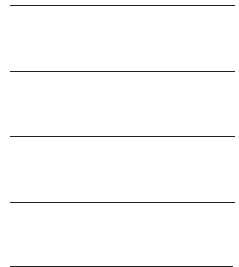


Link Ports 9



9.1 OVERVIEW

The ADSP-2106x SHARC provides additional I/O capability through six dedicated 4-bit link ports. Each link port consists of four bidirectional data lines, a bidirectional clock line, and a bidirectional acknowledge line. The link ports can be clocked twice per processor clock cycle, allowing each port to transfer up to 8 bits of data per cycle. Link port I/O allows a variety of interconnection schemes to I/O peripheral devices as well as coprocessing and multiprocessing schemes. Using link port I/O, it is also possible to configure multidimensional, multiprocessor arrays.

➡ **Note that the ADSP-21061 processor does not have link ports; the discussion in this chapter does not apply to the ADSP-21061.**

Link port features and functions include:

- Link ports can operate independently and simultaneously.
- Link port data is packed into 32-bit or 48-bit words, and can be directly read by the ADSP-2106x core processor or DMA-transferred to on-chip memory.
- Link port data can also be accessed by the external host processor, using direct reads and writes.
- Double-buffered transmit and receive data registers.
- Clock/acknowledge handshaking controls link port transfers which are programmable as either transmit or receive with each link port supported by a separate DMA channel.
- Link ports provide high-speed, point-to-point data transfers to other ADSP-2106x processors. This allows a variety of interconnection schemes between multiple ADSP-2106x processors and external devices, including 1-, 2- and 3-dimensional arrays.

9 Link Ports

The six pins associated with each link port are listed in Table 9.1. Each link port consists of four bidirectional data lines, LxDAT₃₋₀, and two handshake lines, Link Clock (LxCLK) and Link Acknowledge (LxACK). The LxCLK line allows asynchronous data transfers and the LxACK line provides handshaking. When configured as a transmitter, the port drives both the data and LxCLK lines. When configured as a receiver, the port drives the LxACK line.

| <i>Pin(s)</i> | <i>Function</i> |
|----------------------|-------------------------|
| LxDAT ₃₋₀ | Link Port x Data |
| LxCLK | Link Port x Clock |
| LxACK | Link Port x Acknowledge |

Table 9.1 Link Port Pins

“x” denotes the link port number, 0-5.

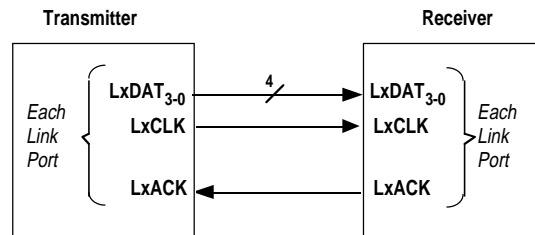


Figure 9.a Link Port Pin Connections

Figure 9.b shows examples of different link port communications schemes. See Chapter 7, *Multiprocessing*, for a discussion of these multiprocessor communications schemes.

9.1.1 Link Port To Link Buffer Assignment

There are six internal data buffer registers which are independent of the actual link ports—these *link buffers*, LBUF0-LBUF5, may be connected to any of the six *link ports*. The link ports receive and transmit data on their LxDAT₃₋₀ data pins, but any of the six link buffers may be assigned to handle data for a particular link port. The link buffers read from or write to internal memory under DMA control.

Remember that “Link Port x” does not automatically mean “Link Buffer x.”

Link Ports 9

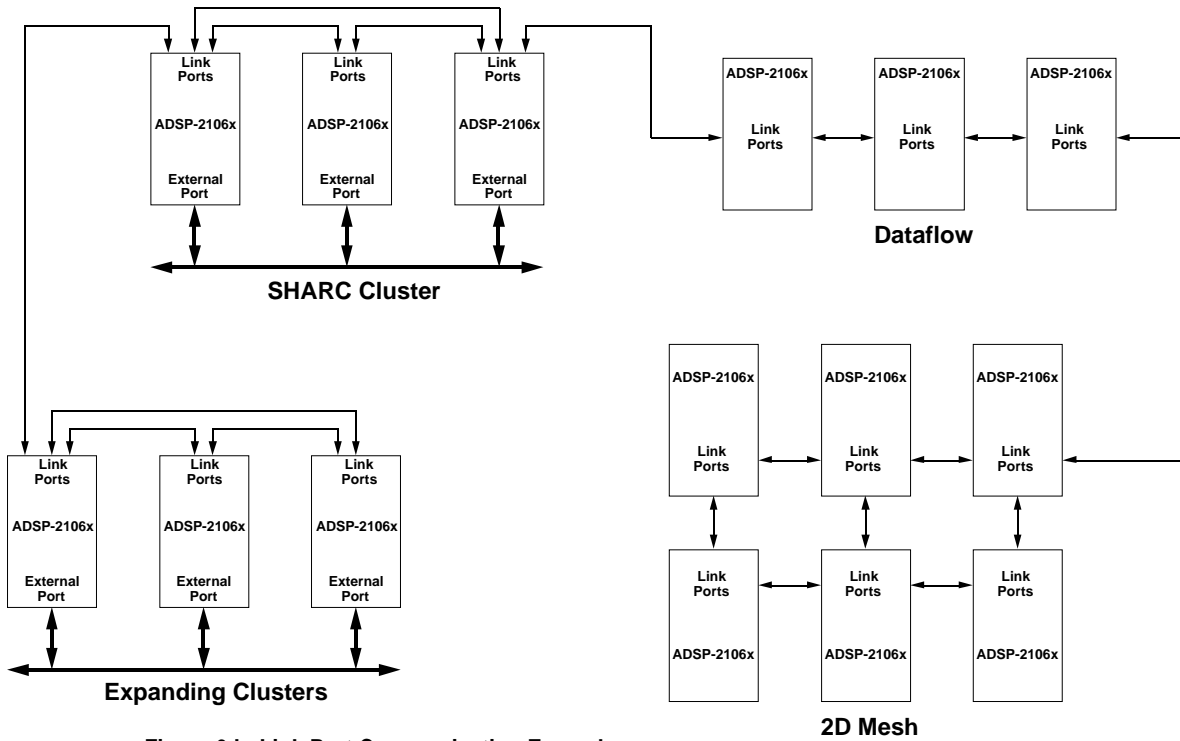


Figure 9.b Link Port Communication Examples

The Link Assignment Register (LAR) is used to assign the link buffer to link port connections. Memory-to-memory transfers may be accomplished by assigning the same link port to two buffers. Details on the LAR register can be found in the “Link Port Control Registers” section of this chapter. Figure 9.1 shows a block diagram of the link ports and link buffers.

9 Link Ports

9.1.2 Link Port DMA Channels

Link buffers 0-5 are supported by DMA channels 1, 3, 4, 5, 6, and 7 respectively:

| | |
|---------------|--|
| DMA Channel 1 | Link Buffer 0 (shared with SPORT1 Receive) |
| DMA Channel 3 | Link Buffer 1 (shared with SPORT1 Transmit) |
| DMA Channel 4 | Link Buffer 2 |
| DMA Channel 5 | Link Buffer 3 |
| DMA Channel 6 | Link Buffer 4 (shared with Ext. Port Buffer 0) |
| DMA Channel 7 | Link Buffer 5 (shared with Ext. Port Buffer 1) |

Some channels are dedicated to link ports. Some are shared with serial ports or the external port as described in the “Link Port DMA Channels” section of this chapter.

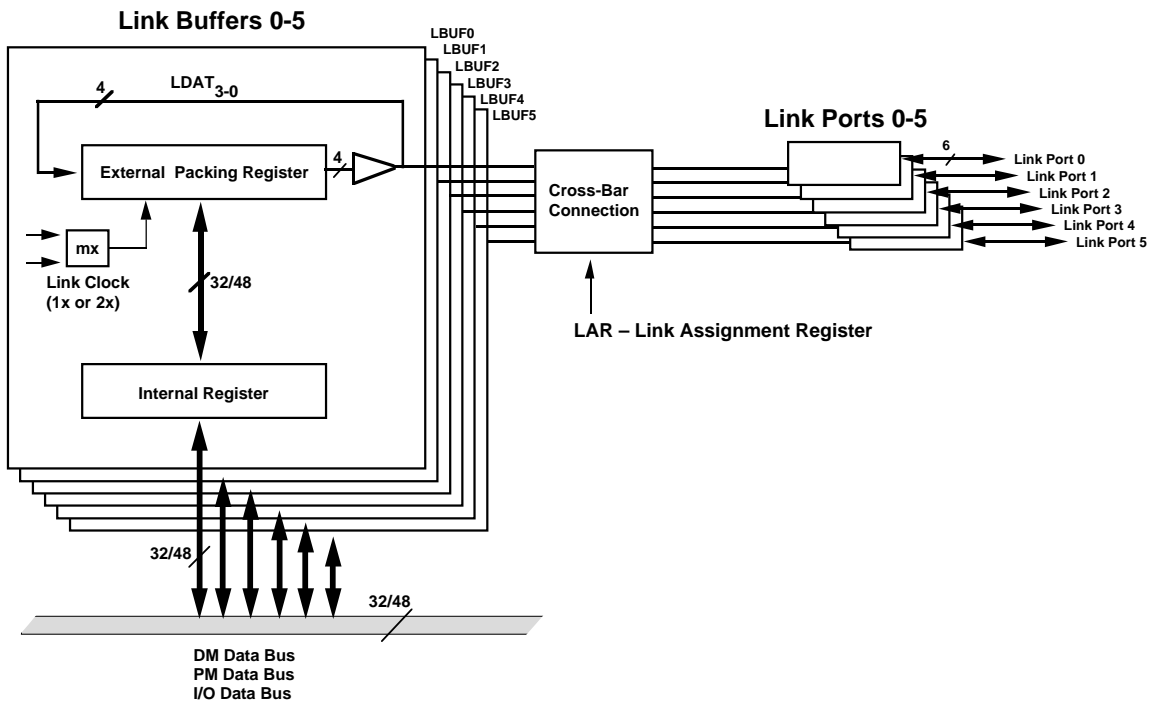


Figure 9.1 Link Ports & Buffers

Link Ports 9

9.1.3 Link Port Interrupts

Three types of interrupts are dedicated to the link ports:

- If DMA is enabled, a maskable interrupt is generated when the DMA block transfer has completed.
- If DMA is disabled, then the link buffer may be read or written by the core processor as a memory-mapped location (part of the IOP register space). A maskable interrupt is generated while DMA is disabled and the receive buffer is not empty, or if the transmit buffer is not full.
- When an external source accesses an unassigned link port (or accesses an assigned link port that has its link buffer disabled), this access causes a maskable LSRQ interrupt.

9.1.4 Link Port Booting

The link ports may be used to load internal memory at reset. Refer to the section “Booting” in the *System Design* chapter of this manual for details of this operation.

9.2 LINK PORT CONTROL REGISTERS

There are three link port control registers: the Link Buffer Control Register (LCTL), the Link Common Control Register (LCOM), and the Link Assignment Register (LAR). To configure link port operations, these registers should be set up in the following order: LAR, LCOM, then LCTL. Before reassigning a link port with the LAR register, disable the link port’s assigned buffer with the LCTL register.

The link port control registers and the serial port control registers share a common internal bus when being written or read. There is a one-cycle latency whenever one of these registers is read after one has been written.

The LCTL and LCOM control registers are initialized to 0x0000 0000 after reset. LAR is initialized to 0x0002 C688, assigning Link Port 0 to Link Buffer 0, Link Port 1 to Link Buffer 1, Link Port 2 to Link Buffer 2, Link Port 3 to Link Buffer 3, Link Port 4 to Link Buffer 4, and Link Port 5 to Link Buffer 5. For complete information about register initialization after reset, see the *Control/Status Registers* appendix of this manual.

9 Link Ports

9.2.1 Link Buffer Control Register (LCTL)

The LCTL register contains control bits unique to each link buffer. Table 9.2 describes the control bits in LCTL.

| <u>Bit(s)</u> | <u>Name</u> | <u>Definition</u> |
|---------------|-----------------|--|
| 0-3 | * | Link buffer 0 controls |
| 4-7 | * | Link buffer 1 controls |
| 8-11 | * | Link buffer 2 controls |
| 12-15 | * | Link buffer 3 controls |
| 16-19 | * | Link buffer 4 controls |
| 20-23 | * | Link buffer 5 controls |
| 24 | LEXT0 | Extended word size: 1=48-bit transfers, 0=32-bit transfers |
| 25 | LEXT1 | Extended word size: 1=48-bit transfers, 0=32-bit transfers |
| 26 | LEXT2 | Extended word size: 1=48-bit transfers, 0=32-bit transfers |
| 27 | LEXT3 | Extended word size: 1=48-bit transfers, 0=32-bit transfers |
| 28 | LEXT4 | Extended word size: 1=48-bit transfers, 0=32-bit transfers |
| 29 | LEXT5 | Extended word size: 1=48-bit transfers, 0=32-bit transfers |
| 30-31 | <i>reserved</i> | |

Table 9.2 Link Control Register (LCTL)

* Each four-bit group includes the following control bits for each link buffer (x=0,1,2,3,4,5):

| <u>Bit#</u> | <u>Name</u> | <u>Definition</u> |
|-------------|-------------|--|
| 0+4x | LxEN | LBUFx enable |
| 1+4x | LxDEN | LBUFx DMA enable |
| 2+4x | LxCHEN | LBUFx chaining enable |
| 3+4x | LxTRAN | LBUFx direction: 1=transmit, 0=receive |

LCTL Control Bits:

- LxEN** Enables a link buffer. As a buffer is disabled (LxEN transitions from high to low), the LxSTAT and LRERR bits are cleared. When its buffer is disabled, an assigned link port stops receiving (driving LxACK) or transmitting (driving LxCLK). To pull the LxACK and LxCLK signals low, enable the pull down resistors with the LCOM register.
- LxDEN** Enables the associated DMA channel.
- LxCHEN** Enables DMA chaining for that channel.
- LxTRAN** Selects the direction of the link buffer, link port and DMA channel: 0 to receive link data, 1 to transmit link data.

Link Ports 9

LEXTx Specifies word size for each link buffer:

LEXTx=1 specifies 48-bit transfers in link buffer x

LEXTx=0 specifies 32-bit transfers in link buffer x

Link buffer data is transmitted and received MSB-first. LEXTx must not be changed while that link buffer is enabled, as this will cause the nibble packing to initialize to an incorrect value.

The LEXTx bits override the setting of the IMDW memory word width bits in SYSCON. If LEXTx=1, data to be transmitted will be read from 48-bit word space in memory, regardless of the setting of IMDW.

Note that when link buffers are enabled or disabled, it is possible to generate unwanted interrupt service requests. This can occur if Link Service Requests (LSRQ) are in use. To avoid this potential problem, the LSRQ register should be masked out while the link buffers are being enabled or disabled. See the “Link Port Interrupts” section of this chapter for more information.

9 Link Ports

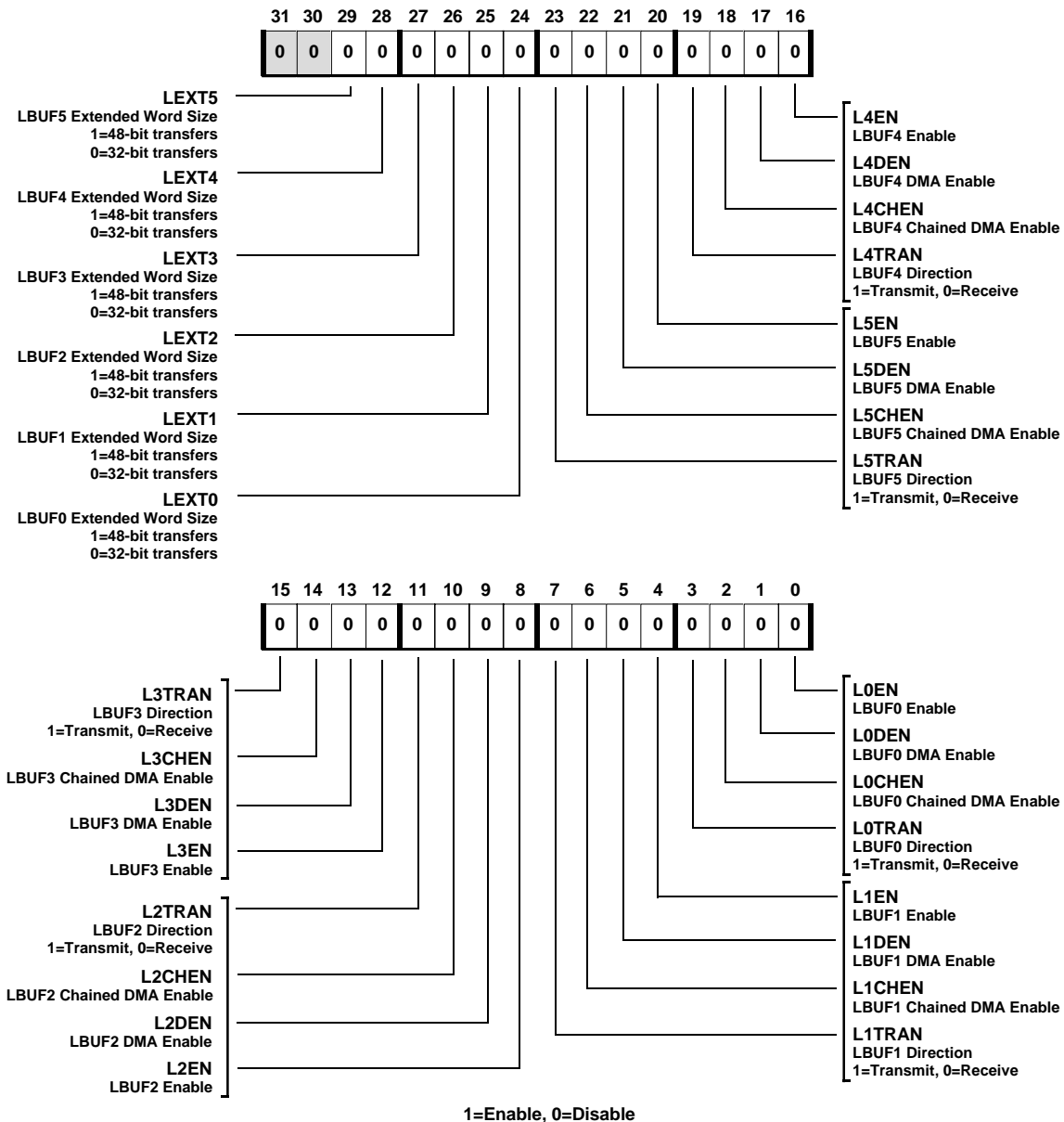


Figure 9.2 LCTL Register

Link Ports 9

9.2.2 Link Common Control Register (LCOM)

The LCOM register contains status bits, packing status bits, and 2X clock rate bits for each buffer. These bits are listed in Table 9.3.

| <u>Bit(s)</u> | <u>Name</u> | <u>Definition</u> |
|---------------|-----------------|--|
| 0-1 | L0STAT(0:1) | Link buffer 0 status: 11=full, 00=empty, 10=one word * |
| 2-3 | L1STAT(0:1) | Link buffer 1 status: 11=full, 00=empty, 10=one word * |
| 4-5 | L2STAT(0:1) | Link buffer 2 status: 11=full, 00=empty, 10=one word * |
| 6-7 | L3STAT(0:1) | Link buffer 3 status: 11=full, 00=empty, 10=one word * |
| 8-9 | L4STAT(0:1) | Link buffer 4 status: 11=full, 00=empty, 10=one word * |
| 10-11 | L5STAT(0:1) | Link buffer 5 status: 11=full, 00=empty, 10=one word * |
| 12 | LCLKX20 | Transfer data at 2X the clock rate on Link Buffer 0 |
| 13 | LCLKX21 | Transfer data at 2X the clock rate on Link Buffer 1 |
| 14 | LCLKX22 | Transfer data at 2X the clock rate on Link Buffer 2 |
| 15 | LCLKX23 | Transfer data at 2X the clock rate on Link Buffer 3 |
| 16 | LCLKX24 | Transfer data at 2X the clock rate on Link Buffer 4 |
| 17 | LCLKX25 | Transfer data at 2X the clock rate on Link Buffer 5 |
| 18 | L2DDMA** | Enable 2-dimensional DMA |
| 19 | LPDRD** | Disable internal pulldown resistor for LxCLK and LxACK |
| 20 | LMSP** | Mesh multiprocessing enable (set to 0 for normal operation) |
| 21-22 | LPTHD** | Mesh multiprocessing LPATH changeover delay: 00=no additional delay, 01=1 additional delay, 10=2 additional delays, 11=3 additional delays |
| 23-25 | <i>reserved</i> | |
| 26 | LRERR0 | Receive pack error status for Link Buffer 0: 1=incomplete, 0=complete |
| 27 | LRERR1 | Receive pack error status for Link Buffer 1: 1=incomplete, 0=complete |
| 28 | LRERR2 | Receive pack error status for Link Buffer 2: 1=incomplete, 0=complete |
| 29 | LRERR3 | Receive pack error status for Link Buffer 3: 1=incomplete, 0=complete |
| 30 | LRERR4 | Receive pack error status for Link Buffer 4: 1=incomplete, 0=complete |
| 31 | LRERR5 | Receive pack error status for Link Buffer 5: 1=incomplete, 0=complete |

Table 9.3 Link Common Control Register (LCOM)

Status bits are read-only.

* The code 01 does not appear as a valid status.

** Common to all link ports.

9 Link Ports

LCOM Control Bits:

- LxSTAT(0:1)** When transmitting, these status bits indicate whether there is room in the buffer for more data. When receiving, these status bits indicate whether new (unread) data is available in the receive buffer. LxSTAT(1)=1 if there is data in the buffer. LxSTAT(0)=0 if there is room in the buffer. These bits are read-only. They are cleared when LxEN changes from 1 to 0. They may subsequently change state when the data buffer is read or written.
- LCLKX2x** This specifies link buffers to transfer at twice the ADSP-2106x clock rate. If LCLKX2x=0, transmit transfers occur at the ADSP-2106x clock frequency, and receive transfers occur at (up to) the ADSP-2106x clock frequency. Set LCLKX2x=1 for receive transfers occurring at greater than the ADSP-2106x clock frequency.
- L2DDMA** This directs the DMA controller to address memory as a two-dimensional array as specified in the DMA address registers. Only DMA channels 0-5 support 2D DMA. Link buffers 4 and 5 on DMA channels 6 and 7 do not support 2D DMA.
- LPDRD** Disables pulldown resistors on signals for unassigned link ports (or on assigned link ports that have their link buffers disabled). These pulldown resistors are 50 k Ω and apply to the LxACK, LxCLK, and LxDAT₃₋₀ signals. Enabling these pulldown resistors keeps the unassigned link port in an inactive state when accessed by another link port. In an application where several ADSP-2106xs share a link port, only one ADSP-2106x should have this bit cleared during operation to prevent too many pulldowns on these lines. External resistors may be used in place of these if needed. LxACK, LxCLK, and LxDAT₃₋₀ should never be left unconnected unless the internal pulldowns are enabled.
- LMSP** Enables mesh multiprocessing mode. Set LMSP=0 for normal operation.
- LPATHD** In a mesh multiprocessing application, these bits allow 1, 2 or 3 additional clock delays to be inserted before changing to the next LPATH register. This allows the current receive operation to complete on the current link port before a new link port is selected. In some mesh multiprocessing applications, this completion delay is significant.
- LRERRx** These bits reflect the status of the receive nibble packer for each link buffer. LRERRx will equal 0 when the nibble packer is set to start receiving a new word. Otherwise it will be 1. If this bit is equal to 1 after a word is received, then an error has occurred (e.g. clock glitch). The LRERRx bits are cleared when LxEN changes from 1 to 0. They may subsequently change state when the link buffer is read or written or while a word is being received.

Link Ports 9

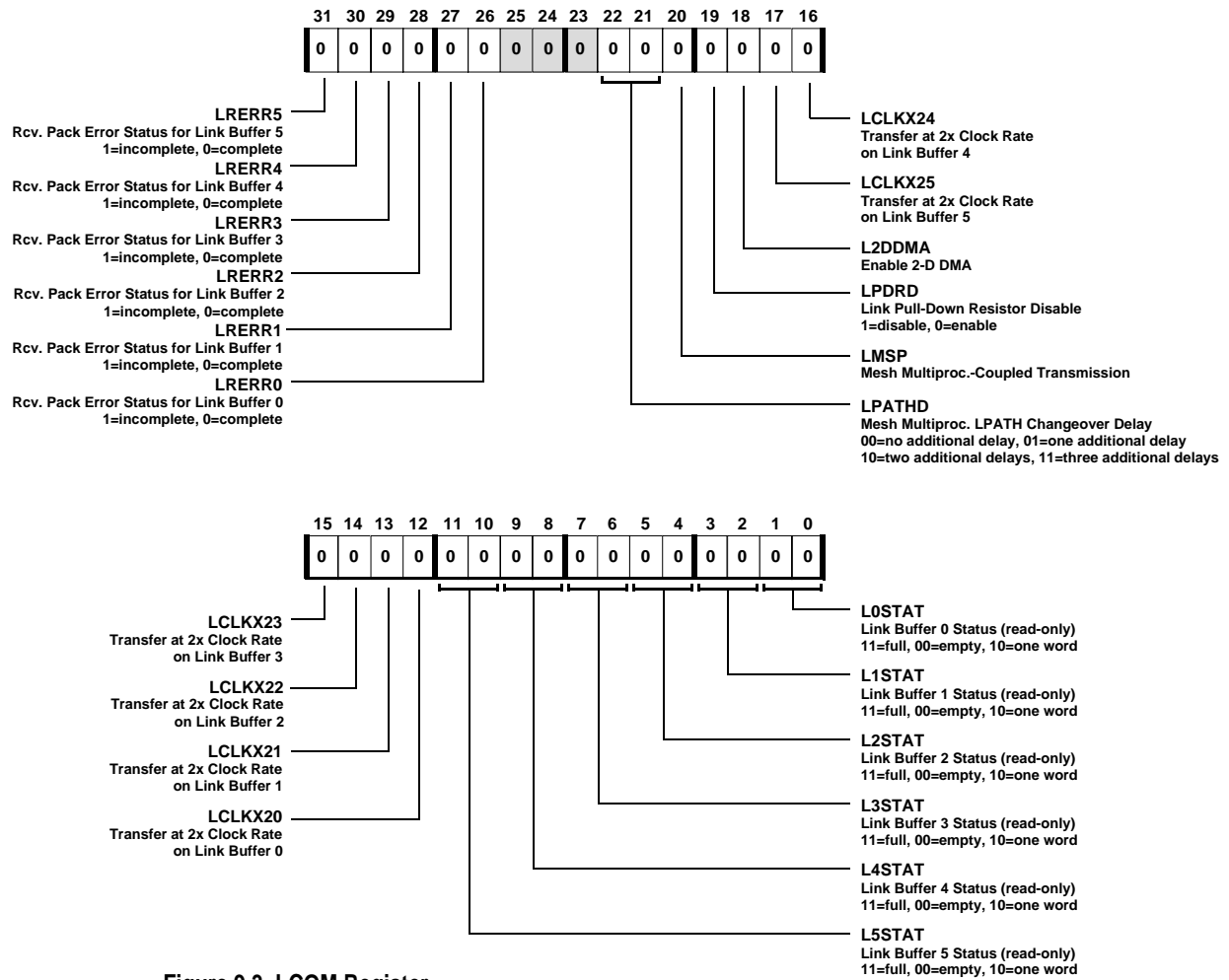


Figure 9.3 LCOM Register

9 Link Ports

9.2.3 Link Assignment Register (LAR)

Each link port is assigned to a link buffer by a 3-bit group in the Link Assignment Register (LAR). There are 6 such groups, one for each buffer, as shown in Table 9.4. The LAR can be thought of as performing a logical (i.e. the link buffer) to physical (i.e. the link port) mapping.

| <u>Bits</u> | <u>Name</u> | <u>Description</u> |
|-------------|-------------|------------------------------------|
| 0-2 | A0LB* | Link port assignment for LBUF0 |
| 3-5 | A1LB* | Link port assignment for LBUF1 |
| 6-8 | A2LB* | Link port assignment for LBUF2 |
| 9-11 | A3LB* | Link port assignment for LBUF3 |
| 12-14 | A4LB* | Link port assignment for LBUF4 |
| 15-17 | A5LB* | Link port assignment for LBUF5 |
| 18-31 | | <i>reserved (must be set to 0)</i> |

Table 9.4 Link Assignment Register (LAR)

| * <u>AxLB</u> | <u>Link Port #</u> |
|---------------|--------------------|
| 000 | Link Port 0 |
| 001 | Link Port 1 |
| 010 | Link Port 2 |
| 011 | Link Port 3 |
| 100 | Link Port 4 |
| 101 | Link Port 5 |
| 110 | <i>reserved</i> |
| 111 | inactive buffer |

The AxLB bits assign a link port to the link buffer *x*. A link port is DISABLED if it has no buffers assigned or if the link port's assigned buffers are disabled. When a link port is disabled, its LxDAT₃₋₀, LxCLK, and LxACK pins are three-stated. If a buffer is intended to be inactive, the corresponding link port assignment field should be set to 7.

Memory-to-memory transfers may be accomplished by assigning the same link port to two buffers, disabling the port partially. One buffer transmits while the other receives. Up to three memory-to memory transfers may occur simultaneously, by using all six link buffers. This partially disabled mode is known as loopback mode. Using this configuration, LxDAT₃₋₀ and LxACK will not be driven or sensed and LxCLK will not be driven. However, LxCLK should not be driven externally in this mode, due to the fact that an LxCLK transition may be sensed and result in a nibble shift in the received data buffer.

Link Ports 9

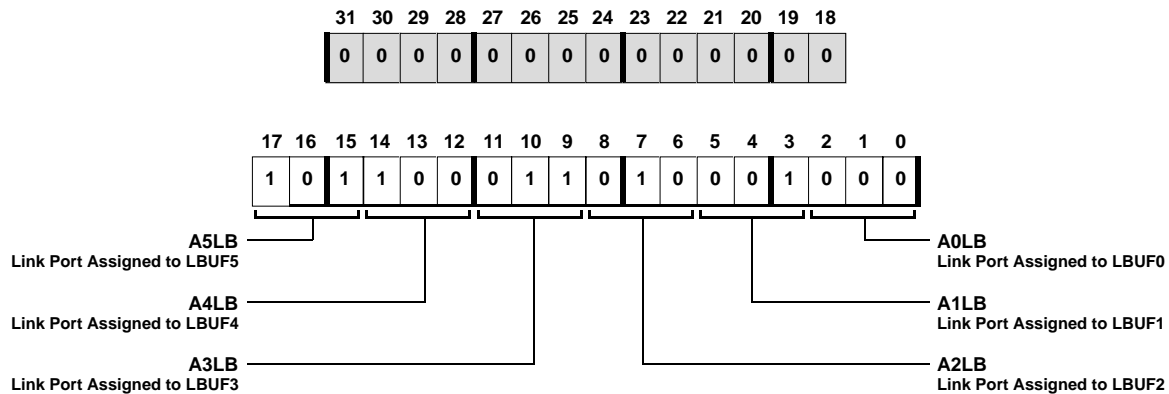


Figure 9.4 LAR Register

9.3 HANDSHAKE CONTROL SIGNALS

The LxCLK and LxACK pins of each link port allow handshaking for asynchronous data communication between ADSP-2106xs. Other devices that follow the same protocol may also communicate with these link ports.

A link-port-transmitted word consists of 8 nibbles (for a 32-bit word) or 12 nibbles (for a 48-bit word). The transmitter asserts the clock (LxCLK) high with each new nibble of data. The falling edge of LxCLK is used by the receiver to latch the nibble. The receiver asserts LxACK when it is ready to accept another word in the buffer. The transmitter samples LxACK at the beginning of each word transmission (i.e. after every 8 or 12 nibbles). If LxACK is deasserted at that time, the transmitter will not transmit the new word—see Figure 9.5. The transmitter will leave LxCLK high and continue to drive the first nibble if LxACK is deasserted. When LxACK is eventually asserted again, LxCLK will go low and begin transmission of the next word. If the transmit buffer is empty, LxCLK will remain low until the buffer is refilled, regardless of the state of LxACK.

The receive buffer may fill if a higher priority DMA or chain loading operation is occurring. LxACK may deassert when it anticipates the buffer may fill. However, LxACK will be reasserted by the receiver as soon as the internal DMA grant signal has occurred, freeing a buffer location.

9 Link Ports

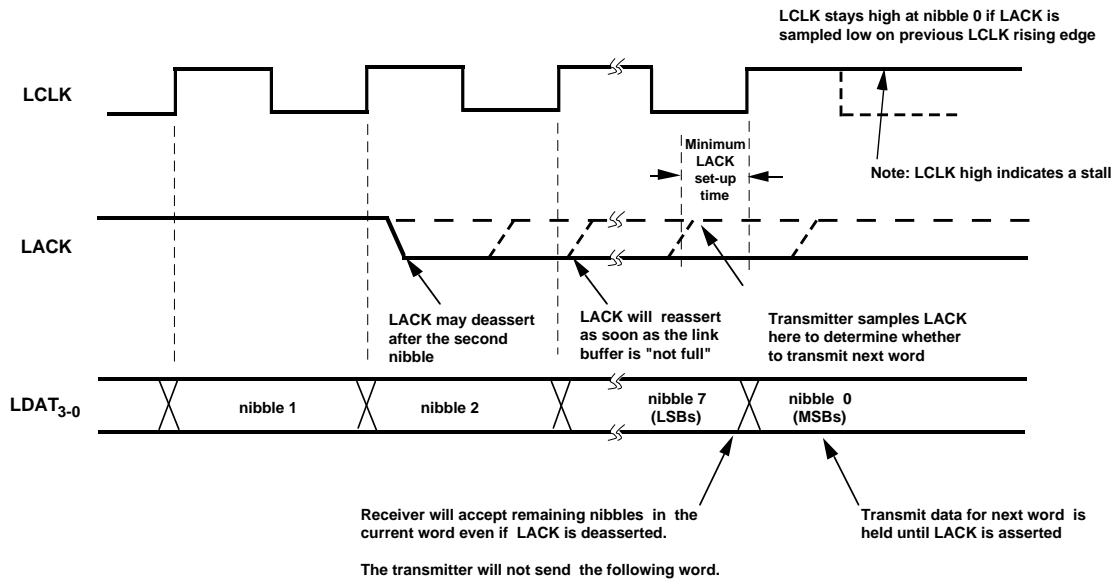


Figure 9.5 Link Port Handshake Timing

Data is latched in the receive buffer on the falling edge of LxCLK. The receive operation is purely asynchronous and can occur at any frequency up to twice CLKIN, the processor clock frequency. **If the receive clock frequency is less than or equal to CLKIN, the LCLKX2 bit for the receive buffer should be set to 0 in LCOM. If the receive clock frequency is between CLKIN and (2 * CLKIN), the LCLKX2 bit should be set to 1 in LCOM.** This causes the buffer status to change, generating an internal DMA request, after the sixth nibble (of eight) or tenth nibble (of twelve) has been received. Because a preemptive DMA request is made, the entire word must be received in a single burst, with no gated clocks used during the word.

When a link port is not enabled, LxDAT₃₋₀, LxCLK and LxACK are tristated. When a port is enabled to transmit, the data pins will be driven with whatever data is in the output buffer, LxCLK will be driven high and LxACK will be tristated. When a port is enabled to receive, the data pins and LxCLK will be tristated and LxACK will be driven high.

Link Ports 9

To allow a transmitter and a receiver to be enabled (assigned and link buffer enabled) at different times, LxACK, LxCLK, and LxDAT_{3:0} may be held low with the 50 kΩ internal pulldown resistors if LPDRD is cleared when the link port is disabled. Thus, if the transmitter is enabled before the receiver, LxACK will be low and the transmission is held off. Similarly, if the receiver is enabled before the transmitter, LxCLK will be held low and the receiver will be held off. If many link ports are bused together, one external resistor should be used to pull down each bused line instead of the internal pulldowns. This will guarantee that the bused lines are not pulled down too strongly.

LxACK, LxCLK, and LxDAT_{3:0} should not be left unconnected unless external pulldown resistors or the internal pulldowns are used.

9.4 LINK BUFFERS

Each link buffer consists of an external and an internal register (see Figure 9.1). When transmitting, the internal register is used to accept the DMA data from internal memory. The external register performs the unpacking for the link port, most significant nibble first. These two registers form a two-stage FIFO, the LBUF_x buffer. Two words can be written into the register (by DMA or from the core) before it signals a full condition. As each word is unpacked and transmitted, the next location in the FIFO becomes available and a new DMA request is made. If the register becomes empty, the LxCLK signal will be deasserted.

Full/empty status for the link buffer FIFOs is given by the LxSTAT bits of the LCOM register. This status is cleared for a link buffer when its LxEN enable bit is cleared in the LCTL register.

During receiving, the external buffer is used to pack the receive port data (most significant nibble first) and pass it to the internal register before DMA-transferring it to internal memory. This buffer is a two-deep FIFO. If the ADSP-2106x's DMA controller does not service it before both locations are filled, then the LxACK signal will be deasserted.

The link buffer width may be selected to be either 32 or 48 bits. This selection is made individually for each buffer with the LEXT bits in the LCTL register. For 40-bit extended precision data or 48-bit instruction transfers, the width must be set to 48 bits.

9 Link Ports

9.4.1 Core Processor Access To Link Buffers

In applications where the latency of link port DMA transfers to and from internal memory is too long, or where a process is continuous and has no block boundaries, the ADSP-2106x processor core may read or write link buffers directly using the full/empty status bit of the link buffer to automatically pace the operation. The *full or empty* status of a particular LBUF_x buffer can be determined by reading the LCOM control/status register. DMA should be disabled (i.e. the LxDEN bit should be cleared) when using this capability. A programming example of core-driven transfers is shown at the end of this chapter.

If a read is attempted from an empty receive buffer, the core will hang until the link port completes transmission of a word. Similarly, if a write is attempted to a full transmit buffer, the core will hang until the external device accepts the complete word. Up to four words (2 in the receiver and 2 in the transmitter) may be sent without a hang before the receiver core must read a link buffer register. To prevent this type of hang condition from occurring, the BHD (Buffer Hang Disable) bit can be set in the SYSCON register.

9.4.2 Host Processor Access To Link Buffers

The link buffers can also be accessed by the external host processor, using direct reads and writes. When the host reads or writes to these buffers, the word width is determined *only* by the host packing mode, as selected by the HPM bits in the SYSCON register, and not by the LEXT bit in LCTL.

9.5 LINK PORT DMA CHANNELS

Link buffers 0-5 are supported by DMA channels 1, 3, 4, 5, 6, and 7 respectively. Some DMA channels are dedicated and others are shared:

- DMA channel 1 is shared by SPORT1 receive and link buffer 0 (LBUF0).
- DMA channel 3 is shared by SPORT1 transmit and link buffer 1 (LBUF1).
- DMA channel 4 is dedicated to link buffer 2 (LBUF2).
- DMA channel 5 is dedicated to link buffer 3 (LBUF3).
- DMA channel 6 is shared by ext. port buffer 0 (EPB0) and link buffer 4.
- DMA channel 7 is shared by ext. port buffer 1 (EPB1) and link buffer 5.

Link Ports 9

DMA Channels 1 and 3 are shared by link buffers 0 and 1, respectively, and by SPORT1. This has several functional implications:

- If the SPORT1 receive DMA enable bit or chaining enable bit is set, then SPORT1 receive is assigned DMA channel 1.
- If the LBUF0 DMA enable bit is set, then link buffer 0 is assigned this DMA channel.
- If both enables are set, the SPORT is selected.
- If neither the SPORT DMA enable or LBUF0 DMA enable is set, then interrupts from both buffers are ORed.

SPORT1 transmit and LBUF1 are shared and selected in the same way.

DMA Channel 6 is shared by the external port buffer EPB0 and link buffer 4 (LBUF4). Functional implications include:

- If the EPB0 DMA enable bit or chaining enable bit is set, then EPB0 is assigned DMA channel 6.
- If the LBUF4 DMA enable bit is set, then link buffer 4 is assigned this DMA channel.
- If both enables are set, EPB0 is selected.
- If neither the external port DMA enable or LBUF4 DMA enable is set, then interrupts from both buffers are ORed.

EPB1 and LBUF5 share DMA channel 7 and are selected in the same way.

A maskable interrupt is generated when the DMA block transfer has completed. A more complete discussion on interrupts can be found in the “Link Port Interrupts” section of this chapter.

If DMA is disabled for a buffer, then the buffer may be read or written by the core processor as a memory-mapped location. A maskable interrupt is generated while DMA is disabled and the receive buffer is not empty or if the transmit buffer is not full.

9 Link Ports

9.5.1 DMA Chaining For Link Ports

In chained DMA operations, the ADSP-2106x automatically sets up another DMA transfer when the contents of the current buffer have been transmitted (or received). The chain pointer register (CPx) is used to point to the next set of buffer parameters stored in memory. The ADSP-2106x's DMA controller automatically downloads these buffer parameters to set up the next DMA sequence. Refer to the *DMA* chapter of this manual for details on how to set up chaining parameters in memory.

DMA chaining is enabled on each link port by setting the LxCHEN bit in LCTL. When chaining is enabled, DMA transfers are initiated by writing a memory address to the CP register.

Six DMA parameter registers are initialized for the chained operation, in the following order:

| | |
|-----|--|
| IIx | Index (start address of memory buffer) |
| IMx | Modify (increment) |
| Cx | Count |
| CPx | Chain Pointer |

9.6 LINK PORT INTERRUPTS

Link ports have 3 different types of interrupts:

1. Interrupts caused by the received or transmitted data when the ADSP-2106x core is accessing the buffers directly and DMA is not enabled.
2. Interrupts generated by the completion of a DMA cycle.
3. Interrupts caused by an attempted external access of a link port that is unassigned or assigned to a link buffer that is not enabled.

Types 1 and 2 are mutually exclusive and use the same interrupt. Type 3 is independent of 1 and 2 and uses a different interrupt vector. Details of each kind of interrupt follow below.

9.6.1 Link Port Interrupts With DMA Disabled

If DMA is disabled for a link port buffer, then the buffer may be written or read by the ADSP-2106x core as a memory-mapped IOP register.

Link Ports 9

If the DMA is disabled but the associated link buffer is enabled, then a maskable interrupt is generated whenever a receive buffer is not empty or when a transmit buffer is not full.

The interrupt latch bit in IRPTL may be masked in the corresponding IMASK register bit. When initially enabling the IMASK bit, the corresponding bit in IRPTL should be cleared first to clear out any request that may have been inadvertently latched.

The interrupt service routine should test the buffer status after each read or write to check when the buffer is empty or full, in order to determine when it should return from interrupt. This will reduce the number of interrupts it must service.

9.6.2 Link Port Interrupts With DMA Enabled

A link port interrupt is generated when the DMA operation is done—i.e. when the block transfer has completed and the DMA count register is zero.

There is an option for implementing a protocol to send additional control information at the end of a block transfer. Because the receive DMA buffer is empty when the DMA block has completed, the external bus master can send up to two additional words to the slave ADSP-2106x's buffer which has space for the two words. The slave's *DMA done* interrupt service routine could then read the buffer and use these control words to determine the next course of action.

9.6.3 Link Port Service Request Interrupts (LSRQ)

Link port service requests allow a disabled (unassigned or assigned and buffer disabled) link port to cause an interrupt when an external access is attempted. The transmit and receive request status bits of the LSRQ register (bits 20-31) allow an ADSP-2106x to determine if another ADSP-2106x is attempting to send or receive data through a particular link port. This lets two processors communicate without prior knowledge of the transfer direction, link port number, or exactly when the transfer is to occur.

When LxACK or LxCLK is asserted externally, a link service request (LSR) is generated in a disabled (unassigned or assigned and buffer disabled) link port. LSRs will not be generated for a link port that is disabled by loopback mode. Each LSR is gated by mask bits before

9 Link Ports

being latched in the LSRQ register. The six possible receive LSRs and the six possible transmit LSRs are ORed together to generate the link service request interrupt. The LSRQ interrupt request may be masked by the LSRQI mask bit of the IMASK register. When the mask bit is set, the interrupt is allowed to pass into the interrupt priority encoder. A diagram of this logic appears in Figure 9.5a.

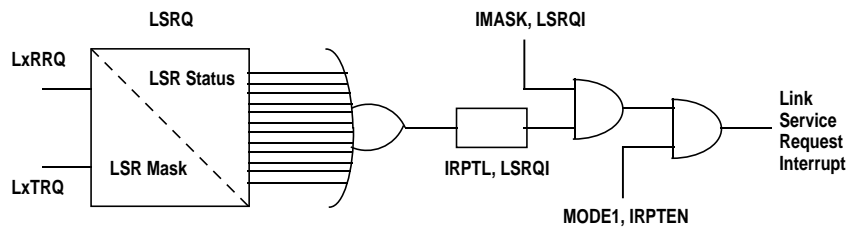


Figure 9.5a Logic For Link Port Interrupts

The interrupt routine must read the LSRQ register to determine which link port to service and whether it is a transmit or receive request.

LSR interrupts have a latency of two cycles. Note that the link service request interrupt is different from the link receive and transmit interrupt—this is also true in IMASK.

The 32-bit LSRQ register holds the masked link status of each link port and the corresponding interrupt mask bits. The link status of the port is set whenever the port is not enabled and one of LxACK or LxCLK is asserted high. The LSRQ status bits are read-only. Table 9.5 and Figure 9.6 show the individual bits of the LSRQ register.

To determine which link port to service, transfer LSRQ to a register Rx (in the register file), then use the leading 0s detect instruction, Rn=LEF TZ Rx. Rn indicates which link port is active in order of priority.

If link service requests are in use, they should be masked out when the assigned link buffers are being enabled, disabled, or when the link port is being unassigned in LAR, otherwise spurious service requests may be generated.

Link Ports 9

This need for masking is due to a delay before LxCLK or LxACK (if already asserted) signals are pulled (if pulldowns enabled) or driven externally (if pulldowns disabled) below logic threshold. During this delay, these signals are sampled asserted and generate an LSRQ.

To avoid the possibility of spurious interrupts, mask the LSRQ interrupt or the appropriate request bit in the LSRQ register and allow an appropriate delay before unmasking. Alternatively, mask the LSRQ interrupt and poll the appropriate request status bit until it is cleared and then unmask the interrupt.

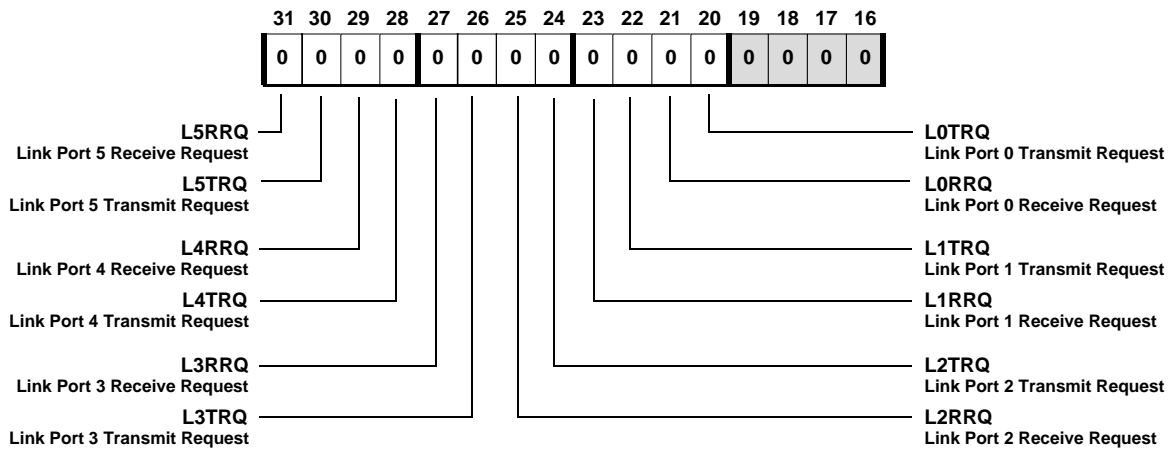
| <u>Bit</u> | <u>Name</u> | <u>Description</u> |
|------------|-----------------|--|
| 0-3 | <i>reserved</i> | |
| 4 | L0TM | Link Port 0 transmit mask |
| 5 | L0RM | Link Port 0 receive mask |
| 6 | L1TM | Link Port 1 transmit mask |
| 7 | L1RM | Link Port 1 receive mask |
| 8 | L2TM | Link Port 2 transmit mask |
| 9 | L2RM | Link Port 2 receive mask |
| 10 | L3TM | Link Port 3 transmit mask |
| 11 | L3RM | Link Port 3 receive mask |
| 12 | L4TM | Link Port 4 transmit mask |
| 13 | L4RM | Link Port 4 receive mask |
| 14 | L5TM | Link Port 5 transmit mask |
| 15 | L5RM | Link Port 5 receive mask |
| 16-19 | <i>reserved</i> | |
| 20 | L0TRQ | Link Port 0 transmit request status (<i>read-only</i>) |
| 21 | L0RRQ | Link Port 0 receive request status (<i>read-only</i>) |
| 22 | L1TRQ | Link Port 1 transmit request status (<i>read-only</i>) |
| 23 | L1RRQ | Link Port 1 receive request status (<i>read-only</i>) |
| 24 | L2TRQ | Link Port 2 transmit request status (<i>read-only</i>) |
| 25 | L2RRQ | Link Port 2 receive request status (<i>read-only</i>) |
| 26 | L3TRQ | Link Port 3 transmit request status (<i>read-only</i>) |
| 27 | L3RRQ | Link Port 3 receive request status (<i>read-only</i>) |
| 28 | L4TRQ | Link Port 4 transmit request status (<i>read-only</i>) |
| 29 | L4RRQ | Link Port 4 receive request status (<i>read-only</i>) |
| 30 | L5TRQ | Link Port 5 transmit request status (<i>read-only</i>) |
| 31 | L5RRQ | Link Port 5 receive request status (<i>read-only</i>) |

Table 9.5 Link Service Request Register (LSRQ)

For *transmit request status* bits, LxTRQ=1 means LxACK=1, LxTM=1, and LxEN=0.

For *receive request status* bits, LxRRQ=1 means LxCLK=1, LxRM=1, and LxEN=0.

9 Link Ports



Request Bits are Read-Only Status

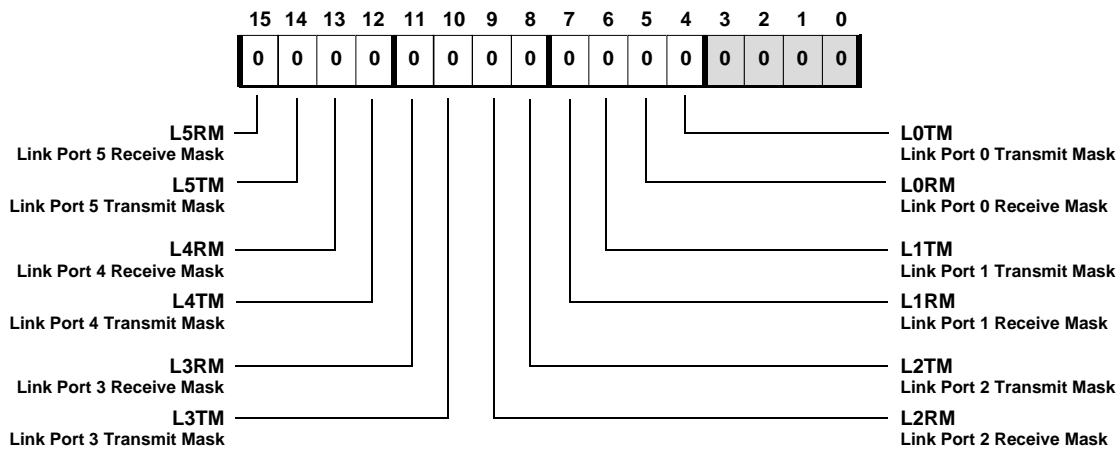


Figure 9.6 LSRQ Register

For *transmit request status* bits, LxTRQ=1 means LxACK=1, LxTM=1, and LxEN=0.

For *receive request status* bits, LxRRQ=1 means LxCLK=1, LxRM=1, and LxEN=0.

Link Ports 9

9.7 TRANSMISSION ERROR DETECTION

Transmission errors on the link ports may be detected by reading the LRERRx bits in the LCOM register. These bits reflect the status of each nibble counter. The LRERRx bit will be zero if the pack counter of the corresponding link buffer is zero (i.e. a multiple of 8 or 12 nibbles have been received). If LRERR is high when a transmission has completed, then an error occurred during transmission. (Note that the DMA word count provides an exact count of the number of words to be transferred.)

To allow checking of this status, the transmitter and receiver should follow a protocol such as the one described below:

Transmitter Protocol—To make use of the LRERRx status, one additional dummy word should always be transmitted at the end of a block transmission. The transmitter must then deselect the link port to allow the receiver to send an appropriate message back to the transmitter.

Receiver Protocol—When the receiver has received the data block, indicated by a *DMA done* interrupt, it checks that it has received an additional word in the link buffer and then reads the LRERR bit. The receiver may then clear the link buffer (LxEN=0) and transmit the appropriate message back to the transmitter on the same, or a different, link port.

9.8 TOKEN PASSING

Two processors wishing to communicate on a link port need to know which of them is currently the transmitter and which is the receiver, otherwise they might both try to transmit at the same time. Token passing is a way of accomplishing this. Figure 9.7 shows a flow chart of the token passing process.

Token passing is a way of establishing which of two ADSP-2106xs is the current transmitter. The token is a software flag, similar to a semaphore, that is passed between the processors. At reset, by convention, the token (flag) is set to reside in the link port of one device, making him the master and the transmitter. When a receiver link port (slave) wants to become the master, he may assert his LxACK line (request data) to get the master's attention. The master will know, through software protocol, whether he is supposed to respond with actual data or whether he is being asked for the token.

9 Link Ports

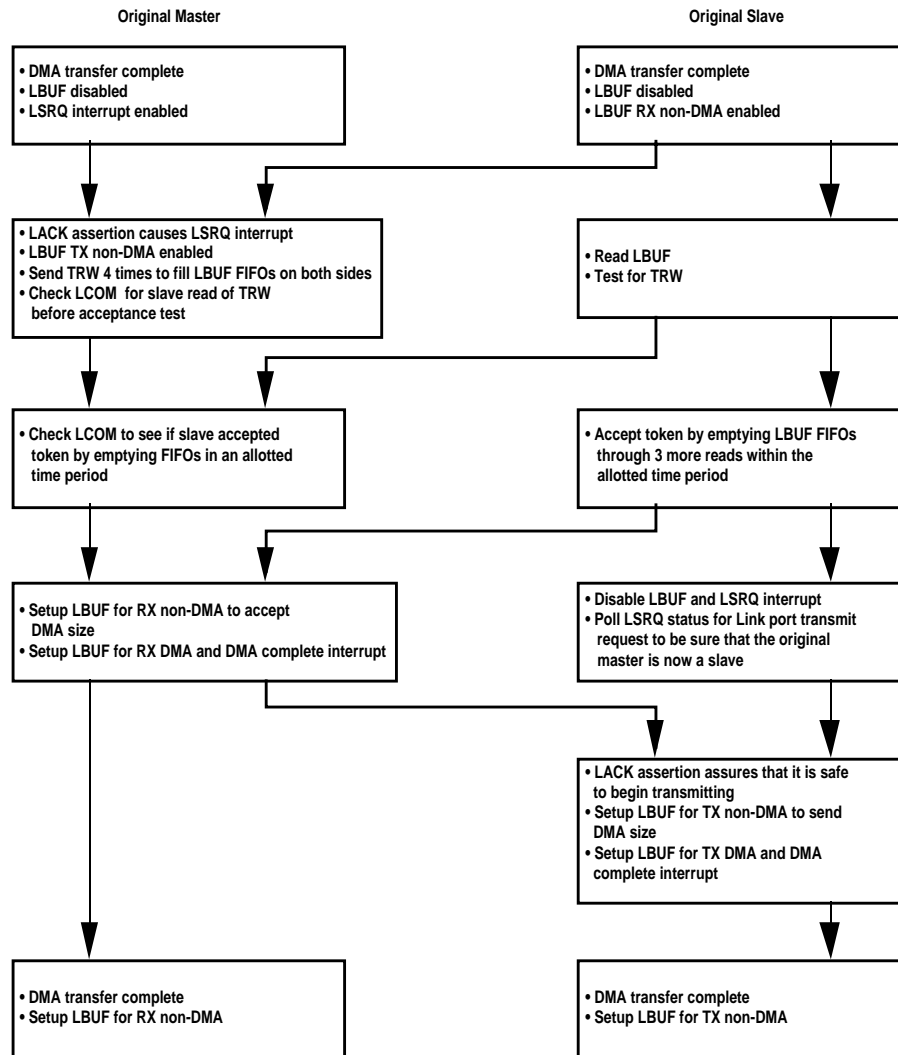


Figure 9.7 Token Passing Flow Chart

Notes:

The token release word can be any user-defined value. Since both the transmitter and receiver are expecting a code word, this need not be exclusive of normal data transmission.

Link Ports 9

If the master wishes to give up the token, he may send back a user-defined token release word and thereafter clear his token flag. Simultaneously, the slave examines the data sent back and if it is the token release word, the slave will set his token, and can thereafter transmit. If the received data is not the token release word, then the slave must assume the master was beginning a new transmission.

Through software protocol, the master can also ask to receive data by sending the token release word without the LxACK (data request) going low first.

An example of software protocol for token passing through the link ports is included at the end of this chapter. Figure 9.7 shows a flow chart of the example code and the following is a description of the process:

The example code is to be loaded on both the original master and the original slave. The code is ID intelligent for multiprocessor systems: ID1 is the original master (transmitter) and ID2 is original slave (receiver). The master transmits a buffer via DMA through LPORT0 using LBUF3 and the slave receives through LPORT0 using LBUF2. The slave then requests the token by generating an LSRQ interrupt in the disabled link port of the master (LPORT0). The master responds by sending the token release word and waiting to see if it is accepted. The slave checks to see that it is the token release word and will accept the token by emptying the master's link buffer FIFO within a predetermined amount of time. If the token is accepted the slave will become the master and transmit a buffer of data to the new slave. If the token is rejected, the master will transmit a second buffer. When complete, the original master will finish by setting up LBUF2 to receive without DMA, and the original slave will set up LBUF3 to transmit without DMA.

The following is a list of the major areas of concern when implementing a software protocol scheme for token passing:

- Ensure that both link buffers are not enabled to transmit at the same time. In the event that this is allowed, data may be transmitted and lost due to the fact that neither link port will be driving LxACK. In the example provided at the end of the chapter, the LSRQ register status bits are polled to ensure that the master becomes the slave before the slave becomes the master, thus avoiding the two transmitter conflict.

9 Link Ports

- Ensure that the link interrupt selection matches the application. If a status detection scheme using the status bits of the LSRQ register is to be used, it is important to note the following: If a link port that is configured to receive is disabled while LxACK is asserted, there will be an RC delay before the 50k ohm pulldown resistor on LxACK (if enabled) can pull the value below logic threshold. Likewise, if a link port that is configured to transmit is disabled while LxCLK is asserted, there will be an RC delay before the 50k ohm pulldown resistor on LxCLK (if enabled) can pull the value below logic threshold. If the appropriate request status bit is unmasked in the LSRQ register (in this instance), then an LSR will be latched and the LSRQ interrupt may be serviced, even though unintended, if enabled.
- Ensure that synchronization is not disrupted by unrelated influences at critical sections where timing control loops are used to synchronize parallel code execution. Disabling of nested interrupts is one of the techniques used in the example to control this.

9.9 LINK TRANSMISSION LINES

The link ports are designed to allow long distance connections to be made between the driver and the receiver. This is possible because the links are self-synchronizing, i.e. the clock and data are transmitted together. Only relative delay, not absolute delay between clock and data is of importance.

In addition, the LACK signal inhibits transmission of the next word, not of the current nibble (i.e. the current word is always allowed to complete transmission). This allows delays of 2 to 3 cycles for the LxACK signal to reach the transmitter.

The links are designed to drive transmission lines with characteristic impedances of 50Ω or greater. A higher transmission line impedance reduces the on-chip effect of driver impedance variations, for distances longer than about 6 inches. It is recommended that an external series termination resistor be used at each link port pin to absorb reflections from the open circuit at the destination. The external resistor should be selected such that its value (plus the internal resistance of the driver) be equal to the characteristic impedance of the transmission line.

Thus, if the typical internal drive resistance is 10Ω and the characteristic impedance is 50Ω , then the link port pin resistor should be 40Ω .

Link Ports 9

9.10 SYSTEM DESIGN EXAMPLE: LOCAL DRAM INTERFACE

The example shown in Figure 9.8 shows how a multiprocessing system can use link ports to connect to local memories and I/O devices. An ASIC implements the interface between the link port and DRAM or an I/O device. This minimal hardware solution frees the ADSP-2106x's external bus for other shared-bus communication. The DRAM and ASIC may be implemented on a single 10-pin SIMM module.

Accesses to the DRAM via a link is most efficient under DMA control. The ASIC receives DMA control information from the link port and sets up the access to the DRAM. It unpacks 16-bit data words from the DRAM or packs 4-bit nibbles from the link. At the end of the DMA transfer, an interrupt will allow the ADSP-2106x to send new control information to the ASIC. The ASIC always reverts to receive mode at the end of a transfer. The LxACK signal is deasserted by the ASIC whenever a page change, memory refresh cycle, or any other access to the DRAM occurs.

Memory modules may be shared by multiple ADSP-2106xs when the link port is used. Each link port supports 40 Mbyte access throughput for either instructions or data. The ASIC is responsible for generating the 2X clock when transmitting to the ADSP-2106x. The ASIC is also responsible for generating sequential DMA addresses based on a start address and word count.

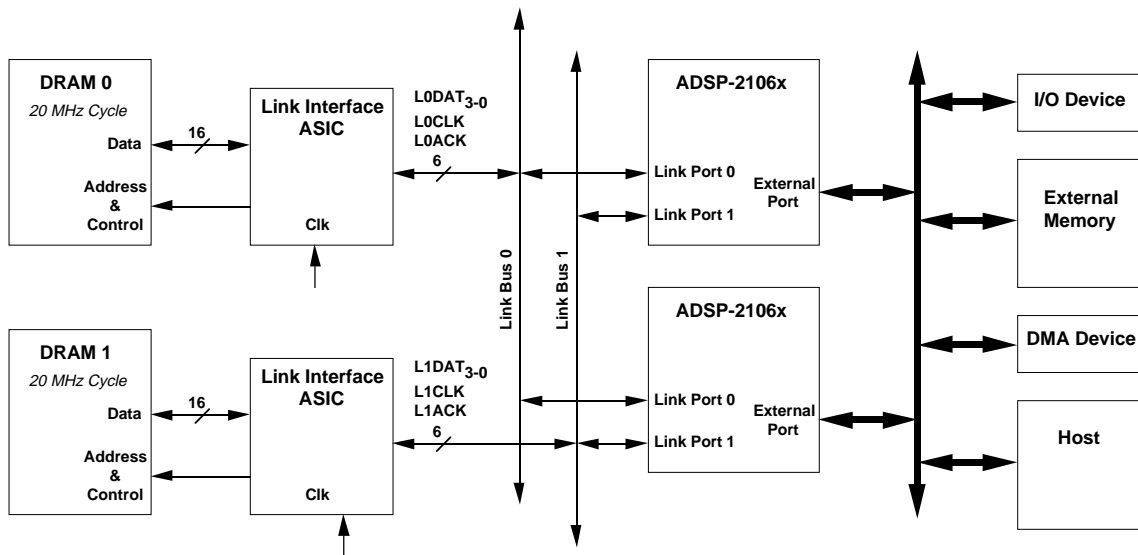


Figure 9.8 Local DRAM With Link Ports

9 Link Ports

9.11 PROGRAMMING EXAMPLES

Listings 9.1 and 9.2 illustrate two ways to perform link port data transfers.

9.11.1 Core-Driven Single-Word Transfers

Listing 9.1 shows an example of single-word data transfers controlled by the ADSP-2106x core.

9.11.2 DMA Transfers

Listing 9.2 shows an example DMA transfer program.

```
/*  
-----  
ADSP-2106x Core-Driven LINK Loopback Example  
  
This example shows an internally looped-back link port transfer. The core directly  
writes to the transfer link buffer (LBUF3) and reads from the receive link buffer  
(LBUF2). The core will hang on the read of LBUF2 until the data is ready. Loopback is  
achieved by assigning the transmit and receive link buffers to the same port (Port  
0).  
----- */  
  
#include "def21060.h"  
  
.segment/pm rst_svc; /* Reset vector from architecture file.*/  
    nop; /* First location is used for booting. */  
    jump start;  
.endseg;  
  
/*----- main routine -----*/  
.segment/pm pm48_lb0; /* Main code segment from arch file.*/  
  
start:  
    r0=0x0000c000; /* LCOM Register: 2x rate */  
    dm(LCOM)=r0; /* note: use r0=0x00010000 on rev. 0.X silicon */  
  
    r0=0x0003f03f; /* LAR Register: LBUF2->Port0, LBUF3->Port0 */  
    dm(LAR)=r0; /* All others inactive. */  
  
    r0=0x00009100; /* LCTL Register: 32-bit data, LBUF2=rx, LBUF3=tx */  
    dm(LCTL)=r0; /* Always write LCTL after LAR. */  
  
    r0=0x12345678; /* Test data to transmit. */  
    dm(LBUF3)=r0; /* Write to LBUF3 to transmit.*/  
  
    r1=dm(LBUF2); /* Read-Core will hang here until data is received.*/  
  
wait: jump wait;  
  
.endseg;
```

Link Ports 9

```
/*
-----
ADSP-2106x DMA-Driven LINK Loopback Example

This example shows an internally looped-back link port transfer.
Two DMA channels are used. Link buffer 3 (LBUF3) and corresponding
DMA channel 5 is used for transmit. Link buffer 2 (LBUF2) and
corresponding DMA channel 5 is used for receive. The LBUF2 interrupt
will occur when the DMA transfer is complete. Loopback is achieved
by assigning the transmit and receive link buffers to the
same port (Port 0).
-----*/

#define N 8
#include "def21060.h"

.segment/dm dm32_b1; /* Data segment name described in arch. file.*/
.var source[N] = 0x11111111, 0x22222222, 0x33333333, 0x44444444,
                0x55555555, 0x66666666, 0x77777777, 0x88888888;
.var destination[N];
.endseg;

.segment/pm rst_svc; /* Reset vector from arch. file. */
    nop; /* First location is used for booting.*/
    jump start;
.endseg;

.segment/pm lp2_svc; /* Link buffer 2 interrupt vector.*/
    jump lp2rx;
.endseg;

/*-----main routine-----*/
.segment/pm pm48_lb0; /* Main code segment from arch. file.*/

start:
    r0=source;
    dm(II5)=r0; /* Set DMA tx index to start of source buffer.*/
    r0=destination;
    dm(II4)=r0; /* Set DMA rx index to start of destination buffer.*/
    r0=1;
    dm(IM5)=r0; /* Set DMA modify (stride) to 1.*/
    dm(IM4)=r0;
    r0=@source;
    dm(C5)=r0; /* Set DMA count to length of data buffer.*/
    dm(C4)=r0;

    r0=0x0000c000; /* LCOM Register: 2x rate */
    dm(LCOM)=r0; /* note: use r0=0x00010000 on rev. 0.X silicon */
```

(listing continues on next page)

9 Link Ports

```
r0=0x0003f03f;      /* LAR Register: LBUF2->Port0, LBUF3->Port0 */
dm(LAR)=r0;         /* All others inactive. */

r0=0x0000b300;     /* LCTL Register: 32-bit data, LBUF2=rx, LBUF3=tx */
dm(LCTL)=r0;       /* Enable DMA on LBUF2 and LBUF3. */
                  /* This will start off the DMA transfer. */
                  /* Always write LCTL after LAR. */

bit set imask LP2I; /* Enable link buffer 2 interrupt. */
bit set mode1 IRPTEN; /* Global interrupt enable. */

wait: idle;        /* Wait for link buffer 2 interrupt.*/
      jump wait;   /* Will end up here after entire DMA completes.*/

/*_____Link Buffer 2 Receive Interrupt Routine_____*/
lp2rx: rti;        /* This interrupt will occur only once.*/

.endseg;
```

Listing 9.2 DMA Transfer Example

Link Ports 9

```
/*
ADSP-2106x LINK Token Pass Example

This is an example of software protocol for token ring passing through the link
ports using LSRQ. This code is to be loaded on both the original master and the
original slave. The code is ID intelligent for multiprocessor systems: ID1 is
the original master (transmitter) and ID2 is original slave (receiver). The
master transmits a buffer via dma through LPORT0 using LBUF3 and the slave
receives through LPORT0 using LBUF2. The slave then requests the token by
generating an LSRQ interrupt in the disabled link port of the master (LPORT0).
The master responds by sending the token release word and waiting to see if it
is accepted. The slave checks to see that it is the token release word and will
accept the token by emptying the master's link buffer FIFO within a
predetermined amount of time. If the token is accepted the slave will become
the master and transmit a buffer of data to the new slave. If the token is
rejected, the master will transmit a second buffer. When complete the original
master will finish by setting up LBUF2 to receive without DMA and the original
slave will set up LBUF3 to transmit without DMA. FLAG0 is used as a software
flac to indicate the original master.
*/

#include "def21060.h"

#define N 8          /* Size of buffer */

#define trw 0x0      /* Token release word */

#define orig_master_id 1 /* ID of SHARC to be original master */
#define orig_slave_id 2 /* ID of SHARC to be original slave */

.SEGMENT/DM    dm_data;

.var source_1[N]= 0x11111111, 0x22222222, 0x33333333, 0x44444444,
                0x11111111, 0x22222222, 0x33333333, 0x44444444;

.var source_2[N]= 0x55555555, 0x66666666, 0x77777777, 0x88888888,
                0x55555555, 0x66666666, 0x77777777, 0x88888888;
```

Listing 9.3 Link Token Passing Example (continues)

9 Link Ports

```
.var source_3[N]= 0x11111111, 0x22222222, 0x33333333, 0x44444444,  
                0x55555555, 0x66666666, 0x77777777, 0x88888888;  
  
.var destination_1[N];  
  
.var destination_2[N];  
  
.var destination_3;  
  
.ENDSEG;  
  
.SEGMENT/PM   isr_tabl; /* Interrupt Service Table   */  
  
        NOP; NOP;NOP;NOP; /* Reserved interrupt   */  
rst_svc:      nop; jump start; nop; nop;  
        NOP; NOP; NOP; NOP;  
sovfi_svc:    RTI; RTI; RTI; RTI;  
tmzhi_svc:    rti; RTI; RTI; RTI;  
vrpti_svc:    RTI; RTI; RTI; RTI;  
irq2_svc:     rti; RTI; RTI; RTI;  
irq1_svc:     rti; RTI; RTI; RTI;  
irq0_svc:     rti; RTI; RTI; RTI;  
        NOP; NOP; NOP; NOP;  
spr0_svc:     RTI; RTI; RTI; RTI;  
spr1_svc:     RTI; RTI; RTI; RTI;  
spt0_svc:     RTI; RTI; RTI; RTI;  
spt1_svc:     RTI; RTI; RTI; RTI;  
lp2_svc:      nop; jump lp2; nop; nop;  
lp3_svc:      nop; jump lp3; nop; nop;  
ep0_svc:      RTI; RTI; RTI; RTI;  
ep1_svc:      RTI; RTI; RTI; RTI;  
ep2_svc:      RTI; RTI; RTI; RTI;  
ep3_svc:      RTI; RTI; RTI; RTI;  
lsrq_svc:     nop; jump lsrq; nop; nop;  
cb7_svc:      RTI; RTI; RTI; RTI;  
cb15_svc:     RTI; RTI; RTI; RTI;  
tmzl_svc:     rti; RTI; RTI; RTI;  
  
.ENDSEG;
```

Listing 9.3 Link Token Passing Example (continues)

Link Ports 9

```
/*_____main routine_____*/

.SEGMENT/PM    pm_code;

start:
    bit set mode2 FLG0; /* Set Flag0 for output          */
    bit clr astat FLG0; /* Clear Flag0 for use as flag to test */
                        /* if this SHARC is the original master */
    r0=dm(SYSTAT);
    r0=FEXT r0 BY 8:3; /* Extract Processor ID from SYSTAT */

    r1=orig_master_id;
    r1=r0-r1;          /* Test if this SHARC is original */
    if eq jump start_as_master; /* master and jump appropriately */

    r1=orig_slave_id;
    r1=r0-r1;          /* Test if this SHARC is original */
    if eq jump start_as_slave; /* slave and jump appropriately */

    idle;
    nop;

/*_____Start of Original Master Routine_____*/

start_as_master:

    bit set astat FLG0; /* Set Flag0 to signify original master */

    r0=source_1;
    dm(II5)=r0; /* Set DMA tx index to start of source buffer */

    r0=1;
    dm(IM5)=r0; /* Set DMA modify (stride) to 1 */

    r0=@source_1;
    dm(C5)=r0; /* Set DMA count to length of data buffer */
```

Listing 9.3 Link Token Passing Example (continues)

9 Link Ports

```
r0=0xc000; /* LCOM Register: 2x rate on LBUF3, */
dm(LCOM)=r0; /* note:use r0=0x00010000 on rev. 0.X silicon */

r0=0x3f1ff; /* LAR Register: LBUF3->port 0 */
dm(LAR)=r0; /* All others inactive */

bit set imask LP3I; /* Enable Link buffer 3 interrupt */
bit set model IRPTEN; /* Global interrupt enable */

r0=0x0000b000; /* LCTL Register: 32-bit data, LBUF3=tx */
dm(LCTL)=r0; /* enable DMA on LBUF3 */
/* This will start off the DMA transfer */
/* Always write LCTL after LAR */

wait_1: idle; /* Wait for Link buffer 3 interrupt */
jump wait_1; /* Will end up here after entire DMA complete */
*/
nop; /* All master interrupts will come back to here */
nop;

/*_____Link buffer 3 Interrupt Routine_____*/

lp3:
if NOT FLAG0_IN jump lp3_orig_slave; /* Test for original master */
nop; /* and jump appropriately */
nop;

/*_____Link buffer 3 Tx finish Interrupt Routine_____*/

lp3_orig_master:
```

Listing 9.3 Link Token Passing Example (continues)

Link Ports 9

```
/*_____Allow for pulldown delay on LxACK of the
slave_____*/

    bit clr imask LSRQI;    /* Disable Link port service request interrupt
*/

    r0=0x10;
    dm(LSRQ)=r0;          /* Unmask LSRQ lport 0 transmit request status    */

    r0=0x00000000;
    dm(LCTL)=r0;          /* LCTL: disable all LBUFs          */

disabled1:
    r0=dm(LSRQ);          /* Check to ensure that the pull down on LxACK    */
    r0=FEXT r0 BY 20:1; /* has pulled the LxACK low. Both the original */
    r0=pass r0;          /* slave and original master will be in sync. */
    if NE jump disabled1; /* The next assertion of LxACK will signify to */
                          /* the master that slave is requesting the token*/

/
*_____*/

    r0=0x00000000;
    dm(LCTL)=r0;          /* disable all LBUFs */

    bit set imask LSRQI; /* Enable Link port service request interrupt
*/

    r0=0x10;
    dm(LSRQ)=r0;          /* Unmask LSRQ lport 0 transmit request status    */

    rti;

/*_____Link buffer 2 Tx finish Interrupt Routine_____*/
```

Listing 9.3 Link Token Passing Example (continues)

9 Link Ports

```
lp3_orig_slave:
    /* Finish by setting up Tx without DMA */
    bit clr imask LP3I; /* disable Link buffer 2 interrupt */

    r0=0x3f1ff;
    dm(LAR)=r0; /* LAR Register: LBUF3->port 0 */

    r0=0x00009000;
    dm(LCTL)=r0; /* enable LBUF3 Tx, No DMA */

    rti;

/* _____Link Service Request Interrupt Routine_____ */

lsrq:
    bit clr imask LP3I; /* disable Link buffer 3 interrupt */

    r0=0x00009000; /* LCTL Register:32-bit data, LBUF3=tx */
    dm(LCTL)=r0; /* disable DMA on LBUF3 */

    r0=trw; /* Get token release word */

    dm(LBUF3)=r0; /* Send token release word to slave */
    dm(LBUF3)=r0; /* fill slave's and master's LP fifos by */
    dm(LBUF3)=r0; /* writing four times, leaving the fifos */
    dm(LBUF3)=r0; /* completely filled on both sides */

token_read:
    r1=0x40; /* check if slave read the token */
    r0=dm(LCOM); /* check if slave read the first word */
    r0=r0 AND r1; /* to be sure they are in sync for the */
    if NE jump token_read; /* reject test */

    LCNTR=10, DO wait_for_slave UNTIL LCE;
wait_for_slave: nop; /* Give slave chance to accept or reject token
```

Listing 9.3 Link Token Passing Example (continues)

Link Ports 9

```
*/

r1=0xc0;          /* check if slave wants the token */
r0=dm(LCOM);      /* check if slave emptied the fifos */
r0=r0 AND r1;    /* within 10 cycles */
if NE jump second_master_mode; /* else second master mode */

slave_mode:

/*_____Protection to avoid two transmitting link
ports_____*/

bit clr imask LSRQI; /* Disable Link port service request interrupt
*/

r0=0x10;
dm(LSRQ)=r0;        /* Unmask LSRQ lport 0 transmit request status */

r0=0x00000000;
dm(LCTL)=r0;       /* LCTL: disable all LBUFs */

slave_disabled:
r0=dm(LSRQ);       /* Check to ensure that the original slave */
r0=FEXT r0 BY 20:1; /* is now disabled. If disabled, will deassert */
r0=pass r0;        /* LxACK and stop generating LxTRQ */
if NE jump slave_disabled;

/
*_____*/

r0=0x3fe3f;
dm(LAR)=r0;        /* LAR Register: LBUF2->port 0 */

r0=0x00000100;
dm(LCTL)=r0;       /* LCTL: enable LBUF2 (Rx), non DMA */
```

Listing 9.3 Link Token Passing Example (continues)

9 Link Ports

```
r0=dm(LBUF2); /* read DMA size */
dm(C4)=r0; /* Set DMA count to length of data buffer */

r0=destination_3;
dm(II4)=r0; /* Set DMA rx index to start of destin buffer */

r0=1;
dm(IM4)=r0; /* step size */

r0=0x00000300;
dm(LCTL)=r0; /* enable LBUF2 DMA Rx */

bit clr irpt1 LP2I; /* clear pending Link buffer 2 interrupt */
bit set imask LP2I; /* Enable Link buffer 2 interrupt */
bit set model IRPTEN; /* Global interrupt enable */

rti;

second_master_mode:

token_read2:
    r1=0xC0; /* check if slave read the tokens */
    r0=dm(LCOM); /* check if slave emptied fifos */
    r0=r0 AND r1; /* to be sure they are in sync for the */
    if NE jump token_read2; /* the second DMA transfer */

    r0=0x3fe3f;
    dm(LAR)=r0; /* LAR Register: LBUF2->port0 */

    r0=0x00000900;
    dm(LCTL)=r0; /* LBUF2: Tx, non DMA */

    r0=@source_2;
    dm(LBUF2)=r0; /* Tx size of DMA to the slave */

    r0=source_2;
    dm(II4)=r0; /* Set DMA tx index to start of source buffer. */

    r0=1;
```

Listing 9.3 Link Token Passing Example (continues)

Link Ports 9

```
dm(IM4)=r0; /* Set DMA modify (stride) to 1 */

r0=@source_2;
dm(C4)=r0; /* Set DMA count to length of data buffer. */

r0=0x00000b00; /* LCTL Register:32-bit data, LBUF2=tx */
dm(LCTL)=r0; /* enable DMA on LBUF2. */
/* This will start off the DMA transfer. */
/* Always write LCTL after LAR. */

bit clr irpt1 LP2I; /* clear pending Link buffer 2 interrupt */
bit set imask LP2I; /* Enable Link buffer 2 interrupt */
bit set model IRPTEN; /* Global interrupt enable */

rti;

/*_____Link buffer 2 Interrupt Routine_____*/

lp2:

if NOT FLAG0_IN jump lp2_orig_slave; /* Test for original master */
nop; /* and jump appropriately */
nop;

/*_____Link buffer 2 Rx finish Interrupt Routine_____*/

lp2_orig_master:
/* Finish by setting up Rx without DMA */
r0=0x3fe3f;
dm(LAR)=r0; /* LAR Register: LBUF2->port0 */

r0=0x00000100;
dm(LCTL)=r0; /* LBUF2: Rx, non DMA */

rti; /* This interrupt will occur only once. */
```

Listing 9.3 Link Token Passing Example (continues)

9 Link Ports

```
/*_____Link buffer 2 Rx Interrupt Routine_____*/
lp2_orig_slave:

/*_____Allow for pulldown delay on LxACK of the
slave_____*/

    bit clr imask LSRQI;    /* Disable Link port service request interrupt
*/

    r0=0x10;
    dm(LSRQ)=r0;    /* Unmask LSRQ lport 0 transmit request status    */

    r0=0x00000000;
    dm(LCTL)=r0;    /* LCTL: disable all LBUFs    */

disabled2:
    r0=dm(LSRQ);    /* Check to ensure that the pull down on LxACK    */
    r0=FEXT r0 BY 20:1; /* has pulled the LxACK low. Both the original */
    r0=pass r0;    /* slave and original master will be in sync. */
    if NE jump disabled2; /* The next assertion of LxACK will signify to */
        /* the master that slave is requesting the token*/

/
*_____*/

    bit clr imask LP2I;    /* disable Link buffer 2 interrupt    */

    r0=0x3fe3f;
    dm(LAR)=r0; /* LAR Register: LBUF2->port 0    */
```

Listing 9.3 Link Token Passing Example (continues)

Link Ports 9

```
r0=0x00000100;
dm(LCTL)=r0;      /* LBUF2=rx (slave), No DMA */

bit clr model NESTM; /* disable interrupt nesting */
                /* to avoid breaking sync */

r0=dm(LBUF2); /* read token permission from master */

/* The following check if the first word after DMA is token */
/* permission. If it is not, the slave read other dummy words */
/* and continue to be slave. If it is token permission, */
/* continue the original flow and the slave will decide if */
/* if will accept the permission. */

r1=trw; /* token release word */

r0 = r0 - r1; /* test received word and set ALU flag */
if NE jump second_slave_mode; /* not token permission */
nop;
nop;

/* The following 3 lines shows how to reject token */
/* If commented-out, this slave will change to master */
/* if uncommented, this slave will continue to be slave */

/* LCNTR=20, DO reject_token UNTIL LCE;
reject_token: nop;
jump second_slave_mode;
nop; /* delay read of incoming message */
nop;

master_mode:

r0=dm(LBUF2); /* Read 3 times to clean the */
r0=dm(LBUF2); /* ex-master's LBUF. So, ex-master */
r0=dm(LBUF2); /* will release the token */
```

Listing 9.3 Link Token Passing Example (continues)

9 Link Ports

```
/*_____Protection to avoid two transmitting link
ports_____*/

    bit clr imask LSRQI;    /* Disable Link port service request interrupt
*/

    r0=0x10;
    dm(LSRQ)=r0;    /* Unmask LSRQ lport 0 transmit request status    */

    r0=0x00000000;
    dm(LCTL)=r0;    /* LCTL: disable all LBUFs    */

disabled3:
    r0=dm(LSRQ);    /* Check to ensure that the pull down on LxACK    */
    r0=FEXT r0 BY 20:1; /* has pulled the LxACK low. Both the original */
    r0=pass r0;    /* slave and original master will be in sync.    */
    if NE jump disabled3; /* The next assertion of LxACK will signify the */
    /* master has become the slave.    */

slave_enabled:
    r0=dm(LSRQ);    /* Check to ensure that the original master has    */
    r0=FEXT r0 BY 20:1; /* become the slave by observing that the */
    r0=pass r0;    /* assertion of LxACK has generated an LxRRQ */
    if EQ jump slave_enabled;

/
*_____*/

    r0=0x3f1fff;
    dm(LAR)=r0;    /* LAR Register: LBUF3->port 0    */

    r0=0x00009000;
    dm(LCTL)=r0;    /* LBUF3=tx, no DMA    */

    r0=@source_3;    /* Tx DMA size    */
    dm(LBUF3)=r0; /* send DMA size across    */
```

Listing 9.3 Link Token Passing Example (continues)

Link Ports 9

```
    r0=0xc0;
wait: r1=dm(LCOM);      /* check if LBUF3 is empty      */
    r0=r0 AND r1;
    if NE jump wait; /* if DMA size not thru, wait      */
    nop;

    r0=source_3;
    dm(II5)=r0; /* Tx DMA setup          */

    r0=1;
    dm(IM5)=r0; /* Set modify to 1          */

    r0=@source_3;
    dm(C5)=r0; /* Set DMA count to length of data buffer */

    r0=0x0000b000; /* LCTL Register:32-bit data, LBUF3=Tx */
    dm(LCTL)=r0; /* enable DMA on LBUF3          */
                /* This will start off the DMA transfer */
                /* Always write LCTL after LAR          */

    bit clr irpt1 LP3I; /* clear pending Link buffer 3 interrupt. */
    bit set imask LP3I; /* Enable Link buffer 3 interrupt      */

    rti;

second_slave_mode:

    r0=dm(LBUF2); /* read 3 times to clean the */
    r0=dm(LBUF2); /* ex-master's LBUF. So, ex-master */
    r0=dm(LBUF2); /* will release the token */

    r0=0x3f1fff;
    dm(LAR)=r0; /* LAR Register: LBUF3->port 0 */

    r0=0x00001000;
    dm(LCTL)=r0; /* enable LBUF3 Rx, No DMA */

    r1=dm(LBUF3); /* read new DMA size */
```

Listing 9.3 Link Token Passing Example (continues)

9 Link Ports

```
r0=destination_2;
dm(II5)=r0; /* Set DMA rx index to start of dest buffer */

r0=1;
dm(IM5)=r0; /* Set modify to 1 */

r0=@destination_2;
dm(C5)=r1; /* real DMA Rx size should be got from master */

r0=0x00003000; /* LCTL Register:32-bit data, LBUF3=Rx */
dm(LCTL)=r0; /* enable DMA on LBUF3 */
/* This will start off the DMA transfer */
/* Always write LCTL after LAR */

bit clr irptl LP3I; /* clear pending Link buffer 3 interrupt. */
bit set imask LP3I; /* Enable Link buffer 3 interrupt */

rti;

/*_____Start of Original Slave Routine_____*/

start_as_slave:

r0=destination_1;
dm(II4)=r0; /* Set DMA rx index to start of dest buffer */

r0=1;
dm(IM4)=r0; /* Set DMA modify (stride) to 1 */

r0=@destination_1;
dm(C4)=r0; /* real DMA Rx size should be from master */

r0=0xc000; /* LCOM Register: 2x rate, */
dm(LCOM)=r0; /* note:use r0=0x10000 on rev. 0.X silicon */
/* original : 0x0000c000 */

r0=0x3fe3f; /* LAR Register: LBUF2->port 0 */
dm(LAR)=r0; /* All others inactive */
```

Listing 9.3 Link Token Passing Example (continues)

Link Ports 9

```
bit set imask LP2I;      /* Enable Link buffer 2 interrupt    */
bit set model IRPTEN;   /* Global interrupt enable */

r0=0x00000300;         /* LCTL Register:32-bit data, LBUF2=Rx */
dm(LCTL)=r0;          /* enable DMA on LBUF2          */
                    /* This will start off the DMA transfer */
                    /* Always write LCTL after LAR      */

wait_2:  idle;          /* Wait for Link buffer 2 interrupt */
        jump wait_2;   /* Will end up here after entire DMA complete */
*/
        nop;          /* All slave interrupts will come back to here */
        nop;

.ENDSEG;
```

Listing 9.3 Link Token Passing Example

9 Link Ports