# Micro-Architecture Specification (MAS): Weighted Round Robin (WRR) Arbiter

## Table of Contents



## 1. Document Control

### 1.1 Title / ID

**Title**: Weighted Round Robin Arbiter Micro-Architecture Specification

**Document ID**: WRR-MAS-001

### 1.2 Version / Revision

- **Version**: v1.2
- **Date**: 2025-04-22

### 1.3 Owner / Authors

- **Primary Owner**: MooresLabAI RTL Design Team
- **Contributors**: Micro-Architecture Group

### 1.4 Approvals

- Design Lead: *Pending*
- Verification Lead: *Pending*
- System Architect: *Pending*

### 1.5 Distribution / Confidentiality

- Internal Use Only
- Confidential and proprietary — Not for external distribution

## 2. Introduction

### 2.1 Purpose and Scope

This document provides a complete specification of the internal logic and operation of the Weighted Round Robin (WRR) arbitration IP. It defines the architectural components, control paths, data paths, parameterization, and runtime behavior necessary to implement and verify the IP.

The WRR arbiter provides fair, weighted access arbitration across multiple requestors. It is suitable for use in Network-on-Chip (NoC) routers, memory controllers, shared buses, and other interconnect scenarios where bandwidth differentiation and fairness are required.

### 2.2 References

- IEEE JSAC Paper: "Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip"

### 2.3 Document Organization

- Sections 1–3 provide context and motivation.
- Sections 4–16 detail the functional behavior and structural design.
- Sections 17–21 offer test scenarios, implementation notes, and historical changes.
- Section 22 provides visualization diagrams using PlantUML.

## 3. Contents Overview

### Design Goals

- **Fairness**: Guarantee all requestors receive proportional bandwidth access
- **Flexibility**: Support runtime updates of requestor priorities
- **Efficiency**: Ensure minimal arbitration latency (1 cycle typical)
- **Scalability**: Enable support for 4 to 64 requestors without architectural changes

### Key Features

- Weighted round-robin scheduling algorithm
- Priority (weight) update interface
- Round-robin pointer for fairness
- 8-bit per-requestor credit counters to enforce weight-based scheduling
- One-hot and encoded ID grant outputs
- Arbitration result within 1 cycle under typical conditions
- Refill credit logic for zero-starvation guarantee

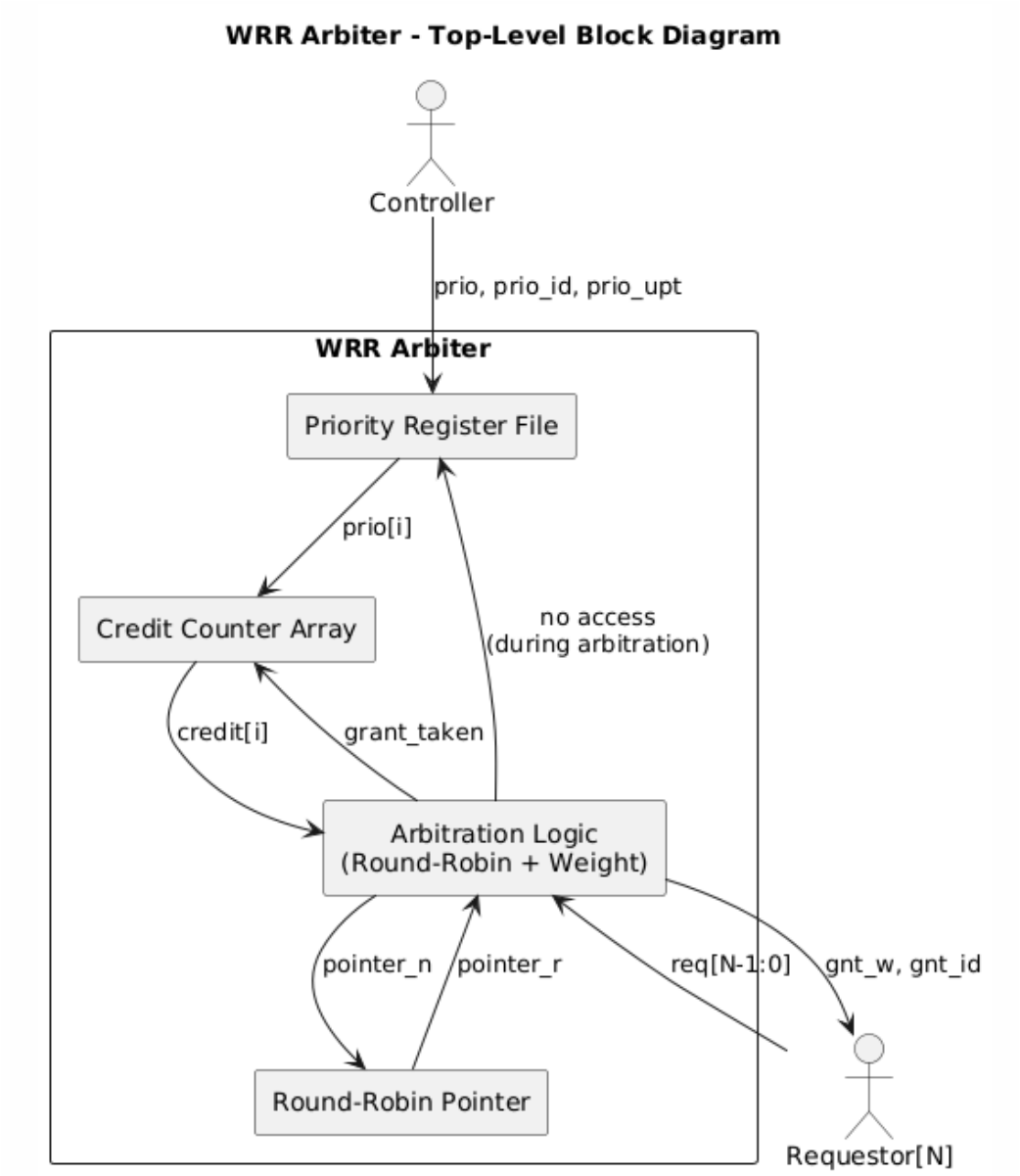
## 4. High-Level IP Overview

### 4.1 Key Features

- **Weight-Based Arbitration**: Each requestor has a priority/weight influencing how frequently it is granted access.

- **Runtime Priority Update**: Priorities can be updated on-the-fly without stalling arbitration.
- **Credit-Based Scheduling**: Credits are decremented upon grant and refilled when exhausted.
- **Zero-Starvation**: The round-robin pointer ensures all requestors will eventually be granted.
- **Combinational Arbitration**: Arbitration is purely combinational for low-latency scheduling.

## 4.2 Top-Level Block Diagram




## 4.3 Module / File Hierarchy

- wrr_arbiter.sv: Top-level arbiter and interface wrapper
- wrr_priority_mgr.sv: Handles the priority register updates
- wrr_credit_logic.sv: Tracks credit counters and performs replenishment

- wrr_arbitration.sv: Combinational arbiter with round-robin rotation

### 4.4 Design Constraints

- Must operate within a single synchronous domain
- Should maintain cycle-accurate grant latency
- Should not exceed critical path delay for target frequency (e.g., < 1 ns for 1 GHz)

## 5. IP Input/Output Ports

This section describes each input and output port, including signal width, function, and timing characteristics.

| Signal | Dir | Width | Description |
|---|---|---|---|
| clk | in | 1 | Clock signal. Rising edge driven synchronous design. |
| rst | in | 1 | Active-high synchronous reset. |
| req | in | N | Requestor request lines. One bit per requestor. |
| ack | in | 1 | Acknowledgment that current grant was accepted. |
| prio | in | PRIORITY_W | New priority value to be written to prio_reg. |
| prio_id | in | ID_BITS | Address of the requestor to which new priority is set. |
| prio_upt | in | 1 | Priority update strobe signal. |
| gnt_w | out | N | One-hot vector indicating which requestor is granted. |
| gnt_id | out | ID_BITS | Encoded ID of the requestor granted this cycle. |

### Timing Description:

- Inputs are sampled on the rising edge of clk.
- Outputs are valid during the same cycle as their corresponding inputs.
- Arbitration occurs every cycle. Priority updates and ack can be asserted simultaneously without functional hazard.

# 6. Internal Sub-Block Descriptions

## 6.1 Priority Register File

**Function**:

- Stores a 4-bit weight (priority) per requestor.
- Controls how often a requestor can be granted relative to others.
- Affects credit replenishment.

**Implementation**:

- Implemented as a register array: logic [PRIORITY_W-1:0] prio_reg[N-1:0]

**Parameterization**:

- Width: PRIORITY_W (typically 4)
- Depth: N (default 32)

**Update**:

- Triggered by prio_upt
- Updated entry is determined by prio_id
- Simultaneous prio_upt and ack is allowed

## 6.2 Credit Manager

**Function**:

- Manages internal credit counters for each requestor.
- Each grant reduces the corresponding credit by 1.
- When no requestor is eligible, all credits are replenished to prio + 1.

**Implementation**:

- Implemented as a register array: logic [CREDIT_W-1:0] credit_r[N-1:0]

**Parameterization**:

- Width: CREDIT_W (typically 8)

- Depth: N (default 32)

**Behavior**:

- Credit decremented only for the granted requestor upon ack=1.
- Credits are replenished when no eligible requestor is found.

## 6.3 Arbiter Logic (Grant Generator)

**Function**:

- Determines which requestor gets access each cycle.
- Uses a rotating pointer and credit eligibility.

**Implementation**:

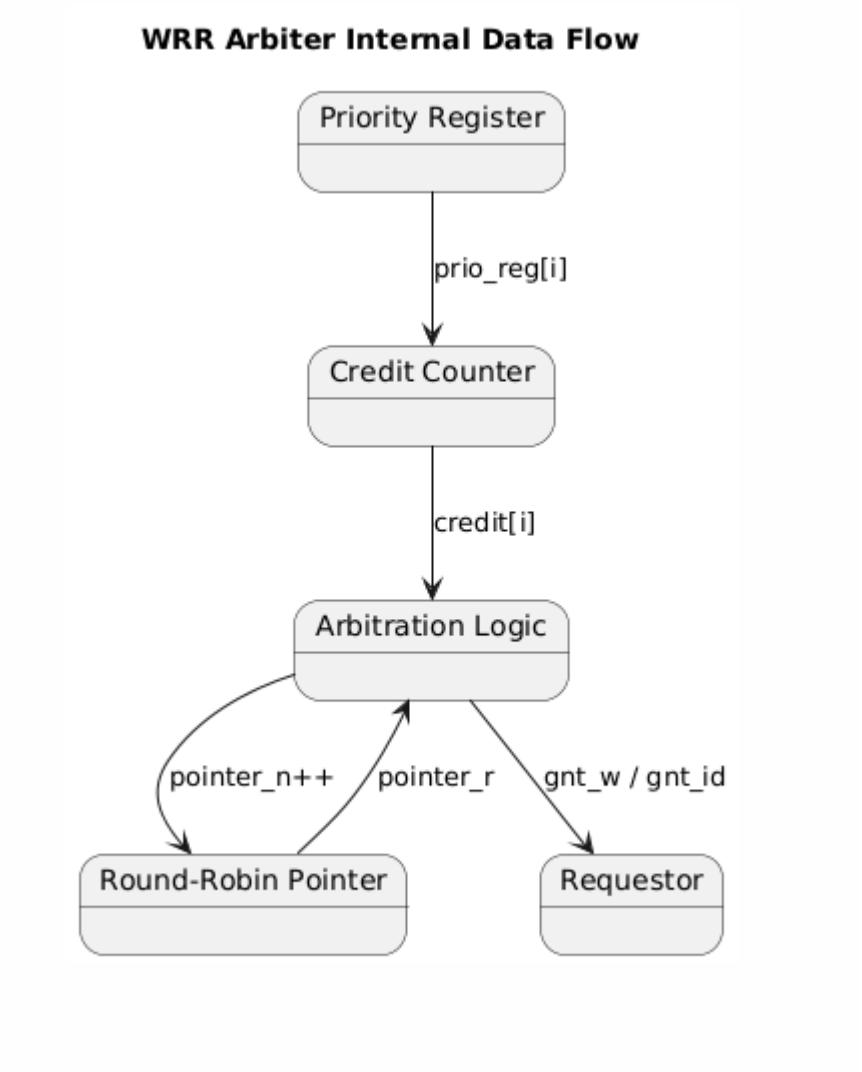
- Rotating pointer scanned for (req[i] == 1 && credit[i] > 0)
- If found, gnt_id = i and gnt_w[i] = 1
- Pointer moves to (i + 1) % N for next cycle

**FSM States**:

- ARBITRATION: Regular match and grant generation
- REPLENISH: Triggered when no eligible requestor

# 7. Data Path Details

## 7.1 Data Flow




1. Incoming req[N-1:0] sampled each cycle.
2. Combinational logic checks credit + request.
3. If match:
    a. Decrement credit, issue grant.
4. If no match:
    a. Refill credit with prio + 1, no grant.
5. Always increment pointer after decision.

### 7.2 Bit Widths & Encodings

| Signal | Width | Notes |
|---|---|---|
| req | N | One-hot per requestor |
| gnt_w | N | One-hot vector output |
| gnt_id | ID_BITS | Binary encoded ID |
| credit | CREDIT_W | 8-bit unsigned counter |
| priority | PRIORITY_W | 4-bit unsigned priority per requestor |

### 7.3 Buffering / Queues

- No FIFOs or queues are used.
- State held entirely in synchronous flip-flops.

## 8. Control Path Details

### 8.1 Status / Debug

The WRR arbiter provides visibility into its internal operational state. For debug and observability purposes, the following signals can be probed:

- credit_r[N-1:0]: Current credit value for each requestor
- prio_reg[N-1:0]: Priority register value for each requestor
- pointer_r: Current position of round-robin pointer
- gnt_w, gnt_id: Current grant decision

All these signals can optionally be exposed to a debug bus or system-level trace mechanism.

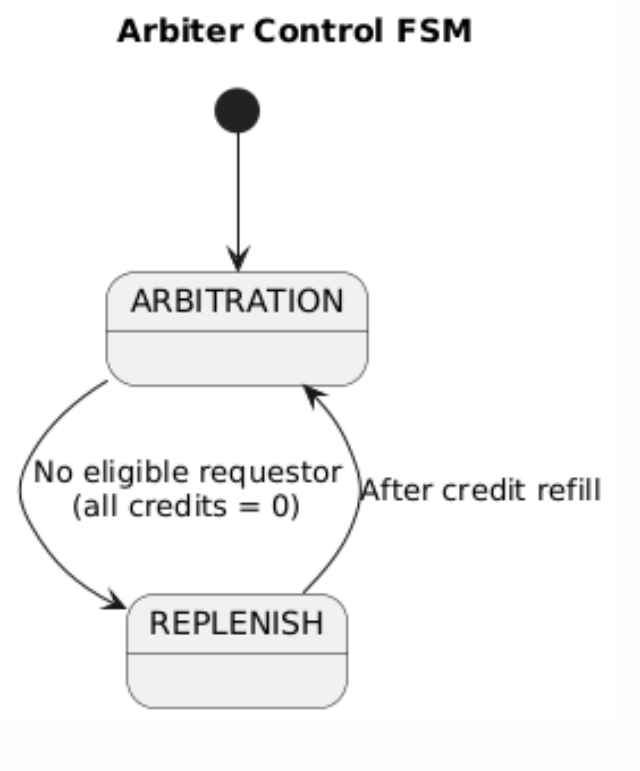
### 8.2 Global Control FSM

The WRR arbiter operates with a simple, two-mode FSM:

| State | Description |
|---|---|
| ARBITRATION | Checks req and credit arrays to find eligible requestor. |
| REPLENISH | Triggered when no requestor is eligible. Refills all credits from prio+1. |

**Transition Logic:**

- If at least one (req[i] && credit[i] > 0) is found → remain in ARBITRATION
- Else → enter REPLENISH
- FSM resets to ARBITRATION after refill




## 8.3 Error / Security Handling

The IP does not include explicit security features, ECC, or error reporting. Integrators are encouraged to wrap the module with any of the following (if required):

- Range checkers for prio_id

- Debug assertions for illegal values
- Wrapper-based boundary checks on configuration inputs

# 9. Pipeline & Timing

## 9.1 Pipeline Stage Breakdown

| Stage | Function |
|---|---|
| 0 | Combinational arbitration |
| 1 | Sequential update of credits, pointer |

The output grant signals (gnt_w, gnt_id) are produced in Stage 0.

## 9.2 Latency

- **Best-case latency**: 1 cycle from req to gnt
- **Worst-case latency**: 2 cycles if credit refill is needed

## 9.3 Hazards & Ordering

- No transaction ordering is required
- All requestors are treated independently
- Hazard-free credit update due to sequential flopping

## 9.4 Multi-Power or Multi-Voltage Domains

- Not supported
- Module is designed for single clock, single power domain operation

# 10. Clocking & Reset

## 10.1 Clock Domains

- Only a single clock domain is supported
- No crossing logic or asynchronous FIFOs

### 10.2 Reset Types

- Synchronous reset (rst) active high
- All internal state (credits, priorities, pointer) is reset synchronously

### 10.3 Reset / Startup Sequence

| Cycle | Signal | Value/Action |
|---|---|---|
| 0 | rst=1 | Priority registers cleared (0) |
| | | Credit registers initialized to 1 |
| | | Pointer reset to 0 |
| 1 | rst=0 | Ready to begin arbitration cycle |

## 11. Registers & Configuration

### 11.1 Configuration

- The WRR arbiter is configured entirely via sideband signals:
  - prio, prio_id, and prio_upt
- Priority updates can happen at any time and do not interfere with arbitration
- No memory-mapped configuration interface is present

### 11.2 Register Map

| Register | Description | Access Type | Width |
|---|---|---|---|
| N/A | Configuration via wires only | N/A | N/A |

For integration into register-based systems, a wrapper may be created externally.

# 12. Performance

## 12.1 Throughput / Bandwidth

- Capable of granting one request per cycle
- Arbitration is pipelined such that input is accepted every cycle

## 12.2 Latency

| Condition | Latency (cycles) |
|---|---|
| Normal operation | 1 |
| Credit refill | 2 |

## 12.3 Resource Utilization

| Component | Approx. Resources |
|---|---|
| Priority registers | N × 4 bits |
| Credit counters | N × 8 bits |
| Pointer register | log2(N) bits |

## 12.4 Scalability

- Works efficiently up to N = 64
- For higher N, hierarchical WRR or pipelined arbitration layers are recommended

# 13. Error Correction / Security / Safety

## 13.1 ECC or Data Integrity

The WRR arbiter IP does not implement any ECC or parity protection over its internal storage arrays, including:

- Priority registers (prio_reg)
- Credit counters (credit_r)

Because the IP uses only small, flip-flop based storage per requestor (e.g., 4-bit priority and 8-bit credit), the design assumes that soft error rates are negligible in target processes and applications. If used in environments with high radiation exposure (e.g., aerospace), wrapper-level ECC is recommended.

### 13.2 Security Model

This IP does not differentiate secure vs. non-secure domains, nor does it support encryption, authentication, or isolation mechanisms. All configuration and control interfaces are treated equally and lack access protection.

Integration suggestions for secure contexts:

- Route control interfaces through a trust zone or secure controller.
- Include access qualifiers in upper-level memory maps if exposed via register wrappers.

### 13.3 Functional Safety

The arbiter does not implement fault containment, fault detection, or recovery logic. It does not support ISO 26262 or ASIL compliance features internally. If used in safety-critical applications, the following must be handled externally:

- Fault detection and reporting
- Redundant path arbitration (dual lock-step modules)
- Diagnostic test sequences for pointer, credit, and priority logic

## 14. Advanced Debug & QoS

### 14.1 Debug / Trace

The WRR IP is optionally compatible with system debug strategies. Although it does not include built-in trace logic, designers may expose the following for debug:

- gnt_w: Active grant vector
- gnt_id: Active grant encoded ID

- pointer_r: Round-robin pointer position
- prio_reg[N]: Priority values per requestor
- credit_r[N]: Current credit per requestor

Optional counters (add-on):

- grant_count[N]: Per-requestor grant counts for QoS accounting

Debug outputs may be used to create live QoS dashboards or validate WRR fairness over simulation cycles.

### 14.2 QoS / Scheduling

This IP enforces a **weighted round robin** (WRR) policy where each requestor receives a number of grants proportional to its priority weight.

Scheduling strategy:

- Each requestor is credited with prio + 1 tokens when its credit reaches 0.
- Requestors consume 1 credit per grant.
- Round-robin pointer ensures even high-weight requestors don't monopolize bandwidth.

Fairness is built-in:

- Requestors with priority = 0 do not get replenished (unless explicitly updated).
- No starvation due to pointer advancement even during refill.

This model ensures proportional access across bursty and latency-sensitive traffic patterns.

## 15. Parameterization & Configurability

| Parameter | Description | Default |
|---|---|---|
| N | Number of requestors supported | 32 |
| PRIORITY_W | Width of priority register (bits) | 4 |

| | | |
|---|---|---|
| CREDIT_W | Width of credit counters | 8 |
| ID_BITS | Number of bits required to encode N requestors | 5 |

### Parameter Constraints

- N must be a power of 2 for pointer wraparound logic.
- CREDIT_W must be wide enough to handle expected max weights.
- PRIORITY_W ≥ 2 is recommended to allow more than binary weight assignment.

### Effects

| Parameter | Effect on Design |
|---|---|
| Higher N | Increases area and logic fanout |
| Wider PRIORITY_W | Enables finer-grained bandwidth control |
| Wider CREDIT_W | Allows higher sustained weights without overflow |

## 16. Error Handling & Debug Hooks

### 16.1 Assertions / Monitors

The following assertions are recommended for integration-level or formal verification:

- assert(!(prio_upt && prio_id >= N));

*Ensures no out-of-bounds writes to the priority register.*

- assert(req != 0 -> gnt_w != 0);

*Guarantees progress: if requestors are active, eventually a grant must be issued.*

- assert(gnt_w[gnt_id] == 1);

*Verifies that the encoded ID matches the one-hot grant.*

- assert(pointer_r < N);

*Ensures the pointer never exceeds the valid range.*

These assertions may be embedded in RTL testbenches or formal environments.

### 16.2 Error Reporting

There are no error output signals in the default IP. If required, integrators may:

- Add a prio_update_error output if prio_id >= N
- Add an illegal_ack output if grant is acknowledged while inactive

### 16.3 Integration with System Debug

For SoC-level debug, WRR state may be exposed via:

- JTAG-based register snapshots (e.g., dbg_gnt_id, dbg_pointer, etc.)
- AXI/APB sideband debug registers (with read-only priority/credit counters)
- Trace logs capturing gnt_id, req, and ack across time for bandwidth profiling

## 17. Example Scenarios & Waveforms

This section illustrates cycle-by-cycle arbitration and dynamic behavior under various request and configuration patterns.

### Scenario A — Basic Arbitration with Multiple Requests

**Initial Conditions**:

- Priorities: #0=4, #1=4, #2=4
- Credits: All initialized to 1
- Requests: Multiple active

| Cycle | clk | req | ack | prio_upt | prio_id | prio | gnt_w | gnt_id | Notes |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ↑ | 0110 | 0 | 0 | -- | -- | 0010 | 0x1 | #1 has credit and requests |
| 1 | ↑ | 0110 | 1 | 0 | -- | -- | 0100 | 0x2 | #1 acked, #2 gets grant |
| 2 | ↑ | 0110 | 1 | 0 | -- | -- | 0000 | 0x0 | No eligible req → refill |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | ↑ | 0110 | 0 | 0 | -- | -- | 0010 | 0x1 | Credit refilled, cycle restarts |

## Scenario B — Mid-Stream Priority Update

**Initial Conditions**:

- Priorities: #0=2, #1=2, #2=2

| Cycle | clk Edge | req | ack | prio_upt | prio_id | prio | gnt_w | gnt_id | Notes |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Rising | 0101 | 0 | 0 | -- | -- | 0001 | 0x0 | #0 granted first |
| 1 | Falling | 0101 | 0 | 1 | 2 | 8 | 0001 | 0x0 | Update #2's priority to 8 |
| 2 | Rising | 0101 | 1 | 0 | -- | -- | 0100 | 0x2 | #2 now has higher weight and wins next |

## Scenario C — Priority Update During ack

| Cycle | clk | req | ack | prio_upt | prio_id | prio | gnt_id | Notes |
|---|---|---|---|---|---|---|---|---|
| 0 | ↑ | 0011 | 1 | 1 | 1 | 6 | 0x0 | Valid priority update during ack |
| 1 | ↑ | 0011 | 0 | 0 | -- | -- | 0x1 | #1 is now higher weight, gets grant |

## Scenario D — Weighted Round Robin

**Priorities**:

- #0=2, #1=8, #2=1

**Behavior**: #1 gets more frequent grants due to higher weight.

| Cycle | clk | req | ack | gnt_w | gnt_id | Notes |
|---|---|---|---|---|---|---|
| 0 | ↑ | 111 | 0 | 0010 | 0x1 | #1 wins — highest weight |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | ↑ | 111 | 1 | 0001 | 0x0 | #0 gets next |
| 2 | ↑ | 111 | 1 | 0100 | 0x2 | #2 gets a turn |
| 3 | ↑ | 111 | 0 | 0010 | 0x1 | #1 back due to higher weight |

### Scenario E — Dynamic Weight Update Under Load

| Cycle | clk | req | ack | prio_id | prio | prio_upt | gnt_w | gnt_id | Notes |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ↑ | 111 | 0 | -- | -- | 0 | 0010 | 0x1 | Initial — #1 wins |
| 1 | ↑ | 111 | 1 | 0 | 8 | 1 | 0100 | 0x2 | #0 boosted to high weight |
| 2 | ↑ | 111 | 0 | 1 | 1 | 1 | 0001 | 0x0 | #1 deprioritized |
| 3 | ↑ | 111 | 1 | -- | -- | 0 | 0001 | 0x0 | #0 repeatedly wins now |

# 18. Implementation & RTL Notes

## 18.1 Coding Style & Synthesis

- RTL uses always_comb for next-state and arbitration logic
- All state (pointers, credits, prio_reg) is stored using always_ff @ (posedge clk)
- Lint-clean under standard tools

## 18.2 Power Optimization

- Idle state detection for clock gating can be added externally
- All logic is active as long as any req is active

## 18.3 DFT & Test

- No internal memory → no BIST required
- Fully scan-friendly
- Credit, prio, and pointer registers are DFT-visible

# 19. Limitations & Future Extensions

## Known Limitations

- Does not scale well beyond 128 requestors without tree or multi-stage WRR
- No support for burst mode grants or fixed-latency slots
- No built-in test/debug trace mechanisms

## Future Work

- Hierarchical WRR for large-scale arbitration
- Integration of optional APB interface for config/debug
- Trust-level separation for secure/unsecure request domains
- Programmable credit step (currently fixed as prio + 1)

# 20. Revision History

| Version | Date | Author(s) | Changes |
|---|---|---|---|
| v1.0 | 2025-03-25 | Architecture Team | Initial specification |
| v1.1 | 2025-04-11 | Simon | Fixed credit logic, pointer, ack+prio allowed |
| v1.2 | 2025-04-28 | Juliet | Fully verbose MAS with diagrams, inline tables |

# 21. Errata

- **None** as of v1.2