**Implementing Low-latency Fair Switch Arbiter**

This project implements the Fair Switch Arbiter (FSA)[1] which is a variant of round-robin arbiter. It enforces absolute fairness compared to switch arbiter (SA)[2].

## 1. Background

### 1-1. Basic round-robin arbiter:

In a basic round-robin arbiter, the critical path is:
At each bit position:
Check "Is this request active and currently highest priority?"
If yes, grant immediately.
If not, check the next position.
This chain of checks forms a sequential critical path.
And the Critical Path Delay (CPD) is
$O(N \times t\_gate)$

### 1-2. Tree-based round-robin arbiter:

In this structure, Requestors are organized hierarchically in a balanced k-ary tree. Each internal node performs local arbitration among its k children and local winners bubble up the tree.
Since each node only need to check k inputs sequentially, the critical path becomes:
$O(\log\_k(N) \times k \times t\_gate)$

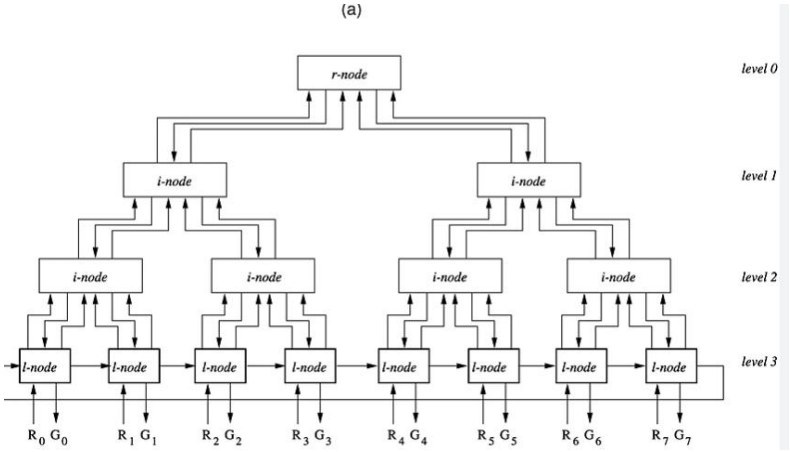


*Figure 1: Tree-based Round-robin arbiter[3].*

### 1-3. Drawbacks of SA:

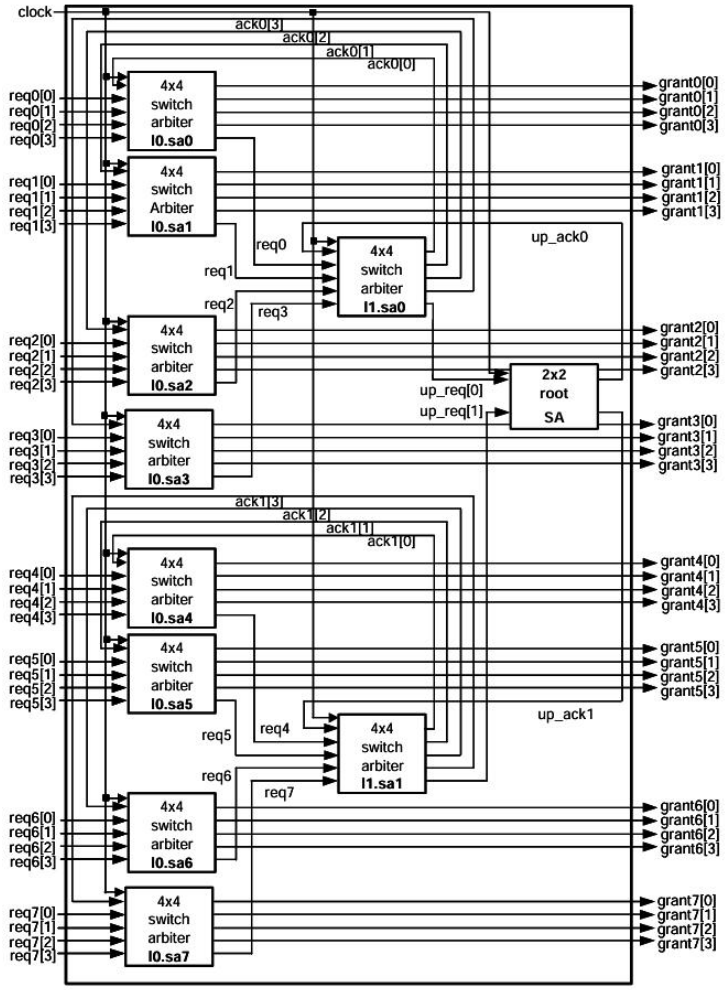


*Figure 2: Switch Arbiter*

In the original SA, Each leaf node will always forward request if any of its children request is active. And Each upper node (internal node and root node) will do round-robin arbitration as well.
Under a non-uniform request stream, SA may become unfair shown in Figure 3:

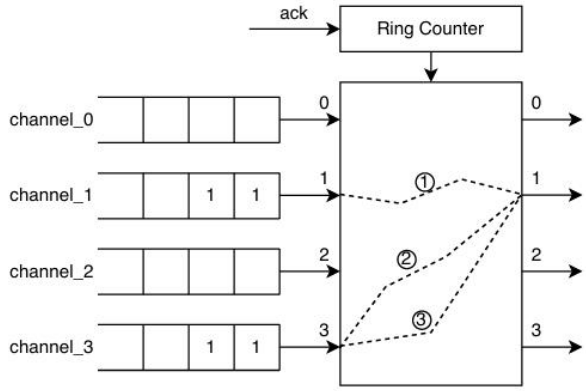


*Figure 3: Example of Unfairness of Switch Arbiter*

Assume all 4 channels request access to output port 1 and the initial priority order is 1>2>3>0
Round 1: port 1 gains access as it has the highest priority. After grant priority becomes 2>3>0>1

Round 2: port 2 gains access but since port 2 didn't request, the next priority, port 3 gains access. After grant priority becomes 3>0>1>2.
Round 3: port 3 gains access as it has the highest priority.
Unfairness arises when it arbitrates in the order of 1,3,3. We expects the order to be 1,3,1,3 for a round-robin behavior when only 1 and 3 request.

## 2. FSA

The FSA exhibits a similar structure as SA. But with several differences.

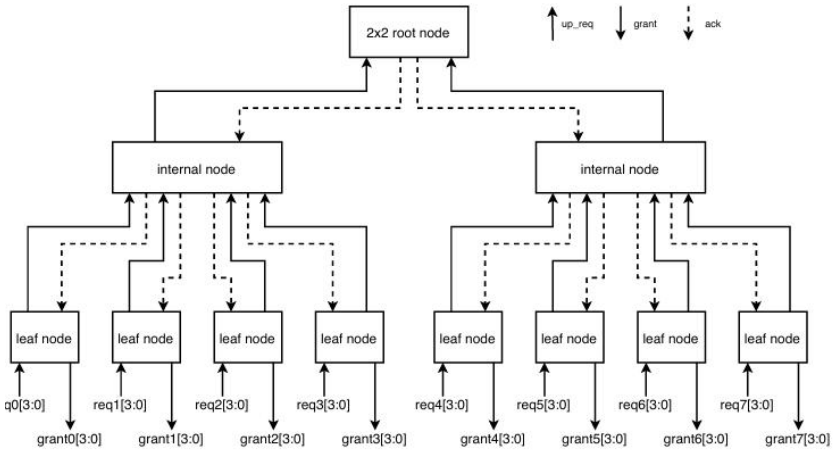


*Figure 4: FSA Block Diagram*

### 2-1. Different priority logic

As shown in Figure 5, after a request is granted in each arbitration cycle, the state jumps such that the highest priority is passed to the next request (Not the next port) and the priority of the granted request is adjusted to the lowest point.

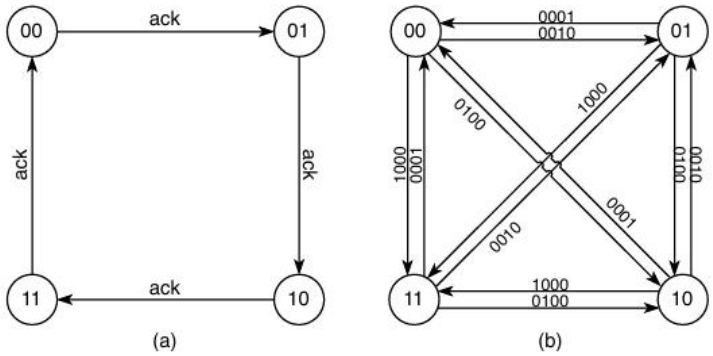


*Figure 5: Priority State Transition in Switch Arbiter(a) and Fair Switch Arbiter(b)*

Consider the example in Figure 3, In SA, after Round 1, the highest priority is always passed to port 2 regardless of whether port 2 requests. But in FSA, the highest priority is passed to port 2 only if port 2 requests. So port 3 will win Round 2, then the port that will win Round 3 will be the first port that requests in the order of 0>1>2>3 which will be 1.
Now the arbitration is fair following the expected order: 1,3,1,3.

### 2-2. Use of Lock

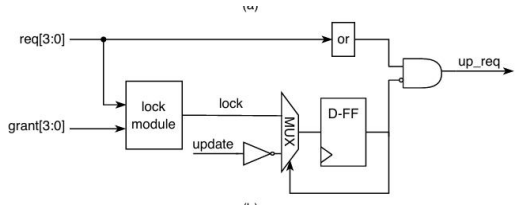


*Figure 6: Lock Logic of FSA*

A lock is used to guard the upward request. A leaf node will only forward request when it is not "locked"

Once all inputs in this leaf node have been serviced, it locks itself from requesting to higher nodes. The nodes "withdraw" itself from competition for this global round.

Because leaf nodes will automatically withdraw when it should not be granted, all upper nodes(internal nodes and root nodes) no longer need to do round-robin arbitration. It can simply do a fixed order (priority) arbitration. Hence we save area and time as fixed priority arbiter is simpler than round-robin arbiter.

**2-3. Behavior Analysis**

(1) Leaf node 0 gains grant and makes local arbitration

(2) Lock in leaf node 0 asserted after making 4 grants, leaf node 0 withdraws and does not request

(3) grant from upper nodes proceeds to leaf node 1.

(4) Leaf node 1 start its arbitration cycle, and so on...

(5) At the end, all leaf nodes withdraw, the root node sees no request (all leaf nodes stop requesting), it asserts update, all leaf nodes are "unlocked": a new global round starts.

## 3. Simulation Result

**1. Local behavior (In leaf node [0])**

Lock is asserted after making 4 grants.

leaf node 0 withdraws and does not request, 1111_111**1** -> 1111_111**0**

grant from upper nodes proceeds to leaf node 1. 0000_00**01** -> 0000_00**10**

The simulation meets the expected behavior discussed in section 2-3.

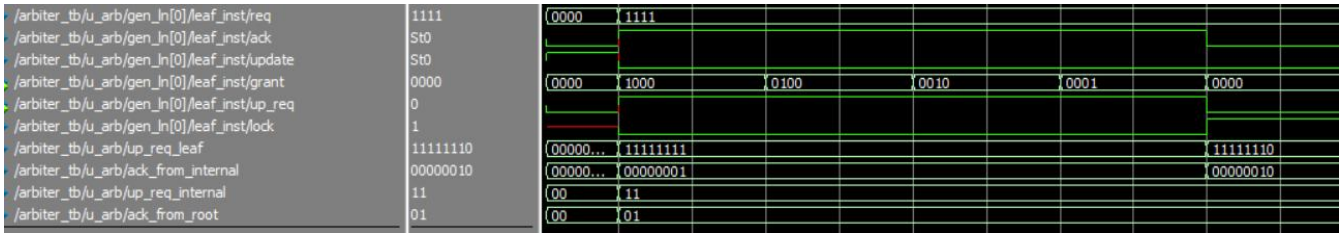


*Figure 2-1: Simulation of one leaf node*

**2. Full simulation**

We use Jain's Index as the metric for fairness of an arbiter to test the correctness of our implementation. (To make sure we've implemented a fair round robin arbiter)

The simulation shows that fairness is achieved for a total of 10000 cycles. (Since requests are randomly generated, it does not ensure an absolute even distribution, so the Jain's Index is not 1, but

should be very close to 1)

```
Starting uniform request test
 Jain's index = 0.9978
 Grant counts:
  client 0: 297
  client 1: 293
  client 2: 279
  client 3: 291
  client 4: 287
  client 5: 306
  client 6: 288
  client 7: 314
  client 8: 299
  client 9: 288
  client 10: 305
  client 11: 311
  client 12: 305
  client 13: 300
  client 14: 295
  client 15: 296
  client 16: 298
  client 17: 306
  client 18: 292
  client 19: 302
  client 20: 291
  client 21: 304
  client 22: 318
  client 23: 288
  client 24: 288
  client 25: 291
  client 26: 284
  client 27: 281
  client 28: 253
  client 29: 267
  client 30: 282
  client 31: 264
```

*Figure 2-2: Simulation result of uniformly random request stream*

If we let all input ports keep requesting, and let the total number of cycles be a multiple of 32, then we can see that all ports evenly get granted, leading to a Jain's index of 1.

```
Starting uniform request test
 Jain's index = 1.0000
 Grant counts:
  client 0: 31
  client 1: 31
  client 2: 31
  client 3: 31
  client 4: 31
  client 5: 31
  client 6: 31
  client 7: 31
  client 8: 31
  client 9: 31
  client 10: 31
  client 11: 31
  client 12: 31
  client 13: 31
  client 14: 31
  client 15: 31
  client 16: 31
  client 17: 31
  client 18: 31
  client 19: 31
  client 20: 31
  client 21: 31
  client 22: 31
  client 23: 31
  client 24: 31
  client 25: 31
  client 26: 31
  client 27: 31
  client 28: 31
  client 29: 31
  client 30: 31
  client 31: 31
All tests completed
```

*Figure 2-3: Simulation result of continuous request stream*

**References**

[1]J. Luo, W. Wu, Q. Xing, M. Xue, F. Yu, and Z. Ma, “A Low-Latency Fair-Arbiter Architecture for Network-on-Chip Switches,” Applied Sciences, vol. 12, no. 23, pp. 12458–12458, Dec. 2022, doi: https://doi.org/10.3390/app122312458.

[2]E. S. Shin, V. J. Mooney, and G. F. Riley, “Round-robin arbiter design and generation,” SMARTech Repository (Georgia Institute of Technology), Jan. 2002, doi: https://doi.org/10.1145/581250.581253.

[3]M. Yang and S. Q. Zheng, "Algorithm-Hardware Codesign of Fast Parallel Round-Robin Arbiters" in IEEE Transactions on Parallel & Distributed Systems, vol. 18, no. 01, pp. 84-95, January 2007, doi: 10.1109/TPDS.2007.3.