QuickRing™ Client Interfaces

National Semiconductor
Application Note 942
Webster (Rusty) Meier
Paul Sweazey
Shilpa Parikh
Stephen Kempainen
June 1994

1.0 INTRODUCTION

This application note contains a general discussion of QuickRing Data Stream Controller (QRDSC) Client interface techniques, Non-Bridge Mode ($\overline{\text{PIPE}}$ asserted). This application note discusses a variety of Client Interface application issues, including:

— QuickRing Control Protocol (QRCP)
— The Receive Client Header Routing Symbol Transformation (Received Stream ID → Transmit Stream ID Transformation)
— Methods of Transmitting/Receiving Data

The following Hardware design options are considered for the QuickRing Client interface:

— Dedicated/Remote CPU/Microcontroller
— Hardware DMA controller
— Local FIFOs
— Multiplexed/Separate QuickRing Transmit/Receive Client Ports
— Programmable Logic to support the particular interface configuration

It is assumed that the reader is familiar with the QuickRing QR0001 Controller Data Sheet, the ''QuickRing Control Transactions'' specification, and FPGA/PAL®/GAL® design.

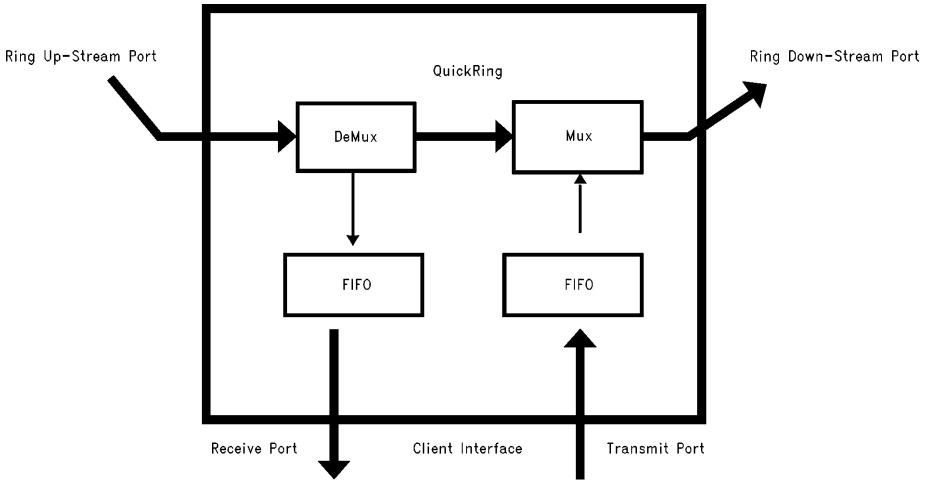
2.0 QuickRing CLIENT PORT DESCRIPTION

The QuickRing Data Stream Controller (QRDSC) contains 4 ports, the Ring Up-Stream and Down-Stream Port and the Transmit and Receive Client Interface. The System Interface is through the Transmit and Receive Client interface. To achieve the highest level of performance the QRDSC Transmit and Receive Ports are separate 32-bit ports, though these two ports can be multiplexed together if desired (see *Figure 1*).

2.1 QuickRing Controller Transmit Port

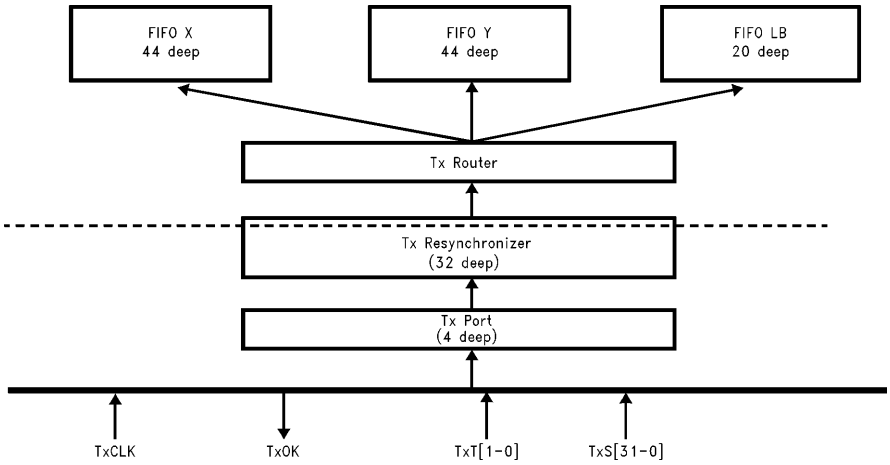
The QRDSC Transmit Port interface consists of (see *Figure 2*):

— 32 Bits of Input Symbol Data
— 2 Bits to represent the QRDSC input type field
— TxOK, Transmit OK HandShake Output Signal
— TxCLK, Transmit Clock Input Signal



TL/F/12044–1

FIGURE 1. The QuickRing Controller Has Four Ports, 2 Ring Ports and the Transmit and Receive Client Ports



TL/F/12044–2

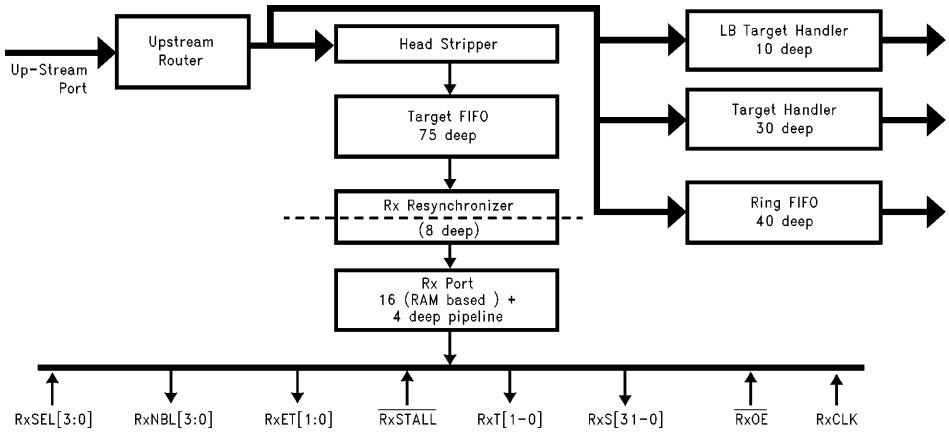
FIGURE 2. Transmit Port Client Interface

The Inputs and Outputs of the Client Transmit interface are synchronous to the ''TXCLK'' input. This application note discusses interfacing to the QRDSC in the Non-Bridge mode ($\overline{\text{PIPE}}$ asserted). In the Non-Bridge Mode the Transmit Port Type field lags the Transmit Port Symbol by one clock. TxOK negating means that the Transmit FIFO can hold a maximum of 20 more Non-Null symbols.

2.2 QuickRing Controller Receive Port

The QRDSC Receive Port Interface consists of (see *Figure 3*):

— 32 Bits of Output Symbol Data
— 2 Bits representing the QRDSC output Type field
— $\overline{\text{RxSTALL}}$ input, when asserted holds the first Non-Null Symbol (RxS[31:0]). This symbol will be held until $\overline{\text{RxSTALL}}$ is negated. This input is sampled on the Falling clock edge of RxCLK.
— $\overline{\text{RxOE}}$ input enables the Receive Symbol when asserted, this input can be used in a system design where the Receive and Transmit Ports are Multiplexed together.
— RxSEL[3:0] Receive Select inputs select the output (Diagnostics Register bits vs Current received Symbol bits) on the RxNBL[3:0] pins.
— RxNBL[3:0] Receive Nibble outputs one of 16 selectable fields of two readable internal areas (Diagnostics Register or Current Output Received Symbol).
— RxET[1:0] Receive Early Type outputs the type information that is entering the Receive Port block, this may be between two to twenty symbols ahead of current output type.
— RxCLK, Receive Clock Input signal.



TL/F/12044–3

FIGURE 3. Receive Port Client Interface

There are three primary signals that control the Receive Port: RxCLK, $\overline{\text{RxOE}}$ and $\overline{\text{RxSTALL}}$. RxCLK should be the constant system clock of the client. $\overline{\text{RxOE}}$ may be permanently asserted in systems where the Client Transmit and Receive Ports are not multiplexed together. In Client systems where the Client Transmit and Receive Ports are Multiplexed together (RxS[31:0] and TxS[31:0] are tied together), or where there is no buffering between the Receive Client Port and the Client bus the $\overline{\text{RxOE}}$ pin must be controlled.

In most all systems $\overline{\text{RxSTALL}}$ must be controlled. *Figures 4* and *5* and Table I give the behavior of the Receive Client Port under different conditions of the $\overline{\text{RxSTALL}}$ input.

There are three recommended methods of interfacing to the QRDSC Receive Client Port (Client Interface Method #1, 2, 3):

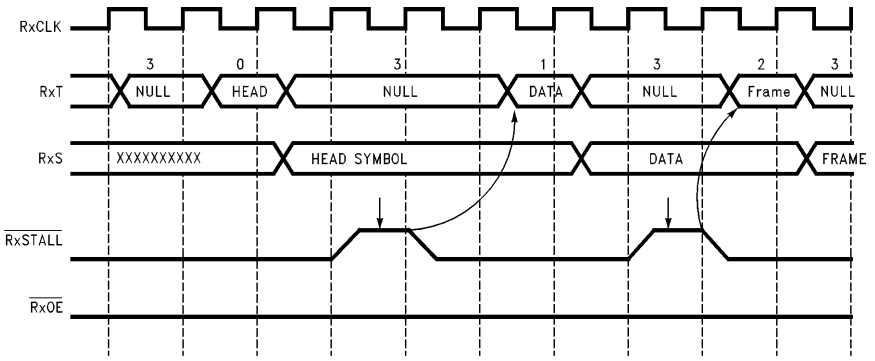
1. Release $\overline{\text{RxSTALL}}$ for one clock cycle only when RxT[ ] takes on a non-null value, leaving $\overline{\text{RxSTALL}}$ asserted at all other times;
2. build a client system that is guaranteed not to require that any symbol remain on RxS[ ] for more than two clock cycles; or
3. provide an external latch to save symbols that might have been lost due to entry to row 10, and stall the next symbol in row 7 or 11 until the external latch can be read.

Client Interface Method #1: Leave $\overline{\text{RxSTALL}}$ normally asserted. This is the easiest method. It restricts the Receive port to rows 1, 2, 5, 6, and 9. $\overline{\text{RxSTALL}}$ is released only to enter rows 1 and 2. This is easily accomplished by releasing $\overline{\text{RxSTALL}}$ in response to each non-null value, but only when that value is no longer needed at RxS[ ]. RxT[ ] will remain non-null for only one clock cycle, and this event will have to be remembered by the client state machine. Unfortunately, because the next state result of row 9 does not update RxT[ ], the fastest that a client may receive data is once every other clock cycle. If there is always another symbol ready in the pipeline, then the Receive Port will toggle between rows 9 and 2, and RxS[ ] will be updated only on the exit from row 9. (See *Figure 4a* and Table I)

Client Interface Method #2: Design a client system that can always consume a received symbol within two clocks. This may be impractical, but if the client system fits this description, then there is no problem. This is because the next state of row 10 is entered in order to stall a symbol that just appeared upon the exit from row 3, and row 10 preserves the symbol for exactly one more clock cycle. Every next-case row out of row 10 will advance RxS[ ] to the next symbol, so the unstallable symbol always appears at RxS[ ] for exactly two clock cycles. (See *Figure 4c* and Table I)

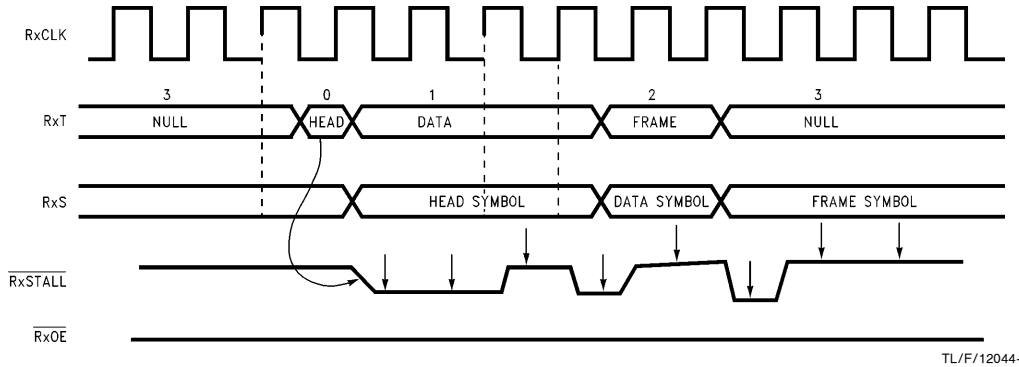
Client Interface Method #3: Provide external storage for the volatile symbol, and stall the next symbol at RxS[ ] until the saved copy can be used. The only entrance to row 10 is from row 3. External logic must monitor the interface for the transition from row 3 to row 10 or 11, and must both save the unstallable symbol and hold $\overline{\text{RxSTALL}}$ asserted until the external copy can be used. This solution requires that external logic emulate the QR0001 Receive Port state machine. Although the T′[ ] input variable and the nullRxS and Stalled state variables are not observable on output pins of the chip, it is possible to compute them externally. An external state machine can deterministically compute the previous state and the value input T′[ ] that helped cause it. Although this information is available one cycle late, there is time to stall the next symbol until the no-stall symbol register is being freed up.

The advantage of alternative 1 is its simplicity, if it is not necessary to receive symbols more often than once every other clock cycle. The advantage of alternative 2 is that, if it happens to describe your system (if it's free), then you will not encounter the no-stall bug. The advantage of alternative 3 is that the client can keep up with high-bandwidth received streams that deliver symbols on every clock cycle.



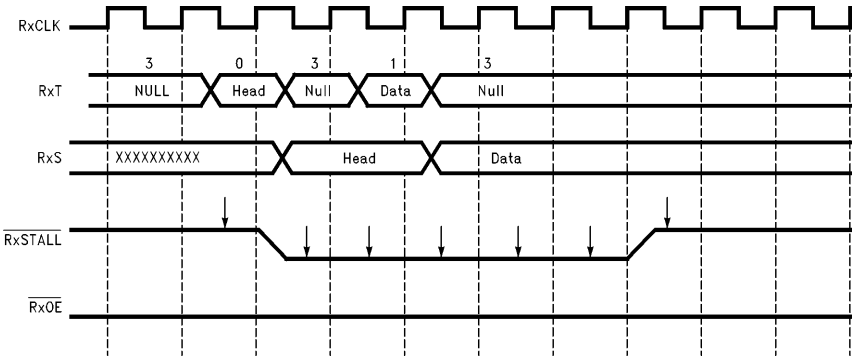
TL/F/12044–19

FIGURE 4a. Client Interface Method # 1, $\overline{\text{RxSTALL}}$ Normally Asserted, $\overline{\text{RxSTALL}}$ Negated only after RxS has been Sampled



TL/F/12044–4

FIGURE 4b. Holding Head Symbols for 2 Clocks and Holding Data and Frame Symbols for 1 Clock Each



TL/F/12044–20

This figure shows how RxS can only be guaranteed to be held for two clocks maximum on QR0001.

FIGURE 4c. Client Interface Method # 2, $\overline{\text{RxSTALL}}$ Normally Asserted

TABLE I

State #	Input		Present State				Next State				Possible Next States
	$\overline{\text{RxSTALL}}$	T[ ]	RxT[ ]	nullRxS	Stalled	RxS[ ]	RxT[ ]	nullRxS	Stalled	Rxs[ ]	
1	F	Null	Null			Sx	Null	T	F	Sx	1, 2, 5, 6
2	F	Ty	Null			Sx	Ty	T	F	Sx	3, 4, 9
3	F	Null	Ty			Sx	Null	F	F	Sy	1, 2, 10, 11
4	F	Tz	Ty			Sx	Tz	F	F	Sy	3, 4, 8
5	T	Null	Null	T		Sx	Null	T	T	Sx	1, 2, 5, 6
6	T	Ty	Null	T		Sx	Ty	T	T	Sx	3, 4, 9
7	T	Ty	Null	F	T	Sx	Null	F	T	Sx	1, 2, 7, 11
8	T		Ty	F		Sx	Ty	F	T	Sx	3, 4, 8
9	T		Ty	T		Sx	Null	F	T	Sy	1, 2, 7, 11
10	T	Ty	Null	F	F	Sy	Ty	T	T	Sx	3, 4, 9
11	T	Null	Null	F		Sx	Null	F	T	Sx	1, 2, 7, 11

$\overline{\text{RxSTALL}}$: Chip input signal; which holds non-mull data at RxS[ ].

T[ ]: (Not externally observable.) The type of the symbol which should appear on RxT[ ] during the next clock cycle unless the symbol marked by RxT[ ] is waiting behind another stalled symbol at RxS[ ].

RxT[ ]: The type code for the symbol that should appear at RxS[ ] during the next clock cycle, unless RxS[ ] is currently non-null and is being paused by $\overline{\text{RxSTALL}}$.

nullRxS: (Not externally provided.) A state variable that is set to TRUE when RxS[ ] is null—its value is volatile (not stallable) and is subject to being overwritten by an arriving symbol.

Stalled: (Not externally provided.) A one-clock-cycle-delayed version of $\overline{\text{RxSTALL}}$.

newRxS: (Not externally provided.) The state machine output that loads the RxS[ ] output with the next non-null value in the pipeline.

RxS[ ]: The 32-bit symbol output bus of the Receive Port.

T: TRUE

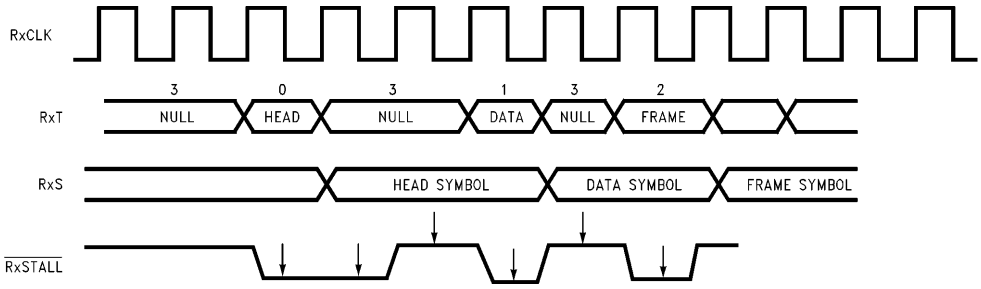
F: FALSE

Null: The symbol type code representing the absence of a symbol.

Ty, Tz: Symbol type codes for a non-null symbol.

Sx: The value of RxS[ ] unchanged from the previous cycle.

Sy: The new value of RxS[ ] whose symbol type code appeared on RxT[ ] during the previous clock cycle.



TL/F/12044–5

FIGURE 5. Null Type is Inserted when $\overline{\text{RxSTALL}}$ is Asserted during the Same Clock Cycle as a Null Symbol (ie. in above Figure Null is Inserted after Head Symbol, See also Table I, State # 9)

3.0 QuickRing PROTOCOL

The QRDSC allows a node to Transmit (Write) data to another node. The data will be transmitted in a Packet. Each Packet will be one Head symbol followed by up to 20 data symbols, each symbol being 32 bits of Data and a 2-bit Type field.

Many applications will benefit from some protocol built on top of the QRDSC basic Write capability. This section will present examples/ideas of higher level protocols. The Reader of this Application Note should also reference the ''QuickRing Control Transactions'', this specification details a higher level protocol similar to the ''QuickRing Control Protocol'' detailed in Section 3.2. Three levels of protocol are presented here:

— No Protocol
— QuickRing Control Protocol (QRCP)
— QuickRing Extended Control Protocol (QRXP)

3.1 No Protocol (Only Write Transactions)

In some dedicated applications no higher level protocol may be needed beyond the basic capabilities inherent in the QR Hardware. For example, an application that was only receiving and buffering data from one other node. In this case just one node might be sourcing data to a particular destination node that is buffering the data being sent (i.e. a graphics display card).

3.2 QuickRing Control Protocol, also Reference ''QuickRing Control Transactions''

Defining a higher level protocol is important to allow multiple vendors to build QuickRing Compatible Nodes. The QuickRing Control Protocol is an example of a higher level protocol that is further described in ''QuickRing Control Transactions''. In the Figures shown below ''SID'' stands for ''Stream IDentification'', ''SID*'' denotes the ''Complement Stream IDentification''. The Complement Stream ID is computed from the Head SID that was received at the Client Receive Port. The Complement SID is used to route data of Responses back to the Requesting Node (see Section 3.5). A QuickRing Control Protocol (see *Figures 6–8*) may include such functions as:

CMD #	Name	Description
0	Read Request	The Read Request Command must be packetized into one QuickRing Packet (1 Head followed by 2 Data Symbols). The Head should be followed by pertinent data (command, address, Byte Enables). The Responder would then send back an Acknowledge/Retry/Illegal Operation Response along with the Requested data. The Responder must be able to complete this transaction even if it is currently handling some other DMA streaming transaction. (See *Figure 6* below.)
1	Write Request	The Write Request Command must be packetized into one QuickRing Packet (1 Head followed by 3 Symbols). This command stores zero to four bytes (within one Quadlet) at the indicated address in the Target node. The Head must be followed by pertinent data (command, address, Byte Enables, data to be written). The Responder would then send back an Acknowledge/Retry/Illegal Operation Response (ACK/RETRY/ILL). The Responder must be able to complete this transaction even if it is currently handling some other DMA streaming transaction. (*Figure 7*.)
2	Event Request	The Write Request Command must be packetized into one QuickRing Packet (1 Head followed by 3 Symbols). This command stores zero to four bytes (within one Quadlet) at the indicated address and generates an interrupt in the Target node. The Head must be followed by pertinent data (command, address, Byte Enables, data to be written). The Responder would then send back an Acknowledge/Retry/Illegal Operation Response (ACK/RETRY/ILL). The Responder must be able to complete this transaction even if it is currently handling some other DMA streaming transaction (similar to *Figure 7*).
3	Atomic Request	The Atomic Request must be packetized into one QuickRing Packet (1 Head followed by 4 Symbols). The Atomic Request can be a Compare and Swap, Mask and Swap, Fetch and Add, and Read and Lock. The Head should be followed by pertinent data (ex. command, address, Byte Enables, Compare data, Swap data). The Responder would then send back the original value of the target location Swap data. For the case where the Swap Data was written to the Target location the Atomic Acknowledge Response (ACK__RSP) should be returned, else the RETRY/FAIL/ILL Response should be returned. The Responder must be able to complete this transaction even if it is currently handling some other DMA streaming transaction. (See *Figure 8* below.)

CMD #	Name	Description
4 to A	RESERVED	
B	Illegal Response (ILL__RSP)	This Response indicates that the requested operation is not supported.
C	Failed Response (FAIL__RSP)	This Response should be sent to indicate that the Atomic Locked Command Request was executed but the attempt Failed (bit not set, the compare did not match and the swap was not performed, the add was not performed, etc).
D	Acknowledge Response (ACK__RSP)	This Response should be sent to indicate that the Read, Write, Atomic Locked Transaction Request completed successfully. If it was a Atomic Locked Transaction Request, then it succeeded (bit set, swap performed, add performed, etc.).
E	Retry Response (RETRY__RSP)	This Response should be sent to indicate that the Read, Write, or Atomic Locked Transaction Request did not complete successfully and can be Re-Tried.
F	Node Reset	This command will place the node in a well defined state, appropriate for re-executing the process of topology discovery, node identification, and system configuration.

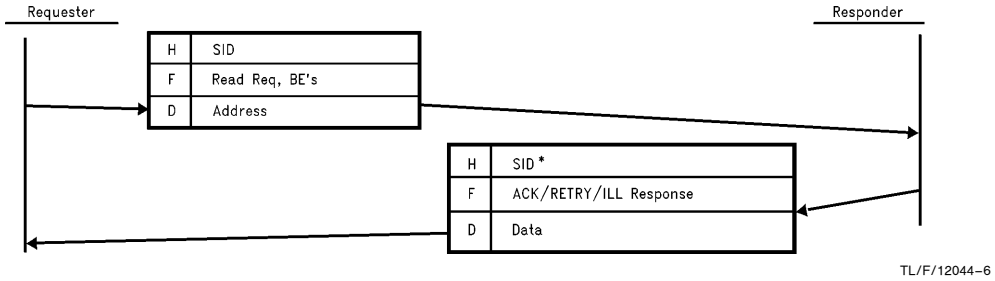


FIGURE 6. Read Request Transaction

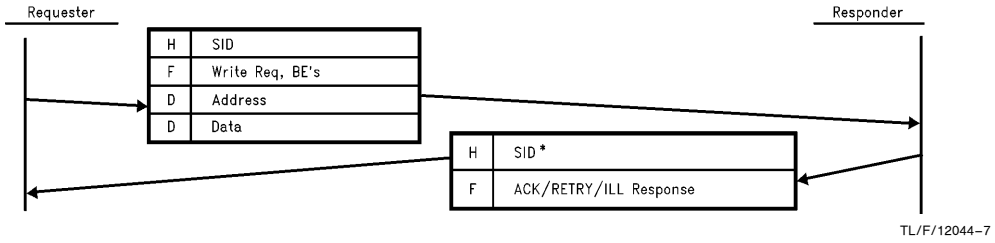


FIGURE 7. Write Request Transaction

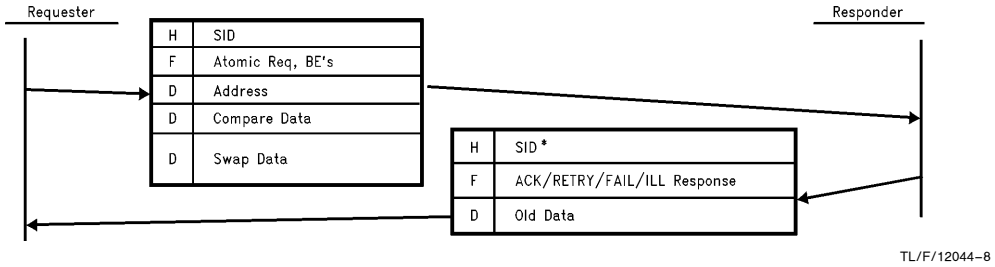


FIGURE 8. Atomic (Compare and Swap) Request Transaction

3.3 QuickRing Extended Control Protocol (QRXP)

This Extended Protocol (QRXP) is an example of extending the QuickRing Control Protocol (QRCP) presented in Section 3.2. Note that this is an example only and not an interoperability standard. This QRXP could be useful in systems that are based upon a Flat Addressing Model. In this model the CPU may perform Memory (or I/O) transactions across QuickRing Nodes. The QRXP (*Figures 6–15*) incorporates QRCP while adding the following features to the Protocol:

- Adds ''Number of Symbols'' (bits 17:14) to the Command Frame Symbol bits. Read/Write/Acknowledge transactions must drive these bits. (Table II and *Figure 9*.)
- The ''#Symbols'' field adds two types of transfers to the protocol: Burst Transfers and Streaming Data Transfers
- Burst Data Transfers are data transfers where more than 1 and 16 or less Data Symbols are transferred. This whole transaction must be incorporated within one QR packet (1 Head followed by up to 20 symbols). The last piece of data must be marked as Frame Type (*Figures 10* and *11*).
- Streaming Data Transfers are data transfers where more than 16 data symbols are to be transferred, thus requiring multiple QR packets sent across the ring. The last piece of data must be marked as a Frame Type (*Figures 12* and *13*).
- Streaming Data Transfers must first make a Request for the transfer and receive an Acknowledge back before any data is transferred (see *Figures 12* and *13*).
- ''Transaction Management'' Command (CMD #A) : This command allows Single/Burst/Streaming Read/Write/Atomic transactions to be ''Terminated'' or ''Inquired'' about (*Figure 14* and *15*).

This Protocol expands the QRCP by adding a four-bit field that indicates the number of symbols to be transferred during the transaction.

TABLE II. Number of Symbols Field Decode Table

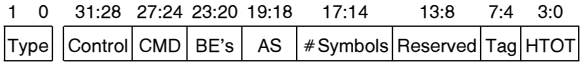
Decode	#Symbols
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	256
8	1k
9	4k
A	16k
B	64k
C	256k
D	1M
E	4M
F	16M

This four-bit field can be added to the following QRXP commands:

CMD#	Name
0	Read Transactions,
1	Write Transactions, and
D	Acknowledge Transactions

The format of the Frame Type Symbol can be seen in the ''QuickRing Control Protocol Specification''. In *Figure 9* ''# Symbols'' (Bits 17:14) have been added to the QROP in the Command Frame Symbol Bit Descriptions.

Command Frame Symbol Bit Descriptions



CMD: Specifies Command (Read, Write,...)

BE's: Specifies Byte Enables for the 32-bit data symbols

AS: Address Space (Memory/IO/Configuration Space)

#Symbols: The number of Data Symbols to be transferred (Read/Write/ACK)

Tag: Data Stream Identification #

HTOT: Hop Count Total (number of Bridges between Source and Destination)

FIGURE 9. QRXP Command Frame Symbol Bit Specification

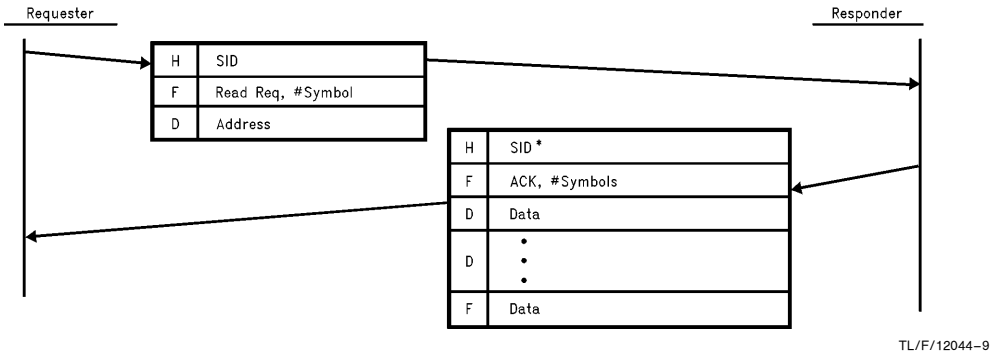
The Added ''Transaction Management'' Command (CMD #A) allows Single/Burst/Streaming Data Read/Write/Atomic transactions to be ''Terminated'' or ''Inquired'' about (*Figures 14* and *15*) . A Requester may Inquire (Terminate) a Single/Burst/Streaming Data Read/Atomic Transaction.

A Responder may Inquire (Terminate) a Single/Burst/Streaming Data Write Transaction. The #Symbols field (bits 17:14) of the Command Frame Symbol (*Figure 9*) are used to encode the 4 Transaction Management commands as follows:

# Symbols Decode	**Name**	**Transaction Management Command Description**
0	Terminate Request	Terminate Data Transaction of specified Stream ID# (Tag)
1	Inquire Request	Inquire about Data Transaction of specified Stream ID# (Tag)
2	Terminate Response	Response to Terminate Request about Data Transaction of specified Stream ID#. The Address and Status of the terminated Transaction will be returned. The Least Significant bit (#0) of the Status field will indicate whether the specified Transaction was In-Progress (''1'' returned) or is Un-Known (Not-In-Progress, ''0'' returned)
3	Inquire Response	Response to Inquire Request about Data Transaction of specified Stream ID#. The Address and Status of the Inquired about Data Transaction will be returned. The Least Significant bit (#0) of the Status field will indicate whether the specified Data Transaction was In-Progress (''1'' returned) or is Un-Known (Not-In-Progress, ''0'' returned).
4:F		Reserved

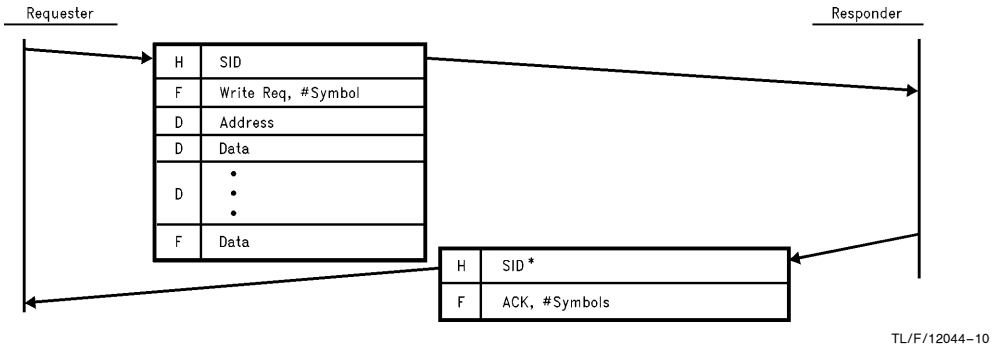
The Burst Read and Write Transactions are first described below:

CMD#	**Name**	**Description**
0	Burst Read Request	The Burst Read Command is the same as the QRDR Read Command except that the Number of Symbols field is equal to or less then 16 but greater than one. This allows the returned data to be in one QR Packet (required). The requester transmits a Burst Read Request to the Destination node along with the number of transfers desired (and BE's, AS, Tag) and the address. The destination (Responder) replies with an Acknowledge and the number of data symbols it can provide. Note, this may be less than requested, if this is the case the Requester must make another request to get all the required data. Also note that the last data symbol is marked by a Frame type (*Figure 10*).
1	Burst Write Request	The Burst Write Command is the same as the QRCP Write Command except that the Number of Symbols field is equal to or less than 16 but greater than one. This allows the transferred data to be in one QR Packet (required) . The requester transmits a Burst Write Request to the Destination node along with the number of transfers desired (and BE's, AS, Tag), the address and the data. Note that the last data symbol is marked by a Frame type. The destination (Responder) replies with an Acknowledge and the number of data symbols it received. Note, this may be less then requested, if this is the case the Requester must make another Write request to transfer the remaining data. See *Figure 11*.



TL/F/12044–9

FIGURE 10. Burst Read Request Transaction

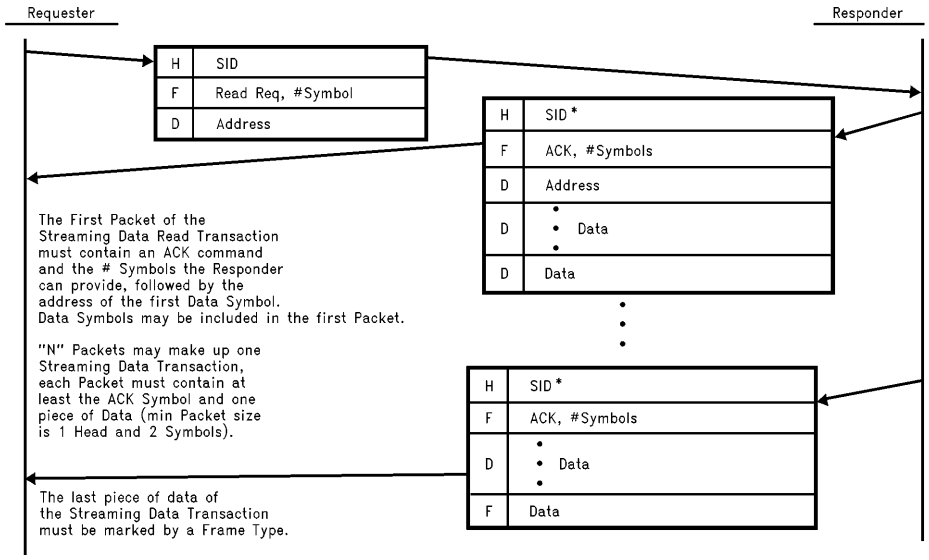


TL/F/12044–10

FIGURE 11. Burst Write Request Transaction

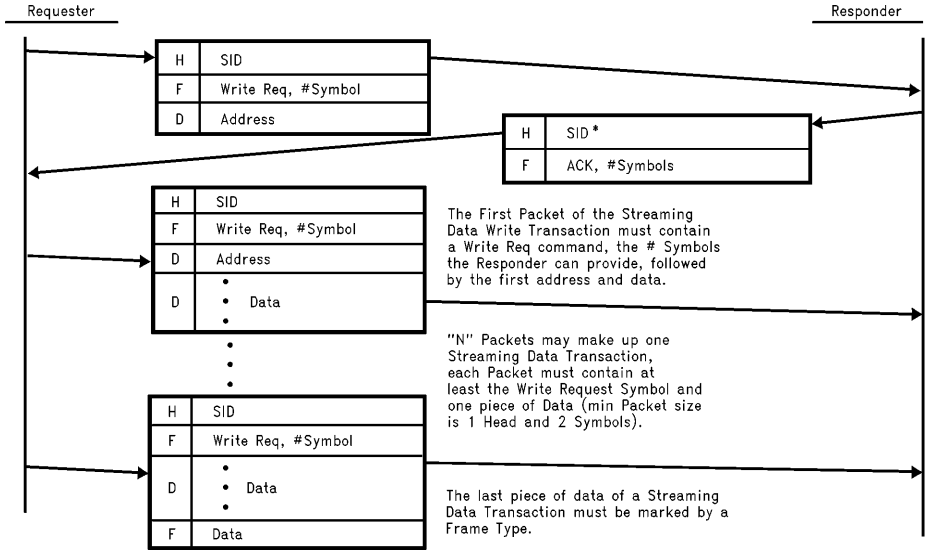
This Protocol has the advantage that Streaming Data Transactions must make a general Request for a Nodes Resource (DMA Channel) across QuickRing before the transaction can take place. In other words Streaming Data transactions are preceded by a Request/Acknowledge Response Handshake. If the Responder does not support Streaming Data Transactions it can return an ACK Command with the specified Number of Symbols (#Symbols) it is capable of handling. The QRXP uses the commands of QRCP with additional features (*Figures 10* to *13*).

CMD#	Name	Description
0	Streaming Data Read Request	The Streaming Data Read Command is the same as the QRCP Read Command except that the Number of Symbols field is greater than 16 symbols (i.e. more than one QR Packet). The requester transmits a Streaming Data Read Request to the Destination node along with the number of transfers desired (and BE's, AS) and the address. The Responder replies with an Acknowledge, the number of data symbols it can provide, an Address (updated each packet), and up to 16 data symbols. Note, the total number of Symbols (#Symbols) may be less than requested, if this is the case the Requester must make another request (after this Streaming Data Transaction is over) to Read the remaining data. Also note that the last data symbol is marked by a Frame type (*Figure 12*).
1	Streaming Data Write Request	The Streaming Data Write Command is the same as the QRCP Write Command except that the Number of Symbols field is greater than 16 symbols (i.e. more then one QR Packet). The requester transmits a Streaming Data Write Request to the Destination node along with the number of transfers desired (and BE's, AS) and the address. Note, No Data is transferred during the initial Streaming Data Request Packet. The destination (Responder) replies with an Acknowledge, and the number of data symbols it can accept. Note, this may be less then requested, if this is the case the Requester must make another request (after this Streaming Data Transaction is over) to transfer the remaining data. Once an ''ACK RSP'' is received the Requesting Node may Stream Data Packets to the Responder along with the address (updated each packet), Byte Enables, Number of Data Symbol Transfers and the data to be written. Also note that the last data symbol is marked by a Frame type. See *Figure 13*.



TL/F/12044–11

FIGURE 12. Streaming Data Read Request Transaction



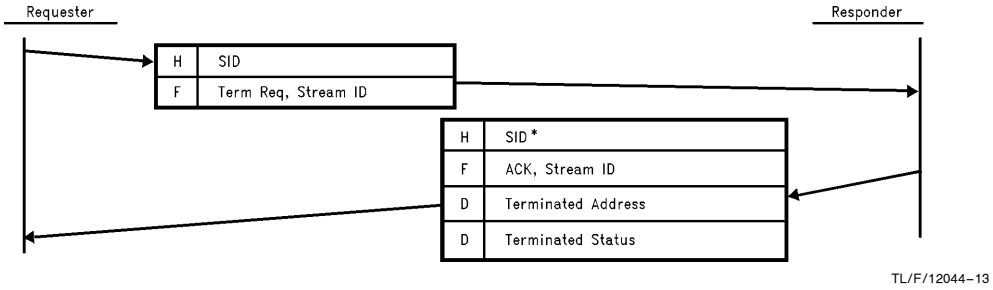
TL/F/12044–12

FIGURE 13. Streaming Data Write Request Transaction

The Added ''Transaction Management'' Command (CMD #A) allows Single/Burst/Streaming Data Read/Write/Atomic transactions to be ''Terminated'' or ''Inquired'' about (*Figures 14* and *15*). This command allows a QuickRing Node:

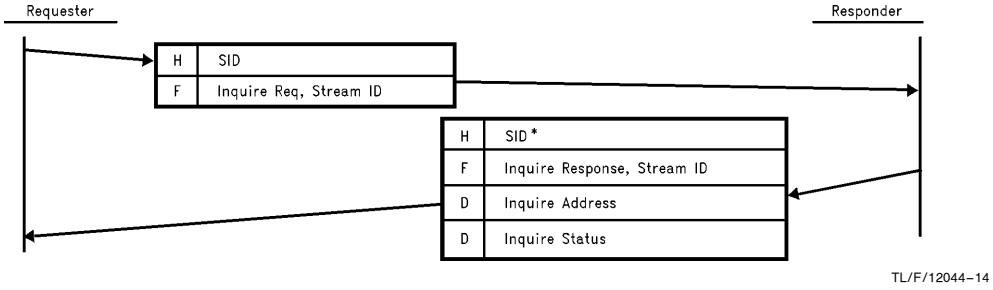
- to inquire about a transaction that it has not received an Acknowledgement from
- to inquire about a large Streaming Data Transaction to receive an Address and Status. The inquiring node can compare the address received to its current DMA address to guarantee no data symbols have been lost or added (DMA count position check)
- to terminate a transaction that has errored

Decode	Name	Transaction Management Commands
0, 2	Terminate Request and Response	Terminate Data Transaction of specified Stream ID# (Tag) and the Response. The Address and Status of the terminated Transaction will be returned. Note that the address returned will be the address of the current data (previous data +1). The Least Significant bit (#0) of the Status field will indicate whether the specified Transaction was In-Progress (''1'' returned) or is Un-Known (Not-In-Progress, ''0'' returned).
1, 3	Inquire Request and Response	Inquire about Data Transaction of specified Stream ID# (Tag). The Address and Status of the Inquired about Data Transaction will be returned. Note that the address returned will be the address of the current data (previous data +1). The Least Significant bit (#0) of the Status field will indicate whether the specified Data Transaction was In-Progress (''1'' returned) or is Un-Known (Not-In-Progress, ''0'' returned)



TL/F/12044–13

FIGURE 14. Terminate Request and Response



TL/F/12044–14

FIGURE 15. Inquire Request and Response

3.4 QuickRing Extended Control Protocol Emulation Using the QuickRing Control Protocol Atomic Lock Command (ex. Compare and Swap)

The QR Extended Control Protocol could be emulated using the QuickRing Control Protocol Atomic Request, such as Compare and Swap. For example, suppose that the Client Receive hardware of a particular node has only a single DMA channel. A bit in a control register in the Destination could be asserted if the DMA channel was in use, if negated then the DMA channel would be known to be available. The Requesting node could perform an Atomic Request Compare and Swap on that bit instead of performing a separate Channel Request/Acknowledge subaction of Section 3.3. Only if an Acknowledge Response (ACK__RSP) was received would the Requesting Channel perform the Streaming Transaction to write data to the Destination Node.

3.5 The Receive Client Header Routing Symbol Return Path Transformation (Received Stream ID → Transmit Stream ID Return Path Transformation)

If a QuickRing Destination Node receives a Packet from a Source Node and wants to return a response to that Source Node the Head symbol of the Received Packet can be complemented to provide the return path back to the Source Node. This process is called the Receive Client Header Routing Symbol Return Path Transformation, or Header Routing Symbol Complement (See *Figure 16* below).

3.6 The QRDSC Client Transmit Port and Packet Size Issues

As data is enqueued into the QRDSC Client Transmit Port, it is Packetized and sent around the ring in Packets. These Packets as large as 1 Head Symbol followed by up to 20 Data/Frame Symbols.

The first Data or Frame symbol enqueued after a Head symbol causes the QRDSC to transmit a Voucher to the Destination Node. The Destination Node returns a Ticket to the Source Node if it has space in its Receive FIFO for 21 symbols (1 Head and up to 20 Data/Frames).

When the Source Node receives the Ticket it sends a Packet to the Destination Node. The Packet will contain the Head Symbol (enqueued in TxFIFO) as well as any data that has been enqueued in the TxFIFO since the initial piece of data.

Table III on next page is based upon simulations and gives an estimate of Packet size based upon how often data is enqueued into the QRDSC Transmit FIFO. The top horizontal line in the chart gives the Number of Nodes in the Ring (2–12), the column on the left side of the table gives how often data is enqueued into the QRDSC Transmit FIFO (i.e. 3 means that one piece of data is enqueued every 3 Tx Clocks).

Fixing the Packet size can allow a performance enhancement in some system designs, particularly those where a CPU or Microcontroller (CPU/M) handles all the Receiving of Data. When the Packet size is fixed the CPU knows that once it sees a Head Symbol it will be followed by ''X'' pieces of data. Therefore the CPU does not need to check the type field for each subsequent symbol (as it would have to otherwise).

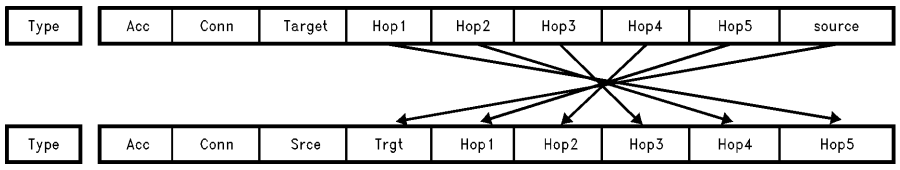
Note that fixing the Packet size means that the Transmitter can guarantee enqueueing data at a certain rate into the QRDSC Client Transmit FIFO, thus guaranteeing a Packet size.

If the system designer wishes to guarantee transmitting Fixed Packets on the Ring there are several considerations to follow:

— Once a full Packet (1 Head followed by up to twenty 32-bit symbols) is enqueued into the QRDSC Transmit FIFO the Client Transmit interface should enqueue another Head Symbol that is different than the original Head symbol (ex. least significant HOP field incremented by a 2-bit counter). This will guarantee that the next packet enqueued does not get concatenated with the previous packet.

— The conditions of Table III must be adhered to. Table III shows the guaranteed Fixed Packet size based upon how often symbols are enqueued into the Client Transmit Interface. Each node, beyond a two node Ring, adds 4 clock cycles to the Voucher/Ticket Latency or 4 clocks of time to accumulate more symbols into the QRDSC Client Transmit FIFO. Also each symbol added to the QRDSC Transmit FIFO adds an additional clock of time to accumulate more symbols.

As a final note it still could be possible to experience a Null between contiguous symbols of a received Fixed Size Packet if RxStall is held negated. This can happen because of Vouchers or Tickets inserted within the Fixed Packet on the Ring. If RxSTALL is asserted and then symbols are received from the ring, Nulls within a packet will be deleted because the Receive FIFO of the QRDSC does not store Nulls. In the case of a CPU/M the condition of Nulls within a Packet should not be encountered because the CPU/M will require multiple clocks to store each received Symbol, thus asserting RxSTALL causing Nulls within the packet to be deleted.



TL/F/12044–15

FIGURE 16. The Header Return Path Transformation (Complement)

TABLE III. QuickRing Packet Size Based upon How Often Data Is Enqueued into the QRDSC Transmit FIFO

CLKs/ Data	Nodes in the Ring										
	2	3	4	5	6	7	8	9	10	11	12
1	20	20	20	20	20	20	20	20	20	20	20
2	16	18	20	20	20	20	20	20	20	20	20
3	8	9	10	11	14	15	16	18	19	20	20
4	6	7	8	9	10	12	13	14	15	17	18
5	4	4	5	6	7	8	8	9	11	12	13

QRDSC Packet Size

4.0 SOME QuickRing CLIENT INTERFACES

There are many methods of Transmitting and Receiving Data at the QuickRing Client Interface. The following list gives several Hardware design options to be considered when designing the QRDSC Client Interface:

— Dedicated/Remote CPU/Microcontroller
— Hardware DMA controller
— Local FIFOs
— Multiplexed/Separate QuickRing Transmit/Receive Client Ports
— Programmable Logic to support the particular interface configuration

The following Client Interface Topics must also be considered when designing a QRDSC Client Interface:

A. How/Where the Transmit Packet is constructed

B. How the Transmit Packet is enqueued into the QRDSC Transmit FIFO

C. How the Receive Packet is dequeued from the QRDSC Receive FIFO

D. Receive Packet Protocol Issues

These Methods and Topics are considered in the following QRDSC Applications.

4.1 CPU Handles All Housekeeping Chores, No Protocol

In this system design the CPU/Microcontroller (CPU/M) takes responsibility for the transmission and reception of data. This method is inexpensive but very low performance since each symbol to be transmitted or received requires multiple CPU/M instruction execution cycles. The efficiency of this method can be enhanced through the use of some of the Protocol options mentioned in Sections 3.2, 3.3, 3.4 and Hardware Options mentioned in Section 4.0 above.

Client Interface Topics

A. The How and Where of Transmit Jacket Construction: The CPU/M could construct each packet (No-Protocol) to be transmitted in its internal RAM (Head, Data/Frame) symbol by symbol. This could be slow since each symbol could take multiple CPU/M instruction cycles to construct/access.

B. Enqueuing the Packet to be Transmitted (Two Methods):

— Method 1 involves the CPU/M transmitting each symbol immediately as it is constructed/accessed in Step A. This method could lead to decreased performance depending upon how many CPU/M instruction (clock) cycles it takes to construct or access the particular symbol. If it takes many clock cycles to construct or access each symbol it would be very hard to guarantee any packet size over several symbols (see Table III).

— Method 2 involves the CPU/M building the packet in its internal Registers. Once the Packet was completed it could be transmitted very rapidly by the CPU, Burst Write to QRDSC Transmit Port. The speed with which the Packet is enqueued into the Transmit FIFO determines the packet size (see Table III). A CPU/M running at 33 MHz or greater will probably take 3–4 clocks for each piece of data transferred.

C. Dequeueing the Received Packet (Two Methods): The Client Interface Method #1 of Section 2.2 would be a good choice for this Client Interface. The ''Client Port Interface Logic'' can watch the Receive type field for a Non-Null Type Field. When a Non-Null type is observed the Interface Logic Block should latch it and hold it available to be accessed by the CPU/M. Once the Symbol (RxS[31:0]) has been captured $\overline{\text{RxSTALL}}$ is negated for one clock cycle to allow the next Non-Null Type and symbol to be output.

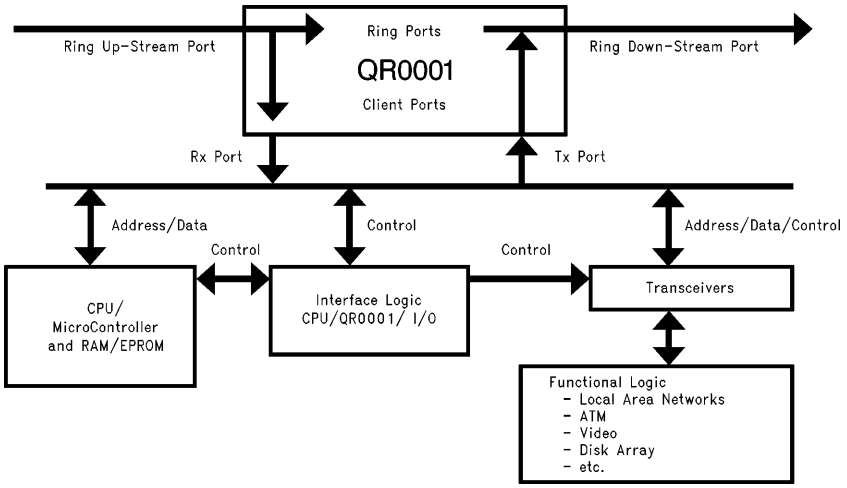
Note that Client Interface Method #3 could be used but would require a 32-bit symbol latch to hold received symbols more then two clocks.

— Method 1 involves the CPU/M accessing the ''Stored Type Field'' (stored in Interface Logic) for each symbol accessed. This method allows the received packets to be multiplexed together (packet size NOT fixed). Since each symbol takes several accesses (type field access, symbol access) this method is very low performance but is probably needed if Method 1 (Section B above) is used to Enqueue the Tx Data.

— Method 2 involves taking advantage of a fixed packet size. A specific pattern of bits in the Head Symbol could denote a Fixed Packet Transfer. As long as the Stored Type is Null the CPU/M interface logic continually dequeues symbols from the QRDSC Receive Port, (Interrupt Logic could be built to Interrupt on Non-Null Types). When the CPU/M reads a Head Type Field and Symbol indicating a Fixed Packet it could then read ''n'' Symbols (''n'' being the fixed Packet size) without looking at the Stored Type Fields. This would have performance advantages and could be used if Method 2 of Section B (above) was used to enqueue the Tx Data.

D. Receive Packet Protocol Issues: The CPU/M can keep track of any number of Received Streams by storing the Received Heads, along with other pertinent Stream information (current RAM address, count, etc.) in RAM. The CPU/M can store the Received Packet in RAM or at an I/O address port depending upon the application.

In this type of Client Interfaces the CPU/M acts as an ''N'' Channel DMA controller sorting and storing the data based upon the Stream ID of the Received Head. Since this Method is not supporting a higher level Protocol it will not be concerned with responding back to the Source node that initiated the transaction.



TL/F/12044–16

FIGURE 17. CPU/Microcontroller QuickRing Interface

4.2. CPU Handles All Housekeeping Chores, QuickRing Control/Extended Protocol (QRCP/QRXP), Fixed Packet Length

This system design is basically the same design as in Section 4.1 except in terms of guaranteeing a Fixed Size Packet and implementation of a QuickRing Control/Extended Protocol (QRCP/QRXP).

Client Interface Topics

A. The How and Where of Transmit Packet Construction: The CPU/M could construct each packet (QRCP/QRXP) to be transmitted in its internal RAM (Head, Frame Command, Data Address and Possible Data, Section 3.2, 3.3) symbol by symbol. This could be slow since each symbol could take multiple CPU/M instruction cycles to construct/access.

B. Enqueuing the Packet to be Transmitted: Method 2 is a much better choice when implementing QRCP/QRXP because it can guarantee a fixed Packet size.

This method involves the CPU/M building the packet in its internal Register Space. Once the Packet is completed it can be transmitted very rapidly by the CPU, i.e. Burst write to QRDSC Transmit Port. The speed with which the Packet is enqueued into the Transmit FIFO determines the guaranteed packet size (see Table III). Guaranteeing a complete Packet (i.e. Read, Write, Atomic Operation, etc.) during transmission makes the Reception and implementation of the Packet Command much simpler.

C. Dequeueing the Received Packet (Two Methods): As mentioned in Section 4.1, The Client Interface Method #1 of Section 2.2 would be a good choice for this Client Interface. The ''Client Port Interface Logic'' can watch the Receive type field for a Non-Null Type Field. When a Non-Null type is observed the Interface Logic Block should latch it and hold it available to be accessed by the CPU/M. Once the Symbol (RxS[31:0]) has been captured $\overline{\text{RxSTALL}}$ is negated for one clock cycle to allow the next Non-Null Type and symbol to be output.

Note that Client Interface Method #3 could be used but would require a 32-bit symbol latch to hold received symbols more then two clocks.

— Method 1 involves the CPU/M accessing the ''Stored Type Field'' (stored in Interface Logic) for each symbol accessed. Since each symbol takes several accesses (type field access, symbol access) this method is very low performance.

— Method 2 involves taking advantage of a fixed packet size. The type of Frame Command (Read, Write, Atomic Command, Sections 3.2, 3.3) following the Head Symbol will denote the Fixed Packet Transfer size. When the CPU/M reads the Fixed Packet Transfer Size it can then read ''n'' Symbols (''n'' being the fixed Packet size) without looking at the Stored Type Fields. This would have some performance advantages. As long as the Stored Type is Null the CPU/M interface logic continually dequeues symbols from the QRDSC Receive Port, (Interrupt Logic could be built to Interrupt on Non-Null Types).

D. Receive Packet Protocol Issues: The CPU/M can keep track of any number of Received Streams by storing the Received Heads, along with other pertinent Stream information (current RAM address, count, etc.) in RAM. In this type of Client Interfaces the CPU/M acts as an ''N'' Channel DMA controller implementing the QRCP/QRXP, see Sections 3.2, 3.3.

For Example, in a Write Operation the DMA controller could Read (Dequeue) the Head Symbol and Frame Command Symbol, and address of the Write Command. The CPU/M could then perform the Write Operation with the following piece of Data dequeued from the QRDSC. The CPU/M would then build (in internal Register Space) and Enqueue the Head Complement and appropriate Acknowledge back to the Source node.

In Packet Transactions an initial Request and Acknowledge Response transaction would take place first before the Write Packet Request could occur (Section 3.3). These initial subactions are useful in limiting the number of Packet Transactions that can take place simultaneously at a particular Target Node.

4.3 CPU Handles All Housekeeping Chores, Extended Control Protocol (QRCP/QRXP), Fixed Packet Length, Single Channel Receive DMA Controller, and Optional External Pending Request FIFO to Hold Transaction Requests

This system design is basically the same design as Sections 4.1 and 4.2 except that several FPGA devices have been added to provide Fixed Packet Length (FPGA TxFIFO), a Single Channel Receive DMA Controller, and optionally a Pending Request FIFO to hold Channel Requests from other nodes when the DMA channel is currently occupied.

This interface represents a good compromise on a high performance/low cost interface between the QRDSC and the Client/Host. The principles behind this interface are as follows:

A. The How and Where of Transmit Packet Construction: The CPU/M could construct each packet (QRCP/QRXP) to be transmitted (Head, Frame Command, Data Address and Possible Data, Sections 3.2, 3.3) symbol by symbol. These Packet Symbols can be enqueued into a FIFO inside of an FPGA (Field Programmable Gate Array).

An FPGA can be used to provide a Transmit Client Interface as well as an Integrated Transmit FIFO (TxFIFO). FPGA devices are available (i.e. Xilinx XC4000 series) that are low cost and allow the Configurable Logic Blocks (CLB) to be configured as 32 x 1-bit or 16-bit deep x 2-bit wide RAM. 34 CLB's could be used to provide two 34-bit (Each FIFO = 32-bit data + 2-bit type field x 16 deep) Tx FIFOs to hold incoming Packets to be transmitted over QuickRing. Note that the TxFIFO may not add much performance over the CPU/M building packets in its internal Register Space (Sections 4.1, 4.2) and burst Writing the Packet to the QRDSC TxFIFO, if this is the case the Transmit FPGA could be considerably simplified.

As discussed earlier, FIFO's are advantageous because they allow the user to guarantee that many 32-bit data symbols are transmitted together as one packet (one Head followed by up to 20 Quadlets of data). The least significant HOP field could be automatically incremented each packet with an integrated two bit counter. This would guarantee that each Transmit packet remains together as a single packet and does not get concatenated with previous packets transmitted.

When responding to another Nodes Requests the received Head routing symbol must be complemented. The complementing of the Head routing symbol can also be done in the Transmit Interface FPGA (see part C below).

B. Enqueueing the Packet to be Transmitted: The FPGA could wait until an entire Packet is enqueued in its internal FIFO. It can then enqueue that packet into the QRDSC, paying attention to the TxOK Handshake signal. This method of the CPU/M enqueueing the Packet into the Transmit FPGA FIFO first allows greater control of a Fixed Packet Size on the Ring (compared to the CPU/M design, Section 4.1). This results from the FPGA enqueueing the Packet into the QRDSC at one symbol every one or two clocks.

C. Dequeueing the Received Packet: The ''Client Port Interface Logic'' as well as a single DMA channel can possibly be implemented in an EPLD (Altera MAX 7000/Xilinx XC7300) or an FPGA (Xilinx XC4000 series).

— The single DMA channel could contain a 32-bit address hold and increment register, a counter, and a Read/Write/Atomic Command register.

— As mentioned in Section 4.1 and 4.2, The Client Interface Method #1 of Section 2.2 would be a good choice for this Client Interface. The ''Client Port Interface Logic'' can watch the Receive type field for a Non-NULL Type Field. When a Non-Null type is observed the Interface Logic Block should latch it and hold it available to be accessed by the CPU/M. Once the Symbol (RxS[31:0]) has been captured $\overline{\text{RxSTALL}}$ is negated for one clock cycle to allow the next Non-Null Type and symbol to be output.

Note that Client Interface Method #3 could be used but would require a 32-bit symbol latch to hold received symbols more then two clocks.

— The FPGA can access the ''Stored Type Field'' (stored in the FPGA) for each symbol accessed, to guarantee only valid symbols are acted upon.

— A Head Comparator could also be built in to allow the DMA channel to determine when its current stream was writing data.

— During Read Accesses the DMA channel could drive the Transmit Interface FPGA and control how data is loaded into the Transmit FIFO(s) of the interface.

The Extended Control Protocol (Sections 3.2 and 3.3) allows the Client Interface to have control of the number of Data Streams being serviced at any one time. In this particular design only one Data Stream would be serviced at a time because of its single DMA channel limitation.

The FPGA also needs a separate address register, Read/Write/Atomic Request register, and Byte Select Register to handle the QuickRing Control Protocol Commands (QuickRing Protocol Specification). Note that QuickRing Control Protocol Commands are single symbol accesses, thus the simple logic needed to perform those types of accesses. This functionality allows the FPGA to handle a QuickRing Control Protocol Command (i.e. Read Request of the Configuration ROM) during a Extended Control Protocol Command (i.e. Packet Write Request Transaction).

D. Receive Packet Protocol Issues: The following scenario describes how the FPGA Receive Client Interface with Integrated DMA channel implements the Extended Control Protocol of Section 3.3:

— All nodes wishing to communicate with another node first send a Transaction Request (Head followed by one Quadlet of Command Request).

— If a Transaction Request is received and no Transaction is currently being serviced by this particular node then an Acknowledge Response can be sent back to the Requesting Node immediately (Head followed by ACK__RSP code Quadlet).

— When responding to another Nodes Request the received Head routing symbol must be complemented. The complementing of the Head routing symbol can be done in the Transmit Interface FPGA.

— If a Transaction Request is received and this node is currently servicing another nodes transaction then two different scenarios could take place:

1. A Non-Acknowledge Response (retry the Request) can be sent back to the Requesting Node immediately (Head followed by RETRY__RSP code Quadlet).
2. An optional small Pending Request FIFO (small Xilinx XC4000 series) could store any further Requests made to this node, the width of this queue could be very small, 4 bits in a one ring system, 24 bits in a Multi-Ring QuickRing System (holds the requesting Node ID). The depth of this queue could be determined by the System Designer based upon the number of nodes in the System and the number of outstanding transactions allowed for each node. If the queue filled up then the protocol could revert to option #1 above. This queue allows a higher performance system by immediately launching the next ACK__RSP once the current DMA Streaming Transaction terminates.

— If a Transaction Request is received and this node cannot service that request (ex. a node that cannot perform Atomic Transactions) then an Illegal Response (ILL__RSP) can be sent back to the Requesting Node immediately (Head followed by ILL__RSP code Quadlet).

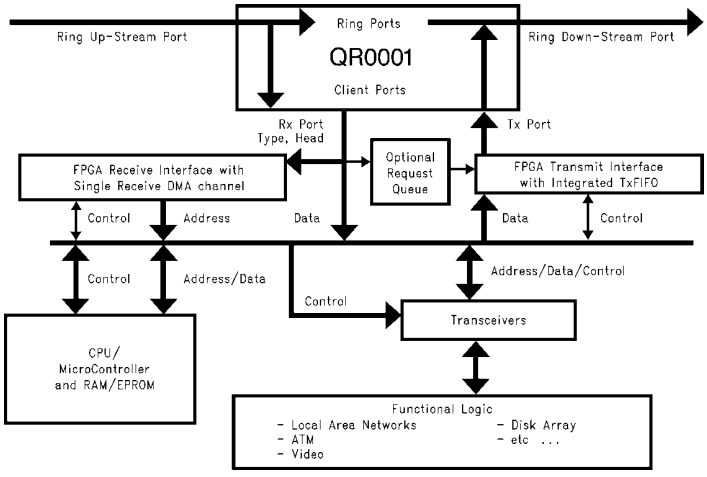
— When a Requesting Node receives an ''ACK__RSP'' then it can send a Head followed by the appropriate control information to set the responder's DMA controller up to perform a Read/Write single/burst DMA Transaction (address/Read or Write Cmmd/Length of DMA/go bit/ ... etc). If the command was a Write command the Requesting Node can also send the stream of data.

— During a Read Packet Request the Responding Node would then return the data followed by an ACK or RETRY Response depending upon the success of the Read Command. During a Write Packet Request the Responding Node returns an ACK or RETRY Response depending upon the success of the Write command.

Proposed Advantages

1. All Burst (Streaming) Requests consist of only a single Quadlet Frame Request and perform 4 handshake transactions. This provides uniformity of length among requests. The Uniform length of requests provides the following advantages:
   — Requests take small amount of QuickRing Bandwidth.
   — Destination storage space required for the Requests is minimal, in a single ring the required storage space could be as small as 4 bits (source node ID).
   — Destination Hardware is simpler because it does not have to be conscious of the details until it is performing the particular transaction.
2. Read, Write, and Atomic Commands are 2 handshake transactions. These higher priority transactions are useful for Control and Status operations. The extra Hardware to support these transactions is fairly simple, an extra Address, Byte Select, and Read/Write Register Bits, and perhaps a small Counter (for Read Block Transfers). This Hardware will allow a DMA Burst (streaming) transaction to be interrupted by an Atomic transaction, followed by the reactivation/continuation of the DMA burst transaction.

Figure 18 shows a Block Diagram of example Hardware that could easily support the QRCP/QRXP and give good system performance.



TL/F/12044–17

FIGURE 18. Programmable Logic QuickRing Interface

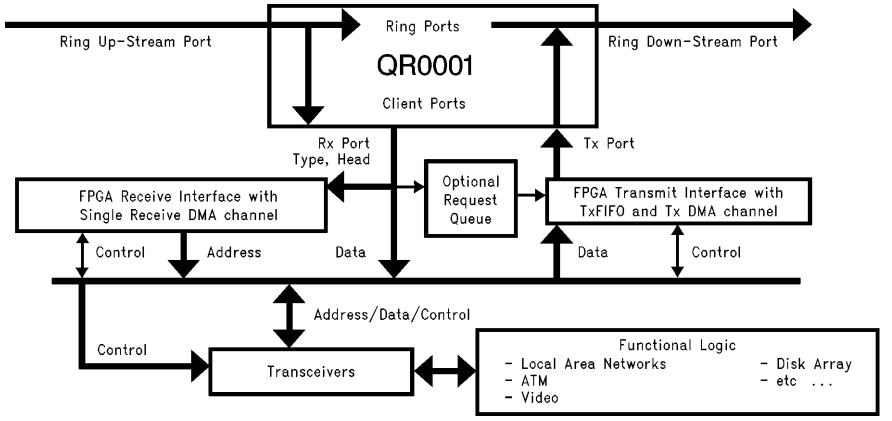
4.4 CPU Handles All Housekeeping Chores, Extended Control Protocol, Fixed Packet Length, Multi-Channel Transmit/Receive DMA Controller, External Receive FIFO to Hold Transaction Requests

This design is a continuation of the design of Section 4.3 except the DMA channels have been expanded (multiple channels, Transmit and Receive) to allow multiple simultaneous transactions.

The Transmit DMA channel allows greater performance, especially for Streaming large amounts of data. The Transmit DMA channel can do all the work of assembling the Packet and enqueueing it into the QRDSC, previously performed by the CPU/M.

4.5 No Local Intelligence, Extended Control Protocol, Fixed Packet Length, Transmit/Receive DMA Controller, External Receive FIFO to Hold Transaction Requests

This design is a continuation of the designs of Sections 4.3 and 4.4 except that there is no Local CPU/M. The Block Diagram for this design is the same as the Section 4.3 Block Diagram except that there is no local CPU/M (see *Figure 19*).



TL/F/12044–18

FIGURE 19. Programmable Logic QuickRing Interface

This Node can function as:

— A Slave only Node. In the slave only mode the FPGA/EPLD client interface must respond to Requests from other nodes as specified in the QRCP/QRXP (Sections 3.2, 3.3).

— A Master Node in a three party transaction, where this Node is the Requester. The three Nodes involved in a three party transaction are:

a. **Instigator:** Provides the intelligence of the transaction by setting up the DMA channels of the Requester to perform the transaction.

b. **Requester:** (This Node) Once its DMA channels are set up it can perform the transaction set up by the Instigator.

c. **Responder:** Functions as the slave node in the transaction, either writing or reading the data from/to the Requester.

The Client Interface to this design is very similar to the designs of Section 4.3 and 4.4 except for Section C.

Dequeueing the Received Packet: The Client Interface Method #2 will be a good choice for this Client Interface given that the Hardware should be able to dequeue a received symbol within two clocks maximum. The ''Client Port Interface Logic'' can watch the Receive type field for a Non-Null Type Field. When a Non-Null type is observed the Interface Logic Block should do two actions:

- Latch the Non-Null Type and hold it available to be accessed by the CPU/M.
- Assert $\overline{\text{RxSTALL}}$ to hold the Received Symbol for up to two clocks.

Once the Symbol (RxS[31:0]) has been captured $\overline{\text{RxSTALL}}$ is negated until the next Non-Null Type and symbol are output.

5.0 FINAL THOUGHTS

This Application Note gives the reader several different QRDSC Client Interfaces, these application interfaces range from low hardware complexity (i.e. lower performance) to greater hardware complexity (higher performance). The system designer can choose the level of complexity needed to give the required performance (see Table IV below).

TABLE IV. Client Interface Performance Approximations

Assume Tx/Rx Client Interface at 50 MHz	Tx Port Average Performance (Clks/Symbol)	Tx Port Max Burst Performance (Clks/Symbol)	Receive Port Average Performance (Clks/Symbol)	Receive Port Max Burst Performance (Clks/Symbol)
#1 CPU/M No Protocol	32	4	40/32	40/32
#2 CPU/M, Higher Level Protocol	40	4	44/36	44/36
#3 CPU/M, Higher Level Protocol, DMA Channel, TxFIFO, Pending Request FIFO	30	2	4	2
#4 CPU/M, Higher Level Protocol, Tx and Rx Multi-Channel DMA, TxFIFO, Pending Request FIFO	4	2	4	2

Assumptions:

Maximum Burst performance is considered for Streaming large amounts of Data.

#1 and #2: CPU/M software takes multiple instructions to Transmit and Receive but Burst Write Transmit from internal Registers is fast. Two Values for the Receive Average and Maximum depend upon Method 1 (examine Type field for each symbol) or Method 2 (only examine type field for the Head). The #2 times are slightly greater because of the Protocol overhead.

#3: CPU/M software takes multiple instructions to Transmit but Burst Write Transmit from FIFO is fast. The Receive Port has a DMA channel to improve performance, but the average is lower because of DMA setup by CPU/M software.

#4: DMA channels on both Transmit and Receive allow maximum performance.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998