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

Chien-Hua Chen, Geeng-Wei Lee, Juinn-Dar Huang, and Jing-Yang Jou
Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan
e-mail: {tony, gwlee, jyjou}@eda.ee.nctu.edu.tw, jdhuang@mail.nctu.edu.tw

Abstract – In shared SoC bus systems, arbiters are usually adopted to solve bus contentions with various kinds of arbitration algorithms. We propose an arbitration algorithm, RT_lottery, which is designed to meet both hard real-time and bandwidth requirements. For fast evaluation and exploration, we use high abstract-level models in our system simulation environment to generate parameters for our configurable arbiter. The experimental results show that RT_lottery can meet all hard real-time requirements and perform very well in bandwidth allocation. The results also show that RT_lottery outperforms several commonly-used arbitration algorithms today.

1. Introduction

Although there are many possible communication architectures for inter-module communications in SoC systems, shared buses are still very popular among these architectures because of their simplicity and area efficiency. The masters on an SoC bus may issue requests simultaneously and hence an arbiter is required to decide which master is granted for bus access. In many applications, masters may have real-time and/or bandwidth requirements on requests. A master with a real-time requirement demands its transactions accomplished within a fixed number of clock cycles. On the other hand, a master with a bandwidth requirement must occupy a fixed fraction of total bandwidth of a bus. If designers find that the implemented arbitration algorithm cannot fulfill some requirements at late design stages, they have to return to a very early design stage to modify the original arbitration algorithm. This would result in a significant schedule delay.

Arbitration algorithms commonly used for shared buses include Static Priority, Time Division Multiplexing (TDM), and Round-Robin [1-4]. Lottery is the arbitration algorithm proposed recently [5] with the advantages of (i) providing designers with good control over bandwidth allocation for each master, and (ii) providing a high-priority master with quite low transaction latency. However, all arbitration algorithms mentioned above cannot well handle bandwidth and hard real-time requirements concurrently.

In this paper, we propose a two-level arbitration algorithm, RT_lottery, which is expected to meet hard real-time and bandwidth requirements of each master at the same time. At the 1st level, we use a Real-Time Handler to satisfy all hard real-time requirements. At the 2nd level, a Lottery-based algorithm with tuned weight is adopted for proper bandwidth allocation.

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

The rest of this paper is organized as follows: previous work including the introduction to some common arbitration algorithms (Static Priority, TDM, and Lottery) is presented in Section 2. Section 3 describes the proposed arbitration algorithm (RT_lottery) and the flow for generating appropriate parameters of RT_lottery to meet bandwidth requirements. The experimental environment and results are shown in Section 4. Section 5 concludes this paper.

2. Preliminaries

2.1 Previous work

In this section, we briefly introduce several previous arbitration schemes [1-4].

1) Static Priority:

Each master is statically assigned a unique priority value. When multiple masters issue requests simultaneously, the master with the highest priority gets granted. The advantage of this arbitration scheme is its simple implementation and small area cost. However, if masters with higher priority issue requests excessively and frequently, other masters with lower priority may rarely be granted. This could introduce severe starvation of low-priority masters and result in extremely unfair bandwidth allocation.

2) TDM:

Time Division Multiplexing (TDM) algorithm divides access time on a bus into time slots and then allocates these slots to masters in certain way. If a master possessing the current time slot does not issue request, the time slot would be wasted. To mend this inefficiency, a 2nd level arbitration algorithm is usually adopted to reallocate the current slot to other requesting masters. Fig. 1 is an example architecture of two-level TDM.

For a two-level TDM arbitration algorithm, the 1st level uses a time wheel where each slot is statically reserved for a unique master and the 2nd level can adopt any arbitration algorithm depending on the target application. For example, if the bandwidth allocation among masters is important, 2nd level can use an arbitration algorithm with better ability of

0-7803-9451-8/06/$20.00

bandwidth allocation. Also note that Round-Robin is actually one kind of TDM algorithm.

3) Lottery [5-6]:

An arbiter implementing the Lottery arbitration algorithm is like a lottery manager deciding which lucky one wins a prize. Each mater on the bus is statically assigned a number of "lottery tickets". The lottery manager generates a pseudo random number, and the master having the ticket matched to this number is granted for access. Obviously, the master having more tickets is more likely granted.

Let the masters be M_1, M_2, …, M_n and the number of tickets held by each master be t_1, t_2, ..., t_n. At any cycle, the set of pending requests is represented by a set of Boolean variables r_1, r_2, ..., r_n, where r_i =1 means that M_i has a pending request, and r_i =0 otherwise. The master to be granted is chosen with the probability given by the equation:

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

The number of tickets of each master can be regarded as its weight. A master with higher weight has higher probability to be granted. We represent the number of tickets possessed by a master as its weight in the following sections. In summary, the Lottery arbitration algorithm is (i) capable of providing designers with good control over bandwidth allocation for each master, and (ii) quite good at providing high priority master with low transaction latency.

2.2 Observations on Lottery arbitration algorithm

A real-time requirement in the previous work [5] is represented in terms of the average transaction latency. However, such a requirement can only be regarded as a loose real-time requirement since there may exist some extremely long-latency transactions. For hard real-time requirements, all transaction latencies (not the average transaction latency) must be smaller than the given requirement all the time.

Meanwhile, to meet the bandwidth requirements, masters are assigned weights according to the ratio of their required bandwidth [5]. Nevertheless, if the bus access behaviors are very diverse among masters, the actual bandwidth ratio would not conform to the weight ratio. The reason may be that the actual traffic load generated by some master is much less or much more than it requests. For example, a master asks for a large fraction of bus bandwidth but rarely issues requests. For this reason, we propose a weight tuning method for better bandwidth allocation.

To meet the real-time requirements, the weight of the master with the minimum latency requirement should be much larger than the others. However, it is really hard to assign a proper weight to each master if there are multiple masters with diverse real-time and bandwidth requirements. For instance, how to assign a proper weight to a master that has a tight real-time requirement but requires only a small fraction of bus bandwidth. Furthermore, if there are masters having hard real-time requirements, a probabilistic arbitration algorithm like Lottery is obviously not appropriate for such applications.

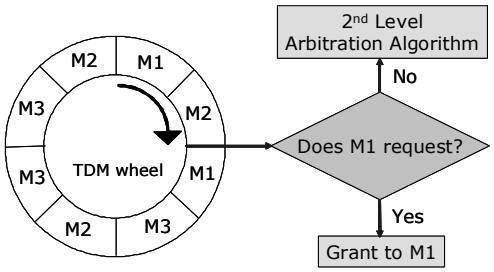


Fig. 1. An example architecture of the two-level TDM

3. Proposed Approach

3.1 Proposed arbiter architecture

Since probabilistic arbitration algorithms cannot handle hard real-time requirements, we propose a two-level arbitration algorithm, RT_lottery (R for **R**eal-time, T for **T**uned weight) to solve the problem. The proposed arbiter architecture is shown in Fig. 2. The 1st level, *Real-Time Handler*, is designed to handle real-time requirements. The 2nd level, *Lottery with tuned weight*, is designed to handle bandwidth requirements. The weight of each master is fine tuned by our weight tuning algorithm based on the evaluation results obtained from system simulation. The details of RT_lottery will be described in later sections.

3.2 Simulation model

In our model, it is assumed that once a master possesses the bus, other masters cannot access the bus until the possessing master releases the bus, i.e., each transaction is non-preemptive. An example of a system architecture containing four masters is shown in Fig. 3. Each master has a traffic generator. The behavior of each traffic generator is given by designers. The arbiter receives requests from all masters then decides which master should be granted.

There are four types of traffic behaviors that can be given for a master:

(1) R_{cycles}:

It is the real-time requirement (in clock cycles) of a master. For those masters without real-time requirements, this information should be left undefined.

(2) Beat number and probabilities:

It defines the probabilities of burst sizes possibly issued by a master. Take Table 2 for example, M3 issues requests of which 50% requests are 8-beat burst and the other 50% requests are 16-beat burst.

(3) Interval cycles and probabilities:

It determines the interval time between two successive requests issued by a master. However, the rule of deciding the interval time varies with different master types (explained later). For example, in Table 2, 10% of the request interval of M1 is 6 clock cycles while 20% is 7 clock cycles and so on.

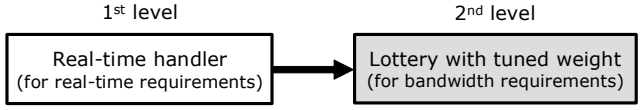


Fig. 2. Proposed arbiter architecture

(4) Type:

In our work, masters are classified into three types based on their traffic behaviors:

1. **D type** (D for Dependency):

D type masters have no real-time requirements and the next request is issued at the time depending on the finish time of the current request. For D type masters, the interval time between two successive requests is the time from the issued time of the former to the finish time of the latter. Fig. 4(a) shows an example. At cycle 2, assume the traffic generator generates a 4-beat burst. The request is not granted until cycle 5 and is finished at cycle 9 (4-beat burst). If the interval time is 10, then the next request is issued at cycle 19 (The issued time of the latter request is the finish time of the former plus 10 cycles).

2. **D_R** type (D for Dependency, R for Real-time):

D_R type masters are the same as D type masters except that they have extra real-time requirements. Fig. 4(b) is an example with the same parameters used in Fig. 4(a). In this example, the master has a real-time requirement, R_{cycle}, which is set to 10 cycles. Thus the request issued at cycle 2 must be finished before cycle 12 (2 + R_{cycles} = 12), which is shown as the dotted line in the figure. If the request is not finished before cycle 12, a real-time violation occurs.

3. **ND_R type** (ND for No Dependency, R for Real-time):

The issued time of a request from an ND_R type master is independent of the finish time of its previous request, and the interval time is the clock cycles between two successive requests. In Fig. 4(c), assume that the interval time is 15. The second request is issued at cycle 17, which directly depends on the issued time (at cycle 2) of the first request but not its finish time (at cycle 9). Since the current request must be finished before the next request, the reasonable value of R_{cycles} is supposed to be smaller than the minimum possible interval time. That is, designers can also assign a tighter real-time requirement. To ensure a reasonable R_{cycles}, we define R_{cycles} = min($t_{min\_interval}$, t_{user_given}), where $t_{min\_interval}$ is the minimum possible interval time and t_{user_given} is the real-time requirement given by designers.

3.3 Proposed arbitration algorithm

In this section, the algorithms of the Real-Time Handler and the weight tuning process for Lottery are described in detail.

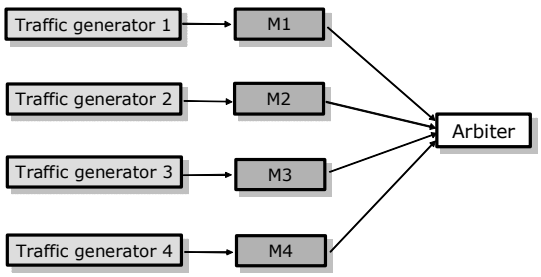


Fig. 3. An example architecture

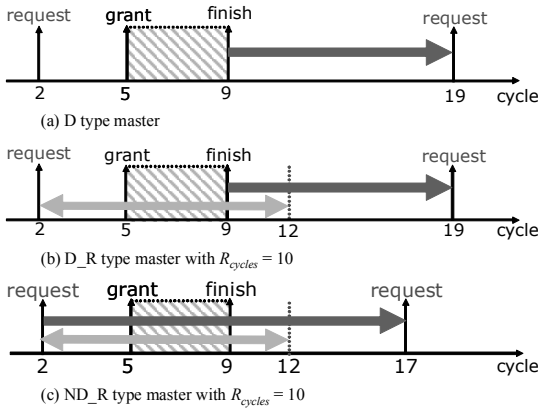


Fig. 4. The example of three types of masters

3.3.1 Real-Time Handler

The Real-Time Handler sets a real-time counter for each master according to their 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. *warning_line* is a global constant value used to remind the arbiter to grant the most urgent master. The master would have higher priority if its corresponding real-time counter value is belowthe *warning_line*. When two or more real-time counters are below *warning_line*, the master with the smallest real-time counter value (more urgent) gets granted. Fig. 5 shows an example of Real-Time Handler's operation. We assume that M1 has R_{cycles} = 30 and the whole system has *warning_line* = 25.

Let us focus on cycle 3 and cycle 11:

(1) Cycle 3 (the left table in Fig. 5):

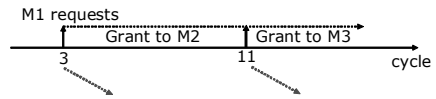
As M1 issues a request at this cycle, the real-time counter of M1 is set to its R_{cycles} , 30. All other masters also issue requests at this time, but only M2's real-time counter value is below *warning_line* and thus it is granted first.

(2) Cycle 11 (the right table in Fig. 5):

M2's request is a 8-beat burst, and therefore the request is finished at cycle 11. At this time, all real-time counters of the pending masters (M1 and M3) are decremented by 8. The values of real-time counters of M1 and M3 are both below *warning_line*. Since the value of M3's real-time counter is smaller, M3 is granted at this cycle.

To meet all real-time requirements in any circumstances, *warning_line* must be carefully set according to the worst contending case. That is,

warning_line = ∑*(maximum possible beat of D_R and ND_R masters) + maximum possible beat number of D masters*



	Real-time counter	Request	Grant
M1	30	Yes	No
M2	25	Yes	Yes
M3	27	Yes	No

	Real-time counter	Request	Grant
M1	22	Yes	No
M2	25	No	No
M3	19	Yes	Yes

Fig. 5. R_{cycles} of M1 = 30, *warning_line* = 25

The idea behind *warning_line* is as follows: in the worst circumstance, the D type master with the maximum possible beat number issues a request of its maximum possible beat number and gets granted. At the next cycle, all other masters with real-time requirements all issue requests. It must be guaranteed that all the real-time requirements of these masters can still be met after the request of this D type master is finished.

Take Table 1 as an example and the worst contending case is shown in Fig. 6:

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 with R_{cycles} smaller than *warning_line*, the proposed arbiter is guaranteed to meet all hard real-time requirements.

3.3.2 Weight tuning flow for Lottery

In this section, we present the 2nd level of RT_lottery, Lottery with tuned weight. Fig. 7 shows the weight tuning flow.

First, we read in the traffic information of each master given by designers. Each master's required bandwidth must be smaller than its maximum bandwidth. The maximum bandwidth of a master is calculated by assuming there is only one master on the bus, i.e., all requests from the master are granted immediately. To screen out unreasonable bandwidth requirements, we evaluate the maximum bandwidth of each master first. Initial weight assignment is based on each master's maximum and required bandwidth.

Second, the weight tuning process tries to move bandwidth share from a master whose allocated bandwidth is more than its required bandwidth to another master whose allocated bandwidth is less than its required bandwidth. We say that a master has extra bandwidth if its allocated bandwidth is more than its required bandwidth. If there are no masters having extra bandwidth, the weight tuning process stops.

3.3.3 Algorithm of weight tuning

In this section, the greedy algorithm of the block named weight tuning in Fig. 7 is presented. First, we introduce some definitions:

- M_i: Each master in the system is marked as M_i, $i = 1 \sim n$, where n is the total number of masters in the system.
- $\mathbf{S_{more}}$: If (M_i's simulated bandwidth – M_i's required bandwidth > 2%), $M_i \in \mathbf{S_{more}}$.
- $\mathbf{S_{less}}$: If (M_i's required bandwidth – M_i's simulated bandwidth > 2%), $M_i \in \mathbf{S_{less}}$.
- $\mathbf{S_{met}}$: If ($|M_i$'s required bandwidth – M_i's simulated bandwidth| < 2%), $M_i \in \mathbf{S_{met}}$.
- m_{most}: The master with the most extra bandwidth in $\mathbf{S_{more}}$.
- m_{least}: The master lacking the most bandwidth in $\mathbf{S_{less}}$.
- t_m: The number of tickets m_{most} has.
- t_l: The number of tickets m_{least} has.

Table 1. A traffic pattern for the explanation of *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

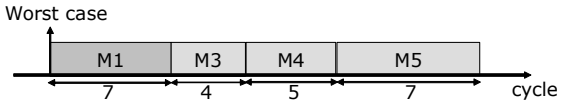


Fig. 6. The worst contending case in Table 1 for real-time

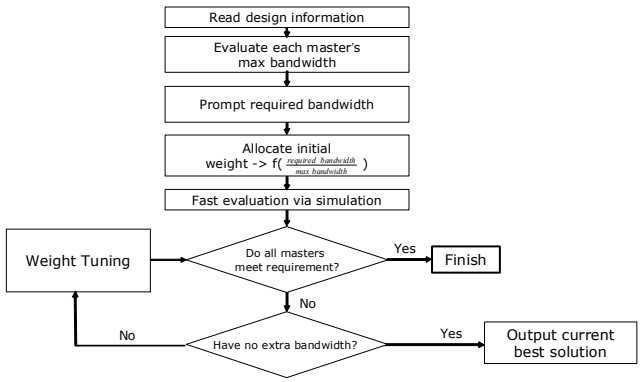


Fig. 7. The weight tuning flow for Lottery

- t_d: The number of tickets that we try to move from t_m to t_l each time.
- B : The bound used for deciding t_d.

The pseudo code of the weigh tuning algorithm is shown in Fig. 8. First, masters are classified into three exclusive sets, $\mathbf{S_{more}}$, $\mathbf{S_{less}}$, and $\mathbf{S_{met}}$ (*line 1*), and then B is initialized (*line 2*). The *while* loop (*line 5-19*) decides t_d (*line 6*) for new t_m and t_l (*line 12 and 13*) in each iteration. It stops on two conditions: (i) $t_d = 0$, the weight cannot be tuned any more (*line 7*); (ii) the new t_m and t_l do not result in moving masters whose bandwidth requirements are met originally ($\mathbf{S_{more}}$ and $\mathbf{S_{met}}$) into $\mathbf{S_{less}}$ (*line 15*). Otherwise, another iteration proceeds and B is reduced to re-calculate a new t_d, t_m and t_l.

4. Experimental Results

4.1 Experimental environment setup

We compare RT_lottery with other three arbitration algorithms, Lottery, Static Priority, and TDM+Lottery. We use a system containing six masters for evaluation. The parameters of these arbitration algorithms are set as follows:

(1) Lottery:

The weight of each master is assigned according to its required bandwidth (weight ratio = required bandwidth ratio).

(2) Static Priority:

Each master is assigned a priority according to its required bandwidth. The master with higher required bandwidth has a higher priority.

```
1:  Classify masters;
2:  Initialize B = 1, finish = 0;
3:  t_m_old = t_m;          // Record the old value of t_m and t_l
4:  t_l_old = t_l;
5:  while ( finish == 0 ) {
6:      t_d = B * t_m / 2;
7:      if (t_d == 0) { // Loop breaks if it is not a meaningful action
8:          t_m = t_m_old;
9:          t_l = t_l_old;
10:         break;
11:     }
12:     t_m = t_m_old - t_d;
13:     t_l = t_l_old + t_d;
14:     simulate();
15:     if(requirements of the masters in S_more and S_met are still met)
16:         finish = 1;
17:     else
18:         B = B / 2;
19:}
```

Fig. 8. The pseudo code of weight tuning

(3) TDM+Lottery:

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

2nd level - Lottery: The weight of each master is assigned according to its required bandwidth (weight ratio = required bandwidth ratio).

4.2 Experiment 1

In this experiment, 6 masters are put on a bus with the traffic behaviors shown in Table 2 [3,7]. For each type of master, we design a heavy-traffic master and a light-traffic master. For example, both M1 and M2 are D type masters, and the requests issued by M1 have larger beat numbers and shorter average interval than those issued by M2. That means M1 generates a heavier traffic load to the bus than M2 does.

The difficulty to meet both real-time and bandwidth requirements generally depends on the total required bandwidth in a system. In the following, we conduct two experiment cases for observations. We consider a given total required bandwidth in one case, and consider a set of 100 different total required bandwidth cases randomly generated in the other case.

First, we evaluate the case that the total required bandwidth utilizes 94% of the entire bus bandwidth, as shown in Table 3. The evaluated maximum bandwidth and the given required bandwidth of masters are also shown in Table 3. From the table, we observe that the maximum bandwidth of each master is very different from each other because there are masters with heavy- and light-traffic loads.

All the experiments are conducted on a PC with a Intel Pentium 4 2.8G processor and 512MB DRAM. Following statistics are recorded during simulation for evaluation:

(1) *bw_miss_num*:
This value represents the number of masters whose bandwidth requirements are missed.

(2) *rt_vio_time*:
This value is calculated by: ∑*(the number of real-time violations of all masters' requests)*. If a request of M_i with real-time requirements is not finished within *Mi's* R_{cycles} cycles, a real-time violation occurs on this request.

(3) *max_latency*:
During a simulation run, we record the latencies of all requests and pick the maximum latency among them as the *max_latency*.

The experimental results are shown in Table 4. On the ability of bandwidth allocation, Static Priority is poor as expected, but Lottery is surprisingly poor as well. This fact indicates that Lottery still needs a good weight tuning strategy for better bandwidth allocation. On the aspect of real-time handling ability, Lottery and Static Priority are failed to meet real-time requirements since they do not take real-time requirements into consideration. Note that Static Priority is even worse than Lottery because its *max_latency* is much longer than that of Lottery (7060 vs. 954). Though TDM+Lottery can handle real-time and bandwidth requirements better, it still fails in bandwidth allocation (*bw_miss_num* = 1).

In general, it is usually harder to meet requirements with higher total required bandwidth summed from all the masters with bandwidth requirements, i.e., the bus utilization is supposed higher. In the second experiment case, we use a generator that can randomly generate the required bandwidth for each master. And let R_{sum} represent the total required bandwidth in terms of the percentage of entire bus bandwidth. Here, we evaluate seven different values of R_{sum}, ranging from 65% to 95%. For each R_{sum}, 100 random cases are conducted to compare four arbitration algorithms. R_{sum_i} represents the i_{th} case (i = 1 ~ 100) of simulation for R_{sum}. The simulation time for each R_{sum_i} is less than one minute on our equipment. Following statistics are recorded during simulation for evaluation:

(1) *rt_vio_time_sum*:
∑*(rt_vio_time* in each R_{sum_i}*)*

(2) *rt_fail_case_sum*:
The number of cases which contain one or more real-time violations among all 100 cases (R_{sum_i} is a case failed to meeting real-time requirements if *rt_vio_time* > 0 in R_{sum_i}).

(3) *bw_fail_case_sum*:
The number of cases which fail to meet bandwidth requirements among all 100 cases (R_{sum_i} is a case failed to meeting bandwidth requirements if *bw_miss_num* > 0 in R_{sum_i}).

(4) *fail_case_sum*:
The number of cases which fail to meet real-time or bandwidth requirements among all 100 cases, (R_{sum_i} is a failed case if *rt_vio_time* > 0 **or** *bw_miss_num* > 0 in R_{sum_i}).

The experimental results are shown in Table 5. We can see that it is harder to meet the requirements with larger R_{sum}. The value of *fail_case_sum* decreases as R_{sum} goes low. The summary of experimental results is shown in Table 6. RT_lottery can not only meet real-time requirements but also be good at bandwidth allocation for the masters.

4.3 Experiment 2

The objective of experiment 2 is to observe the impact of different burst beat numbers on the arbitration algorithms. The traffic patterns are given that all masters send the same beat numbers of 8, 16, and 32, respectively. Similar to the experiment 1, we run 100 random cases for each R_{sum}.

The experimental results are shown in Fig. 9. RT_lottery is the best among the four algorithms for fixed 8, 16, and 32-beat. RT_lottery and TDM+Lottery, which are capable of handling both bandwidth and real-time requirements, perform much better than the other two algorithms. Nevertheless, it is harder to meet requirements with larger fixed beat number for RT_lottery and TDM+Lottery, since the numbers of failed cases arise with larger beat numbers. The reason is that with larger beat number, the granularities of weight (ticket number) for RT_lottery and TDM+Lottery get coarser. Each time a fixed amount of weight is transferred from M_i to M_j, the influence of weight transfer on cases of 8 or 16 fixed beat number is smaller than that on the case of 32 fixed beat number.

5. Conclusions

The two-level arbitration algorithm, RT_lottery, is proposed in this paper. We use high abstract-level models and a fast simulation-based evaluation environment to generate appropriate parameters for RT_lottery. RT_lottery is guaranteed to meet all hard real-time requirements and perform very well in bandwidth allocation. Three existing arbitration algorithms, Static Priority, Lottery, and TDM+Lottery are compared with RT_lottery. The experimental results show that RT_lottery is the best among these four algorithms in the ability to handle real-time and bandwidth requirements.

Hence, the RT_lottery-based arbiter can be a better choice for those SoC systems containing masters with hard real-time and diverse bandwidth requirements.

Table 2. The traffic pattern for the experiment 1

Heavy traffic | Light traffic

	type	R_{cycles}	beat/prob.		interval/prob.				
M1	D		8/50	16/50	6/10	7/20	8/40	9/20	10/10
M2	D		1/50	4/50	10/10	11/20	12/40	13/20	14/10
M3	D_R	65	8/50	16/50	6/10	7/20	8/40	9/20	10/10
M4	D_R	85	1/50	4/50	10/10	11/20	12/40	13/20	14/10
M5	ND_R	65	8/50	16/50	65/10	66/20	67/40	68/20	69/10
M6	ND_R	85	1/50	4/50	85/10	86/20	87/40	88/20	89/10

Table 3. An example bandwidth requirement

	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

Table 4. The experimental results of the experiment 1

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

Table 5. The results 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

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

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

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_case_sum*
bw_f : *bw_fail_case_sum* *fail* : *fail_case_sum*

Table 6. The summery of the experimental results

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 Fixed Priority	No consideration	Poor

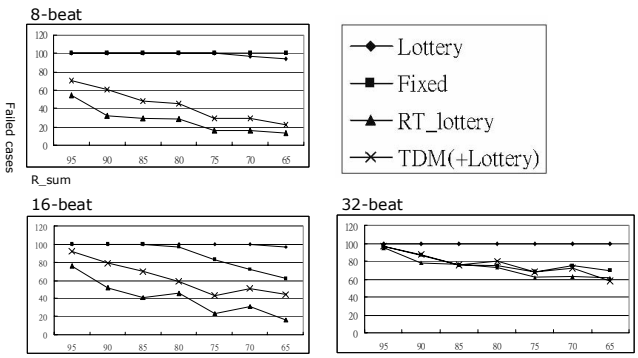


Fig. 9. Number of failed cases for 100 random cases

References

[1] 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.

[2] 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.

[3] 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, Page(s):618 – 621.

[4] 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.

[5] K. Lahiri, A. Raghunathan, and G. Lakshiminarayan, "LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs," *Design Automation Conference*, 2001, Page(s):15 – 20.

[6] A. C. Waldspurger and W. E. Weih., "Lottery Scheduling: Flexible Proportional Share Resource Management," Symp. on *Operating Systems Design and Implementation*, 1994.

[7] 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.