

# 應用於系統單晶片之可同時保證頻寬及即時要求的仲裁器演算法

研究生： 陳建華      指導教授： 周景揚 博士

國立交通大學

電子工程學系 電子研究所碩士班

## 摘要



對系統單晶片(SoC)的系統而言，若有多個元件同時使用通訊管道會產生資源衝突。為了解決這問題，需要一個仲裁器(arbiter)來決定哪個元件可以使用通訊管道。每個元件可能會有不同的頻寬或是即時(real-time)的要求，若仲裁演算法無法滿足元件的這些要求，會導致系統不符合應用。在系統單晶片設計流程的初期，仲裁演算法和它的參數必須決定，若在設計流程後期的階段，才發現無法滿足頻寬和即時的需求，設計者很可能需要回到設計流程前期來修改系統架構，因此延誤了出產時間。

為了解決這個問題，首先，我們提出了一個仲裁演算法(命名為RT\_lottery)期許能同時符合所有元件的頻寬和即時的需求。為了能合宜地決定RT\_lottery的參數，我們用高階抽象度的方法模擬一個系統單晶片的系統以評估仲裁器的效用。根據評估的結果及我們的調整參數流程，來自動產生合宜的參數。實驗結果證明經過我們的方法而決定的仲裁演算法，對於頻寬和即時保證的需求，表現得比實驗中其他對照的演算法來的好。

# **A Real-Time and Bandwidth Guaranteed Arbitration Algorithm for SoC Communication**

Student: Chien-Hua Chen

Advisor: Dr. Jing-Yang Jou

Department of Electronics & Institute of Electronics  
National Chiao Tung University



On an SoC bus, an arbiter is required to decide which master is granted for access when multiple masters on the same shared bus issue requests at the same time. We propose an arbitration algorithm, RT\_lottery, which intends to meet bandwidth and real-time requirements simultaneously. To decide suitable parameters for our arbiter, we model the SoC system at a high abstract level for evaluation. Based on the evaluation model and our weight tuning flow, the parameters are decided appropriately. We compare our arbitration algorithm, RT\_lottery, with Static Priority, Lottery, and TDM + Lottery, and the experimental results show that RT\_lottery handles both bandwidth and real-time requirements better than the other arbitration algorithms.

# Acknowledgments

I felt an immense gratitude to my adviser, Professor Jing-Yang Jou and professor Juinn-Dar Huang, for their insightful suggestion and patient guidance throughout the course of this work. Together, I gratefully acknowledge Geeng-Wei Lee, Che-Hua Shih, and Cheng-Yeh Wang, whose constructive suggestion really helped me a lot. Special thanks to all members in the EDA lab for their friendship and company. Finally, I have to show my sincere appreciation to my friends who give me a wonderful memory up to now.



# Contents

Chinese Abstract.....	i
English Abstract.....	ii
Acknowledgements.....	iii
Contents.....	iv
List of Tables.....	vi
List of Figures.....	vii
Chapter 1 Introduction.....	1
Chapter 2 Preliminaries.....	4
2.1 Purpose of an arbiter.....	4
2.2 Performance evaluation of an arbiter.....	4
2.3 Introduction of some common arbitration algorithms.....	6
2.4 Performance evaluation.....	9
2.5 Our observations on previous work.....	10
2.6 Motivations.....	11
Chapter 3 Proposed Approach.....	13
3.1 Proposed arbiter architecture.....	13
3.2 Evaluation model.....	14
3.3 Proposed arbitration algorithm.....	17
3.3.1 Algorithm of Real-Time Handler.....	18
3.3.2 Weight tuning flow for Lottery (2 <sup>nd</sup> level).....	20
3.3.3 Acceptance range.....	22
3.3.4 Algorithm of weight tuning.....	23

Chapter 4 Experimental Results.....27

    4.1 Experimental environment.....27

    4.2 Experiment 1.....31

    4.3 Experiment 2.....36

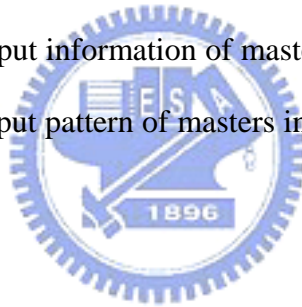
Chapter 5 Conclusions.....38

Reference.....39



# List of Tables

Table 2.1. Performance of the ATM switch.....	10
Table 3.1. The example of input traffic information of masters.....	15
Table 3.2. Input pattern for explanation of the warning line.....	20
Table 4.1. The input pattern for experiment 1.....	32
Table 4.2. The required bandwidth pattern for single case in experiment 1.....	33
Table 4.3. The result of the single case in experiment 1.....	33
Table 4.4. The result of 100 random cases.....	35
Table 4.5. The summary of experiment 1.....	36
Table 4.6. The fixed 8-beat input information of masters in experiment 2.....	36
Table 4.7. The fixed 16-beat input information of masters in experiment 2.....	36
Table 4.8. The fixed 32-beat input pattern of masters in experiment 2.....	37



# List of Figures

Figure 2.1. The simple architecture of SoC system.....	5
Figure 2.2. A simple architecture of two-level TDM arbitration algorithm.....	7
Figure 2.3. A diagram to explain Lottery arbitration algorithm.....	8
Figure 2.4. Cell forwarding in a 4 ports ATM switch.....	9
Figure 3.1. The proposed arbiter architecture.....	13
Figure 3.2. The simplified flow of weight tuning in 2 <sup>nd</sup> level of RT_lottery.....	14
Figure 3.3. The architecture of the evaluation.....	14
Figure 3.4. The example of D type master.....	16
Figure 3.5. The example of D_R type master with $R_{cycles} = 10$ .....	17
Figure 3.6. The example of ND_R type master with $R_{cycles} = 10$ .....	17
Figure 3.7. The example for Real-Time Handler ( $R_{cycles}$ of M1 = 30, Warning line = 25).....	18
Figure 3.8. The worst case of Table 3.2.....	20
Figure 3.9. The detailed weight tuning flow for Lottery.....	21
Figure 3.10. The flow of weight tuning.....	24
Figure 4.1. The example of the 1 <sup>st</sup> level of TDM + Lottery.....	31
Figure 4.2. Trend of failed cases for 100 random cases.....	37

# Chapter 1

## Introduction

System-on-Chip (SoC) is a major revolution taking place in the design of integrated circuits. It is a technology that integrates heterogeneous system components (CPU, Memory, and DSPs, etc.) into a single chip and offers several benefits, including improvements in system performance, power dissipation, and design time [1].

The massive real-time data transfer among IP cores is essential to keep the system function properly. Therefore, a systematic On-Chip-Bus (OCB) standard now becomes a widely adopted key technology for IP integration in SoC development. This advanced concept provides a systematic, modular, and reusable bus interface circuit to easily and instantly integrate the data flow from all IPs into a system-level backbone for data transfer [2].

Network-on-Chip (NoC) is the design methodology proposed recently to solve following three problems: (1) Buses usually handle 3 to 10 communication components efficiently, but they do not scale to higher numbers [3-5]. (2) Wire delay is no longer negligible since technology scaling works better for transistors than for interconnecting wires. Global and long wires make system performance unpredictable hence it is hard to maintain global synchrony [6]. (3) In a complex system, each component may be designed by different teams at different times with different tools and languages. At system level, the components are not easily compatible since a tiny change in one component may result in unexpected effects on other



seemingly unrelated components of the system [1]. Thus, more design efforts are spent on verification, which in turn lowers the design productivity and delays product development.

Our work focuses on shard bus architecture. Up to the present, there are many On-Chip-Bus (OCB) protocols. Future SoC designs demand (1) increasing levels of system complexity, (2) increasing performance demands, (3) increasing clock speed requirements, (4) reduction of system power consumption. To meet these demands, bus protocol should be updated [7]. For Advanced Microcontroller Bus Architecture (AMBA) series, ASB protocol is presented in 1995 and then AHB protocol is presented in 1999. Advanced eXtensible Interface (AXI) Protocol proposed in 2003 is the latest generation of AMBA.

The masters on an SoC bus may issue requests simultaneously and an arbiter is required to decide which master is granted for access. In many applications, masters may have real-time or bandwidth requirements. A master with real-time requirements demands tasks accomplished within fixed clock cycles. The master with bandwidth requirements must occupy a fixed fraction of total bandwidth of the interconnect channel. If designers find that an arbitration algorithm cannot fulfill requirements at late design stages, they need jumping back to a very early design stage to significantly modify the arbitration algorithm. This results in a significant schedule delay.

Arbitration algorithms commonly used for shared buses include Static Priority, TDM, and Round-Robin [8-13]. Lottery is the arbitration algorithm proposed recently [14] with advantages of (i) providing designers with great control over the bandwidth allocated to each SoC component, and (ii) providing high priority SoC component with quite low traffic latencies. However, all arbitration algorithms mentioned above cannot well handle bandwidth and hard real-time requirements concurrently.

In this thesis, we propose a two-level arbitration algorithm, RT\_lottery (R for real-time, T for weight Tuning), which is expected to meet hard real-time and bandwidth requirements of each master at the same time. For the 1<sup>st</sup> level, we develop an arbitration algorithm whose

purpose is to handle hard real-time requirements (named Real Time Handler). For the 2<sup>nd</sup> level, we adopt a Lottery based arbitration algorithm with weight tuning for bandwidth requirements. Although Lottery is good at bandwidth allocation, we observe that if request conditions (traffic behaviors) of masters vary a lot, it is not adequate to decide Lottery parameters (weight of each master) merely according to the required bandwidth ratio. For this reason, we propose a weight tuning algorithm to decide appropriate parameters of Lottery automatically.

We compare RT\_lottery with other three arbitration algorithms, Static Priority, Lottery, and TDM + Lottery (1<sup>st</sup> level : TDM, 2<sup>nd</sup> level : Lottery). The experimental results show that RT\_lottery with parameters fine tuned by our weight tuning flow can handle real-time and bandwidth requirements of each master better than the other arbitration algorithms.

The remainder of this thesis is organized as follows: the previous works including introduction of some common arbitration algorithms (Static Priority, TDM, and Lottery) and the motivations of this work are presented in Chapter 2. Chapter 3 describes the proposed arbitration algorithm (RT\_lottery) and method of automatically generating suitable parameters of RT\_lottery to meet bandwidth requirements. The experiment environment and results are shown in Chapter 4. Chapter 5 concludes the thesis.

# Chapter 2

## Preliminaries

### 2.1 Purpose of an arbiter

In an SoC system, masters may request to access interconnect channel simultaneously and thus the contention occurs. In the situation, an arbiter is required to decide which master can access the interconnect channel. Like Fig. 2.1, masters which need to access interconnect channel assert request signals high [15]. An arbiter gathers pending request signals of all masters and asserts the corresponding grant signal high for certain master which has the authority to access interconnect channel according to the arbitration algorithm. Thus it can be seen that different types of arbitration algorithms could lead the different behavior of an SoC system.

### 2.2 Performance evaluation of an arbiter

Depending on different demands of designers, there are several aspects of performance evaluation for an arbiter:

1) Low average latency:

In our work, latency means the cycles between request's generation time and request's finish time.

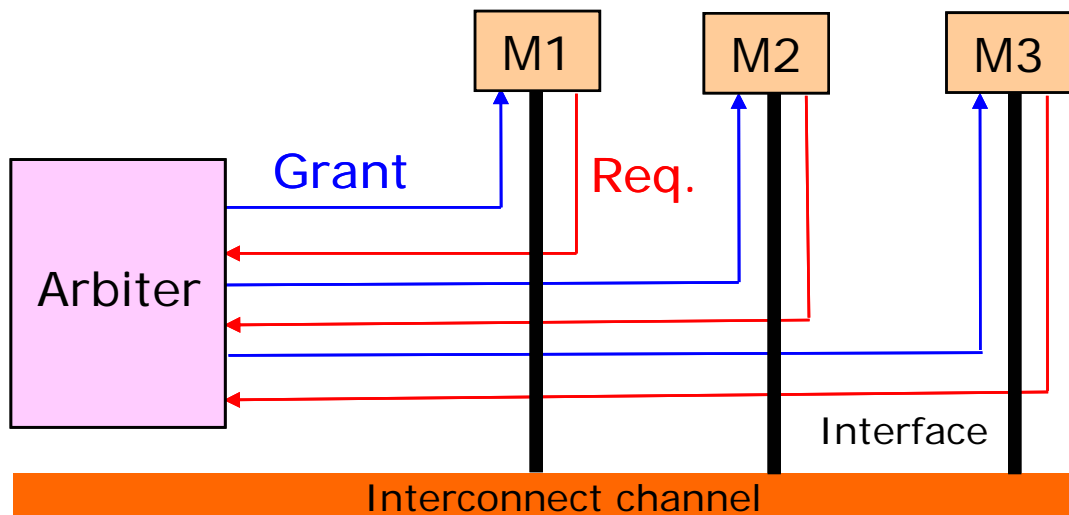


Fig. 2.1. The simple architecture of SoC system

2) Real-time handling:

Some masters require tasks accomplished within fixed cycles.

3) Guarantee fraction of communication bandwidth:

It is somewhat like the concept of QoS (Quality of Service). Designers may hope that each master can get nearly a certain fixed fraction of bandwidth at least.

4) High channel utilization:

If the channel utilization is low, the bandwidth of channel is wasted. It violates the principle of efficiency.

5) Low hardware complexity:

It not only means smaller area of an arbiter, but also simpler implementation of an arbiter.

In our work, we focus on the aspects of real-time handling and guaranteed fraction of communication bandwidth of each master.

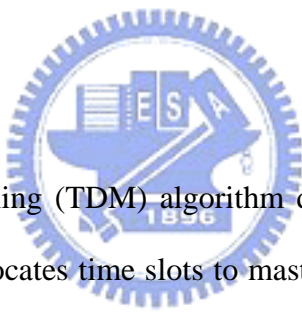
## 2.3 Introduction of some common arbitration algorithms

In this section, we present several arbitration schemes:

### 1) Static Priority [16-18]:

Each master is statically assigned a unique priority value. When multiple masters issue request simultaneously, the master with the highest priority would be granted. The advantages of this arbitration scheme include simpler implementation and smaller area cost. However, if masters with higher priority request successively and frequently, masters with lower priority may rarely be granted. This could produce the severe starvation of low priority masters and result in unfair bandwidth allocation.

### 2) TDM [8-11]:



Time Division Multiplexing (TDM) algorithm divides execution time on the channel into time slots and then allocates time slots to masters. If the master owning the current time slot does not request, the current time slot may be wasted. To mend this inefficiency, a 2<sup>nd</sup> level arbitration algorithm is usually adopted to reallocate the available slot to other requesting masters. Fig. 2.2 is a simple architecture of two-level TDM.

For a two-level TDM arbitration algorithm, the 1<sup>st</sup> level uses a time wheel where each slot is statically reserved for a unique master and the 2<sup>nd</sup> level could be any arbitration algorithm depending on applications. For example, if bandwidth reservation for masters is important, an arbitration algorithm with better ability of bandwidth allocation can be used as the 2<sup>nd</sup> level. In our work, when the arbiter grants some master, regardless of which level makes a decision, the time wheel will be rotated by one slot. For example, as shown in Fig. 2.2, the current time slot is reserved for M1. If M1 does not have a request, the decision is made by the 2<sup>nd</sup> level and the time wheel is rotated by one slot.

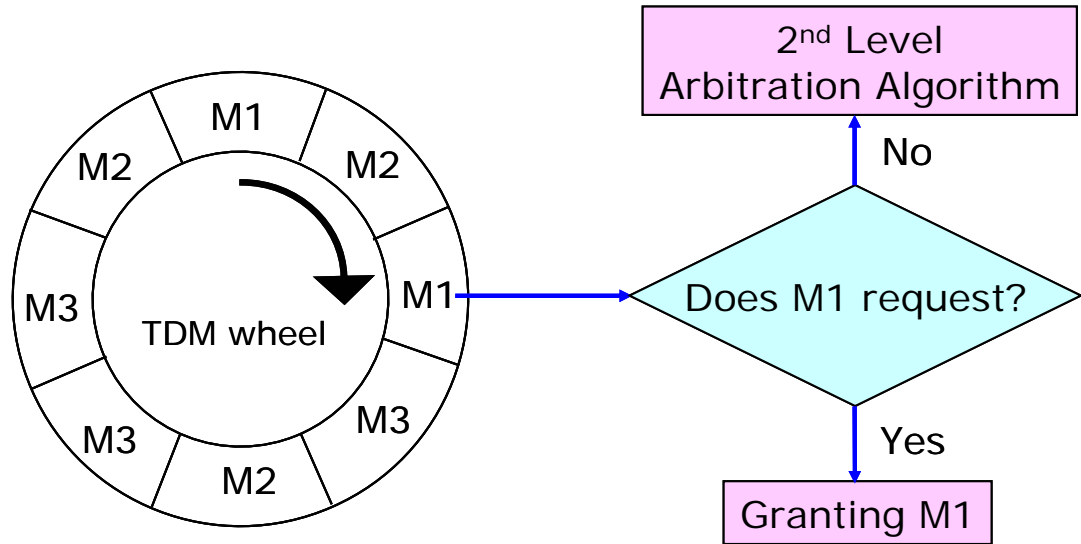


Fig. 2.2. A simple architecture of two-level TDM arbitration algorithm

### 3) Lottery [14][19]:

For the Lottery arbitration algorithm, an arbiter is just like a lottery manager deciding which lucky one will win a prize. The lottery manager accumulates requests for ownership of the channel from one or more masters, each of which is (statically or dynamic, statically for our work) assigned a number of 'Lottery tickets'. The lottery manager generates a pseudo random number that corresponds to one ticket number. The master having the most thickets is most likely to be granted.

Let the set of bus masters be  $M_1, M_2, \dots, M_n$  and the number of tickets held by each master be  $t_1, t_2, \dots, t_n$ . At any 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  $M_i$  has a pending request, and  $r_i = 0$  otherwise. The master to be granted is chosen randomly, with the probability of

$$\text{granting component } M_i \text{ given by: } P(M_i) = \frac{r_i \cdot t_i}{\sum_{j=1}^n r_j \cdot t_j}$$

To decide which master to be granted, a lottery manager sums up the total number of tickets possessed by the masters which request, given by  $\sum_{j=1}^n r_j \cdot t_j$ . Then, it generates a pseudo random number from the range  $[0, \sum_{j=1}^n r_j \cdot t_j)$  to decide which master to be

granted. If the random number falls in the range  $[0, r_1t_1)$ , the channel is granted to  $M_1$ ; if it falls in the range  $[r_1t_1, r_1t_1 + r_2t_2)$ , the channel is granted to  $M_2$ , and so on. In general, if it falls in the range  $[\sum_{j=1}^i r_j \cdot t_j, \sum_{j=1}^{i+1} r_j \cdot t_j)$  it is granted to component  $C_{i+1}$ . Fig 2.3 gives an example.

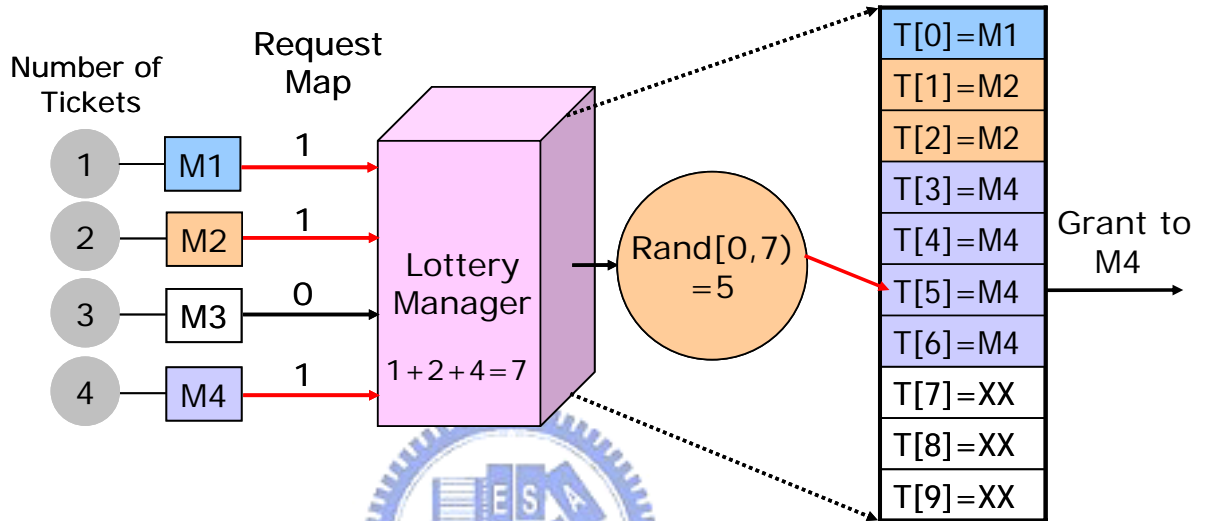


Fig. 2.3. A diagram to explain Lottery arbitration algorithm

In the example, there are totally four masters  $M_1$ ,  $M_2$ ,  $M_3$  and  $M_4$  assigned 1, 2, 3, and 4 tickets respectively. Request map indicates the request conditions of masters. As shown in the Fig. 2.3, all masters request for access except  $M_3$ . The Lottery manager sums up the tickets number of masters which request ( $\sum_{j=1}^n r_j \cdot t_j = 1+2+4=7$ ). Then, the lottery manager randomly generates the number 5 from the range  $[0, \sum_{j=1}^n r_j \cdot t_j = 7)$ . The random number 5 falls into the range  $[r_1t_1+r_2t_2=3, r_1t_1+r_2t_2+r_4t_4=7)$ , so  $M_4$  is granted.

The number of tickets of each master for Lottery arbitration algorithm is like the weight associated to the master. A master with a higher weight has better chance to be granted. We present the number of tickets as weight in the following sections.

## 2.4 Performance evaluation

In this section, we show the experimental results and conclusions of reference [14]. The experiment compares Lottery with Static Priority, and TDM arbitration algorithm in the aspects of bandwidth allocation and low average latency. Fig. 2.4 shows the experiment environment.

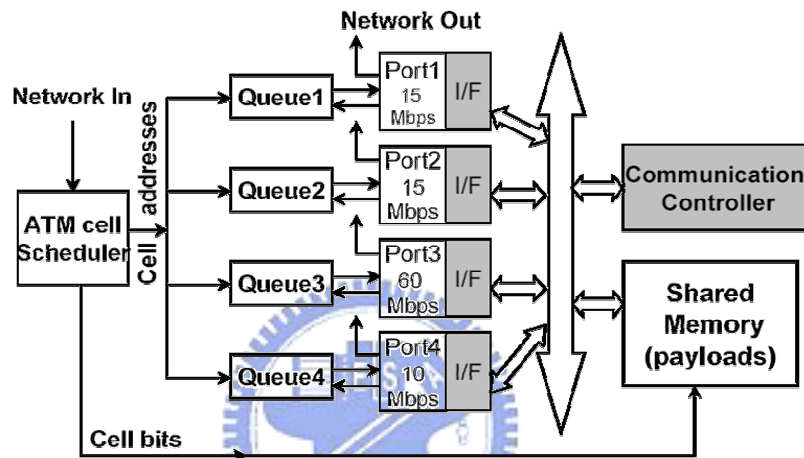


Fig. 2.4. Cell forwarding in a 4 ports ATM switch [14]

The following quality-of-service requirements are imposed: (1) Port4 requires minimum latency. (2) Port1, Port2, Port3 share the bandwidth in the ratio 1 : 1 : 4. Lottery tickets, time-slots, and priorities were assigned uniformly in the ratio 1 : 1 : 4 : 6, for ports 1, 2, 3, 4, respectively. The results of the experiments are shown in Table 2.1.

The columns present performance metrics for each output port (bandwidth fraction and latency for Port4, and bandwidth fraction for Ports 1, 2, and 3). The rows present the performance under each alternative arbitration algorithm. For example, Port3 receives about 59% of the total bus bandwidth based on the Lottery arbitration algorithm. From the Table 2.1, we can make following observations: (1) In latency aspect : The latency observed at Port4 based on Lottery is comparable to that based on Static Priority, while the latency observed at

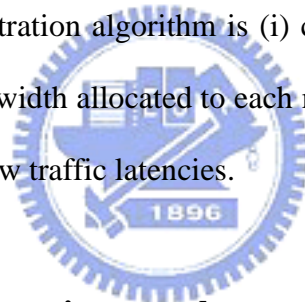


Table 2.1. Performance of the ATM switch

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

port4 based on TDM is 7 times larger. (2) In bandwidth aspect: Based on the Lottery arbitration algorithm, the bandwidth ratio of Ports 1, 2, and 3, is 1 : 1.19 : 4.13 which is the best result, since it is 1 : 1 : 2.22 based on TDM and Static Priority even generates the starvation at Port1 (only 0.01 %).

To sum up, the Lottery arbitration algorithm is (i) capable of providing the designer with the great control over the bandwidth allocated to each master, and (ii) quite good at providing the high priority master with low traffic latencies.



## 2.5 Our observations on previous work

From above, we make some observations. Low average latency can be seen as loose real-time requirements since some extreme long latencies may exceed cycles of real-time requirements. However, for hard real-time requirements, the worst case latency (not average latency) must be smaller than certain amount. According to Table 2.1, Static Priority and TDM cannot handle real-time and bandwidth requirements at the same time. The results in section 2.3 demonstrate that the Lottery arbitration algorithm (a) provides low latency for bursty traffic with real-time latency constraints, and (b) at the same time, provides effective bandwidth guarantees for traffic generated by each port.

Based on Lottery, how do we allocate the weight of each master on an SoC bus to meet

real-time and bandwidth requirements at the same time? Let the problem be easier for just considering bandwidth requirements first. In the experiment of [14], for good bandwidth allocation, the weight of each component is allocated according to the ratio of required bandwidth. Nevertheless, as shown in Table 2.1, the bandwidth ratio (1 : 1.19 : 4.13 : 0.68, Port1 -> Port4) would not conform with the weight ratio (1 : 1 : 4 : 6, Port1 -> Port4) for Port4. The reason should be that the traffic of discordant Port4 is much less than the other ports (request ratio 1 : 1 : 4 : 0.67, Port1 -> Port4). Thus it can be seen that weight allocation is not good enough just according to the ratio of the required bandwidth. If request conditions (traffic behaviors) of masters vary a lot, weight tuning is required to improve bandwidth allocation.

To meet the real-time requirements, the experiment of reference [14] requires Port4 with minimum latency. The weight of Port4 is much larger than the other ports. However, there is no rule to allocate the suitable weight of a certain component much larger than others. It is even harder to allocate weight of each master to meet real-time requirements if there are multiple components with real-time requirements, not to mention meeting bandwidth and real-time requirements at the same time. Furthermore, if a certain component requires **hard** real-time, a probabilistic arbitration algorithm like Lottery is obviously not appropriate.

## 2.6 Motivations

- (1) Since Lottery is not suitable for **hard** real-time, can we develop an arbitration algorithm being capable of handling **hard** real-time requirements?
- (2) For Lottery, can we develop a method of weight tuning considering both request

condition and required bandwidth to allocate weight of masters to meet bandwidth requirements?

- (3) Can we develop the arbitration algorithm to meet hard real-time and bandwidth requirements at the same time?



# Chapter 3

## Proposed Approach

### 3.1 Proposed arbiter architecture

Since probabilistic arbitration algorithms cannot handle hard real-time requirements, we propose the two-level arbitration algorithm, RT\_lottery (R for Real-time, T for weight Tuning). The proposed arbiter architecture is shown in Fig. 3.1. The 1<sup>st</sup> level, Real Time Handler, intends to handle real-time requirements. The 2<sup>nd</sup> level, Lottery with weight tuning, intends to handle bandwidth requirements. The simplified flow of weight tuning for the 2<sup>nd</sup> level is shown in Fig. 3.2 and the details will be presented in the following sections.

In section 2.4, we find that doing a good bandwidth allocation should consider both request conditions and required bandwidth. As shown in Fig. 3.2, we simulate the whole system and then analyze the result. If the result meets bandwidth requirements, the flow ends; else we perform weight tuning according to the result analysis and then simulate again. Since we need to simulate the whole SoC system for evaluation, we should model the SoC system at the early design stages.

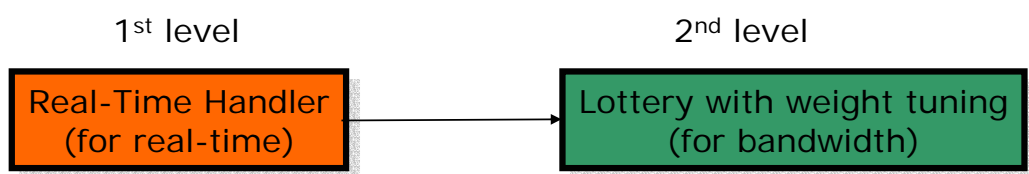


Fig. 3.1. The proposed arbiter architecture

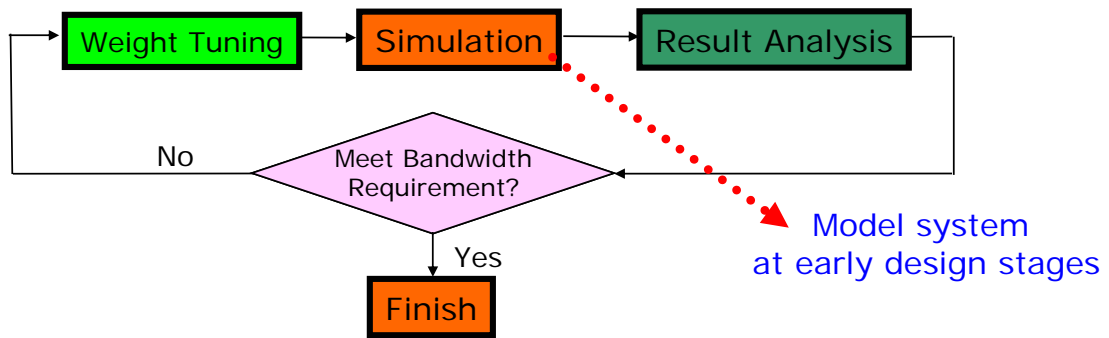


Fig. 3.2. The simplified flow of weight tuning in 2<sup>nd</sup> level of RT\_lottery

### 3.2 Evaluation model

In our model, we assume that once a master possesses the channel, the other masters cannot access the channel until the possessing master releases the channel. The architecture of the evaluation model is shown in Fig. 3.3.

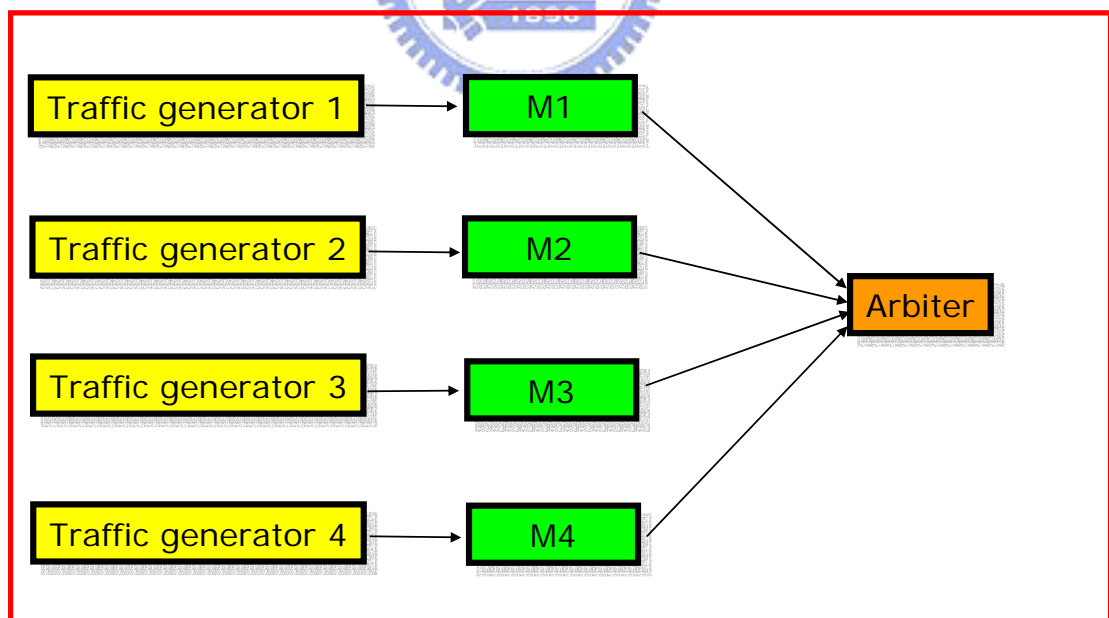


Fig. 3.3. The architecture of the evaluation

Each master has a traffic generator. The behavior of each traffic generator is given by

designers. An arbiter accumulates requests of all masters and decides which master should be granted. We also construct the monitor to check the correctness of the model. There are four types of information for a master:

(1)  $R_{cycles}$  :

It is the real-time requirements (in cycles) of a master. For those masters without real-time requirements, this information should be left undefined. As shown in Table 3.1, the  $R_{cycles}$  of M2 is 100 cycles for example.

Table 3.1. The example of input traffic information of masers

	type	$R_{cycles}$	beat/prob.			interval/prob.	
M1	D		4/50	5/20	6/30	60/20	70/80
M2	D_R	100	3/20	4/50	5/30	80/10	90/90
M3	ND_R	120	5/30	6/50	7/20	14/50	16/50

(2) Beat number and probabilities :

It defines the burst size and its probability of each request. Take Table 3.1 for example, there is 30% chance for M3 to generate a 5-beat burst request.

(3) Interval cycles and probabilities :

It determines the next request time of the master. Nevertheless, the rule of deciding the next request time varies with master types (explained later). There is 20 % of probability for the interval of M1 to be 60 in Table 3.1.

(4) Type :

Designers must define each master's type. Take Table 3.1 for example, M3 belongs to ND\_R type. There are three possible types:

a. D type (D for Dependency) :

D type masters have no real-time requirements and its next request depends on the finish time of its current request. Interval means the cycles between the next request and the finish time of the current request. Fig. 3.4 shows an example.

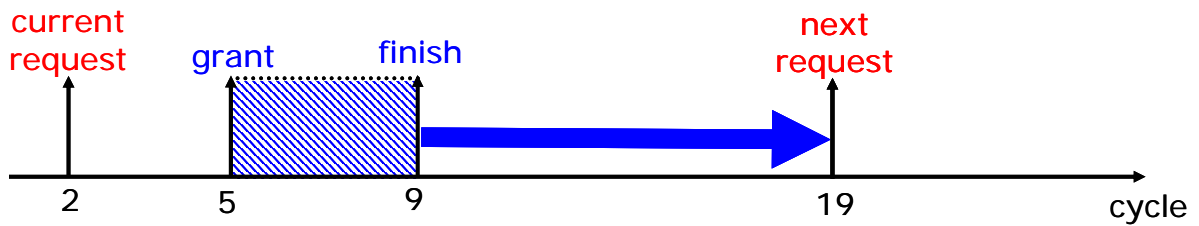


Fig. 3.4. The example of D type master

At cycle 2, the traffic generator randomly generates a 4-beat burst according to beat probability. The request is not granted until cycle 5 and is finished at cycle 9 (4-beat burst). Based on the probability of interval cycles, the interval time generated randomly is 10 and it determines the next request time. Since the interval of D type master means the cycles between the next request and the finish time of the current request, the next request occurs at cycle 19.

b. D\_R type (D for Dependency, R for Real-time) :

D\_R type masters are like D type masters except that they have real-time requirements. If beat and interval numbers generated randomly are the same as Fig. 3.4, Fig. 3.5 is the same as Fig. 3.4 except that the request occurred at cycle 2 must be finished before cycle 12 for  $R_{cycles} = 10$ . If the request is not finished before cycle 12, a real-time violation occurs.

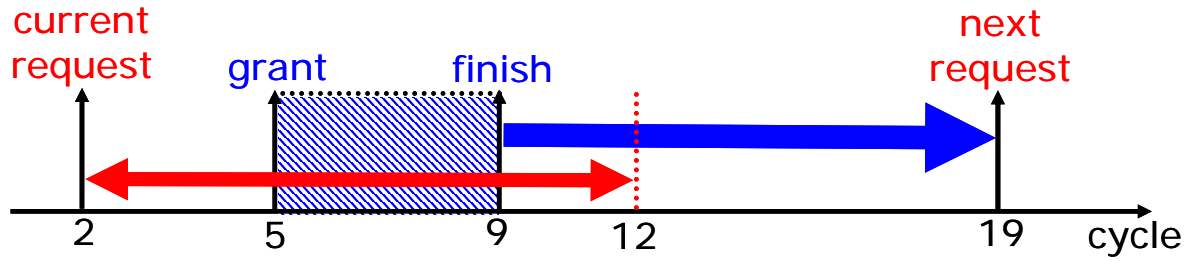


Fig. 3.5. The example of D\_R type master with  $R_{cycles} = 10$

c. ND\_R type (ND for No Dependency, R for Real-time) :

The next request of ND\_R type master is independent of the finish time of the current request and interval means cycles between two successive requests. In Fig. 3.6, the random result of interval distribution is 15. The next request is issued at cycle 17 regardless of the finish time of the current request. The MPEG encoder belongs to this type of master, for example. Since the current request must be finished before the next request, the reasonable value of  $R_{cycles}$  should be smaller than the minimum possible interval. For reasonable  $R_{cycles}$  constraints, we define  $R_{cycles} = \min(t_{min\_interval}, t_{user\_given})$ .  $t_{min\_interval}$  is the minimum possible interval and  $t_{user\_given}$  is  $R_{cycles}$  given by designers.

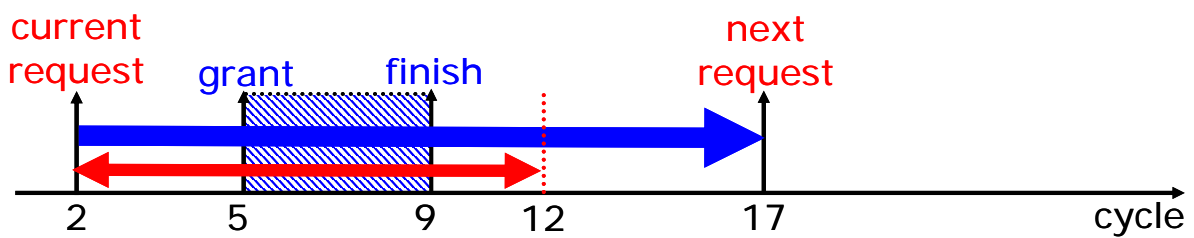


Fig. 3.6. The example of ND\_R type master with  $R_{cycles} = 10$

### 3.3 Proposed arbitration algorithm

We have presented the architecture of RT\_lottery (section 3.1 and Fig. 3.1). In this section, the algorithms of Real-Time Handler and weight tuning for Lottery are described in detail.



### 3.3.1 Algorithm of Real-Time Handler

The Real-Time Handler sets a real-time counter for each master with real-time requirements. When a master issues a request, the corresponding real-time counter is set to this master's  $R_{cycles}$ . The real-time counter is decremented by 1 every cycle until the master is granted to access the channel. *Warning line* is the value used to remind an arbiter to give the grant to the emergent master. The master would have higher priority if its corresponding real-time counter is below the warning line. When two or more real-time counters are below the warning line, the master with the smallest real-time counter value gets granted. Fig. 3.7 shows an example of Real-Time Handler's operation.

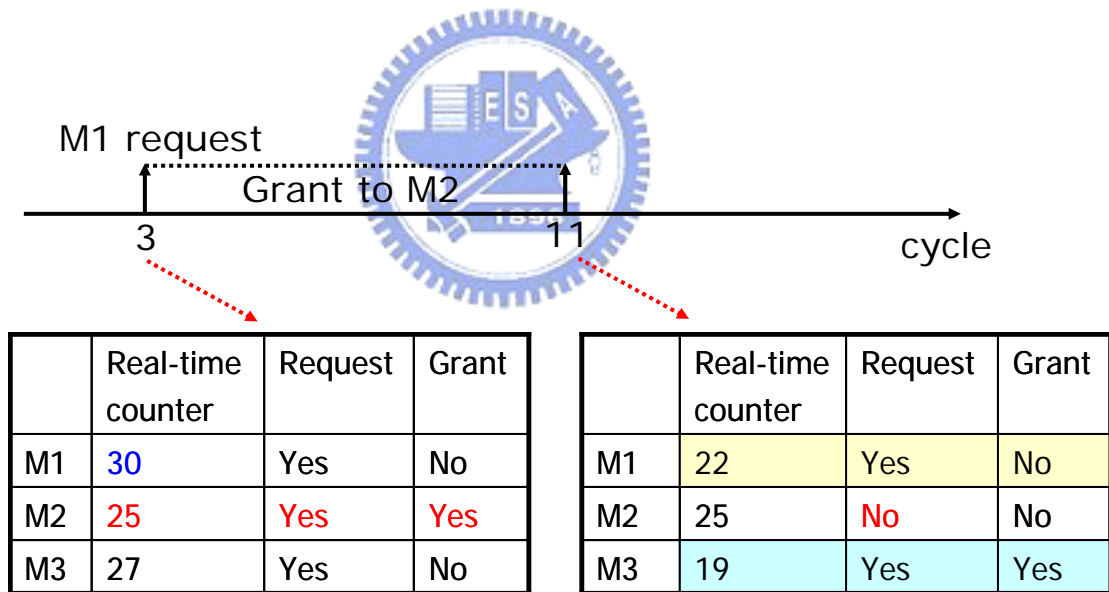


Fig. 3.7. The example for Real-Time Handler ( $R_{cycles}$  of M1=30, Warning line = 25 )

In Fig. 3.7, we assume that  $R_{cycles}$  of M1 = 30 and Warning line = 25. Let us focus on cycle 3 and cycle 11:

- (1) Cycle 3 (see the left table of Fig. 3.7)

As M1 requests at this cycle, the real-time counter of M1 is reset to its  $R_{cycles}$  ( real-time counter = 30). All masters issue requests but only M2's real-time counter is below the warning line, so M2 is granted to access the bus.

(2) Cycle 11 (see the right table of Fig. 3.7)

We assume that M2's transaction is a 8-beat burst and finishes at cycle 11. Each real-time counters of pending masters are decrements by 8. At this cycle, M1 and M3 request and their real-time counters are both below the warning line. Since the real-time counter of M3 is smaller than M1's, M3 is granted.

To meet real-time requirements, we set the appropriate value of warning line considering the worst contending case.

So,

$$\text{Warning line} = \text{SUM (max possible beat of type D\_R or type ND\_R)} \\ + \text{max possible beat of all type D}$$



In the worst case, a master without real-time requirements (D type) gets grant before all real-time counters being below the warning line. After this D type master finishing the task, all real-time counters are below the warning line at this arbitration time , hence all emergent masters with real-time requirements still must queue up to be granted. Take Table 3.2 for an example and the worst contending case is shown in Fig. 3.8:

$$\text{Warning line} = \max (5,6,7,4,5,6) + \max (2,3,4)+\max(3,4,5)+\max (5,6,7) = 23$$

If there is no master whose  $R_{cycles}$  is smaller than the warning line (like Table 3.2), we can meet all hard real-time requirements.

Table 3.2. Input pattern for explanation of the warning line

	type	$R_{cycles}$	beat/prob.			interval/ prob.	
M1	D		5/20	6/40	7/40	40/50	50/50
M2	D		4/50	5/20	6/30	60/20	70/80
M3	D_R	200	2/30	3/30	4/40	40/50	60/50
M4	D_R	100	3/20	4/50	5/30	80/10	90/90
M5	ND_R	120	5/30	6/50	7/20	14/50	16/50

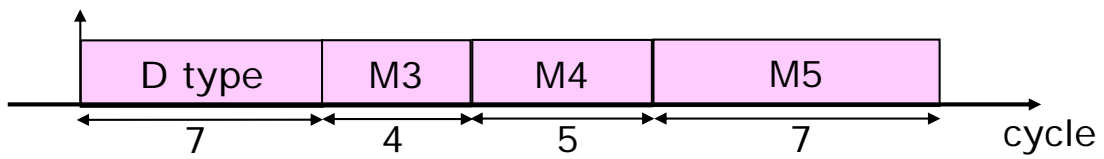


Fig. 3.8. The worst case of Table 3.2

### 3.3.2 Weight tuning flow for Lottery (2<sup>nd</sup> level)



In this section, we present the 2<sup>nd</sup> level of RT\_lottery, Lottery with weight tuning. The simplified weight tuning flow has been shown in Fig. 3.2. Fig. 3.9 is the detailed tuning flow for Lottery to meet bandwidth requirements.

Explanation of each block :

- (1) First, we read the information about traffic behaviors of masters from designers. The input pattern is like Table 3.1.
- (2) Each master's required bandwidth must be smaller than its maximum bandwidth. The maximum bandwidth of a master is obtained by assuming that there is only one master on the channel, i.e., any request from the master will be granted immediately. To screen out unreasonable required bandwidth constraints, we evaluate the maximum bandwidth of each master first.

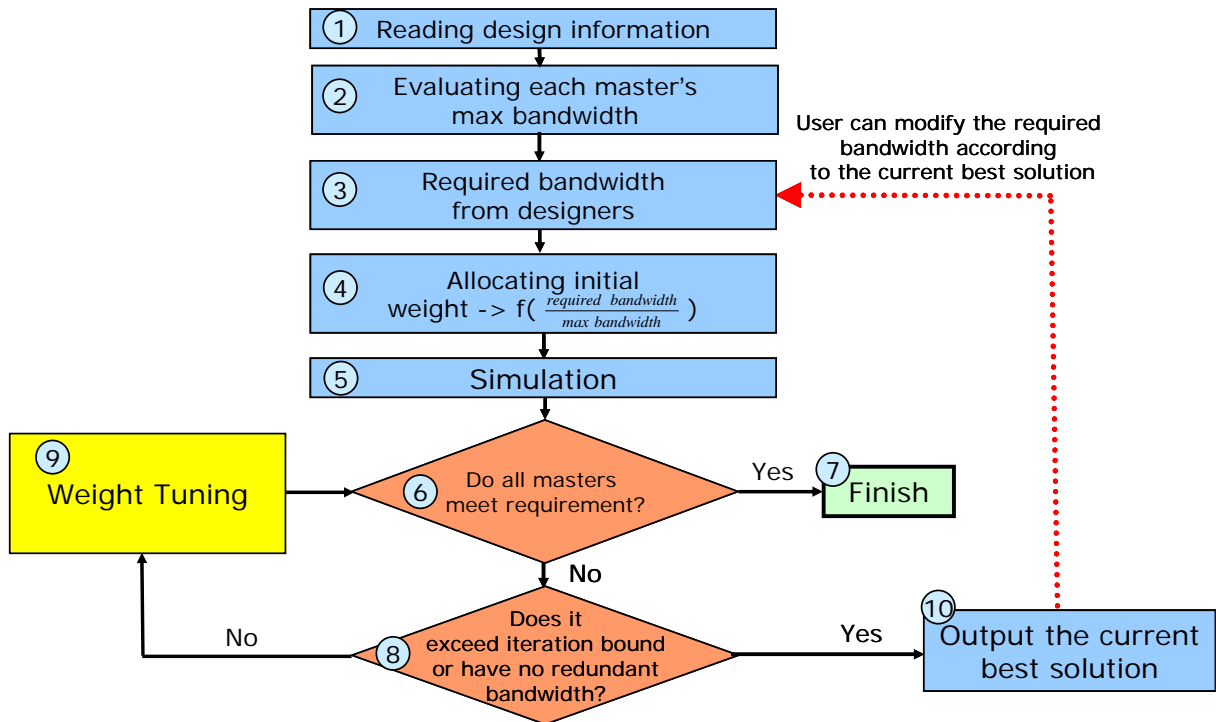


Fig. 3.9. The detailed weight tuning flow for Lottery

(3) Designers input required bandwidth of each master.

(4) The maximum bandwidth of each master is related to traffic condition. Initial weight allocation is based on each master's maximum bandwidth and required bandwidth. We introduce some definitions first .

(a)  $R_i$ : he max bandwidth of  $M_i$  .

(b)  $r_i$ : represents the required bandwidth of  $M_i$ .

(c)  $O_i$ : the value which is equal to  $r_i / R_i$  .

(d)  $T_i$ : initial tickets of  $M_i$  .

We set  $T_i = \text{int} [(total\ tickets * (O_i / \text{SUM} (O)))]$

(5) Based on masters' information and evaluation architecture (see Fig. 3.3), we simulate the whole system.

(6) If all masters meet bandwidth requirements, we get what we want and go to block 7, else go to block 8.

(7) Output the results of the flow.

- (8) Weight tuning intends to move bandwidth from the master whose bandwidth is more than its required bandwidth to the master whose bandwidth is less than required bandwidth. We say that the master has redundant bandwidth if its bandwidth is more than required bandwidth. If no master has redundant bandwidth, the weight tuning process is no more useful and stops (block 10). Otherwise, go to block 9 for weight tuning. Although there is any master with redundant bandwidth, it is possible that we still cannot meet bandwidth requirements with weight tuning. We set the iteration bound for our flow. If the iteration number exceeds the iteration bound, go to block 10.
- (9) Based on evaluation result, we try to tune weigh for meeting bandwidth requirements. The arbitration algorithm of weight tuning is presented in section 3.3.4.
- (10)Output the current best solution for designers. Designers may change the required bandwidth of each master according to the current best solution.

### 3.3.3 Acceptance range



How do we define meeting bandwidth requirements? If we demand that the required bandwidth is exactly equal to the simulated bandwidth, it is somewhat unreasonable. Based on the same input information of masters and required bandwidth, the results of our experiment may even vary with different random seeds. We design the experiment to decide the acceptance range from required bandwidth to simulated bandwidth. The bandwidth requirements are met if the difference of required bandwidth and simulated bandwidth is within this acceptance range. Based on the same input information of masters and the required bandwidth, we simulate the whole system with different seeds (0 ~ 65535) to measure the max difference of simulated bandwidth. The result shows that the max difference of simulated bandwidth is 4% of total bandwidth and we choose 2% as the acceptance range.

### 3.3.4 Algorithm of weight tuning

In the section, the algorithm of the block named weight tuning in Fig. 3.9 is presented (see Fig 3.10). Weight tuning intends to move bandwidth from the master which has the most extra bandwidth to the master which lacks bandwidth the most. Since we do not know the suitable amount of weight transfer to meet bandwidth requirements, the amount of weight transfer is initialized as certain value (described later) and is decremented with binary method. To be more detail, we introduce some definitions first:

$S_{\text{more}}$  :

$S_{\text{more}}$  is the set of masters having more bandwidth than required.

Eq.:

If ( $M_i$ 's simulated bandwidth –  $M_i$ 's required bandwidth > 2%),  $M_i \in S_{\text{more}}$   
(2% is acceptance range decided in section 3.3.3)

$S_{\text{less}}$  :

If some master's simulated bandwidth is less than required bandwidth, this master lacks bandwidth.  $S_{\text{less}}$  is the set of masters lacking bandwidth.

Eq.:

If ( $M_i$ 's required bandwidth –  $M_i$ 's simulated bandwidth > 2%),  $M_i \in S_{\text{less}}$

$S_{\text{met}}$  :

If some master's simulated bandwidth is about equal to required bandwidth, this master meets bandwidth requirements.  $S_{\text{met}}$  is the set of masters meeting bandwidth requirements.

Eq.:

If ( $| M_i$ 's required bandwidth –  $M_i$ 's simulated bandwidth | < 2%),  $M_i \in S_{\text{met}}$

- $m_{most}$  : The master having the most extra bandwidth in  $S_{more}$
- $m_{least}$  : The master lacking the most bandwidth in  $S_{less}$
- $t_m$  : The number of tickets  $m_{most}$  has
- $t_l$  : The number of tickets  $m_{least}$  has
- $t_d$  : The number of tickets that we try to tune each time
- $t_m'$  : The number of tickets  $m_{most}$  has after weight tuning
- $t_l'$  : The number of tickets  $m_{least}$  has after weight tuning
- $R$  : Right bound used for deciding  $t_d$
- $L$  : Left bound used for deciding  $t_d$

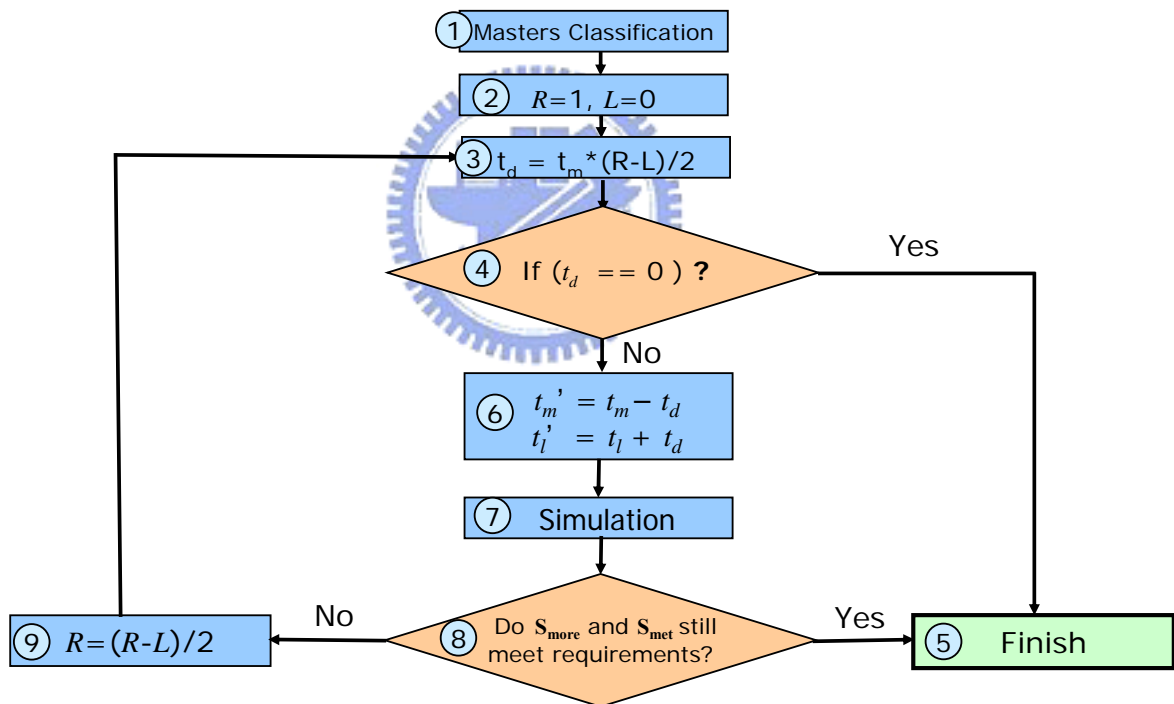


Fig. 3.10. The flow of weight tuning

Explanation of each block :

- (1) First, masters are classified into three sets -  $S_{more}$ ,  $S_{less}$ , and  $S_{met}$ .
- (2) R and L are initialized.

(3) Decide  $t_d$ ,  $t_d = (R-L) t_m / 2$

$t_d$  is used for deciding  $t_m'$  and  $t_l'$ .

(4) The block checks whether the iteration is meaningful or not. If the iteration is meaningful, go to block 6, else go to block 5. If  $t_d$  is equal to zero,  $t_m'$  and  $t_l'$  cannot be changed (see block 6) and weight tuning does not need going on.

(5) Finish the weight tuning.

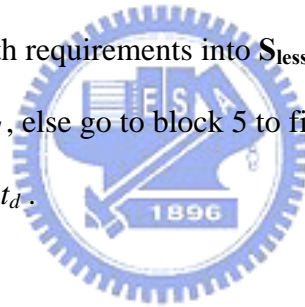
(6) Decide  $t_m'$  and  $t_l'$  according to  $t_d$ .

(7) Simulate the whole system with new weight allocation.

(8) The proposed flow of weight tuning is required to merely improve bandwidth allocation.

This block intends to prevent weight tuning from worsening bandwidth allocation.  $S_{\text{more}}$  and  $S_{\text{met}}$  are sets that meet bandwidth requirements. If  $t_m'$  and  $t_l'$  make these sets which originally meet bandwidth requirements into  $S_{\text{less}}$  which violate bandwidth requirements, go to block 9 to reduce  $t_d$ , else go to block 5 to finish weight tuning.

(9) Reduce  $R$  for decreasing  $t_d$ .



The purpose of weight tuning is transferring the fixed amount ( $t_d$ ) of weight from  $m_{\text{most}}$  to  $m_{\text{least}}$ . If the current  $t_d$  transfers any master from  $S_{\text{more}}$  or  $S_{\text{met}}$  to  $S_{\text{less}}$ , we would decrease  $t_d$  in the proposed flow.

Pseudo code is shown as follows:

Master classification ;

Initialize  $R = 1$  ,  $L = 0$ , finish = 0;

// record the old value of  $t_m$  and  $t_l$

$t_{m\_old} = t_m$  ;

$t_{l\_old} = t_l$  ;



```

while (finish == 0)
{
     $t_d = (R - L) * t_m / 2 ;$ 
     $t_m = t_m - t_d ;$ 
     $t_l = t_l + t_d ;$ 
    if ( $t_d = 0$ ) // Loop breaks if it is not a meaningful action
    do
    {
         $t_m = t_{m\_old} ;$ 
         $t_l = t_{l\_old} ;$ 
        break;
    }
    simulate ();
    if ( $S_{more}$  and  $S_{met}$  still meet requirements)
    do
    {
        finish = 1;
    }
    else
    {
         $R = (R-L) / 2 ;$ 
    }
}

```



# Chapter 4

## Experimental Results

### 4.1 Experimental environment

We compare RT\_lottery with Lottery, Static Priority, and TDM + Lottery (two-level). The parameters of these arbitration algorithms are set as follows:

(1) Lottery:

The weight of each master is allocated according to each master's required bandwidth (weight ratio = required bandwidth ratio)

(2) Static Priority:

The priority of each master is according to its required bandwidth. The master with higher required bandwidth has a higher priority.

(3) TDM + Lottery

To meet real-time and bandwidth requirements at the same time, we design a two-level arbitration algorithm.

1<sup>st</sup> level – TDM :

The 1<sup>st</sup> level intends to meet real-time requirements. Masters which have real-time

requirements are allocated time slots. There are three steps for our algorithm of allocating time slots to masters which have real-time requirements.

a. Calculate the distance of time slots for each master to meet real time requirements:

$D_i$ : the distance of time slots of  $M_i$

$$D_i = \text{int} ( R_{cycles} / B_{max} ) - 1$$

(  $B_{max}$ : the max possible beat number of all masters

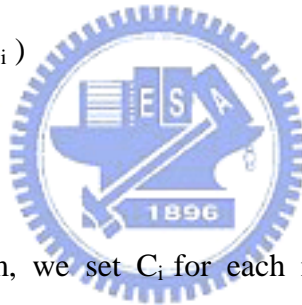
$R_{cycles}$ : the real-time requirements (in cycles) of the  $M_i$  )

b. Calculate  $S_i$ :

Since the 1<sup>st</sup> level is the time wheel, we must decide the size of the wheel.  $S_i$  is used to decide the size of the time wheel. It can be seen as the number of time slots which we hope possessed by  $M_i$  in the whole time wheel.

$$L = \text{LCM} ( \text{all } D_i )$$

$$S_i = L / D_i$$



c. Allocate time slots:

In the algorithm, we set  $C_i$  for each master.  $C_i$  is like counter to indicate allocating time slots.

$C_i$ : the counter of  $M_i$

First, we initialize  $C_i=D_i$ . Then, run through time slots of wheel to allocate each time slot to the suitable master. At each time slot, the master with minimum  $C_i$  gets this time slot. Once  $M_i$  gets time slot,  $C_i = D_i$ ,  $S_i = S_i - 1$  and other masters' counters are decremented by 1 (Ex. For  $j^{\text{th}}$  master,  $C_j = C_j - 1$ ). If there is any  $M_i$  with zero  $S_i$ , allocate remaining masters to following time slots one more time (according to  $C_i$ ) and the algorithm stops.

Pseudo code is shown as follows:

//  $M_i$  means the  $i^{\text{th}}$  master which has real-time requirements;

```

for each  $M_i$ 
do
{
     $D_i = \text{int} ( R_{cycles\_i} / B_{max} ) - 1;$ 
}
 $L = \text{LCM} ( \text{all } D_i );$ 
for each  $M_i$ 
do
{
     $C_i = D_i;$     // initialize all  $C_i = D_i$ 
     $S_i = L / D_i;$ 
}
Finish = 0;
while (Finish == 0)
do
{
    for  $M_i$  whose  $C_i$  is the minimum
    do
    {
         $C_i = D_i ;$ 
         $S_i = S_i - 1;$ 
        time_slots.allocate( $M_i$ ); //  $M_i$  is allocated the current time_slot
    }
    for  $M_j$  whose  $C_j$  is not the minimum
    do
    {

```



```

    Cj = Cj - 1 ;
    Sj = Sj - 1 ;
}
if any Mi whose Si = 0
do
{
    Finish = 1;
    while (there is any Mj whose Sj != 0)
    do
    {
        for Mk whose Sk != 0 && Ck is the minimum
        do
        {
            time_slots.allocate(Mk);
            Sk = 0;
        }
    }
}
}

```



Fig. 4.1 shows an example.

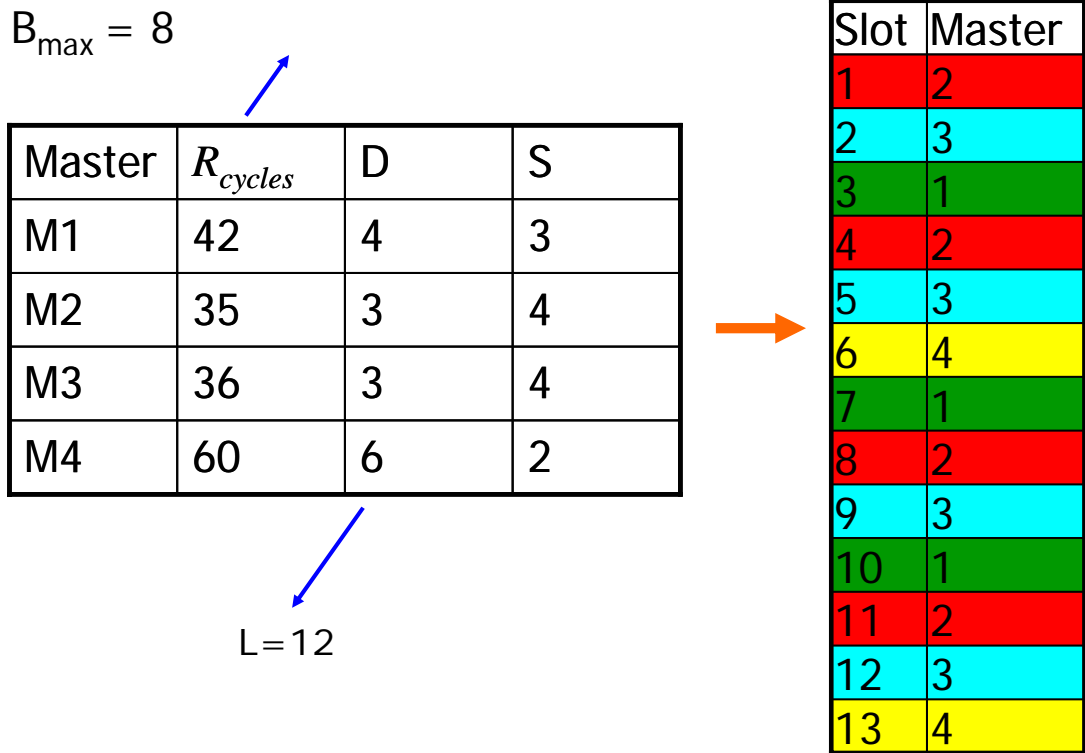


Fig. 4.1. The example of the 1<sup>st</sup> level of TDM + Lottery

In Fig. 4.1, we assume that  $B_{\max} = 8$  of all masters. According to the given  $R_{cycles}$ , we calculate  $D_i$  for each  $M_i$ . In the example,  $D_1 = \text{int}(42 / 8) - 1 = 4$ . Thus, we get  $L = 12$  ( $\text{LCM}(4,3,3,6) = 12$ ). Based on  $L$ , we calculate  $S_i$  for each  $M_i$ . In Fig. 4.1,  $S_3 = 12 / 3 = 4$ . The result of time slots allocation is just like the right table of Fig. 4.1.

2<sup>nd</sup> level – Lottery :

The weight of each master is allocated according to each master's required bandwidth (weight ratio = required bandwidth ratio).

## 4.2 Experiment 1

In experiment 1, we use the input information of masters shown in Table 4.1 [10][20].

Table 4.1. The input pattern for experiment 1

	Heavy	Light								
	type	$R_{cycles}$	beat/prob.		interval/prob.					
Master1	D		8/50	16/50	6/10	7/20	8/40	9/20	10/10	
Master2	D		1/50	4/50	10/10	11/20	12/40	13/20	14/10	
Master3	D_R	65	8/50	16/50	6/10	7/20	8/40	9/20	10/10	
Master4	D_R	85	1/50	4/50	10/10	11/20	12/40	13/20	14/10	
Master5	ND_R	65	8/50	16/50	65/10	66/20	67/40	68/20	69/10	
Master6	ND_R	85	1/50	4/50	85/10	86/20	87/40	88/20	89/10	

For each type of master, we design a heavy traffic master (intensive use bus) and a light traffic master (infrequent use bus). For example,  $M_1$  belongs to D type and its possible beat number is 8 or 16, which is larger than  $M_2$ 's 1 or 4. In the aspect of interval,  $M_1$ 's possible interval cycle is shorter than  $M_2$ 's on average.

Although the input information of masters is given, the required bandwidth of each master is undecided. The difficulty of meeting real-time and bandwidth requirements varies with the different required bandwidth pattern. Let us see the single case of required bandwidth pattern first. We record following values for evaluation:

(1)  $bw\_miss\_num$ :

The value represents the number of masters which miss bandwidth requirements.

(2)  $rt\_vio\_time$ :

This value is calculated by: SUM ( the number of real-time violations of *all* masters' requests ). If the request of  $M_i$  which has real-time requirements is not finished within the  $R_{cycles}$ , a real-time violation occurs.

(3)  $max\_latency$ :

During the simulation time, we record the latencies of all requests and choose the maximum one among these as the *max\_latency* .

Table 4.2 is the given required bandwidth pattern.

Table 4.2. The required bandwidth pattern for single case in experiment 1

	M1	M2	M3	M4	M5	M6	
Maximum Bandwidth(%)	63	18	63	19	17	2	
Required Bandwidth(%)	20	5	40	10	17	2	=> 94 % in total

The maximum bandwidth of each master is very diverse from one to one, since the input information of masters includes heavy and light traffics. The results are shown in Table 4.3.

Table 4.3. The result of the single case in experiment 1

	<i>bw_miss_num</i>	<i>max_latency</i> (cycle)	<i>rt_vio_time</i>
Static Priority	3 (50%)	7060	244
Lottery	3 (50%)	954	160
TDM+Lottery	1 (17%)	314	0
RT_lottery	0 (0%)	170	0

For the ability of bandwidth allocation, Static Priority is poor and Lottery still needs weight tuning for better results. In the aspect of real-time, Lottery and Static Priority are failed to meet real-time requirements since these two arbitration algorithms do not take real-time



requirements into consideration. The Static Priority is worse than Lottery for real-time requirements, because  $max\_latency$  of Static Priority is much larger than that of Lottery (7060 VS. 954). Even though TDM + Lottery has advantages of handling real-time (1<sup>st</sup> level) and bandwidth (2<sup>nd</sup> level), it still has bad bandwidth allocation capability ( $bw\_miss\_num = 1$ ).

It is not quite fair to conclude the comparison of these four arbitration algorithms just by a single case of required bandwidth pattern. We design a random required bandwidth pattern generator. This generator can randomly generate the required bandwidth for each master and the sum of these required bandwidth is equal to  $R_{sum}$ . In general, it is usually harder to meet requirements with higher  $R_{sum}$ . For each  $R_{sum}$ , the experiments are conducted to compare four arbitration algorithms with 100 different random required bandwidth patterns, and  $R_{sum\_i}$  represents the  $i^{th}$  iteration of simulation for  $R_{sum}$ . We record the following values during simulation for evaluation:



(1)  $rt\_vio\_time\_sum$  :

SUM(  $rt\_vio\_time$  in each  $R_{sum\_i}$  )

(2)  $rt\_fail\_sum$  :

The count of real-time failed cases in 100 iterations

( if  $rt\_vio\_time > 0$  in  $R_{sum\_i} \Rightarrow R_{sum\_i}$  is a real-time failed case )

(3)  $bw\_fail\_sum$  :

The count of bandwidth failed cases in 100 iterations

( if  $bw\_miss\_num > 0$  in  $R_{sum\_i} \Rightarrow R_{sum\_i}$  is a bandwidth failed case )

(4)  $fail\_sum$  :

The count of failed cases in 100 iterations

(if  $rt\_vio\_time > 0$  or  $bw\_miss\_num > 0$  in  $R_{sum\_i} \Rightarrow R_{sum\_i}$  is a failed case)

The results are shown in Table 4.4.

Table 4.4. The result of 100 random cases

RT_lottery	$R_{sum}$	$rt_v$	$bw_f$	$rt_f$	$fail$
	95	0	87	0	87
	90	0	80	0	80
	85	0	79	0	79
	80	0	68	0	68
	75	0	66	0	66
	70	0	57	0	57
	65	0	38	0	38

TDM+ Lottery	$R_{sum}$	$rt_v$	$bw_f$	$rt_f$	$fail$
	95	1	99	1	99
	90	8	96	8	96
	85	8	95	8	96
	80	6	91	6	91
	75	6	83	6	84
	70	3	75	3	75
	65	2	58	2	58

Lottery	$R_{sum}$	$rt_v$	$bw_f$	$rt_f$	$fail$
	95	12915	99	100	100
	90	12150	97	100	100
	85	11159	98	100	100
	80	10535	86	100	100
	75	9007	73	100	100
	70	9022	58	100	100
	65	8274	45	100	100

Static Priority	$R_{sum}$	$rt_v$	$bw_f$	$rt_f$	$fail$
	95	18577	100	100	100
	90	17396	100	100	100
	85	13739	100	99	100
	80	14235	98	100	100
	75	11200	88	99	100
	70	11076	83	97	97
	65	10345	82	96	98

$rt_v$  :  $rt\_vio\_time\_sum$        $rt_f$  :  $rt\_fail\_sum$   
 $bw_f$  :  $bw\_fail\_sum$            $fail$  :  $fail\_sum$

As can be seen, it is harder to meet requirements with larger  $R_{sum}$ . The number of *fails* decreases with lower  $R_{sum}$ . In real-time aspect, Lottery and Static Priority do not consider real-time requirements and hence the  $rt_f$  of these two arbitration algorithms are much larger than others. Static Priority is worse than Lottery since its  $rt_v$  is much larger than Lottery's. The 1<sup>st</sup> level of TDM + Lottery handles real-time requirements but it still fails for certain critical cases. In bandwidth aspect, Lottery and TDM + Lottery handle bandwidth requirements better than Static Priority, but RT\_lottery with weight tuning is the best.

Number of failed cases in different  $R_{sum}$  :

$$RT\_lottery < (TDM + Lottery) < Lottery \approx \text{Static Priority}$$

Table 4.5 is the summery of experiment 1.

Table 4.5. The summary of experiment 1

Arbitration algorithm	Real-time capability	Bandwidth allocation capability
RT_lottery	Always holds	Best
TDM + Lottery	Only fails for critical cases	Good but requiring weight tuning
Lottery	No consideration	Good but requiring weight tuning
Static Priority	No consideration	Poor

### 4.3 Experiment 2

The objective of experiment 2 is to observe the effect of beat number on arbitration algorithms. The input patterns are designed that all masters send the same beat number of 8, 16, and 32, respectively. The input information of masters for fixed 8-beat, 16-beat, and 32-beat is shown respectively in Table 4.6, Table 4.7, and Table 4.8. We run 100 random cases for each  $R_{sum}$ . Results are shown in Fig. 4.2.

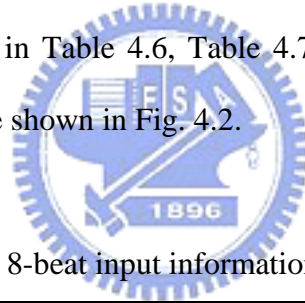


Table 4.6. The fixed 8-beat input information of masters in experiment 2

	type	$R_{cycles}$	beat/prob.	interval/prob.				
Master1	D		8/100	6/10	7/20	8/40	9/20	10/10
Master2	D_R	100	8/100	6/10	7/20	8/40	9/20	10/10
Master3	ND_R	100	8/100	100/10	101/20	102/40	103/20	104/10

Table 4.7. The fixed 16-beat input information of masters in experiment 2

	type	$R_{cycles}$	beat/prob.	interval/prob.				
Master1	D		16/100	6/10	7/20	8/40	9/20	10/10
Master2	D_R	100	16/100	6/10	7/20	8/40	9/20	10/10
Master3	ND_R	100	16/100	100/10	101/20	102/40	103/20	104/10

Table 4.8. The fixed 32-beat input information of masters in experiment 2

	type	$R_{cycles}$	beat/prob.	interval/prob.				
Master1	D		32/100	6/10	7/20	8/40	9/20	10/10
Master2	D_R	100	32/100	6/10	7/20	8/40	9/20	10/10
Master3	ND_R	100	32/100	100/10	101/20	102/40	103/20	104/10

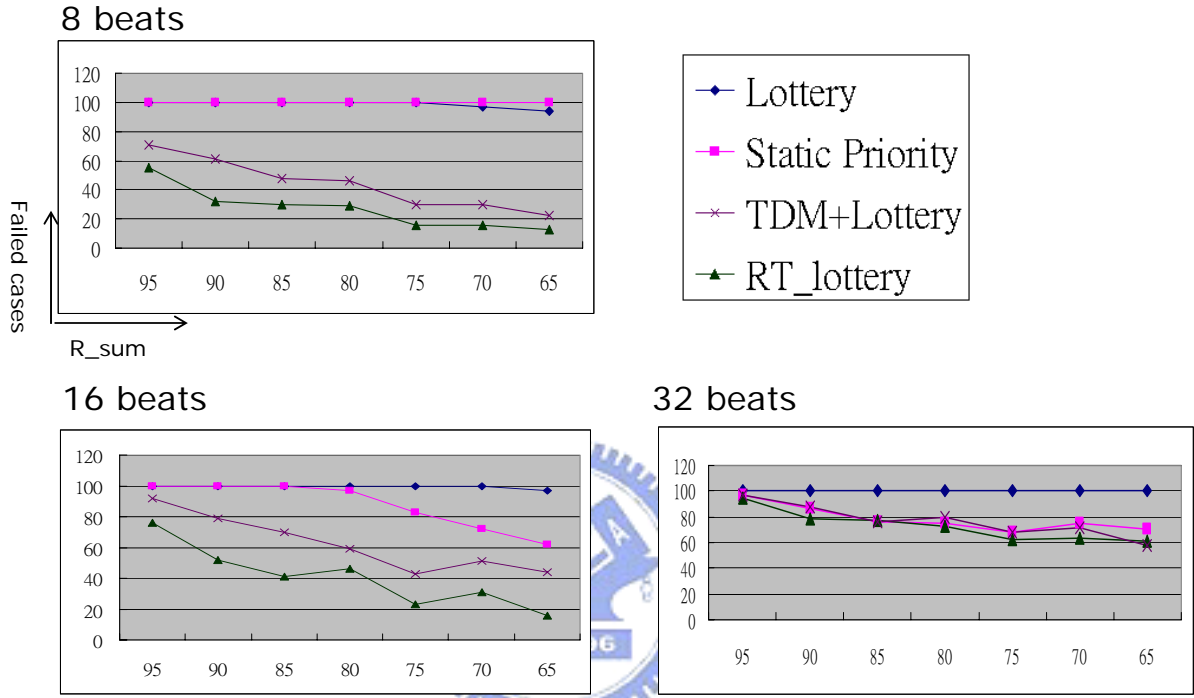


Fig. 4.2. Trend of failed cases for 100 random cases

As shown in Fig. 4.2, RT\_lottery is the best among four arbitration algorithms for fixed 8, 16 and 32-beat. As well, RT\_lottery and TDM + Lottery, which have capability of handling both bandwidth and real-time requirements, are much better than the other arbitration algorithms. Nevertheless, it is harder to meet requirements with larger fixed beat number for RT\_lottery and TDM + Lottery and the numbers of failed cases arise with larger beat number. The reason is that with larger beat number, the granularities of weight (ticker number) for RT\_lottery and TDM + Lottery are more coarse-grained. If there is fixed amount weight transfer from  $M_i$  to  $M_j$ , the influence of weight transfer on 8 or 16 fixed beat pattern is smaller than 32 fixed beat pattern.

# Chapter 5

## Conclusions

The two-level arbitration algorithm **RT\_lottery** with weight tuning is proposed in this thesis. Our evaluation model and weight tuning algorithm set the suitable parameters of RT\_lottery. The experiments compare RT\_lottery with Static Priority, Lottery, and TDM + Lottery. We make some conclusions as:

- (1) The results show that RT\_lottery is the best among four arbitration algorithms for meeting bandwidth and real-time requirements at the same time.
- (2) Our evaluation model and weight tuning algorithm really set the parameters of RT\_lottery well and make it more powerful.
- (3) With smaller possible beat number, the outstanding of RT\_lottery is more obvious because the granularity of its weight is more fine-grained.

# References

- [1] <http://www.webopedia.com/>
- [2] <http://www.arm.com>
- [3] A. Jantsch, and H. Tenhunen, *Networks on Chip*, Kluwer Academic Publisher, 2003.
- [4] A. Adriahtenaina, H. Charlery, A. Greiner, L. Mortiez and C. A. Zeferin, "SPIN: A Scalable, Packer Switched, On-Chip Micro-Network," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2003, supplements 70-73.
- [5] C. A. Zeferino and A. A. Susin, "SoCIN: A Parametric and Scalable Network-on-Chip," in *proceedings of the symposium on Integrated Circuits and Systems Design*, 2003, Page(s) 169-174.
- [6] P. P. Pande, C. Grecu, A. Ivanov and R. Saleh, "Design of A Switch for Network on Chip Applications," in *Proceedings of the 2003 International Symposium on Circuits and Systems*, 2003, volume 5, Page(s) 217 – 220.
- [7] AMBA AXI Protocol Specification rev. r0p0.
- [8] C. H. Pyoun, C. H. Lin, H. S. Kim, and J. W. Chong, "The Efficient Bus Arbitration Scheme In Soc Environment," *International Workshop on System-on-Chip for Real-Time Applications*, 2003, Page(s):311 – 315.
- [9] M. Yang, S.Q. Zheng, Bhagyavati, and S. Kurkovsky, "Programmable Weighted Arbiters for Constructing Switch Schedulers," *Workshop on High Performance Switching and Routing*, 2004, Page(s):203 – 206.
- [10] M. Conti, M. Caldari, G. B. Vece, S. Orcioni, and C. Turchetti, "Performance Analysis of Different Arbitration Algorithms of the AMBA AHB Bus," *Design Automation Conference*, 2004.

- [11] F. Poletti, D. Bertozzi, L. Benini, and A. Bogliolo, "Performance Analysis of Arbitration Policies for SoC Communication Architectures," *Journal of Design Automation for Embedded Systems*, 2003, Page(s):618 – 621.
- [12] E. S. Shin, V. J. Mooney, G. F. Riley, "Round-Robin Arbiter Design and Generation," *Symposium on System Synthesis*, 2002, Page(s):243 – 248.
- [13] K. C. Lee, "A Variable Round-Robin Arbiter for High Speed Buses and Statistical Multiplexers," *International Phoenix Conference on Computers and Communications*, 1991, Page(s):23 – 29.
- [14] K. Lahiri, A. Raghunathan, and G. Lakshminarayan, "LOTTERYBUS: A New High-Performance Communication Architecture for System-On-Chip Designs," *Design Automation Conference*, 2001, Page(s):15 – 20.
- [15] [acalab1.csie.ntu.edu.tw/aca2002/slides/lecture11.pdf](http://acalab1.csie.ntu.edu.tw/aca2002/slides/lecture11.pdf)
- [16] C. Li, R. Bettati, W. Zhao, "Static Priority Scheduling for ATM Networks," *Proceedings on Real-Time Systems Symposium*, 1997, Page(s):264 – 273.
- [17] S. Ramamurthy, M. Moir, "Static-Priority Periodic Scheduling on Multiprocessors," *Proceedings on Real-Time Systems Symposium*, 2000, Page(s):69 – 78.
- [18] B. Andersson, S. Baruah, J. Jonsson, "Static-Priority Scheduling on Multiprocessors," *Proceedings on Real-Time Systems Symposium*, 2001, Page(s):193 – 202.
- [19] A.C Waldspurger and W.E Weih., "Lottery Scheduling: Flexible Proportional Share Resource Management," *Symp. on Operating Systems Design and Implementation*, 1994.
- [20] K. Lahiri, A. Raghunathan, and S. Dey., "Evaluation of The Traffic Performance Characterization of System-on-Chip Communication Architectures," *International Conference on VLSI Design*, 2001, Page(s):29 – 35.