A Precise Bandwidth Control Arbitration Algorithm for Hard Real-Time SoC Buses

Bu-Ching Lin, Geeng-Wei Lee, Juinn-Dar Huang, Jing-Yang Jou
Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan
Outline

- Introduction
- Previous Works
  - fixed priority
  - time division multiple access (TDMA)
  - Lottery
  - RT_lottery
- Proposed Arbitration Architecture
- Experimental Results
- Conclusions
Introduction (1/2)

- Shared bus is widely used in current SoC designs
  - Master - initiate communication transactions
  - Slave - respond to transactions initiated by masters
  - Arbiter - manage the usage of bus
Introduction (2/2)

- Requirements in different applications
  - complete transactions of all requests before the corresponding deadlines in real-time applications
  - take at least a fixed fraction of total bandwidth in multimedia applications

- Difficult to satisfy both real-time and bandwidth requirements simultaneously
  - an innovative arbitration algorithm is required
Previous Works

- Existing arbitration algorithms
  - fixed priority
  - time division multiple access (TDMA)
  - Lottery
  - RT_lottery
Fixed Priority

- Among the requesting masters, the one with the highest priority gets granted

- Pros
  - simple, low hardware cost and easy to implement

- Cons
  - starvation problem – the masters with lower priority hardly get the service
  - lack of control over real-time and bandwidth requirements
TDMA (1/2)

- Execution time is divided into time slots which are **statically** assigned to masters.
- 2\(^{nd}\)-level of arbitration is usually adopted to alleviate the wasted time slots.
TDMA (2/2)

Pros

- **deterministic** worst-case response latency
- **reserved** bandwidth for each master

Cons

- difficult to design time slot sequences in an **unpredictable** system
- more slots ➔ more bandwidth and shorter latency
  - what if a master with **LOW** bandwidth requirement but needs **SHORT** response latency?
Arbiter grants a master *stochastically* from contending requests

- i.e., a weighted random mechanism

<table>
<thead>
<tr>
<th>Tickets (weight)</th>
<th>Request Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>Y</td>
</tr>
</tbody>
</table>

- 1 + 3 + 4 = 8

Random selection: Rand[0, 8) = 5

- T[0] = M1
- T[7] = M4
- T[8] = XX
- T[9] = XX

Grant: M4
Why Lottery is not suitable in SoC?

- **Similar** requesting rate for each master is assumed
- **What if the requesting rates are not similar?**
  - e.g., 3 masters with the same tickets assignment, one with the requesting rate varies from 80% to 40%, the rate of the other 2 is fixed at 40%

  - Requesting Rate of M1
    - 80% | 70% | 60% | 50% | 40%
    - M1 | 49% | 46% | 42% | 38% | 33%
    - M2 | 24% | 26% | 28% | 30% | 33%
    - M3 | 24% | 26% | 28% | 30% | 33%

  - **ticket ratio ≠ bandwidth ratio**
  - **weight tuning** is required
Latency and Bandwidth in Lottery

- Response latency and bandwidth allocation both controlled by the number of tickets.

- e.g., 3 masters have similar traffic behaviors.

More tickets ⇒ shorter response latency
⇒ more bandwidth allocation.
Summary of Lottery

Pros
- good control over bandwidth allocation in network switching applications
- fair average response latency

Cons
- no hard real-time consideration
- no independent controllability over response latency and bandwidth allocation
RT_lottery

- A 2-level arbitration algorithm dealing with real-time and bandwidth requirements simultaneously
- The proposed architecture
  - 1st level – real-time handler
    - handles the hard real-time requirements
  - 2nd level – Lottery with tuned weight
    - reserves the bandwidth allocation for each master
Real-Time Handler

- Similar to earliest deadline first scheduling (EDF)
  - the request with earliest deadline and below the warning line gets granted

- Deadline
  - the time limit for a master to complete a request
  - missing the deadline is regarded as the real-time violation

- Warning line
  - the worst case of scheduling the contending requests
Weight Tuning

- A ticket redistribution mechanism to meet the required bandwidth by simulation

- Initially, $T_d = T_m / 2$

- If any master in $S_{more}$ or $S_{meet}$ -> $S_{less}$
  
  \[ T_d = T_d / 2 \]

\[ T_m' = T_m - T_d \]

\[ T_l' = T_l + T_d \]
Fail Case of Weight Tuning

- Fail to meet bandwidth requirements due to diverse masters’ requesting rate
  - e.g., the requesting rate of each master is 80%, 30%, 30%, respectively and each of them requires at least 30% bandwidth

<table>
<thead>
<tr>
<th>Ticket Assignment of M1, M2 and M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
</tr>
<tr>
<td>M2</td>
</tr>
<tr>
<td>M3</td>
</tr>
</tbody>
</table>

⇒ weight tuning is not a panacea!
## Summary of Previous Works

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed priority</td>
<td>simplicity, area efficiency</td>
<td>no real-time consideration, no means for bandwidth control</td>
</tr>
<tr>
<td>TDMA</td>
<td>deterministic worst-case latency, reserved bandwidth allocation</td>
<td>no hard real-time guarantee, no precise bandwidth control</td>
</tr>
<tr>
<td>Lottery</td>
<td>reserved bandwidth allocation, fair average latency</td>
<td>no real-time consideration, no precise bandwidth control</td>
</tr>
<tr>
<td>RT_lottery</td>
<td>hard real-time guarantee</td>
<td>limitation of Weight Tuning</td>
</tr>
</tbody>
</table>
RB_lottery Architecture

- 3-level arbitration algorithm
  - real-time handler - handles the hard real-time requirements
  - Lottery with tuned weight - reserves the bandwidth allocation for each master
  - bandwidth regulator - provides fine-grained control over bandwidth allocation
An Example of RB_lottery

- Bandwidth regulator monitors the bus traffic
  - record the transactions of each master
  - temporarily block the requests from masters that have already got the required bandwidth in a period

```
<table>
<thead>
<tr>
<th>Tickets (weight)</th>
<th>Request map</th>
<th>Block signals</th>
<th>Grant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M1</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>M2</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>M3</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M4</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
```

```
T[ 0 ] = M1  
T[ 1 ] = M3  
T[ 4 ] = XX  
T[ 5 ] = XX  
T[ 6 ] = XX  
T[ 7 ] = XX  
T[ 8 ] = XX  
T[ 9 ] = XX  

Rand[ 0, 4 ] = 2

Grant M3
```
The Implementation

- Observation window ($w$) — the execution time is divided into windows of size $w$ cycles
  - block the requests of over-served masters temporarily

- Bandwidth register — the allocated bandwidth in the current window

required bandwidth = 30%

allocated bandwidth

30%  30%  30%  30%

execution time
A Precise Bandwidth Control Arbitration Algorithm for Hard Real-Time SoC Buses

**RB_lottery Algorithm Flow**

1. check whether a new window starts
2. grant the most urgent master
3. stochastically grant an un-blocked master
4. record the transaction cycles
5. check the allocated bandwidth
6. reset all the blocked signals and proceed to the next window
Experimental Environment

- Transaction level model in SystemC

- different arbitration algorithms are used in the experiments, such as fixed priority, lottery, TDMA + Lottery, RT_lottery, RB_lottery
3 Types of Masters

- **D type (D for Dependency)**
  
  Example:
  
  \[
  \text{beat} = 4, \quad \text{interval} = 15, \quad R_{cycles} = 10
  \]

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

- **ND_R type (ND for No Dependency, R for Real-time)**
## Experiment Setup

### Behavior of masters

<table>
<thead>
<tr>
<th></th>
<th>type</th>
<th>beat/prob.</th>
<th>interval/prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master 1</td>
<td>D</td>
<td>8/50</td>
<td>7/20</td>
</tr>
<tr>
<td>Master 2</td>
<td>D</td>
<td>1/50</td>
<td>10/10</td>
</tr>
<tr>
<td>Master 3</td>
<td>D</td>
<td>8/50</td>
<td>7/20</td>
</tr>
<tr>
<td>Master 4</td>
<td>D</td>
<td>1/50</td>
<td>10/10</td>
</tr>
<tr>
<td>Master 5</td>
<td>D_R</td>
<td>8/50</td>
<td>10/10</td>
</tr>
<tr>
<td>Master 6</td>
<td>D_R</td>
<td>1/50</td>
<td>10/10</td>
</tr>
<tr>
<td>Master 7</td>
<td>ND_R</td>
<td>8/50</td>
<td>65/10</td>
</tr>
<tr>
<td>Master 8</td>
<td>ND_R</td>
<td>1/50</td>
<td>85/10</td>
</tr>
</tbody>
</table>

- **Heavy-Traffic**
- **Light-Traffic**

- 4 D type masters, 2 D_R type masters and 2 ND_R type masters in the simulation system
- Half of masters are heavy-traffic
Performance Comparisons (1/2)

- Fail cases of different arbitration algorithms
  - 100 random required-bandwidth combinations for each workload
  - 102400 simulation cycles for each combination

<table>
<thead>
<tr>
<th>Workload (%)</th>
<th>Fixed Priority</th>
<th>Lottery</th>
<th>TDMA + Lottery</th>
<th>RT _lottery</th>
<th>RB _lottery</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>100</td>
<td>100</td>
<td>95</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>65</td>
<td>100</td>
<td>100</td>
<td>98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>70</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>75</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>85</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>37</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>55</td>
<td>12</td>
</tr>
<tr>
<td>95</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>74</td>
<td>44</td>
</tr>
</tbody>
</table>
Performance Comparisons (2/2)

- **Hardware comparisons**

<table>
<thead>
<tr>
<th></th>
<th>Fixed Priority</th>
<th>Lottery</th>
<th>TDMA + Lottery</th>
<th>RT_lottery</th>
<th>RB_lottery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gate counts</td>
<td>215</td>
<td>4296</td>
<td>4917</td>
<td>5134</td>
<td>5814</td>
</tr>
</tbody>
</table>

- **Summary**

<table>
<thead>
<tr>
<th></th>
<th>real-time capability</th>
<th>bandwidth capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed priority</td>
<td>no consideration</td>
<td>poor</td>
</tr>
<tr>
<td>Lottery</td>
<td>no consideration</td>
<td>good but weight tuning is required</td>
</tr>
<tr>
<td>TDMA+Lottery</td>
<td>no guarantee</td>
<td>good only in low loaded bus (workload &lt; 60%)</td>
</tr>
<tr>
<td>RT_lottery</td>
<td>always hold</td>
<td>good but still fails in high loaded bus (workload &lt; 75%)</td>
</tr>
<tr>
<td>RB_lottery</td>
<td>always hold</td>
<td>good even in extremely high loaded bus</td>
</tr>
</tbody>
</table>
Observation Window Comparisons

- Fail cases in different size of observation window of RB_lottery
  - 100 random required-bandwidth combinations for each workload
  - 102400 simulation cycles for each combination
  - the size of observation window ranges from 128 to 2048

<table>
<thead>
<tr>
<th>Workload (%)</th>
<th>The size of observation window</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>128</td>
</tr>
<tr>
<td>85</td>
<td>4</td>
</tr>
<tr>
<td>87</td>
<td>11</td>
</tr>
<tr>
<td>89</td>
<td>25</td>
</tr>
<tr>
<td>91</td>
<td>37</td>
</tr>
<tr>
<td>93</td>
<td>42</td>
</tr>
<tr>
<td>95</td>
<td>57</td>
</tr>
</tbody>
</table>
Conclusions

- RB_lottery is proposed to provide
  - hard real-time guarantee
  - fine-grained bandwidth control

- The observation window in the bandwidth regulator
  - the larger size of observation window, the better controllability over bandwidth requirements
Thank you!