**VLSI Technology**

**X2X Synchronous Bridge**

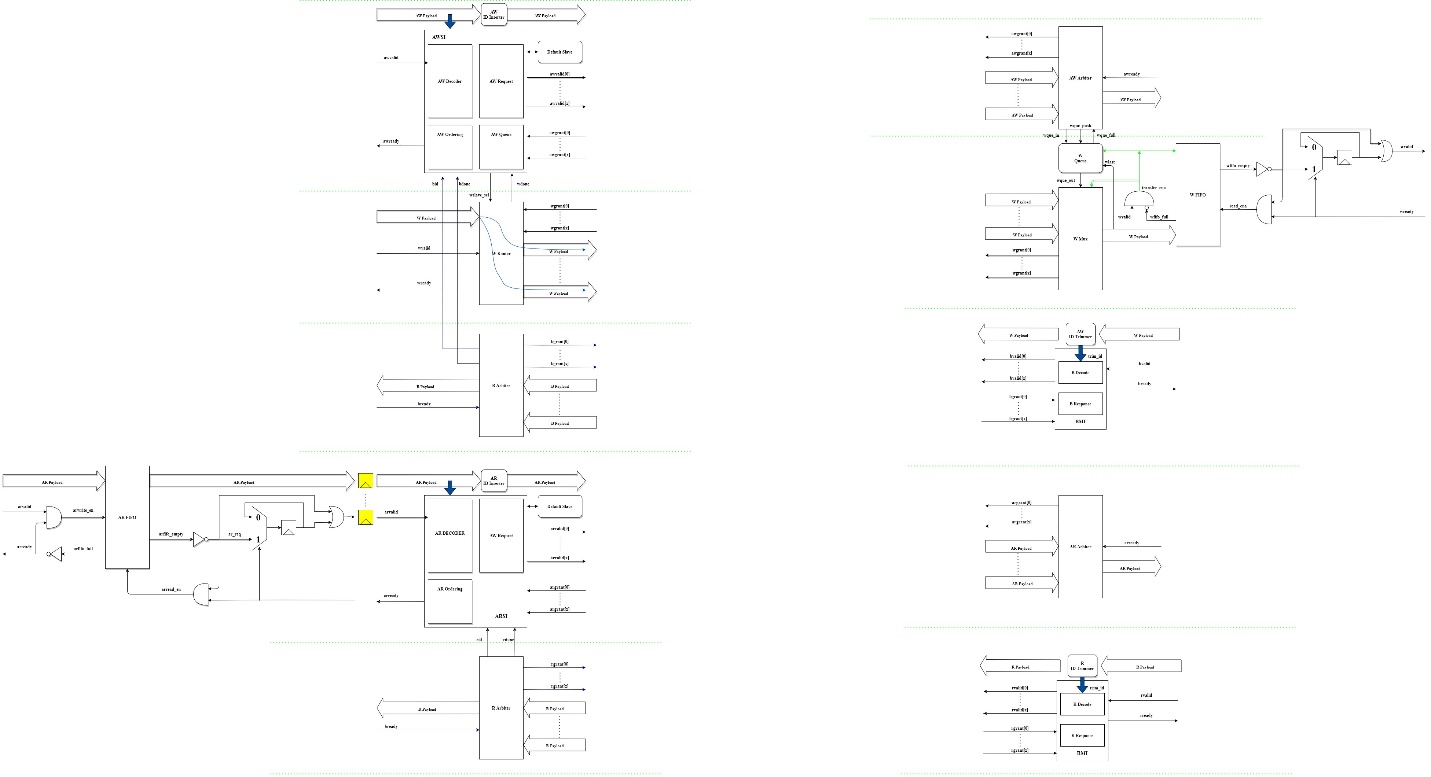
**Lê Quang Hưng**

**Nguyễn Hùng Quân**

**2020.12.17**

INDEPENDENT REPEATED REGISTERS DESIGN

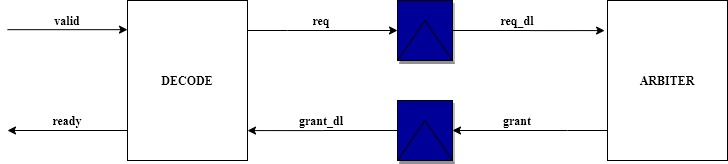
1. TỔNG QUAN



**Hình 1: Independent repeated registers design**

Vấn đề của thiết kế X2X trước

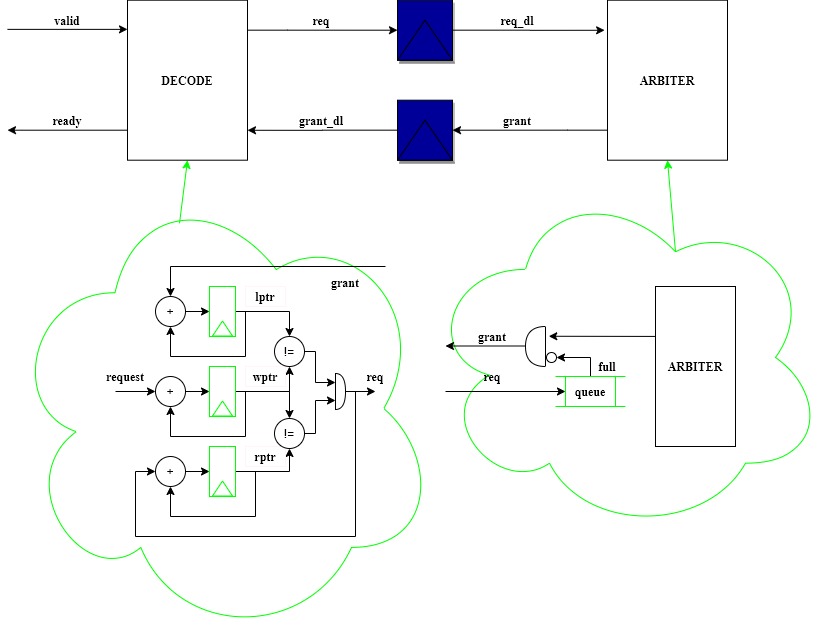
Với cách thiết kế trước đây, các ràng buộc về giữa bên gửi và bên nhận của X2X có thể gây ra các vấn đề timing. Khi Master và Slave nằm cách xa nhau, delay trên đường truyền yêu cầu phải chèn thêm các Repeated Registered (Pipeline) giữa các đường kết nối, tuy nhiên điều này lại gây giảm throughput của X2X



**Hình 2: X2X problems**

Xét trường hợp Master muốn gửi một chuỗi dữ liệu cho Slave tương ứng của nó. Sau khi gửi dữ liệu đầu tiên, Master không được phép truyền dữ liệu tiếp theo cho đến khi DECODE nhận được tín hiệu grant\_dl từ ARBITER. Thêm vào đó, tín hiệu req từ Master vừa được grant chỉ hợp lệ trở lại sau khi Master gửi dữ liệu tiếp theo. Do đó, X2X sẽ tiêu tốn ít nhất là 3 chu kỳ clock trước khi Master được phép gửi dữ liệu tiếp theo.

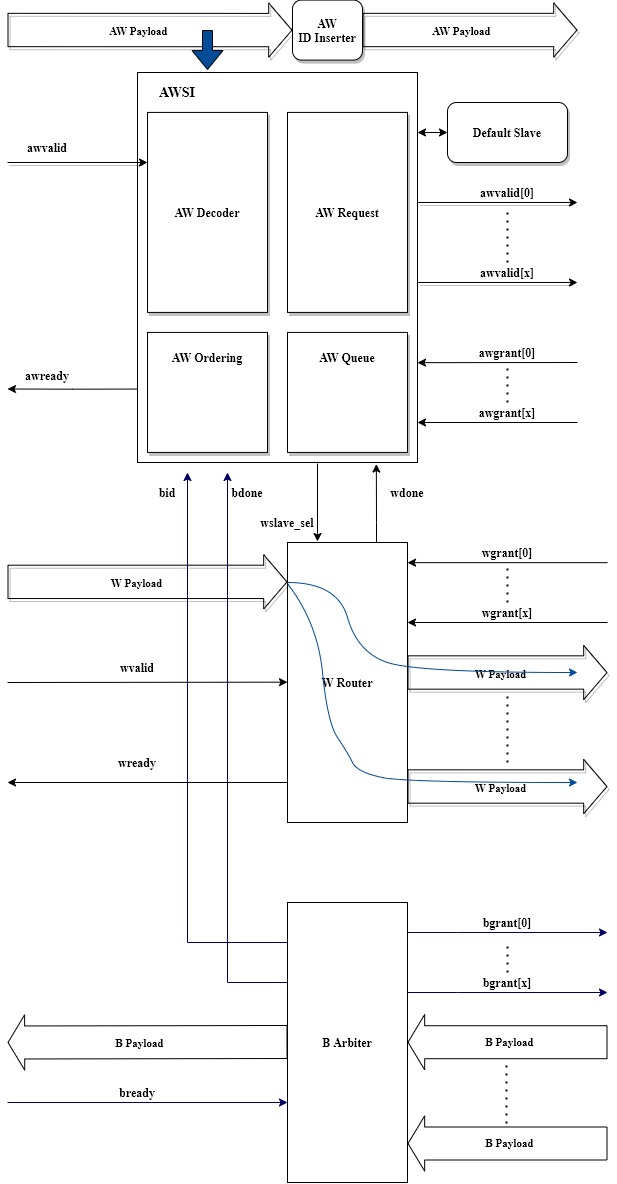
Cách giải quyết



**Hình 3: Repeated Register Solution**

Để giải quyết vấn đề trên, ARBITER sẽ dùng thêm các queue chứa dữ liệu gửi từ Master, đồng thời các thanh ghi lptr(limit pointer), wptr(write pointer), rptr(read pointer) cũng được thêm vào các bộ DECODE. Ban đầu wptr và rptr sẽ được reset về 0, tuy nhiên giá trị reset của lptr chính là độ sâu của queue tương ứng trên phía ARBITER. Master sẽ không được phép gửi dữ liệu tiếp theo nếu như queue đầy (lptr = wptr).

1. WRITE TRANSACTION
2. MASTER SIDE (SI)

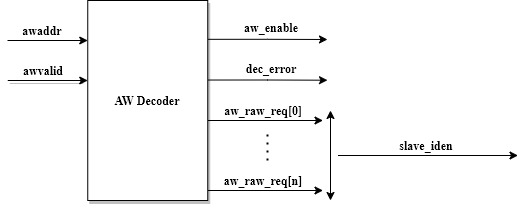


**Hình 4: Write transaction - Master side**

* 1. AWSI

AWSI quản lý kênh AW từ phía Master. Nhiệm vụ chính của nó là kiểm tra địa chỉ awaddr để phân phối dữ liệu đến các Slave tương ứng, đảm bảo các yêu cầu của CDAS. Nếu địa chỉ đưa vào không thuộc vùng địa chỉ của bất kì Slave nào, một Default Slave sẽ được gọi để trả tín hiệu DECERR về cho Master.

**AW Decoder**

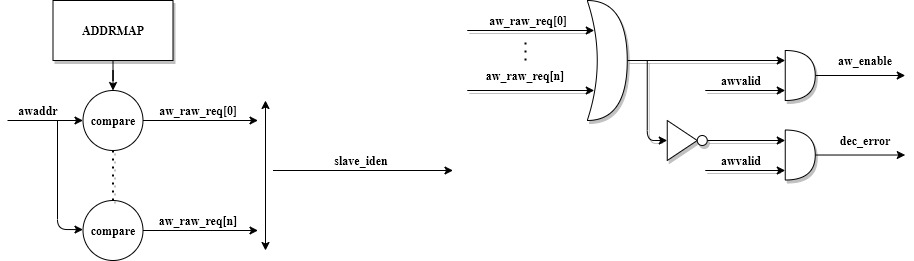


**Hình 5: AW Decoder interfaces**

+ slave\_iden chính là tín hiệu aw\_raw\_req.

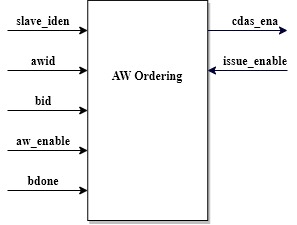
+ dec\_error: báo hiệu địa chỉ awaddr là không hợp lệ.

+ aw\_enable: báo hiệu địa chỉ awaddr là hợp lệ.



**Hình 6: AW Decoder logic**

**AW Ordering**

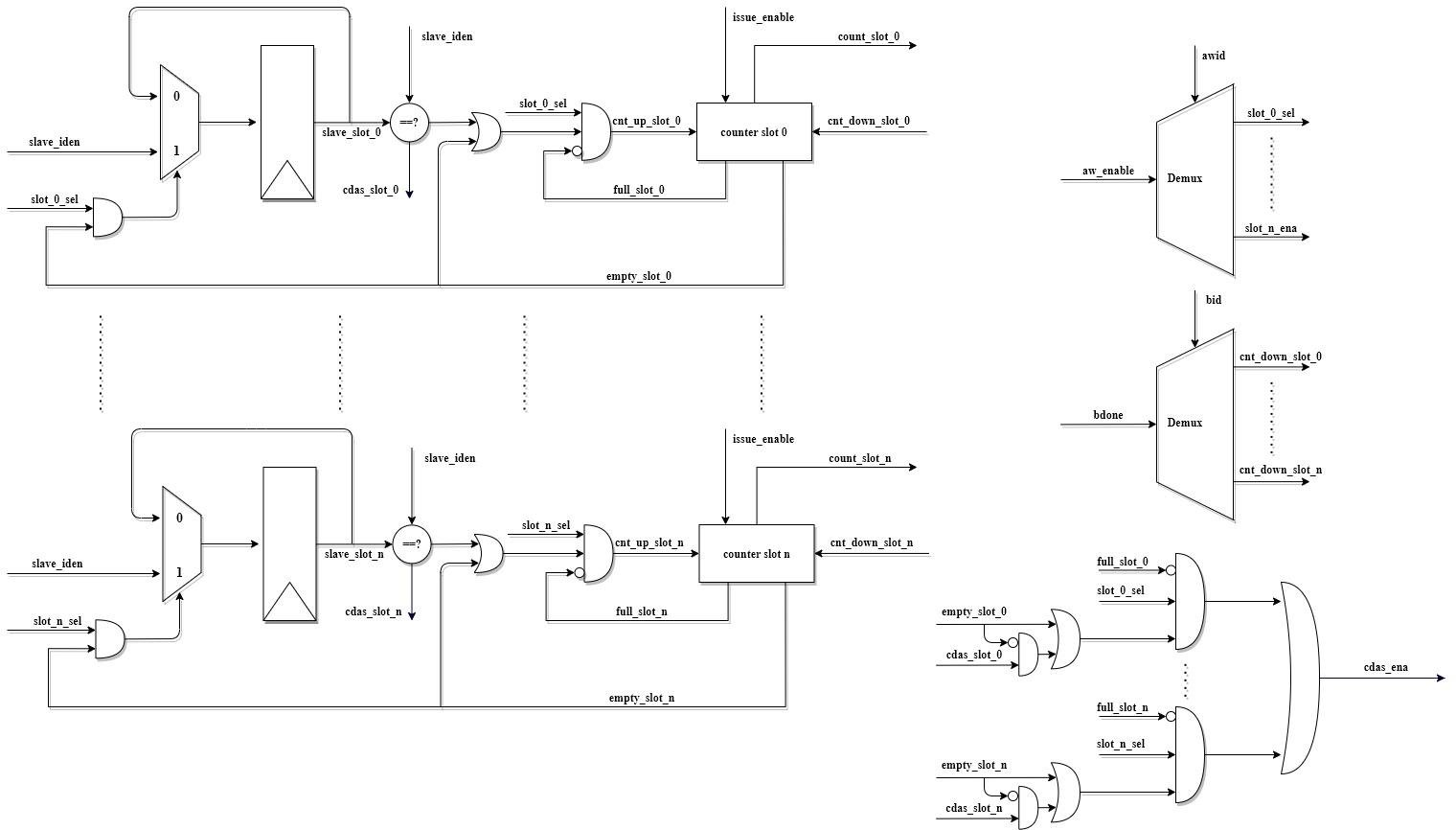


**Hình 7: AW Ordering interfaces**

+ bdone: tín hiệu báo tín hiệu trên kênh B đã được trả cho Master.

+ issue\_enable: tín hiệu báo AWSI được phép gửi request cho Transaction hiện tại.

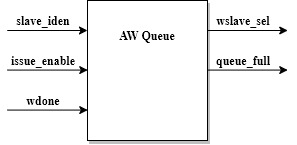
+ cdas\_ena: tín hiệu báo các yêu cầu về ordering theo CDAS được thỏa mãn.



**Hình 8: AW Ordering logic**

AW Ordering lưu thông tín về số outstanding transaction đang được gửi đi và Slave tương ứng của mỗi ID. Tín hiệu cdas\_ena sẽ được kéo xuống mức 0 nếu như các yêu cầu về CDAS không được thỏa mãn hoặc số lượng outstanding transaction của ID hiện tại đã đạt đến giời hạn.

**AW Queue**

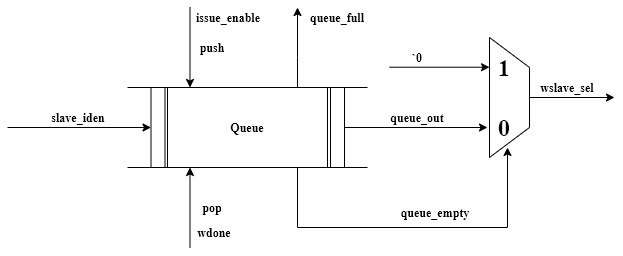


**Hình 9: AW Queue interfaces**

+ wdone: tín hiệu báo dữ liệu trên kênh W đã được nhận trên Slave tương ứng.

+ wslave\_sel: tín hiệu chọn Slave đích.

+ queue\_full: báo số lượng outstanding transaction hiện tại đã đạt tới giới hạn.



Hình 10: AW Queue logic

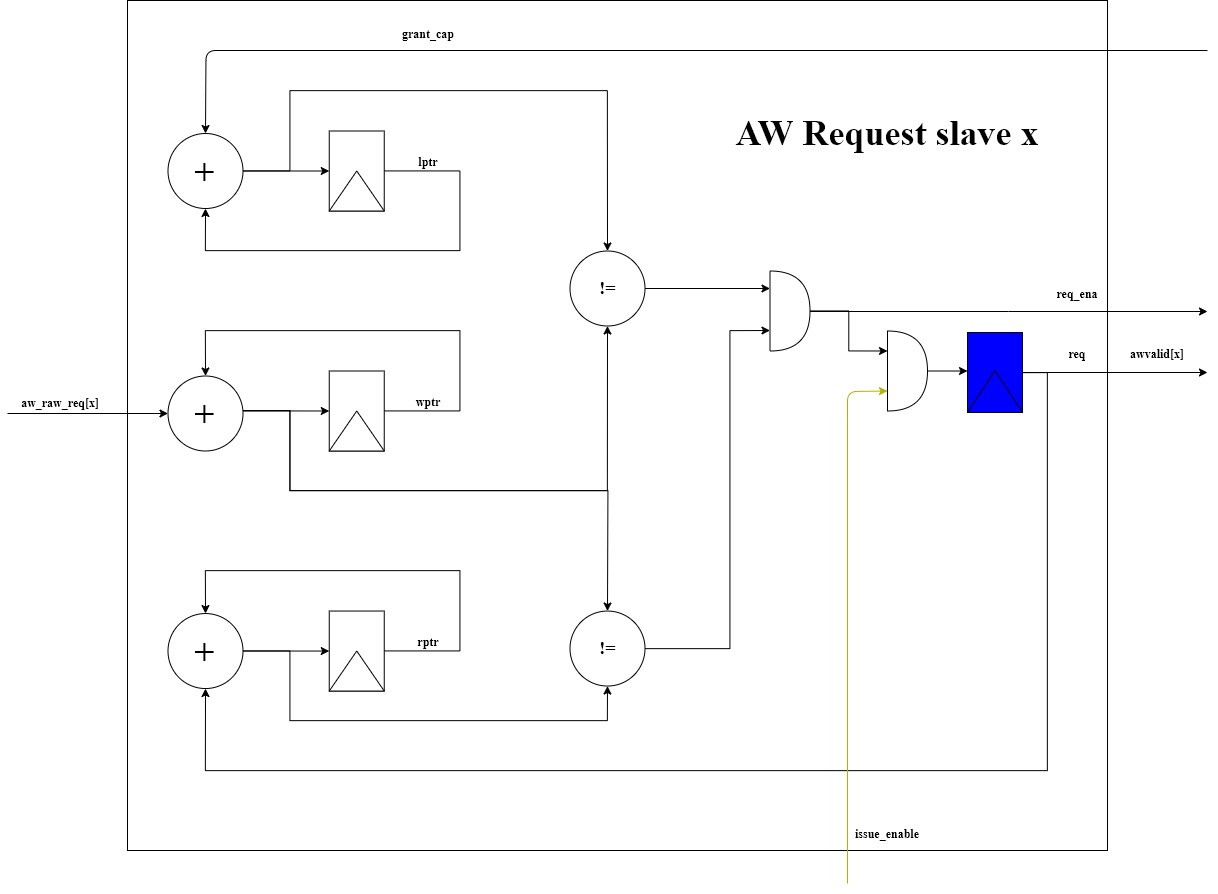
**AW Request**

AW Request gồm nhiều cell AW Request slave x, mỗi cell quản lý một Slave tương ứng. Nhiệm vụ của mỗi cell là lưu thông tin của queue bên phía MI, nhờ đó AWSI biết được dữ liệu hiện tại có được phép xuất hiện trên đường truyền hay không. Đây là một phần của bộ **Repeated Register Solution** được mô tả ở trên.

+ req\_ena: cho biết queue bên phía Slave hiện tại vẫn chưa đầy, Master được phép gửi dữ liệu nếu như issue\_enable được bật.

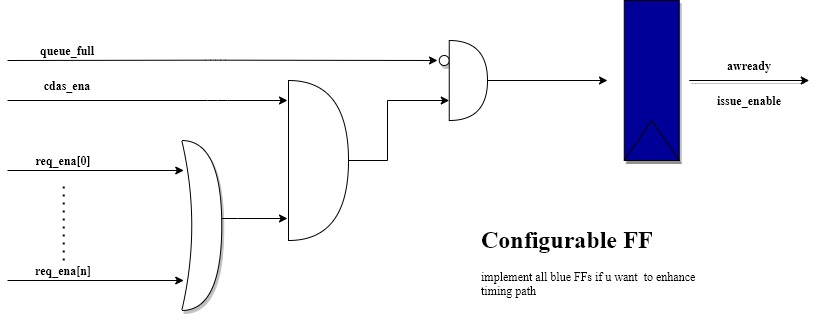
+ grant\_cap: chính là tín hiệu grant được trả về từ phía MI.

+ Flip Flop trong hình có thể có hoặc không tùy theo cấu hình của DECODE.



**Hình 11: AW Request Slave x**

**OTHER LOGIC**

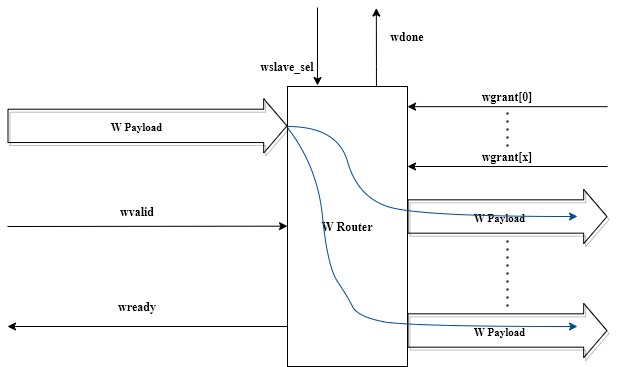


**Hình 12: Other logic AWSI**

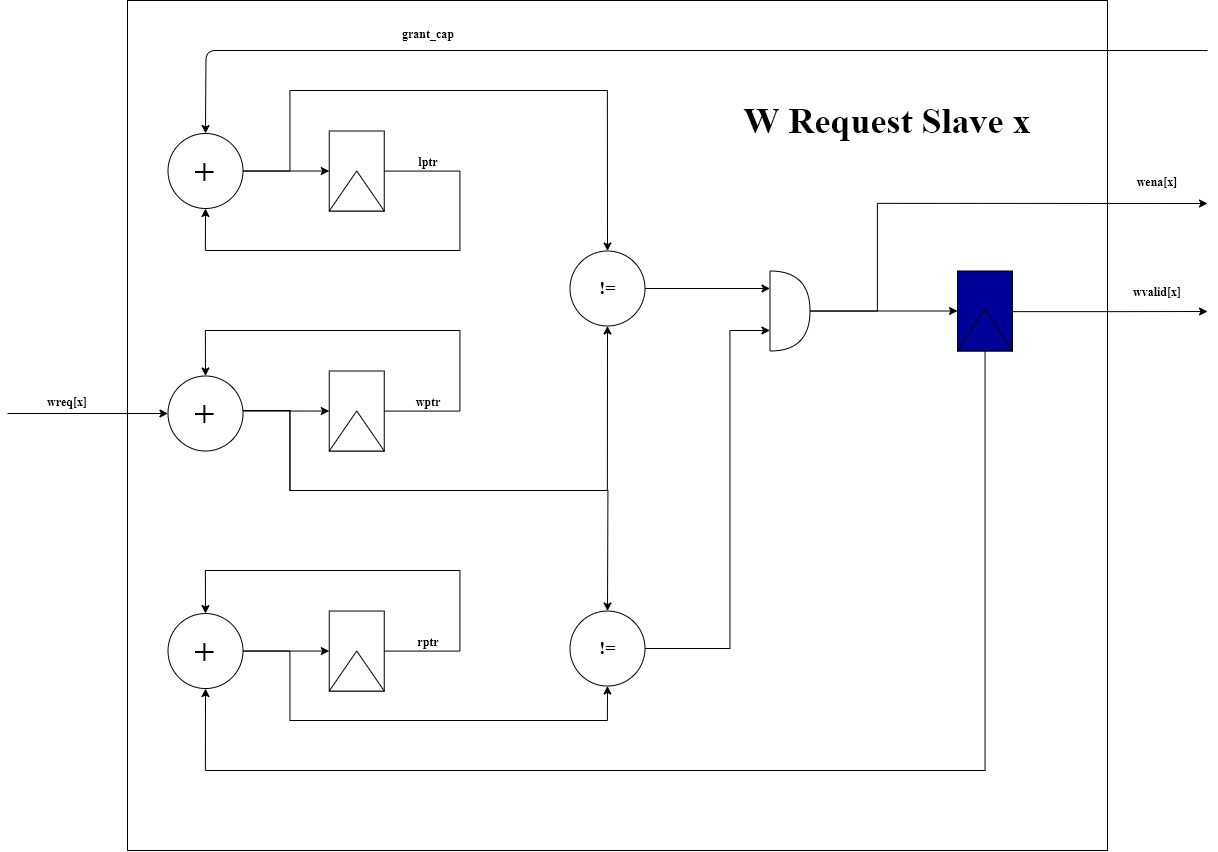
+ issue\_enable cũng chính là tín hiệu awready trả lại cho Master.

+ Flip Flop trong hình có thể có hoặc không tùy theo cấu hình của DECODE.

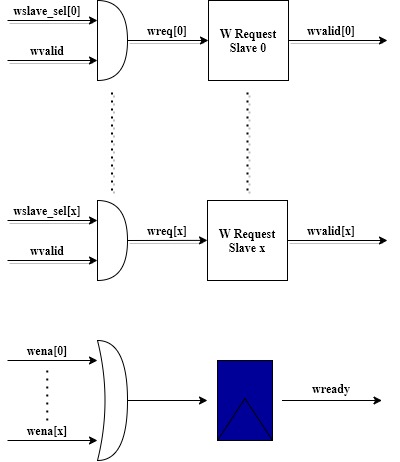
* 1. W ROUNTER



**Hình 13: W Router interfaces**



**Hình 14: W Request Slave x**



**Hình 15: W Router logic**

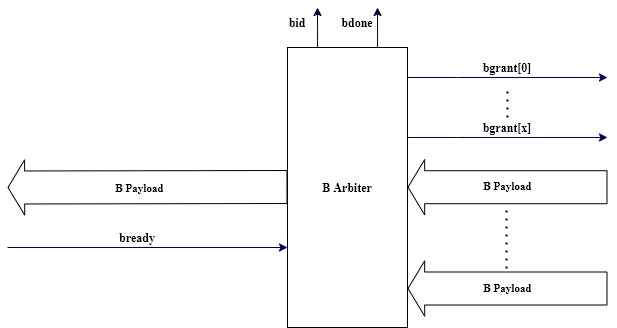
+ W Payload sẽ được phân phối đến các Slave tương ứng, tuy nhiên tín hiệu wvalid ở mỗi kênh W Payload sẽ được điều chỉnh bằng các mạch logic phía trên.

+ wdone: tín hiệu báo kết thúc của một chuỗi W transaction, được bật lên khi có wvalid và wlast.

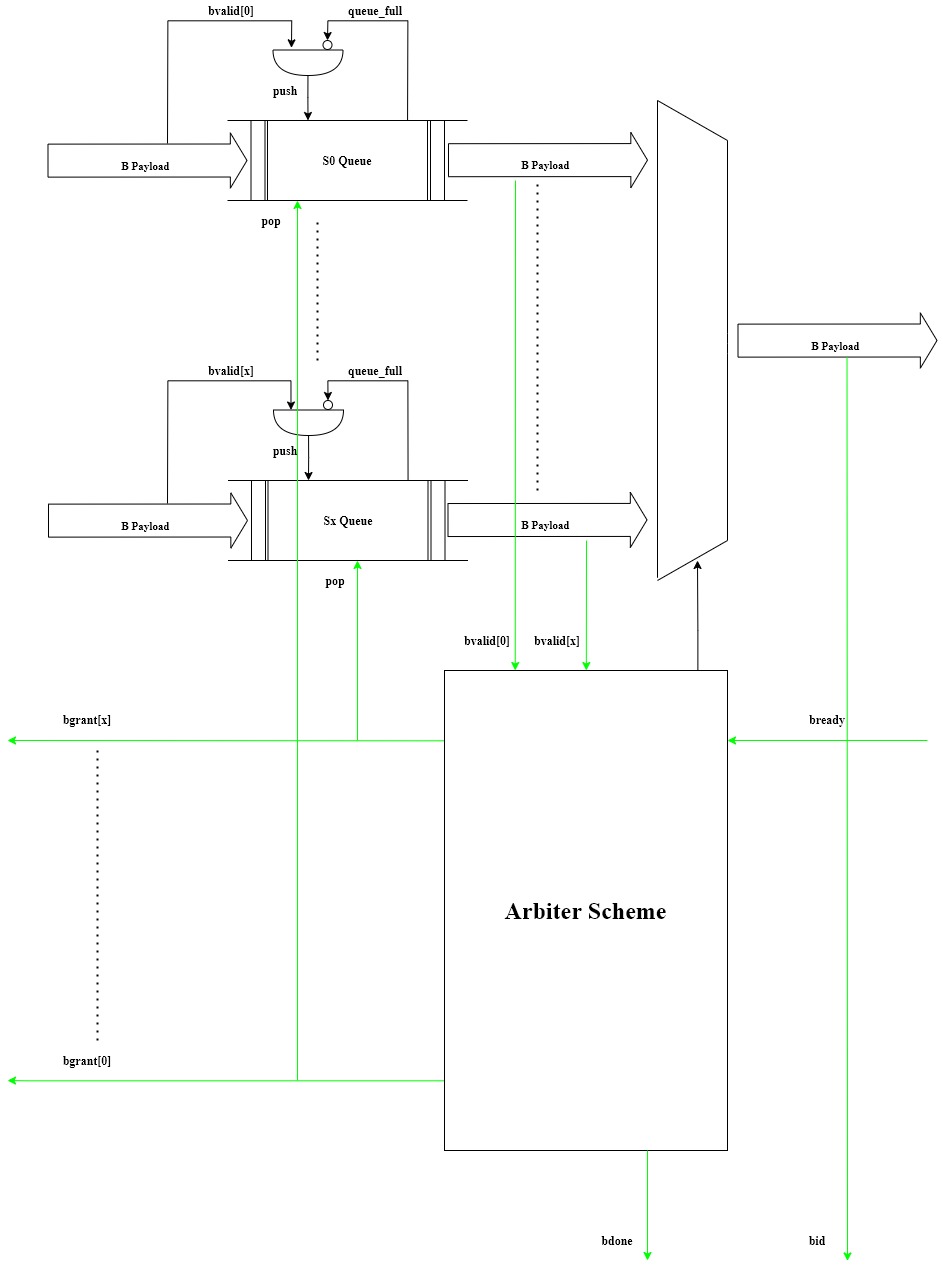
* 1. B ARBITER

B Arbiter làm nhiệm vụ phân xử lựa chọn B Payload trả lại cho Master. Các kênh B Payload đang yêu cầu grant B Arbiter là các kênh có bvalid tích cực mức cao.

+ bdone: tín hiệu báo tín hiệu trên kênh B đã được trả cho Master.



**Hình 16: B Arbiter interfaces**

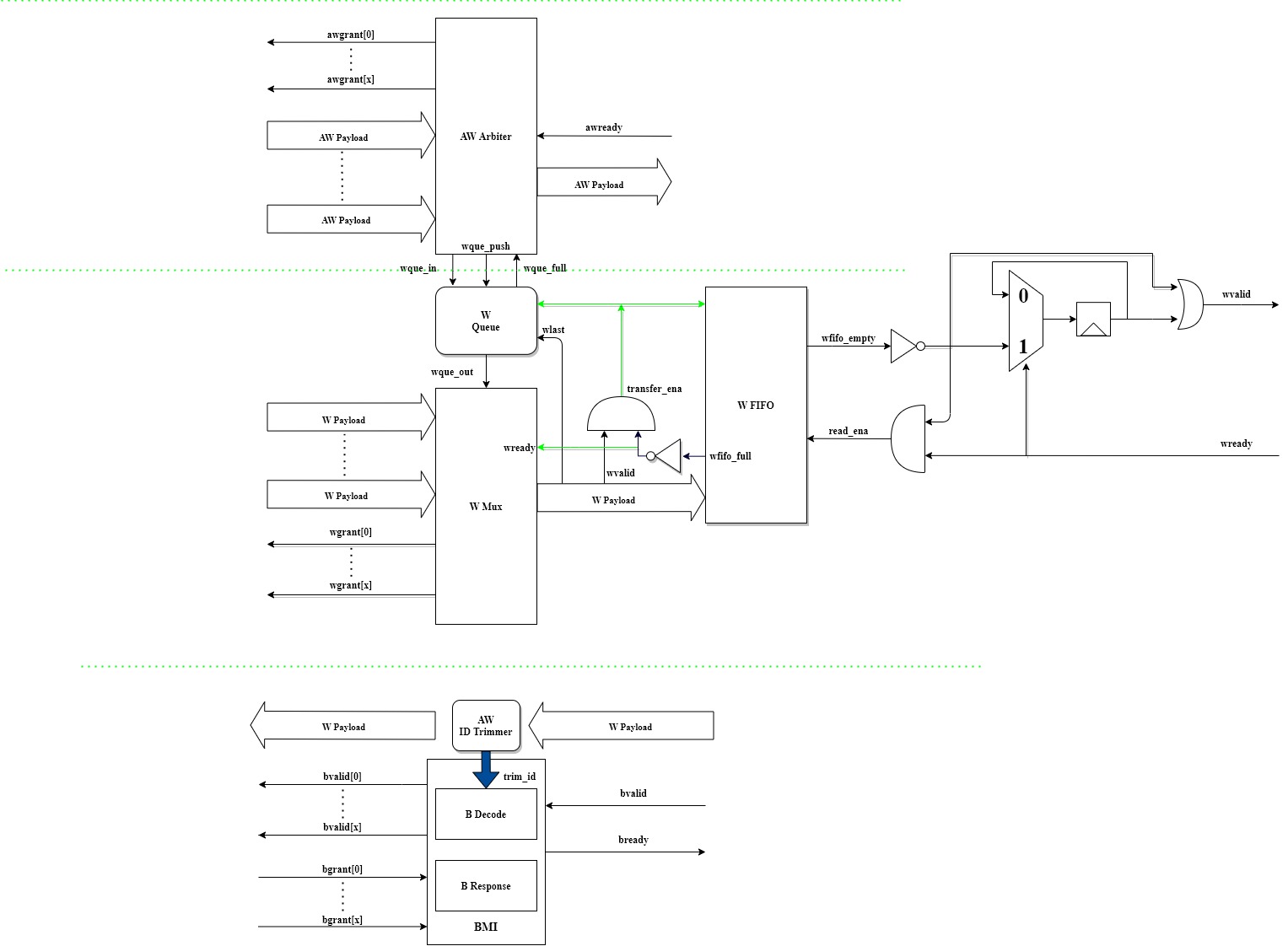


**Hình 17: B Arbiter logic**

+ bvalid: chính là các tín hiệu req của Arbiter Scheme. Arbiter Scheme chỉ được phép grant tín hiệu khi bready tích cực mức cao.

+ bdone: bdone sẽ được bật lên mỗi khi có một tín hiệu grant tích cực.

1. SLAVE SIDE (MI)



**Hình 18: Write transaction - Slave side**

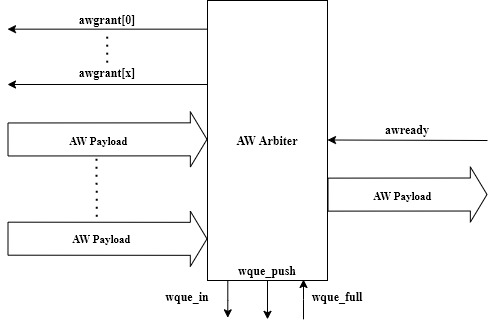
W FIFO và các logic xung quanh ở hình trên minh họa cho cấu trúc của X2X khi người dùng muốn sử dụng một thêm FIFO ở phía MI.

Nếu không sử dụng W FIFO, tín hiệu wfifo\_full ở hình trên sẽ được thay bằng tín hiệu wready. Các logic phía sau W FIFO sẽ được bỏ đi.

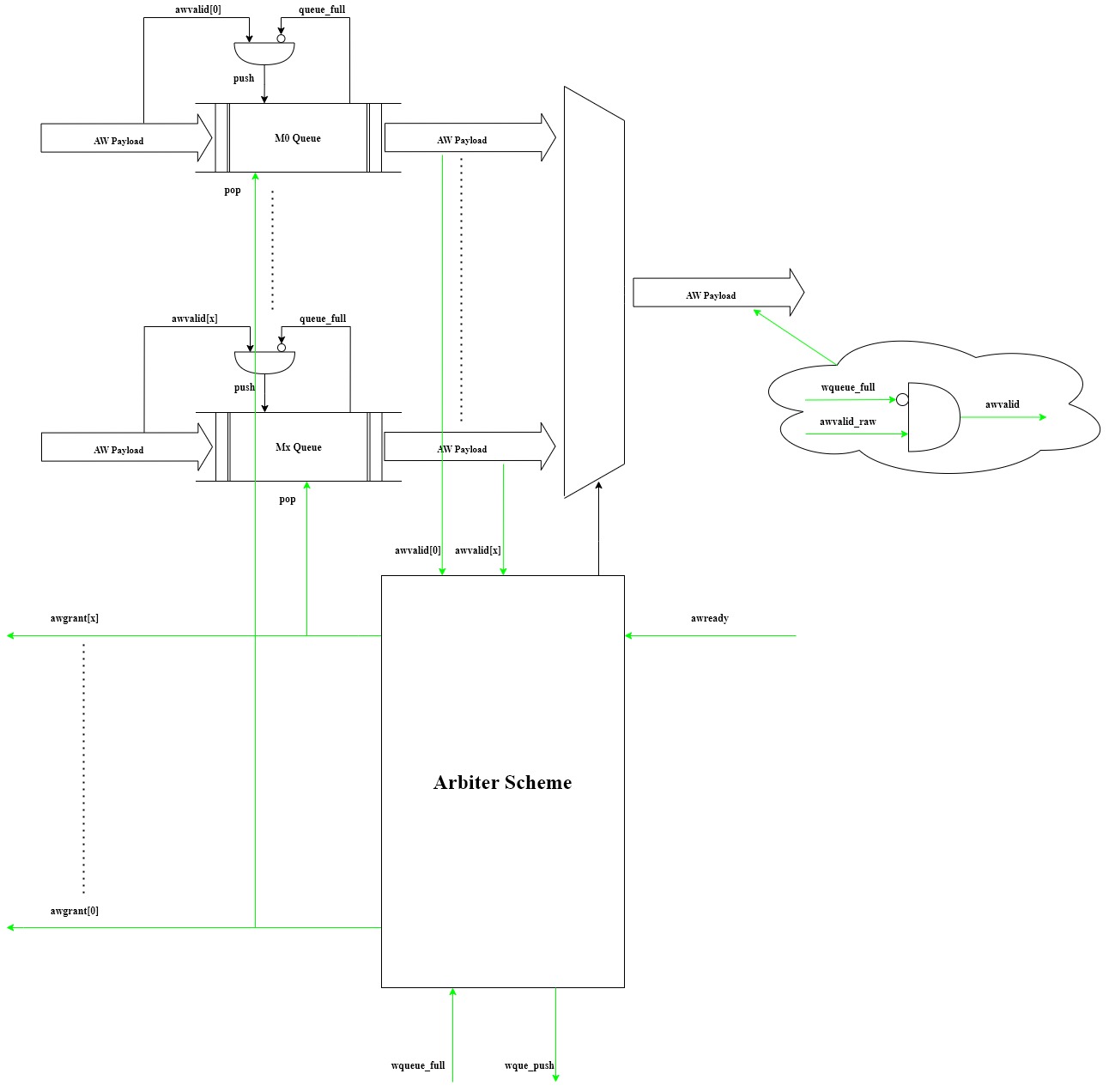
* 1. AW Arbiter

+ wque\_push: tín hiệu yêu cầu push identifier của Master được chọn vào W Queue.

+ wque\_full: tín hiệu báo W Queue đã đầy.



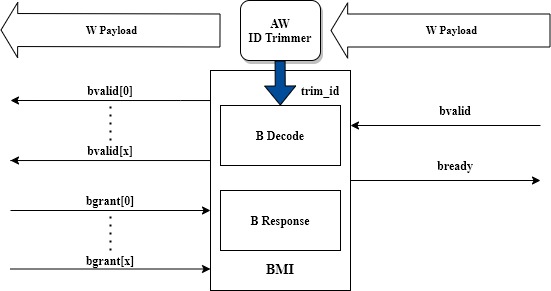
**Hình 19: AW Arbiter interfaces**



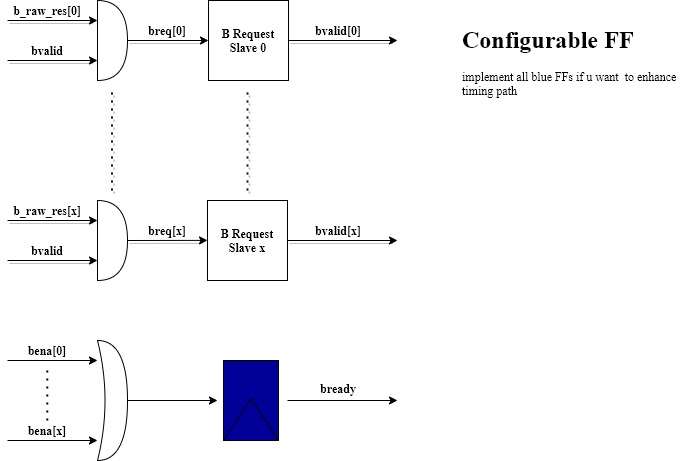
**Hình 20: AW Arbiter logic**

Hoặc động của AW Arbiter gần giống với B Arbiter. Tuy nhiên, AW Arbiter phải xét thêm tín hiệu wque\_full mới được phép trả tín hiệu grant về cho các Master.

* 1. BMI



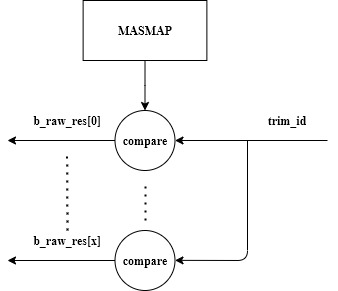
Hình 21: BMI interfaces



Hình 22: BMI logic

BMI hoặt động tương tự như W ROUTER. Tuy nhiên BMI phải sử dụng thêm một bộ decode để biết dữ liệu hiện tại cần trả cho Master nào.

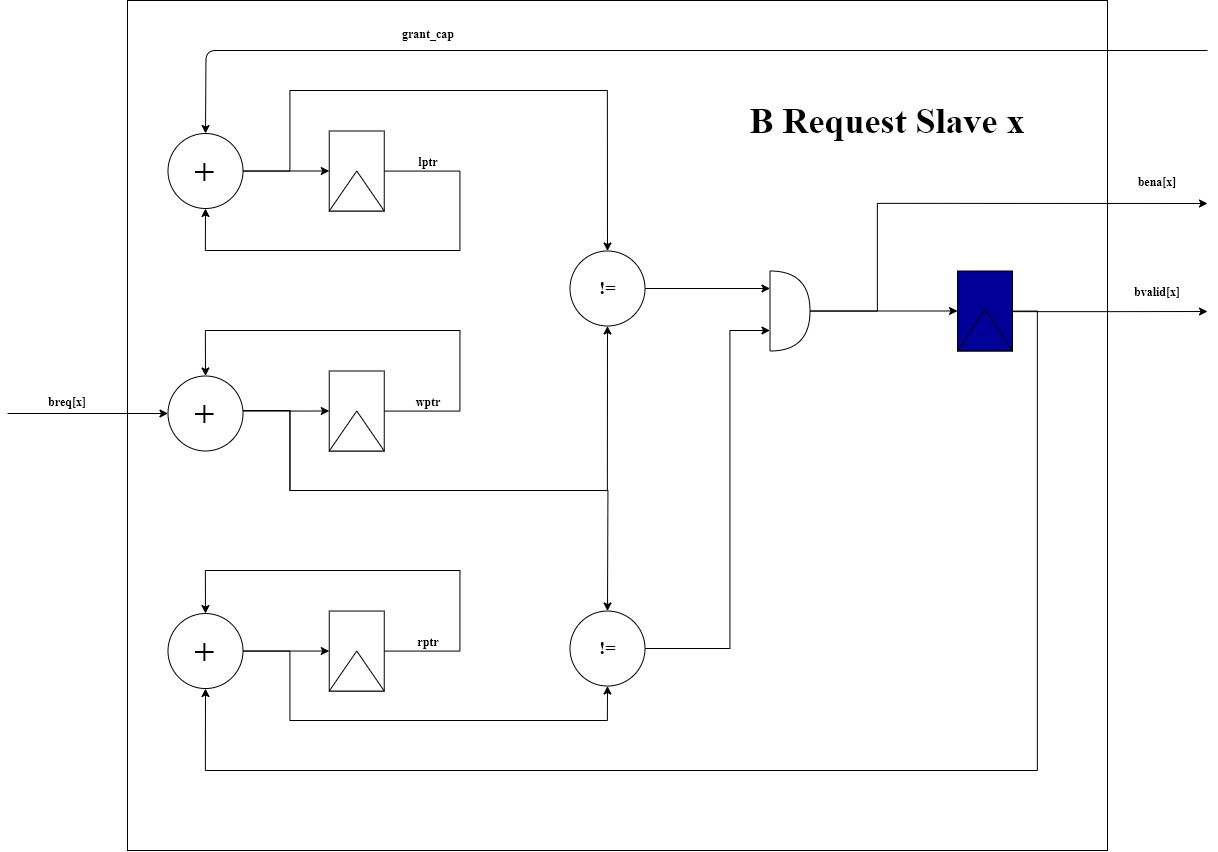
**B Decode**



**Hình 23: B Decode**

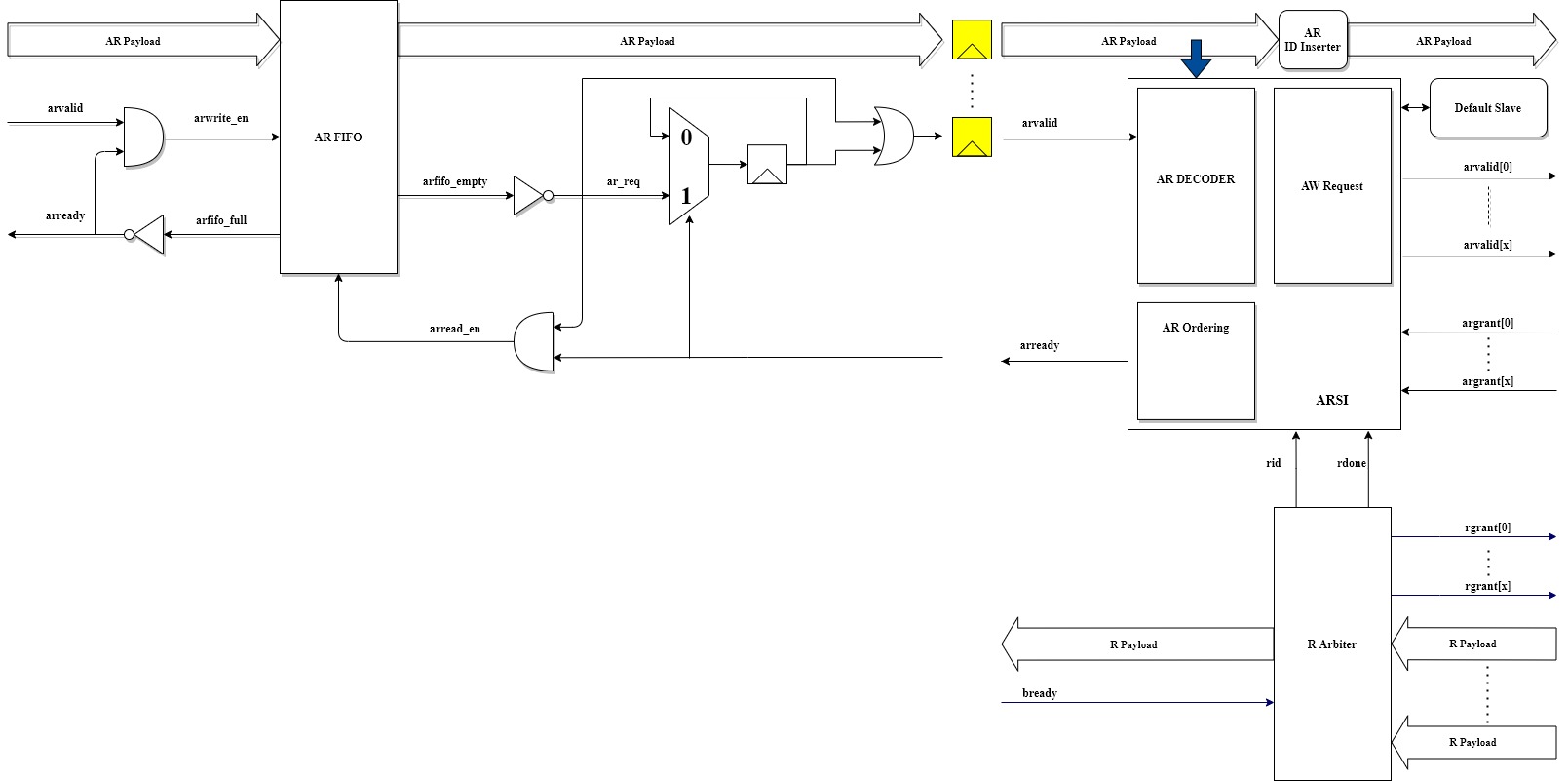
+ trim\_id: là phần ID được tách ra từ BID, đây chính là ID được thêm vào sau bộ AW ID Inserter.

**B Request Slave x**



**Hình 24: B Request Slave x**

1. READ TRANSACTION
2. MASTER SIDE (SI)



**Hình 25: Read transaction – Master side**

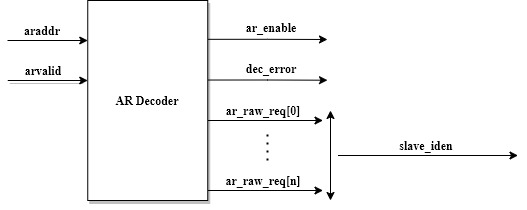
AR FIFO và các logic xung quanh ở hình trên minh họa cấu trúc X2X khi người dùng muốn sử dụng thêm FIFO ở phía SI.

* 1. ARSI

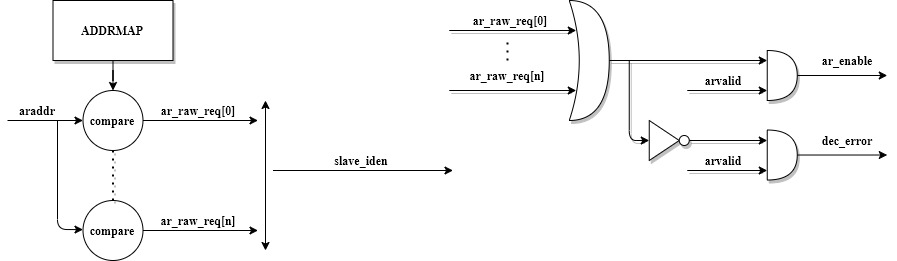
ARSI quản lý kênh AR từ phía Master. Nhiệm vụ chính của nó là kiểm tra địa chỉ araddr để phân phối dữ liệu đến các Slave tương ứng, đảm bảo các yêu cầu của CDAS. Nếu địa chỉ đưa vào không thuộc vùng địa chỉ của bất kì Slave nào, một Default Slave sẽ được gọi để trả tín hiệu DECERR về cho Master.

Hoạt động của ARSI tương tự AWSI, điểm khác biệt duy nhất là ARSI không cần lưu thông tin về Slave đích của mỗi AR transaction.

**AR Decoder**

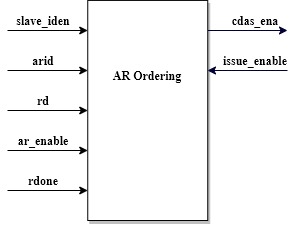


**Hình 26: AR Decoder interfaces**

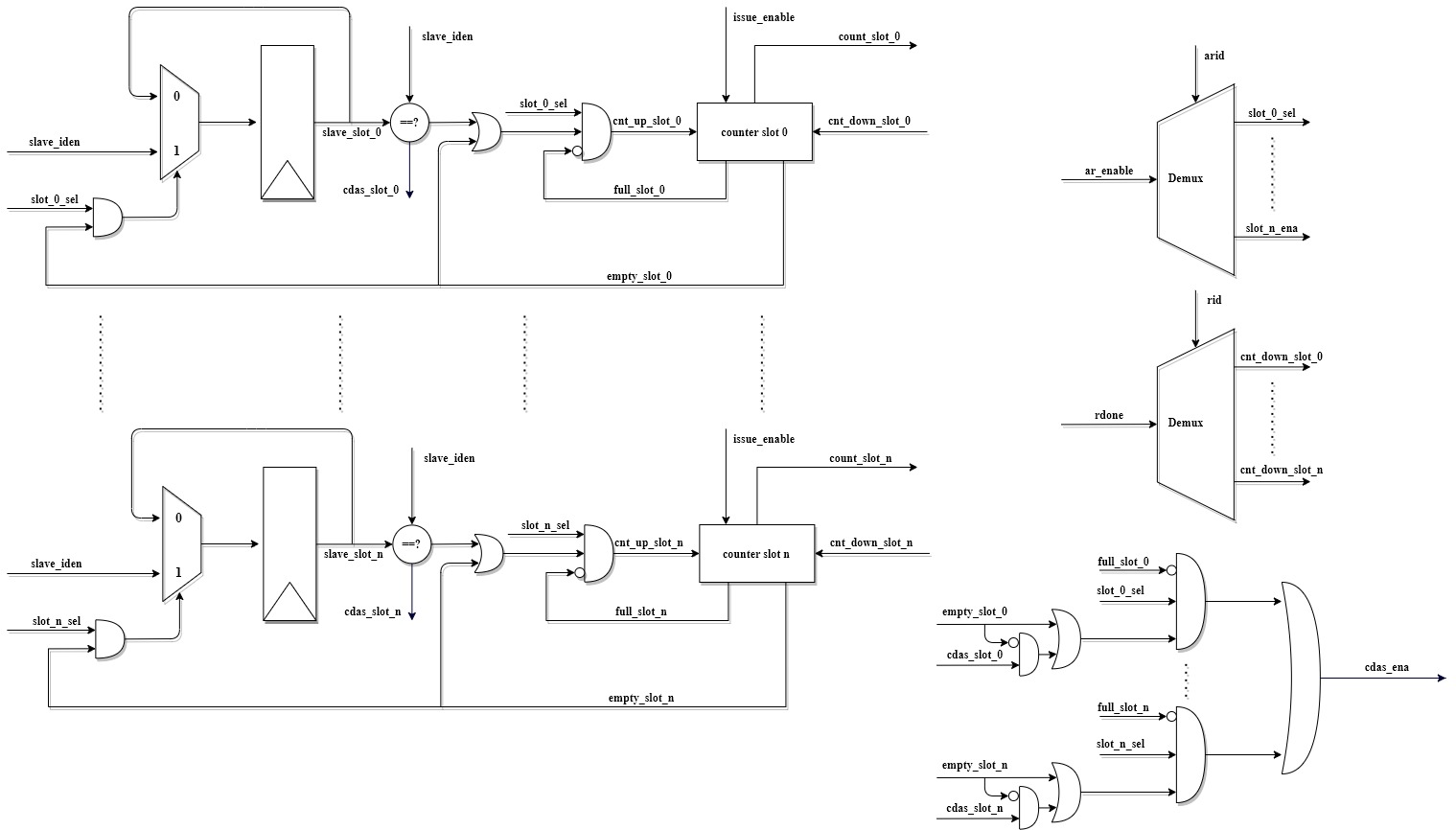


**Hình 27: AR Decode logic**

**AW Ordering**

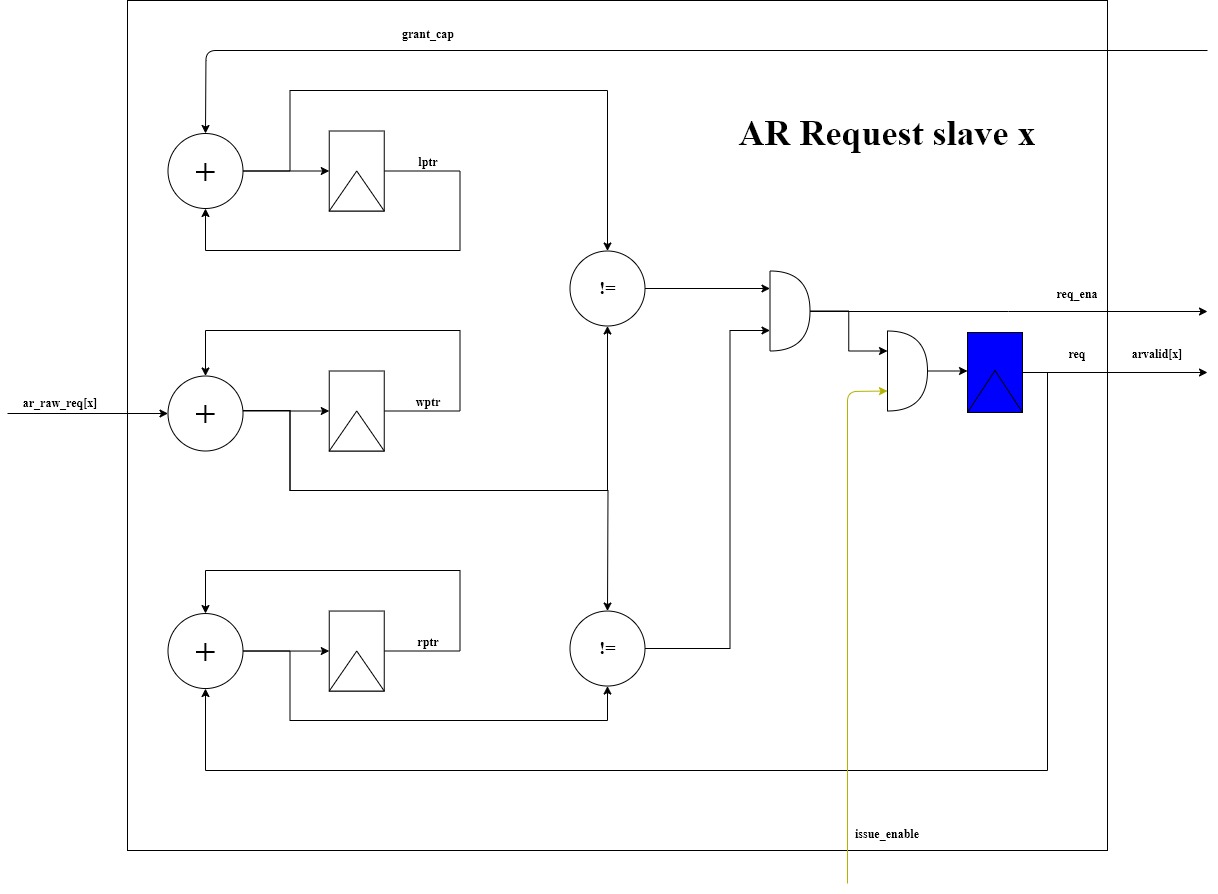


Hình 28: AR Ordering interfaces



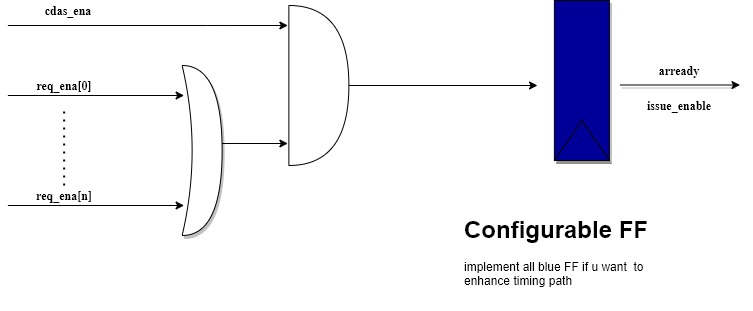
Hình 29: AR Ordering logic

**AR Request**



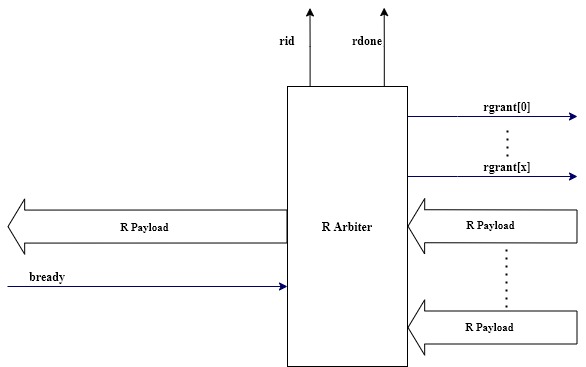
**Hình 30: AR Request**

**OTHER LOGIC**



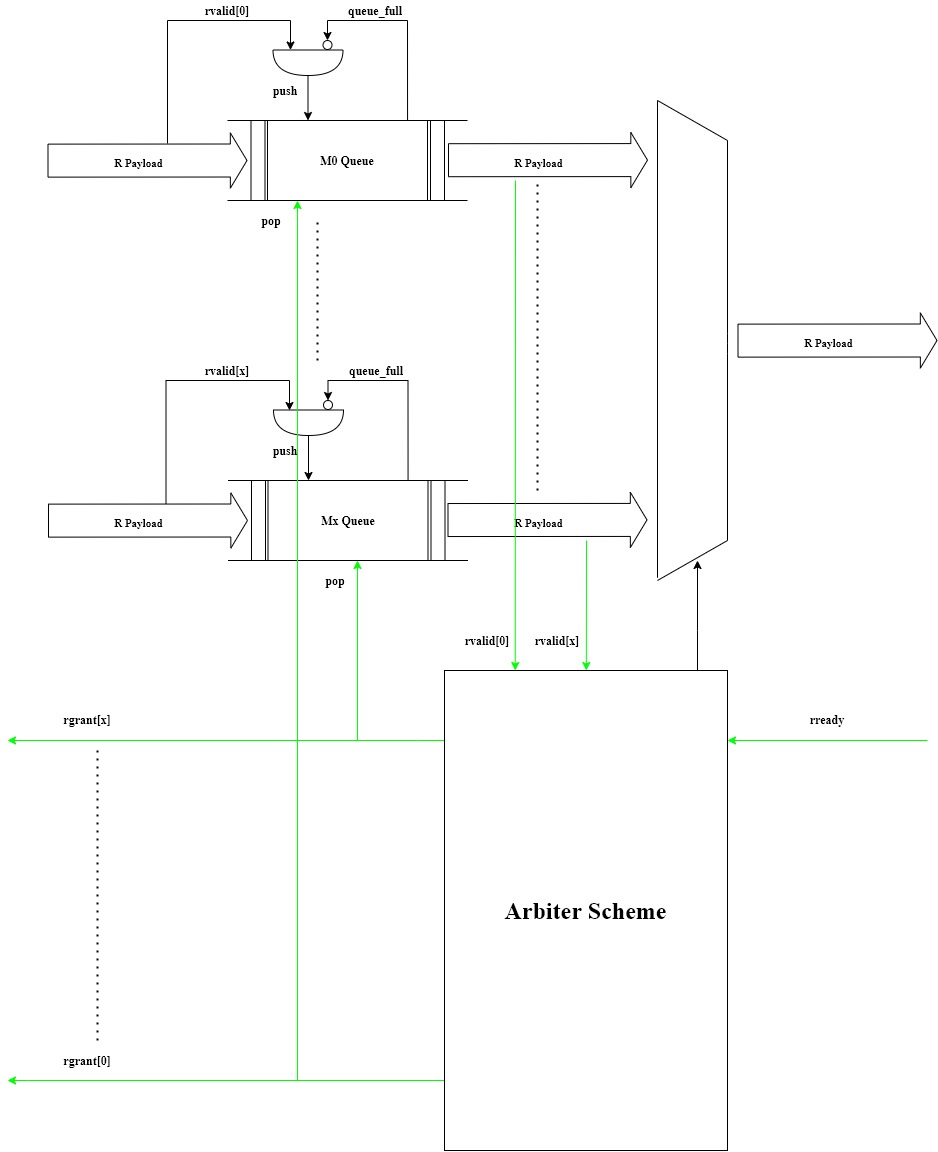
**Hình 31: Other logic ARSI**

* 1. R Arbiter



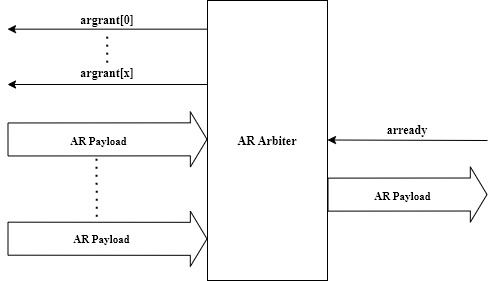
**Hình 32: R Arbiter interfaces**

+ wdone: tín hiệu báo kết thúc của một chuỗi R transaction, được bật lên khi có rvalid và rlast.

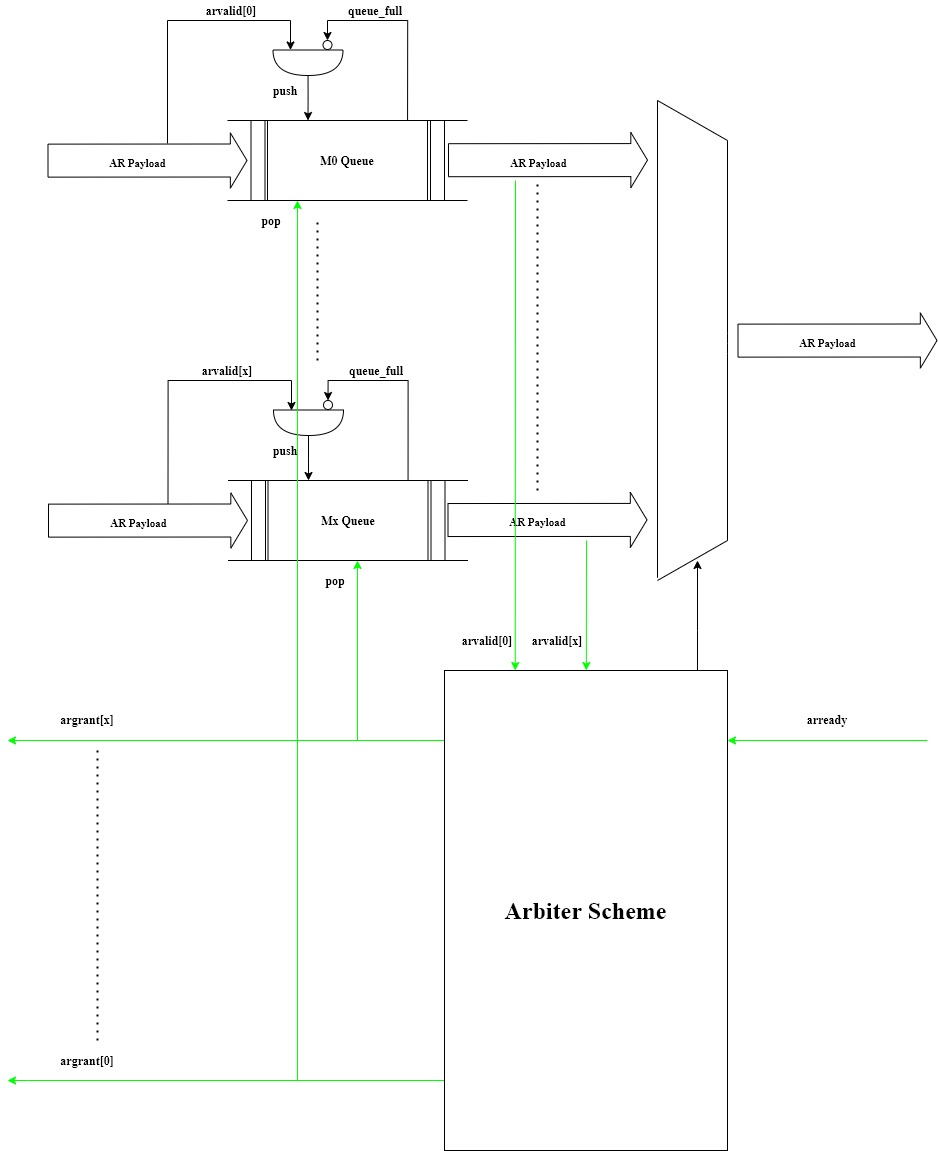


**Hình 33: R Arbiter logic**

1. SLAVE SIDE (MI)
   1. AR Arbiter

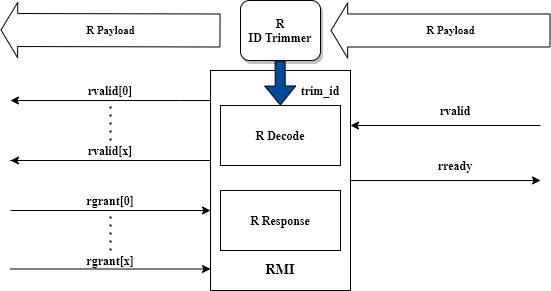


**Hình 34: AR Arbiter interfaces**

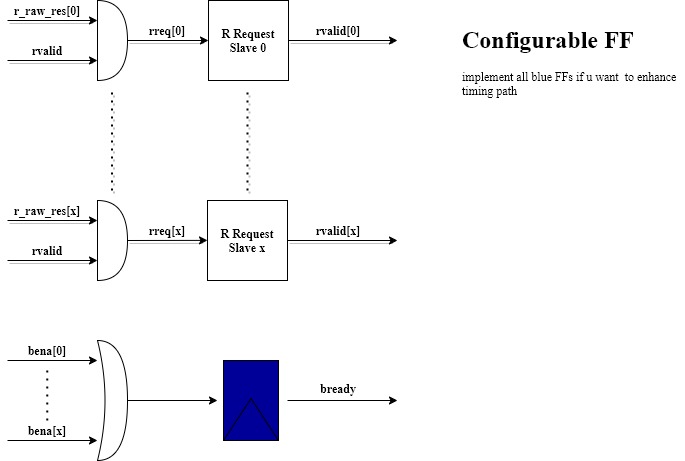


**Hình 35: AR Arbiter logic**

* 1. RMI

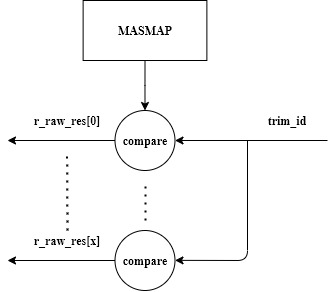


**Hình 36: RMI interfaces**



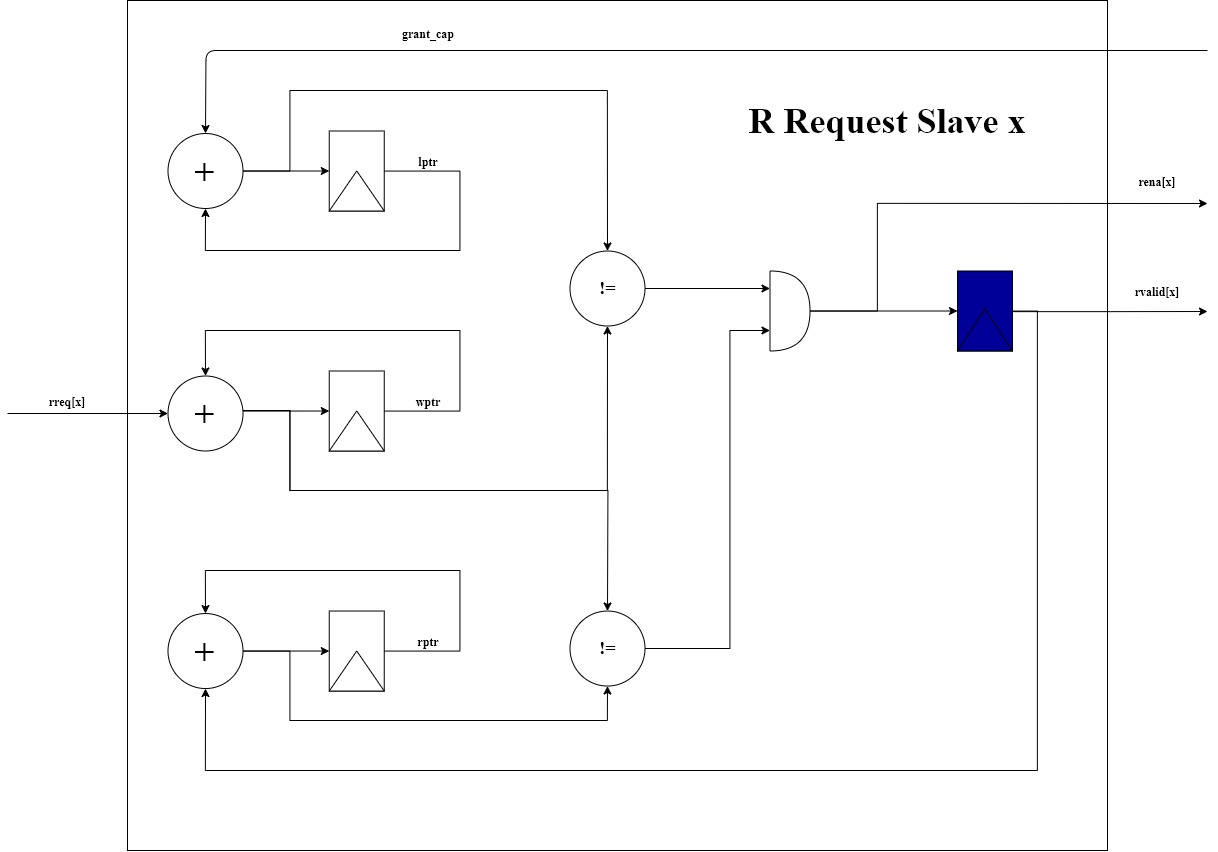
**Hình 37: RMI logic**

**R Decode**



**Hình 38: R Decode**

**R Request Slave x**



**Hình 39: R Request Slave x**