CFG REGBUS updates

Requirements NOTES

Joji PHILIp, Jay TOMLINSON

Table of Contents

[3 Table of Tables 4](#_Toc18669087)

[4 REGBUS functional requirements 6](#_Toc18669088)

[4.1 RBC 6](#_Toc18669089)

[4.2 RBM 6](#_Toc18669090)

[4.3 Transport 6](#_Toc18669091)

[4.3.1 Routing on regbus 6](#_Toc18669092)

[4.4 Slave side 6](#_Toc18669093)

[4.4.1 Ring masters 7](#_Toc18669094)

[4.4.2 Ring slaves 8](#_Toc18669095)

[4.4.3 CSR 8](#_Toc18669096)

[4.5 PMON 8](#_Toc18669097)

[4.6 Message mastering from slaves (interrupts and errors) 8](#_Toc18669098)

[4.7 Domain crossings 9](#_Toc18669099)

[4.8 Functional safety 9](#_Toc18669100)

[4.9 Low power and clock gating 9](#_Toc18669101)

[4.10 Broad next level questions 9](#_Toc18669102)

[5 Micro Architecture 10](#_Toc18669103)

[5.1 RegBus Overview 10](#_Toc18669104)

[5.2 RegBus Packet formats 11](#_Toc18669105)

[5.3 RegBus Controller (RBC) 13](#_Toc18669106)

[5.4 RegBus Master (RBM) 13](#_Toc18669107)

[5.5 RegBus Slave decoder (SD) 14](#_Toc18669108)

[5.5.1 Overview 14](#_Toc18669109)

[5.5.2 Block Diagram 14](#_Toc18669110)

[5.5.3 RX NoC Interface to Ring Interface(s) 15](#_Toc18669111)

[5.5.4 Ring Interface(s) to TX NoC 15](#_Toc18669112)

[5.6 RegBus Slave (RBS) 16](#_Toc18669113)

[5.6.1 Overview 16](#_Toc18669114)

[5.6.2 Block Diagram 16](#_Toc18669115)

[5.6.3 Interfaces 19](#_Toc18669116)

[5.6.4 Interface FIFO 20](#_Toc18669117)

[5.6.5 CSR Request/Response Registers. 21](#_Toc18669118)

[5.6.6 Receive Packet Processing 22](#_Toc18669119)

[5.6.7 Transmit Response Packet 24](#_Toc18669120)

[5.6.8 Response Credit Management 28](#_Toc18669121)

[5.6.9 Interface to CSR modules 28](#_Toc18669122)

[5.6.10 Reg Slave Generated messages. 28](#_Toc18669123)

[5.7 Future 29](#_Toc18669124)

Table of Figures

[Figure 1 RegBus Star Topology 10](#_Toc18669125)

[Figure 2 RegBus Ring Topology 11](#_Toc18669126)

[Figure 3 RBS Read Request Packet format 12](#_Toc18669127)

[Figure 4 RBS Write Request Packet format 12](#_Toc18669128)

[Figure 5 RBS Read Response Packet format 13](#_Toc18669129)

[Figure 6 RBS Write Response Packet format 13](#_Toc18669130)

[Figure 7 Slave Decoder Block Diagram 15](#_Toc18669131)

[Figure 8 RBS Block Diagram 17](#_Toc18669132)

[Figure 9 36-bit REGBUS\_RDATA\_WIDTH Read32 Transaction Timing 18](#_Toc18669133)

[Figure 10 36-bit REGBUS\_RDATA\_WIDTH Write32 Transaction Timing 19](#_Toc18669134)

[Figure 11 Interface FIFO functional behavior -- Writes 20](#_Toc18669135)

[Figure 12 RX FSM 23](#_Toc18669136)

[Figure 13 RX Timing: 36-bits 23](#_Toc18669137)

[Figure 14 RX Timing: 18-bits 24](#_Toc18669138)

[Figure 15 TX FSM 25](#_Toc18669139)

[Figure 16 FX Timing: 36-bits 26](#_Toc18669140)

[Figure 17 FX Timing: 36-bits with credit delays 27](#_Toc18669141)

# Table of Tables

[Table 1 RBS Command Encoding 12](#_Toc18669142)

[Table 2 Ring Interface Signals 19](#_Toc18669143)

# REGBUS functional requirements

## RBC

Regbus controller will replace existing regbus tunnel module. It allows multiple requesters to get on the regbus layer to access registers in the NoC and user register space. Multiple protocol interfaces will be supported: AXI4, AXI4-Lite, APB, IOSF-SB etc.

## RBM

Current RBM is created out of a subset of the ace master bridge. The new regbus master will connect user interfaces from RBC to the regbus transport layer. It will have a single command channel to support ordering requirements of IOSF-SB. RSSB will be used in place of TX/RX switches.

RBC-RBM is under development currently.

* Low power support (fence/drain)
* Need low power support even when user regbus is enabled.

## Transport

Primary goal is to reduce area and wire count of the regbus transport layer.

* New transport layer will be designed using RSSB
* A single layer will be utilized
* A single virtual channel would suffice if the layer is only used for register access traffic
  + Expect to eventually need multi-VC, bi-directional traffic (traffic to RBM and back down tree).
* Data width of the transport layer is configurable from 36 down to 9-bits
* Sideband reduction can be done using look up routing mode in RSSB or by sending the route information in-band instead of in sideband.

### Routing on regbus

Regbus currently has a tree topology, with the single regbus master forming the root of the tree. All the slaves have a single fixed route to the master. Routes from RBM to slaves can require multiple turns. If the topology requires a single turn, lookup routing has no advantage over the current source routing. For multi-turn routes, other route information passing options being developed for RSSB can be deployed on regbus layer to reduce the side band wiring and flop cost on the regbus transport. These other route information passing options are look-up routing in RSSB or by passing the route information in-band in the packet.

## Slave side

Two classes of slaves are currently supported, internal NoC registers and user agent registers.

### Ring masters

This module has traditionally been the NoC destination node for a register transaction from RBM. This module in turn passes the register access to target modules at that node position using rings. Address lookup at RBM produces route to the ring master of the node and also the ring ID of destination module at that node.

#### Changing to a star topology

* Ring master can be replaced by the node RSSB
* Non-direction port of the RSSB (up to 24 ports) can serve as connections to CSR slaves
* Route lookup only needs to provide the route to target RSSB and its exit port (if using source routing)
* This is more suitable for power, clock and voltage domain crossing
  + A unique ring is not needed for each power domain at the node. Low power handling is easier
* Domain change can be supported using ILDC on the RSSB link ring slave device
* Number of wires required by a star network can be higher than a uni-directional ring. However,
  + Within a node region, when the distances spanned are smaller, the wire cost difference is negligible
* Fits better with nocstudio model of NoC construction
* All links are credit flow controlled and pipelinable like other NoC links.
* A start topology reduces the latency of register accesses compared to the ring

#### Advantages of staying with a ring

* As noted earlier, ring implementation uses lower number of wires and is a suitable approach if the ring is expected to span across nodes
* A ring approach has smaller size of arbitrations compared to a star. A ring is likely to operate at higher frequency and so might be more suited to running the regbus layer at-speed.
* Hierarchical routing achieved by the ring approach might offer advantages
* Regbus to ring transition stop ‘Ring-master’ offers a suitable point for hosting functions like PMON registers.

#### Alternate star topology

Another disadvantage of a star topology is that each slave in a different clock domain ends up requiring a crosser from the RSSB RM. This is in comparison to a ring per clock domain, where a single crosser is required to get on the ring assigned to that clock domain and shared by all slaves in that clock domain.

Alternatively, one RSSB host ports is assigned for each clock domain. There is a single ILDC on this link which feeds a slave decoder module, which acts as a root to a star connecting to all slaves in that clock domain. A ring per clock domain is replaced by a star per clock domain.

RING MASTER >> STAR RSSB

RING >> STAR SLAVE DECODER

### Ring slaves

* When used in the star toplogy, ring slaves are not really handling any ring functionality
* They are simple register slave bridges converting from the narrow regbus packet to the wider register read/write interface to CSR blocks

If a ring based architecture is continued, ring slave must have

* 1-input (req), 1-output(req and resp), 1-bidi port (req to CSR, resp from CSR)
* Single VC, but credited flow control
* Support resizing between ring and CSR interfaces

### CSR

This likely doesn’t need any major architectural changes. However, automation of RTL and collateral generation from a single register definition source file needs to be completed. Standard interfaces, standard register types and common module usage has to be continued and enhanced as needed.

## PMON

* Support is required for Intel PMON architecture and programming model
* PMON configuration, event selection and counters have be shared among modules in the NoC
* A centralized location of the PMON modules is required wrt to the NoC modules sharing it.
* Writes to the PMON control registers have to have some multicast/broadcast semantics to convey them to the participating modules
* Event propagation from modules to the PMON block can use a circuit switched network consisting of a daisy chain of muxes, whose select lines are based on the local PMON select control states obtained from the PMON modules
* In a ring based approach, ring-master seems to be a suitable place to host the PMON controller. It can use the ring for broadcasting PMON setup and another ring circuit for gathering the events. This is the ATS architecture.
* In a star topology, PMON controller can be considered as a host on the regbus layer. It acts as slave for register accesses to the PMON registers. It also acts as a master to broadcast these control states to participating modules. Wired event network has to be constructed from the participating modules to the PMON host.

## Message mastering from slaves (interrupts and errors)

* Interrupt and error events should have the capability of sending messages to a suitable host on the NoC layer.
* This requires the CSRs to have mastering capability to inject messages into the NoC in addition to the read and writes responses when it functions as a slave for registers accesses.
* Packet based interrupt and error messages allow inclusion of syndrome details and ID of the originator

## Domain crossings

RBC, RBM and regbus transport layer can operate in a single clock domain. CSRs and ring slaves need to be synchronous to the module hosting them. Clock crossing from transport to slaves will add latency and register costs (for maintaining full throughput across the crossing). Options have been described in earlier sections.

## Functional safety

Regbus layer can reuse the FuSa features developed in the transport layer. However the slave end modules will have to be enhanced to work with these transport features

## Low power and clock gating

Regbus infrastructure should support the new AMBA low power scheme for arbitrary power domains across the system. Fence and drain functionality and support for new handshake mechanism with NSPS will have to be built into RBM. Low power support plan from RBC’s perspective is yet to be developed.

RBM, transport infrastructure, slave bridges and CSRs should all support aggressive activity based coarse clock gating and synthesis based fine clock gating. At the broadest level RW registers only need to be clocked during a write. Status registers need clock during status update and writes. RSSB based transport will inherit the coarse clock gating functionality built into the normal transport layer. Ring master and ring slaves need to be clocked based on register activity targeting the node.

## Broad next level questions

Requirements and implementation details for these are not clear at this point. But we have to put enough thought here to be able to add these seamlessly to the architecture in future.

* Supporting write multicast capability
  + HW autonomous multiple unicasts?
  + True network multicast?
  + Has implications on address space definition, lookup etc.
* Register reset requirements from Intel model? Cold reset, warm reset?
* Any retention register requirements?
* Save and restore requirements for low power?
* Multiple masters on the same regbus layers?
* Regbus layer hardware used for other sideband traffic? E.g Low power handshake messages.

# Micro Architecture

## RegBus Overview

The Regbus is used to access NoC CSRs and optionally allow access to user CSRs. Users have the option to configure access to the regbus using a separate interface agent or to tunnel into it from the NoC to allow the NoC hosts to access the RegBus.

The RegBus Controller, RBC, provides interfaces and the optional tunnel to allow user access to the RegBus. The RegBus Master, RBM, is the root of the regbus network. Register Bus transactions are routed to the CSR modules through RSSB routing elements, Slave Decoders and ultimately to RegBus Slave modules that interface directly with the CSR blocks.

Two network topologies are planned, a “Star” connection of RBS modules and “Ring” connection of RBS modules. High level block diagrams are shown below.

Figure 1 RegBus Star Topology



Figure 2 RegBus Ring Topology



The signals used between SD and RBS are the same whether in Star Mode or Ring Mode. The naming, will be the same as the previous Regbus generation names, *ring\_\**. But there are a few functional differences in some of the signals involved. In the ring mode, the credit signals operate as true credit signals, indicating space in a FIFO. In the Star mode, the credit signals operate more like a ready signal.

## RegBus Packet formats

The packet format for regbus operations is being designed to transfer only needed information in as few flits as possible. The packets are made up of 36-bit flits, but transfers may be 9-bits or 18-bits. When in the Ring configuration, the RingID needs to be in the first transfer so the slave knows whether to forward or not. The Command also needs to be in the first transfer to so that the Slave can calculate end-of-packet, EOP.

The Command encoding is dependent upon the RingID. It is encoded as shown in Table 1.

Table 1 RBS Command Encoding

|  |  |  |
| --- | --- | --- |
| **Ring ID** | **Cmd Encoding** | **Meaning** |
| !=0x00 & != 0x1f | 3’b000 | RD32 |
| 3’b001 | RD64 |
| 3’b010 | WR32 |
| 3’b011 | WR64 |
| 3’b100 | Msg to Slave (32b or 64b?) |
| 3’b1x0 | Reserved |
| 3’b11x | Reserved |
| 0x00 | 3’b000 | RD32 Resp |
| 3’b001 | RD64 Resp |
| 3’b010 | WR32 Resp |
| 3’b011 | Reserved |
| 3’b100 | Msg to Master (32b or 64b?) |
| 3’b1x0 | Reserved |
| 3’b11x | Reserved |
| 0x1f | 3’bxx0 | Broadcast32 (WR) |
| 3’bxx1 | Broadcast64 (WR) |

Do we need both 32-bit and 64-bit Broadcast operations?  
Do we need both 32-bit and 64-bit Message formats (to/from Slave).

It is important to note that unneeded address bits are not transmitted. This includes bits beyond bit 14 and lower 2 bits of the address. Since the minimum request size is 32-bits, the lower 2 bits of address will always be zero.

The Host Regbus bit is used to access Host Registers when Host Regbus is asserted. The Converter bit is used for those AMBA protocol that have a converter for first converting the AMBA protocol to AXI4L prior to sending to the common core module (ns\_acemstrbrdg\_core or ns\_aceslvbrdg\_core).

Figure 3 RBS Read Request Packet format

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Flit 0 | 35:23 | 22:16 | 15 | 14:10 | 9 | 8:6 | 5 | 4:0 |
| Address[14:2] | Reserved | Secured | SeqNum | Converter | Command | Host Regbus | Ring ID |

Figure 4 RBS Write Request Packet format

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Flit 0 | 35:23 | 22:16 | 15 | 14:10 | 9 | 8:6 | 5 | 4:0 |
| Address[14:2] | Reserved | Secured | SeqNum | Converter | Command | Host Regbus | Ring ID |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Flit 1 | 35 | 34:27 | 26 | 25:18 | 17 | 16:9 | 8 | 7:0 |
| Wstrb[3] | Data[31:24] | Wstrb[2] | Data[23:16] | Wstrb[1] | Data[15:8] | Wstrb[0] | Data[7:0] |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Flit 2 | 35 | 34:27 | 26 | 25:18 | 17 | 16:9 | 8 | 7:0 |
| Wstrb[7] | Data[63:56] | Wstrb[6] | Data[55:48] | Wstrb[5] | Data[47:40] | Wstrb[4] | Data[39:32] |

Flit 2 only exists for Write64 request packets. When there is an error response both data flits will be returned with all zeroes.

Figure 5 RBS Read Response Packet format

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Flit 0 | 35:25 | 15 | 14:10 | 9 | 8:6 | 5 | 4:0 |
| Reserved | Error | SeqNum | Reserved | Cmd=3'b00x | Reserved | RingID=0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Flit 1 | 35 | 34:27 | 26 | 25:18 | 17 | 16:9 | 8 | 7:0 |
| 1'b0 | Data[31:24] | 1'b0 | Data[23:16] | 1'b0 | Data[15:8] | 1'b0 | Data[7:0] |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Flit 2 | 35 | 34:27 | 26 | 25:18 | 17 | 16:9 | 8 | 7:0 |
| 1'b0 | Data[63:56] | 1'b0 | Data[55:48] | 1'b0 | Data[47:40] | 1'b0 | Data[39:32] |

Flit 2 only exists for Read64 response packets.

Figure 6 RBS Write Response Packet format

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Flit 0 | 35:25 | 15 | 14:10 | 9 | 8:6 | 5 | 4:0 |
| Reserved | Error | SeqNum | Reserved | Cmd=3'b010 | Reserved | RingID=0 |

## RegBus Controller (RBC)

## RegBus Master (RBM)

## RegBus Slave decoder (SD)

### Overview

The RegBus Slave Decoder (SD) module is a bridge with the primary responsibility to bridge between a NOC packet and a RegBus Packet. It sends/receives RegBus packets in Star or Ring mode. The format of the RegBus packets are shown in section 5.2 above.

In Star mode, the SD will have multiple ring interfaces that are directly connected to a Regbus Slave component of a router or bridge. In Ring mode, SD will just have a single ring interface which connects to up to 31 RegBus Slave components of a router or bridge. The 31 limit comes from the 5-bit Ring ID and 1 Ring ID, 0, being reserved for the Slave Decoder.

The NoC and Ring interfaces are all REGBUS\_RDATA\_WIDTH bits wide which can be a minimum of 9-bits. The packets all have ring ID and command in the first flit to allow this to happen. The Ring ID is needed so that a RegBus Slave component knows if it needs to assemble the packet locally or simply forward the packet the next component on the ring. The Command is needed in the first flit to determine the packet framing, end-of-packet.

When Star mode is enabled, the SD will decode the ring ID from the RX packet and use that to determine which interface is to receive the packet. NocStudio will need to provide a mapping table to map a Ring ID to an SD port.

### Block Diagram

The high-level Block Diagram for the Slave Decoder data path is shown in Figure 7. The SD supports both a ring topology and a star topology. When in the Star mode and there are multiple outstanding responses pending it is possible that multiple responses (or RegBus slave generated messages) to be pending. This creates the condition that arbitration is needed to choose a responder to service. Because most of the CSR modules will have the same delay, it is likely that the responses will be returned in order but it isn’t guaranteed.

The Request and Response paths are completely decoupled because the Slave Decoder does not need to track outstanding responses. When a response is returned, it is always destined for the RBM. This also simplifies the handling of RBS generated messages as SD doesn’t need to do anything special other than to convert to a NoC packet and send it on to RBM.

Figure 7 Slave Decoder Block Diagram



### RX NoC Interface to Ring Interface(s)

The SD has a single RX interface to RSSB. This is a standard RSSB interface that is REGBUS\_RDATA\_WIDTH bits wide that uses credit flow control. An interface FIFO is instantiated for receiving request flits and managing RX credits. Whenever a flit is read from the FIFO, a credit is returned to RX NoC interface.

Since RBM has already formatted the packet correctly there is minimal conversion that needs to be done. SD doesn’t not need to track the packet boundaries going out on the ring interface(s) as the NoC interface already has an end-of-packet (EOP) indication that can be used. That should already match what is needed for the ring.

RTL assertions to detect a mismatch between NOC and RegBus packets?

### Ring Interface(s) to TX NoC

Responses or slave generated messages that arrive at the input ring interface(s) are simply re-formatted to become NoC flits and forwarded. But when in Star mode, there needs to be some arbitration and flow control to prevent an incoming flit from transmitting until the previous packet has completed. In general most CSR modules will have the same fixed delay so it is typical that multiple packets never start at the same time. But with User RegBus and Converter cases it is possible that there is additional latency that does result in simultaneous responses from the multiple interfaces in Star mode. A simple Round-Robin arbitration will be used to ensure no starvation. Once a packet has been granted, it will remain selected until the entire packet ends.

## RegBus Slave (RBS)

### Overview

The RegBus slave supports two topologies, Star mode and Ring mode. In Star mode, all requests are destined for the associated CSRs or Host CSRs (User RegBus). The goal is to keep the data path as similar as possible between the 2 modes, only the control changes and that should be as minimal as possible.

One of the goals of this module is to be as small as possible since there will be many instances of it in a NoC. Another goal is improve the throughput of the RegBus requests both for latency and bandwidth.

### Block Diagram

The high-level block for the RBS is shown in Figure 8. It shows the major registers and the data paths needed for both the Star mode and the Ring mode.

The CSR Request register contains a Header and 2 36-bit data registers in order to be able to atomically perform a 64-bit register write. These registers will be loaded from the interface or the optional 2-deep interface fifo. The interface fifo is REGBUS\_RDATA\_WIDTH bits wide so the CSR Request register needs to support partial loading to form a full 36-bit word. The data registers can be re-used for Read Data responses because they are only needed for Request or Response and never both. There is a separate register, Hdr\_rsp, for the response from the CSR module. This allows for the Hdr\_req register to be re-used sooner in order to support a higher request rate. The SeqNum field of the Hdr\_req register is loaded into the Hdr\_rsp when a CSR response is returned. The Hdr\_rsp register only has Command, SeqNum, and Error fields as the remaining fields are 0’s.

When operating in the Star topology all responses go directly to the interface and divided down to the REGBUS\_RDATA\_WIDTH for sequencing back to the Slave Decoder. When operating in the Ring topology, responses must be coordinated with the ring transfers that are destined for other slaves on the ring.

Figure 8 RBS Block Diagram



Figure 9 shows the logical timing of RegBus read transactions to RBS for a 36-bit REGBUS\_RDATA\_WIDTH. Figure 10 shows the case for a 32-bit Write operation. The Interface FIFO is not shown as that only add delay to the interface. As can be seen, the RBS is capable of processing a request every other cycle.

One thing to note with this timing is that it requires a change to all CSR modules to remove the internal delay that it asserts in its existing design inserts. This change allows for *regslv\_rsp\_valid* to be asserted in the same cycle as *regslv\_req\_valid*. The overall response timing is the same as the *regslv\_rsp\_valid* loads the Hdr\_rsp register and possibly D0/D1 registers. This provides a direct registered out interface instead of having the read data mux directly in the data out path.

Figure 9 36-bit REGBUS\_RDATA\_WIDTH Read32 Transaction Timing



Figure 10 36-bit REGBUS\_RDATA\_WIDTH Write32 Transaction Timing



### Interfaces

The RegBus Slave interface doesn’t change from the earlier ring interface signals.

Table Ring Interface Signals

|  |  |  |
| --- | --- | --- |
| **Signal Name** | **Direction** | **Interface** |
| ring\_data\_in\_valid | input | input ring |
| ring\_data\_in | input | input ring |
| ring\_credit\_out | output | input ring |
| ring\_data\_out\_valid | output | output ring |
| ring\_data\_out | output | output ring |
| ring\_credit\_in | input | output ring |

Independent of the RegBus topology these same signals will be used. In the Ring mode, these signals will operate as they did in previous generations of the Regbus ring. In the Star mode, these signals will mostly operate the same. The only difference is that the ring\_credit signals will act as a “ready” signal that indicates that the interface is ready to take another transfer.

### Interface FIFO

The RegBus Slave contains an Interface FIFO to allow for credit based flow control on the ring input interface when in Ring mode. This FIFO is typically 2-deep to allow for 1 entry to be written while the other is being read from. This FIFO has the same width as the register bus (REGBUS\_RDATA\_WIDTH). Whenever an entry is read out and written into the CSR Request register, credit is returned to the interface to allow for more transfers. This also logically removes the flit from the interface FIFO.

When in the Star mode, the interface FIFO becomes a pipe stage register along with a valid signal. When not valid, the “ready” (*ring\_credit\_out*) signal will be asserted. When the valid signal is asserted, the flit will be loaded into the next portion of the CSR Request register if it is not busy.

The functional behavior of the interface FIFO with respect to the Request register and response out is shown in Figure 11 for the Write32 case. The 32-bit write case is shown as it requires more interface transfers for a complete request than a read request would. In the case of read requests, multiple requests would be loaded into the interface FIFO when the CSR Request register is busy.

Figure 11 Interface FIFO functional behavior -- Writes



### CSR Request/Response Registers.

The CSR Request Header (Hdr\_req), Response (Hdr\_rsp), and Data registers are used to hold the request information for the all transactions to the CSR module(s) of the component. The Data registers are dual purpose in order to reduce area, the can hold Write request data or Read response data. For bandwidth reasons there are separate request and response header registers. This allows the request register reload to happen while a response is being sent out to the ring or SD response interface.

The two Data registers, D0 and D1 are 36-bits wide and are loadable at the REGBUS\_RDATA\_WIDTH granularity. The D1 register is only loaded for 64-bit write requests or 64-bit read responses. The Hdr\_req register only saves the needed bits, the reserved and RingID fields are not saved in this register. This makes the Hdr\_req register only a total of 24-bits.

When the Regbus data width is less than 36-bits, a counter will be used to indicate which portion of a CSR Request/Data Registers is to be loaded with the next load. That counter will be part of the Receive Request Packet tracking logic. It will be used to send Register and segment write enable signals to the CSR request registers.

The CSR response register is loaded all in the same cycle when the response from CSR module is valid, *regslv\_rsp\_valid*.

The register are loaded using the *req\_reg\_{hdr, d0, d1}\_wren* signals from the RX FSM. These signals are all 4-bits to support the 9-bit REGBUS\_RDATA\_WIDTH. When the Regbus is wider than 9-bits then multiple bits will be asserted at a time. Should I be worried about the minor code coverage difference due to this?

In Ring Mode, requests not destined for this regslv are not written into the CSR Request register.

#### Request Register Ready (csr\_reqreg\_ready)

The *csr\_reqreg\_ready* signal is used to indicate when the CSR Request register is ready to receive something from the intf\_fifo. The RX FSM, which indicates the type of flit in the next entry of the intf\_fifo, will generate write enable signals for the Header (hdr\_req\_reg), Data word0 (D0\_reg), and Data word1 (D1\_reg) registers. See 5.6.6 below.

Once *req\_reg\_hdr\_wren*[msb] has been asserted, the *csr\_hdr\_valid* register will be set to indicate that a valid command is present. If the command is a read type command, the *csr\_reqreg\_ready* will be de-asserted until the response has been received and processed. If the command is a write, the *csr\_reqreg\_ready* signal will remain asserted until the proper amount of data has been written into the D0 and/or D1 registers. When the *req\_reg\_d0\_wren*[msb] asserts with *csr\_reqreg\_ready* asserted, the *csr\_d0\_valid* will be set to indicate that D0\_reg has been written. For WR64 requests, *csr\_d1\_valid* signal will use the *req\_reg\_d1\_wren[*msb*]* to set. The *csr\_\*\_valid* signals will be cleared when regslv\_rsp\_ready asserts and *regslv\_req\_valid* is asserted. At that point it is the TX FSM tracking validity of the CSR interface.

The tricky part is that in order to meet the performance goals, read request packet every other cycle, we need to able to accept the next header as soon as the response header has been sent on the *ring\_data\_out* interface. This means that if read response data is delayed going out to the ring due to lack of credits, then csr\_reqreg\_ready must be de-asserted to prevent the data register from being overwritten. It is for this reason that *regslv\_rsp\_ready* signal from the TX FSM must be factored into the csr\_reqreg\_ready signal.

#### CSR Request Valid

The CSR Request Valid, *regslv\_req\_valid*, signal is used to indicate to the CSR module that there is a

#### Assertions

Potential assertions that could be written to check consistency between the RX FSM and the CSR Request register.

* (rx\_state != HDR) && regslv\_req\_valid
* (rx\_subword\_count != 0) && regslv\_req\_valid

### Receive Packet Processing

All transfers that are received from the Slave Decoder are written into the Interface Fifo (section 5.6.3). The RX FSM (*rx\_state*) is used to track the packet boundaries of the flit at the head of the FIFO. The FSM state indicate the contents of that flit, whether it is a header or which part of the data it is. Since the Interface FIFO width is REGBUS\_RDATA\_WIDTH, a counter will be used to indicate which portion of flit is present. The combination of the FSM State and the Counter are used to indicate which part of the request register should be written into when it is ready to receive a request. The RX FSM is shown in Figure 12 for the Star Mode configuration.

When in the Ring Mode configuration the transition equations will change slightly to take into account whether there is a ringID match or not.

Figure 12 RX FSM



The logical timing is shown in Figure 13 and Figure 14 for REGBUS\_RDATA\_WIDTH of 36-bits and 18-bits, respectively.

The *full\_word* signal is generated based upon the REGBUS\_RDATA\_WIDTH to determine the counter value to use that along with the command is used to determine the next state. When REGBUS\_RDATA\_WIDTH is 36-bits, the Intf FIFO data out needs to be used to determine *is\_write* when in the HDR state. In all other cases, the Hdr\_req command field can be used since it is the first thing loaded.

Note that for the timing diagrams shown in Figure 13 and Figure 14 the *csr\_reqreg\_ready* is just approximate and may not actually be possible due to actual time required to send responses on the link. A realistic result would likely have *csr\_reqreg\_ready* de-asserted longer.

Figure 13 RX Timing: 36-bits



Figure 14 RX Timing: 18-bits



### Transmit Response Packet

An FSM will be used to track CSR access responses being sent to the *ring\_data\_out*. This Tx FSM is responsible for generating the *regslv\_rsp\_ready* signal to indicate that the Hdr\_rsp and Data registers can be reloaded. The state diagram for the TX FSM is shown in Figure 15. The FSM shows the major states. When the REGBUS\_RDATA\_WIDTH is less than 36, there will also be a counter to track the minor states. The minor states indicate which chunk of the 36-bit word is currently being sent.

These states are used to select the correct REGBUS\_RDATA\_WIDTH piece to put on the *ring\_data\_out* interface signals.

Nothing extra is required when in Star mode. When in Ring mode extra information, *tx\_src\_is\_fwd,*  is required to know whether to select an internal packet or if we are simply forwarding a packet from the ring (interface FIFO). Whenever there is an internal packet to send, it will take precedence over a packet to be forwarded. This decision is only made at packet boundaries, that is after the current packet as completed then a new choice can be made.

Figure 15 TX FSM



This signal will unconditionally de-assert following the sampling of *regslv\_rsp\_valid* being asserted. It will reassert as soon as it knows that it can complete the current response prior to the register being re-used. For example a RD32 response requires 2 *ring\_data\_out* cycles. Once the response Header is on *ring\_data\_out* interface, *regslv\_rsp\_ready*, will assert in the following cycle if there is another credit available for sending the data. If it is a RD64 response then it will check again when the first data is on the *ring\_data\_out* and de-assert *regslv\_rsp\_ready* if there isn’t an extra credit for the second data flit to go out.

Figure 16 FX Timing: 36-bits



Figure 17 FX Timing: 36-bits with credit delays



#### Reg slave Response Ready (regslv\_rsp\_ready)

The *regslv\_rsp\_ready* signal is used to indicate whether it is ok to load a response into the response header resgister (Hdr\_rsp) and the data into the D0/D1 registers. When asserted and a response is ready from the CSR module, then the response and data will be written into the response registers.

If there are enough credits available to send the response header and any response data, then the *regslv\_rsp\_ready* signal will remain asserted. Otherwise it must remain de-asserted to prevent a new write request from storing data in the D0/D1 register that contains read response data that is still to be sent.

*reglsv\_rsp\_ready\_nc = (tx\_next\_state == IDLE) || (tx\_next\_state == HDR) || ((tx\_next\_state != HDR) && (ring\_credit\_count > 1))*

#### Ring Data Out

The ring data out interface in the star mode simply needs to take into account the out response from the CSR module. See the equations below. The Ring mode behavior is similar, the interface fifo not empty indication is used in place of the *regslv\_rsp\_valid* signal.

*ring\_data\_out\_valid = rsp\_credit\_avail && (tx\_state != IDLE)*

*ring\_data\_out = tx\_src\_is\_fwd ? intf\_fifo\_dout: 🡸 Ring mode only  
 (tx\_state == HDR) ? Hdr\_rsp\_reg :   
 (tx\_state == D0) ? d0\_reg :  
 (tx\_state == D1) : D1\_reg;*

### Response Credit Management

Regbus slave will maintain a counter, *ring\_credit\_counter*, to keep track of ring\_out interface credits. It will be incremented whenever *ring\_credit\_in* is asserted and will be decremented whenever *ring\_data\_out\_valid* is asserted. At reset this counter will be set to the value of 2.

Whenever the *ring\_credit\_counter* is non-zero, there are credits available and a response can be passed to the ring or an incoming flit, destined for another ring slave, can be forwarded to the *ring\_data\_out* interface.

Should the ring\_credit\_counter be initialized to a constant value of 2 like today or should this be a parameter?

### Interface to CSR modules

Requests that are destined for this Regbus Slave module are sent to the CSR interface via the CSR interface registers. It is expected that the typical CSR module will provide the response immediately, in the same cycle as *regslv\_req\_valid* asserts. Those CSR modules that have User Regbus enabled or have a converter may take longer to generate a response.

Because Regbus slave provides this interface from a register there is no need for the typical CSR module to register any of the request signal or response modules. It is implementation specific how that

### Reg Slave Generated messages.

As of this writing (Aug 2019), there are not any defined messages from Reg Slave. It is envisioned that when full PMON support is added NocStudio, certain error conditions will result in an error message will be generated to report this condition.

This section will simply discuss the characteristics that will need to be supported.

* The message will be sourced from the CSR module such as for the PMON error indicated above.
* The message will have a header and 1 (2?) word(s) of data which is the actual message.
  1. In the case of a PMON error, the expectation is that data word will be the error log information.
* A separate interface, msgif, will be used with the CSR to manage these messages.
* The Regbus Slave will need to inject this onto the ring\_data\_out interface and it will be with fixed priority relative to other ring operations.
  1. CSR generated message
  2. CSR generated response
  3. When in Ring mode, a ring forward to another ring element is the lowest priority.

## Future

List of future things to be added to the RegBus design.

1. Broadcast/Multicast messages
   1. Always to all components or just to some?
   2. Need to define the high-level flow of these messages.
2. Slave Generated Messages
   1. What messages should we support?
   2. Need to define the high-level flow of these messages.
3. RBM Generated Messages
   1. Need to define the high-level flow of these messages and their content.