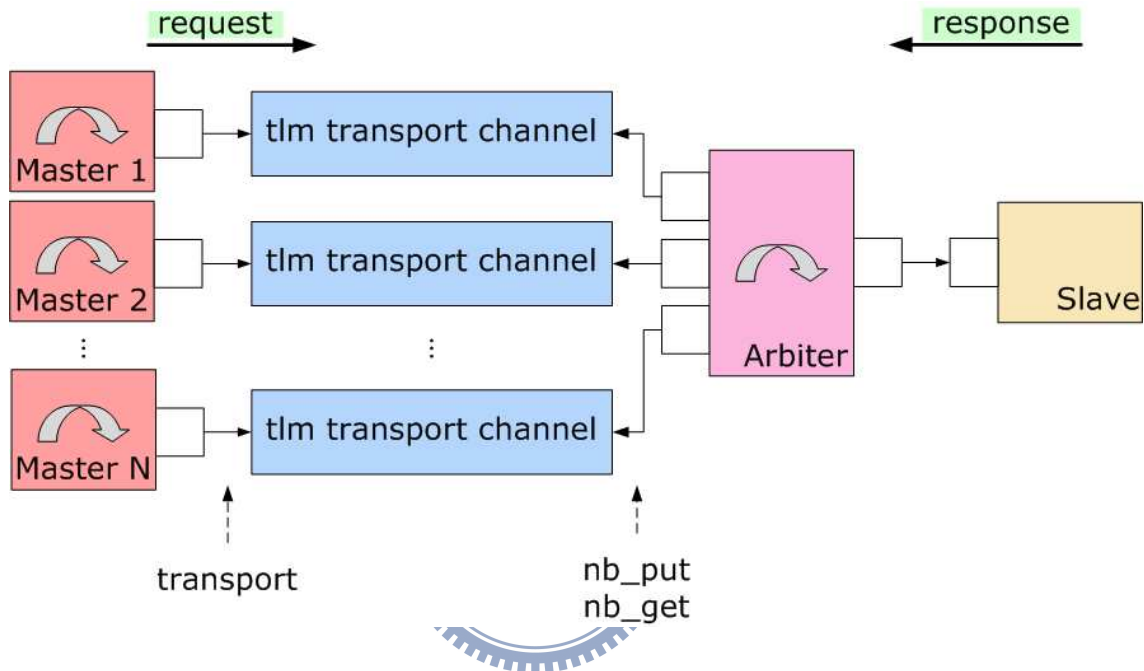


Figure 4.1: An example system using transaction level model in SystemC

4.1.2 Traffic types

We classify three high abstract-level traffic models to emulate the IP cores behavior in SoC systems. They have some parameters defined as follows. First, the transactions of a request is represented as the beat number. For example, if the beat number of a request is 4, it means that it is a 4-beat transaction. Second, the time of next request can be initiated is determined as the interval time. Third, the real-time requirement is represent as R_{cycle} which is the dead-line of a request.

- D type(D for dependency):

A D type master has no real-time requirements and initiates the next request

at the time depending on the finish time of the current request. As show in Figure 4.2, the beat number is 5 and the interval time is 10. If the first request is initiated at cycle 2 and is granted at cycle 5, the request will be completed at cycle 9. The next request will be initiated at cycle 19 which is 10 cycles later than the finish time.

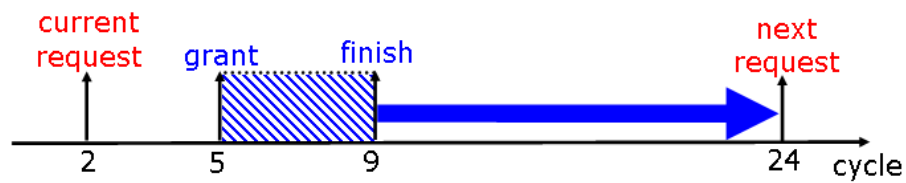


Figure 4.2: D type master (beat number = 5; interval time = 15)

- D_R type(D for dependency, R for real-time):

A D_R type master behaviors like a D type master with the real-time requirement. Each request must be completed before its deadline. Figure 4.3 is an example with the same parameters used in Figure 4.2. But the master has an extra real-time requirement, R_{cycle} , which is set to 10 cycles. Consequently, the request initiated at cycle 2 must be completed before cycle 12 which is 10 cycles later than the initiated time. It is a real-time violation, if the request is not completed until cycle 12.

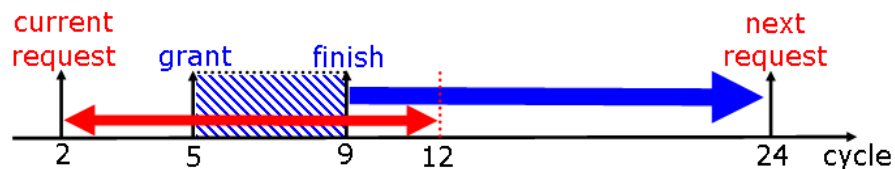


Figure 4.3: D_R type master (beat number = 5; interval time = 15; $R_{cycle} = 10$)

- ND_R type(ND for non-dependency, R for real-time):

A ND_R type master is another kind of master with the real-time requirement.

But the initiated time of the current request from an ND_R type master is independent of the completed time of its previous request. In other words, the ND_R type masters issue requests periodically. As shown in Figure 4.4, the same parameters used in Figure 4.3. Since the interval time is 15, the second request is initiated at cycle 17, which directly depends on the initiated time of the first request.

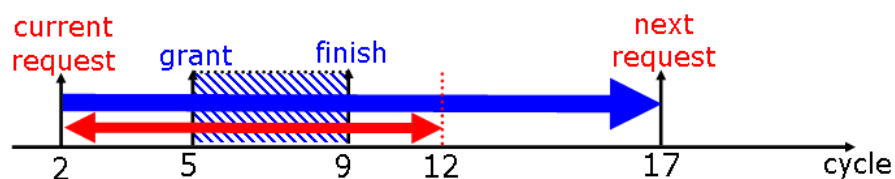


Figure 4.4: ND_R type master (beat number = 5; interval time = 15; $R_{cycle} = 10$)

4.1.3 Traffic behavior

In the following experiments, we set up a system with eight masters. As shown in Table 4.1, the second column is the master type and the third column is the real-time requirement, R_{cycle} . However, the R_{cycle} of a D type master is left undefined since they do not have real-time requirements. The fourth and the fifth column are the probability of the beat size and the interval time between two successive requests initiated by a master, respectively.

For example, the second row shows that the Master 1 may initiate 50% chance of 8-beat transactions and 50% chance of 16-beat transactions. And the next request of Master 1 may wait for 6, 7, 8, 9 or 10 cycles with the probability of 10%, 20%, 30%, 40% and 50%, respectively.

Moreover, Master 1, Master 2, Master 3 and Master 4 are D type masters which only require a fraction of bandwidth allocation. Master 5, Master 6, Master 7 and Master 8 are masters with real-time requirements. They not only require a

fraction of bandwidth allocation but also restrict the completed time of each request. Master 1, Master 3, Master 5 and Master 7 are heavy traffic masters and the others are light traffic masters. The heavy traffic masters have larger burst beats and shorter interval time than the light ones. In other words, the heavy traffic masters generate a heavier traffic load to the shared bus than the light ones do.

4.2 Experiment 1

In Experiment 1, we compare the performance of different arbitration algorithms, static priority, Lottery, TDMA + Lottery(the second level arbitration is Lottery), RT_lottery and RB_lottery. The level of difficulty to meet both real-time and bandwidth requirements generally depends on the bus workload in terms of the percentage of bus bandwidth utilization. As a result, we randomly generate patterns for different bus workloads and compare the results. As shown in Table 4.2, the first column gives the bus workload varying from 60% to 95%. For each bus workload, 100 random patterns of different required bandwidth combinations for the eight masters are generated. And then we simulate the input patterns with different arbitration algorithms. The results in 102400 simulation cycles are recorded and analyzed to see if the arbitration algorithms can meet the real-time and bandwidth

Table 4.1: The behavior of each master in the experiments

	type	R_{cycle}	beat/probability		interval/probability				
Master 1	D		8/50	16/50	6/10	7/20	8/40	9/20	10/10
Master 2	D		1/50	4/50	10/10	11/20	12/40	13/20	14/10
Master 3	D		8/50	16/50	6/10	7/20	8/40	9/20	10/10
Master 4	D		1/50	4/50	10/10	11/20	12/40	13/20	14/10
Master 5	D.R	128	8/50	16/50	10/10	11/20	12/40	13/20	14/10
Master 6	D.R	196	1/50	4/50	10/10	11/20	12/40	13/20	14/10
Master 7	ND.R	65	8/50	16/50	65/10	66/20	67/40	68/20	69/10
Master 8	ND.R	85	1/50	4/50	85/10	86/20	87/40	88/20	89/10

requirements simultaneously. If the real-time requirements are not all met or the allocated bandwidth is less than the required bandwidth with 2% error range during simulation, it is a failed pattern.

The parameters of compared arbitration algorithms are set as follows:

- fixed priority:

The priority of each master is assigned according to the required bandwidth.

The master with higher required bandwidth has a higher priority.

- Lottery:

The weight of each master is assigned according to the required bandwidth.

The required bandwidth ratio is the weight ratio.

- TDMA + Lottery:

1st level – TDMA: Masters with real-time requirements are allocated with time slots accordingly.

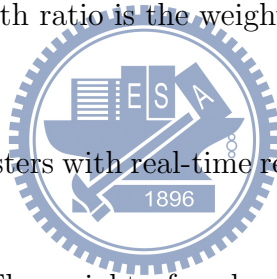
2nd level – Lottery: The weight of each master is assigned according to the required bandwidth. The required bandwidth ratio is regarded as the weight ratio.

- RT_lottery:

The weight of each master is assigned according to their bandwidth requirements and the traffic behaviors initially. To achieve better bandwidth allocation, a weight tuning mechanism is used to redistribute tickets among masters. More details can be found in [21].

- RB_lottery:

The weight of each master is assigned and tuned as the process of RT_lottery.



The size of observation window and bandwidth variance are set to 256 and 10 cycles, respectively.

As shown in Table 4.2, the first column is the total required bandwidth varying from 60% to 95%. For each case of total required bandwidth, we generate 100 random patterns of different required bandwidth combinations for the eight masters. And then we simulate the input patterns with different arbitration algorithms. We record and analyze the results in 102400 simulation cycles to see if the arbitration algorithms can meet the real-time and bandwidth requirements simultaneously. If one of the requirements is violated, it is a failed pattern.

Table 4.2: The number of fail patterns under different arbitration algorithms

Workload(%)	Fixed Priority	Lottery	TDMA + Lottery	RT_lottery	RB_lottery	
					FRB	ARB
60	100	100	95	0	0	0
65	100	100	98	0	0	0
70	100	100	100	0	0	0
75	100	100	100	10	0	0
80	100	100	100	18	0	0
85	100	100	100	37	1	0
90	100	100	100	55	12	10
95	100	100	100	74	44	39

The second and third column show the simulation results of the fixed priority and Lottery, respectively. Since fixed priority and Lottery do not take real-time requirements into consideration, they fail in the 100 patterns under different workload. The fourth column shows the simulation results of TDMA + Lottery. It handles the real-time requirements at the first level and provides control over bandwidth requirements at the second level. As a result, it may survive when the workload is less than 70%, but it still fails to meet the requirements in high workload. The fifth column is the results of RT_lottery. Compared to other arbitration algorithms,

RT_lottery is outstanding. Because of the hard real-time guarantees and the fine-control over bandwidth, it successfully meets the requirements of most patterns. However, it still loss the bandwidth controllability in the high workload. The sixth and seventh column show the results of the proposed algorithm, RB_lottery. The first fail pattern appears when the workload is 85%, which is an extremely high traffic load. Under the same observation window, we also find that ARB performs better than FRB.

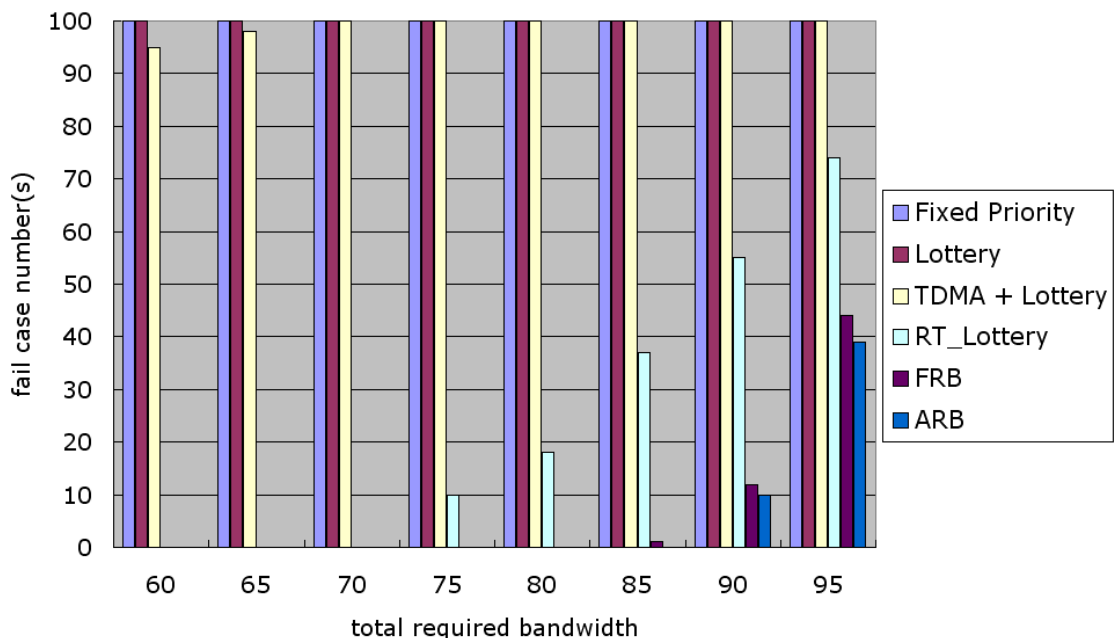


Figure 4.5: Figure of Table 4.2

As shown in Figure 4.5, the number of fail patterns monotonically increases while the workload increases. And the proposed algorithm, RB_lottery, still meets the requirements in more than half of patterns when the workload is 95%.

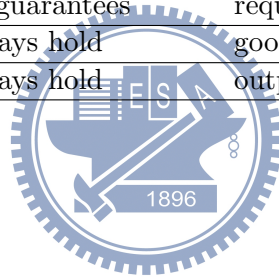
From this experiment, we have the following summaries. Since fixed priority and lottery do not consider the real-time requirements, they fail in the 100 random

patterns under different bus workloads. TDMA + Lottery may survive in the cases of low bus workload. Compared to other existing arbitration algorithms, RT_lottery is remarkable good. However, RB_lottery performs even better. The first failed case appears when the bus workload reaches 85%, which is an extremely high traffic load.

The number of fail cases monotonically increases while the total required bandwidth rises. And the proposed algorithm, RB_lottery, still holds more than 50% successful cases even when workload is 95%.

Table 4.3: The summaries of experiment 1

	real-time capability	bandwidth capability
fixed priority	no consideration	poor
Lottery	no consideration	required weight tuning
TDMA + Lottery	no guarantees	required weight tuning
RT_lottery	always hold	good except the highly loaded bus
RB_lottery	always hold	outperform than others



4.3 Experiment 2

The observation window size is one of the key parameters in our proposed algorithm. The performance under different window sizes is compared in the experiment. We experiment the size of observation window from 256 to infinite and observe the performance of RB_lottery included FRB and ARB. Similar to experiment 1, we generate 100 random required bandwidth combinations for each workload and simulate 102400 cycles for each case.

As shown in Table 4.4, larger size of observation window in FRB can provide better performance. In the highly loaded bus, for example, 95% of workload in the seventh column, only 27 patterns miss the requirements in the 100 random patterns while the observation window is large enough. However, large observation window leads to higher hardware cost. There is a tradeoff between performance and cost.

Table 4.4: The number of fail patterns under different size of observation window in FRB

Workload(%)	the size of observation window in FRB					
	128	256	512	1024	2048	∞
85	4	1	0	0	0	0
87	11	1	0	0	0	0
89	25	11	4	2	0	0
91	37	25	10	7	7	4
93	42	31	24	20	14	12
95	57	44	33	32	28	27

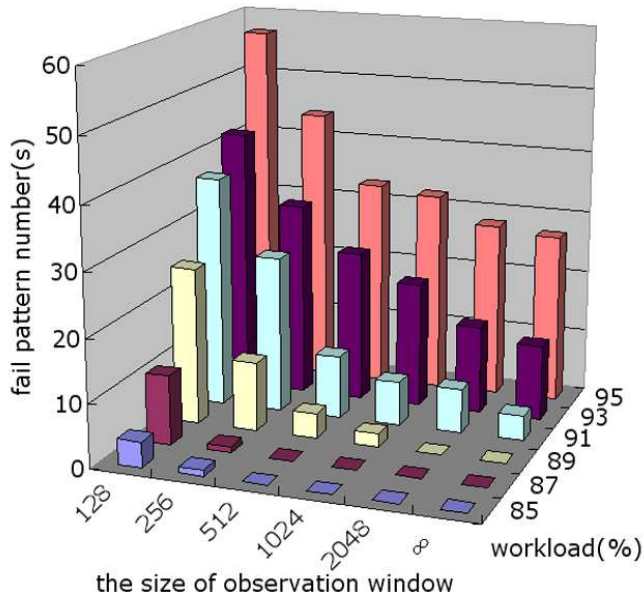
ARB shows the same results with FRB in Table 4.5. Larger observation window results in better performance. However, the unlimited size of observation window for ARB is not useful. The bandwidth variance cooperates with the windows. If there are no windows during operation, ARB works like FRB. The seventh column shows that the number of fail pattern is the same as FRB in Table 4.4.

Comparing the performance of ARB and FRB in Figure 4.6, ARB shows better ability on handling real-time and bandwidth requirements simultaneously

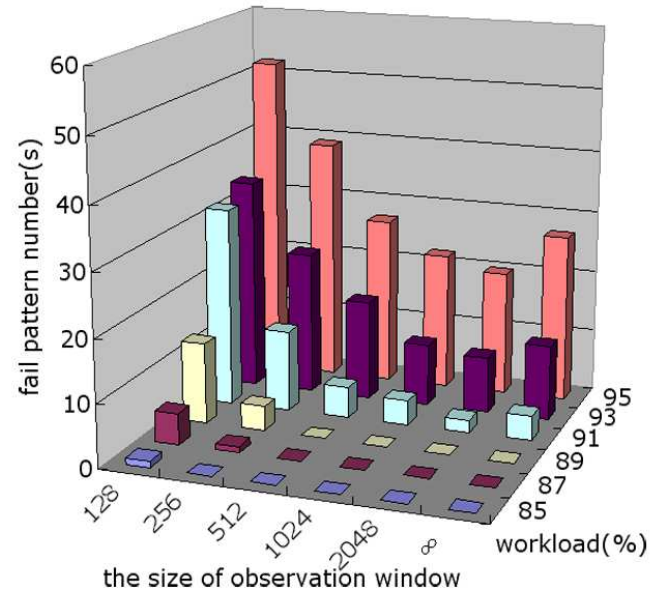
Table 4.5: The number of fail patterns under different size of observation window in ARB

Workload(%)	the size of observation window in ARB					
	128	256	512	1024	2048	∞
85	1	0	0	0	0	0
87	5	1	0	0	0	0
89	13	4	0	0	0	0
91	32	13	5	4	0	4
93	34	23	16	10	9	12
95	52	39	27	22	20	27

under the same situation. About 30% to 100% fail cases in FRB could be solved in ARB due to the dynamic bias boundary.



(a) Figure of Table 4.4



(b) Figure of Table 4.5

Figure 4.6: Figures of Table 4.5 and Table 4.4

4.4 Experiment 3

In the following experiment, we compare the different bandwidth variance in our proposed architecture. Similar to experiment 1, we generate 100 random patterns with different required bandwidth combinations for each workload and simulate 102400 cycles for each case. The bandwidth variance varies from 5% of required bandwidth to 30% of required bandwidth and the observation window is set to 256 in the experiment.

As shown in Table 4.6, the larger bandwidth variance in ARB can provide better performance. For example, when the workload is 95%, there are 42 fail patterns when the bandwidth variance is 5% in the second column and 33 fail patterns when the bandwidth variance is 30% in the seventh column. The bandwidth variance reflects the relationships between the adjacent windows. The larger size of bandwidth variance records more communication behaviors of windows while ARB dynamically tunes the bandwidth allocation. However, larger bandwidth variance also leads to higher hardware cost. We need to make the tradeoff between performance and cost during design.

Table 4.6: The number of fail patterns under different size of bandwidth variance in ARB

Workload(%)	the bandwidth variance in ARB					
	5%	10%	15%	20%	25%	30%
85	0	0	0	0	0	0
87	2	1	1	1	1	1
89	8	7	6	5	5	5
91	21	20	16	15	15	11
93	30	30	27	25	23	21
95	42	39	39	37	35	33

Chapter 5

Conclusions

A three-level arbitration algorithm, RB_lottery, is proposed in this paper. It provides not only the hard real-time guarantee but also the better capability of bandwidth control. The bandwidth regulator is utilized to dynamically monitor the bus communication and thus can precisely control the bandwidth allocation. Four existing arbitration algorithms, static priority, Lottery, TDMA + Lottery, and RT_lottery, are compared with RB_lottery. The experimental results clearly show that RB_lottery is the best among these five algorithms.

Hence, the lottery-based arbiter with a bandwidth regulator can be a better choice for those SoC buses with both the real-time and bandwidth constraints.

Bibliography

- [1] “Peripheral Interconnect Bus Architecture.” <http://www.omimo.be>
- [2] “Open Core Protocol Specification – v1.0.” <http://www.sonics.com>, 1999.
- [3] Virtual Socket Interface Alliance, <http://www.vsi.org>
- [4] “IBM Microelectronics CoreConnect Bus Architecture.”
<http://www.chips.ibm.com/products/coreconnect>
- [5] “AMBA 2.0 Specification.” <http://www.arm.com/armtech/AMBA>
- [6] “Sonics Integration Architecture.” Sonics Inc., <http://www.sonicsinc.com>
- [7] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 2002
- [8] J. L. Hennessy and D. A. Patterson, *Computer Organization and Design: The Hardware/Software Interface* Morgan Kaufmann Publishers, 2004
- [9] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and, L.Todd, *Surviving the SoC Revolution*. Kluwer Academic Publishers, 1999.
- [10] J. Liang, S. Swaminathan and, R. Tessier, *ASOC: A Scalable, Single-Chip Communications Architecture*, Parallel Architectures and Compilation Techniques, 2000, Page(s):37-46.
- [11] F. Poletti, D. Bertozzi, L. Benini and, A. Bogliolo, “Performance Analysis of Arbitration Policies for SoC Communication Architectures,” *ACM Transactions on Embedded Computing Systems*, 2003, Page(s):189-210.
- [12] M. Yang, S. Q. Zheng, Bhagyavati, and, S. Kurkovskyt, “Programmable Weighted Arbiters for Constructing Switch Schedulers,” *Workshop on High Performance Switching and Routing*, 2004, Page(s): 203-206
- [13] C.-H. Pyoun, C.-H. Lin, H.-S. Kim and, J.-W. Chong, “The Efficient Bus Arbitration Scheme in SoC Environment,” *System-on-Chip for Real-Time Applications*, 2003, Page(s):311-315
- [14] I.E. Sutherland and J. Ebergen, “Computers Without Clocks,” *Scientific American, INC*, 2002, Page(s):62-69

- [15] L. Benini and G. De Micheli, "Powering Networks on Chips: Energy-Efficient and Reliable Interconnect Design for SoCs," *14 International Symposium on Systems Synthesis*, 2001, Page(s):33-38
- [16] K. Goossens, J. van Meerbergen, A. Peeters and, P. Wielage. "Networks on Silicon: Combining Best-Effort and Guaranteed Services." *Design, Automation and Test in Europe*, 2002, Page(s):423-425
- [17] E. Rijpkema, K. Goossens, A. R adulescu, J. van Meerbergen, P. Wielage and, E. Waterlander, "Trade-offs in the Design of a Router with both Guaranteed and Best-effort Services for Networks on Chip," *Design Automation and Test in Europe*, 2003, Page(s):294-302
- [18] K. Lahiri, A. Raghunathan, G. Lakshminarayana and, S. Dey, "Design of High-Performance System-on-Chips using Communication Architecture Tuners," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2004, Page(s):620-636
- [19] K. Lahiri, A. Raghunathan and, G. Lakshminarayana, "LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs," *Design Automation Conference*, 2001, Page(s): 15-20
- [20] C. A. Waldspurger and W. E. Weih, "Lottery Scheduling: Flexible Proportional-Share Resource Management," *Proceeding of the First Symposium on Operating Systems Design and Implementation*, 1994, Page(s): 1-11
- [21] C.-H. Chen, G.-W. Lee, J.-D. Huang and, J.-Y. Jou, "A Real-Time and Bandwidth Guaranteed Arbitration Algorithm for SoC Bus Communication," *Asia South Pacific Design Automation Conference*, 2006, Page(s):600-605
- [22] Y. Zhang, "Architecture and Performance Comparison of A Statistic-Based Lottery Arbiter for Shared Bus on Chip," *Asia South Pacific Design Automation Conference*, 2004, Page(s):1313-1316
- [23] S. Ross, *A First Course in Probability*, Prentice Hall, 2002.
- [24] C. Liu and J. Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment," *Journal of the ACM*, 1973, Page(s):46-61
- [25] J. Lehoczky, L. Sha and, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-time Systems Symposium*, 1989, page(s):201-209
- [26] L. Sha and J. B. Goodenough, "Real-Time Scheduling Theory and Ada," *IEEE Computer*, 1990, Page(s):53-62

- [27] W. D. Weber, J. Chou, I. Swarbrick and, D. Wingard, “A Quality-of-Service Mechanism for Interconnection Networks in System-on-Chips,” *Design, Automation and Test in Europe*, 2005, Page(s):1530-1591
- [28] The Open SystemC Initiative. <http://www.systemc.org>
- [29] A. Rose, S. Swan, J. Pierce, and J.-M. Fernandez, “Transaction Level Modeling in SystemC,” <http://www.systemc.org>, 2005

