Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

Using Token Passing to Control SHARC Link Port Bi-directional Communication

Last Modified: *9/23/97*

Contributed by: *Richard G.*

Overview

Link port communication on the SHARC is bi-directional and can be switched at runtime through software control. For bi-directional communication to work, each link port must know which is the transmitter and which is the receiver at any given time. You can use token passing to accomplish this type of software control. This document describes how the link token pass example code implements token passing on the SHARC.

Token Passing

Token passing is a software control scheme designed to control the direction of communication between two or more devices. It enables a program to establish a master, or transmitter, in a given communication system and to transfer that mastership to any of the slaves at a given time. The token, a software flag, resides in the master link port and can be released to a slave upon request or at the discretion of the master.

The link ports of the SHARC implement this protocol through the link service request interrupt. A slave can use this feature to request the token from the master. The master, through a link service request interrupt service routine, recognizes the request for the token and decides whether or not to release the token to the slave. Releasing the token causes the master to set up its link port to receive thus becoming a slave. The slave, having verified that it has been given the token, will then set up its link port to transmit and becomes the master.

Using the Link Token Pass

To use the link token pass example code, you must load it on both the original master and the original slave. The example code is ID intelligent for multiprocessor systems: ID#1 is the original master (transmitter) and ID#2 is original slave (receiver).

To implement token passing, the example code follows this procedure:

1) The master transmits a buffer via DMA through LPORT0 using LBUF3, and the slave receives through LPORT0 using LBUF2.

2) The slave requests the token by generating an LSRQ (link port service request) interrupt in the disabled link port of the master (LPORT0).

3) The master responds by sending the token release word and waits to see if it is accepted.

4) The slave checks the token release word and accepts the token by emptying the master's link buffer FIFO within a predetermined amount of time.

5) If the token is accepted the slave becomes the master and transmits a buffer of data to the new slave. If the token is rejected, the master transmits a second buffer.

6) When the transmission is complete the original master finishes by setting up LBUF2 to receive without DMA, and the original slave sets up LBUF3 to transmit without DMA.

When implementing a software protocol for token passing, it is important to note the following:

- Make sure that only one link buffer is enabled to transmit at a time. Otherwise data will be lost since neither link port will drive LxACK. The example code polls the LSRQ register status bits to ensure that the master becomes the slave before the slave becomes the master to avoid the two transmitter conflict.
- When using the status bits of the LSRQ register for status detection and a link port that is configured to receive is disabled, there will be an RC delay before the 50k ohm pull-down resistor on LxACK can pull the value below logic threshold. In this instance, if the LxTM (link port transmit request) bit is set in the LSRQ register, then an LSR (link port service request) will be latched and the LSRQ interrupt may be generated unintentionally.
- In an application that uses timing control loops to synchronize parallel code execution (such as the example code), synchronization must not be disrupted by unrelated influences at critical sections. Disabling interrupt nesting is one of the techniques the example code uses to control synchronization.

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


#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;

.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;


/*_____________________MAIN ROUTINE_____________________________________________*/

.SEGMENT/PM   pm_code;


start:
      bit set mode2 FLG0O;    /* 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       */

      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 mode1 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:


/*_________________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_________________*/

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  */


      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             */

      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 destination buffer
      */

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

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

```
      bit clr irptl LP2I;     /* clear pending Link buffer 2 interrupt        */
      bit set imask LP2I;     /* Enable Link buffer 2 interrupt               */
      bit set mode1 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;
      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 irptl LP2I;     /* clear pending Link buffer 2 interrupt        */
      bit set imask LP2I;     /* Enable Link buffer 2 interrupt               */
      bit set mode1 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.         */



/*_________________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                */

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

      bit clr mode1 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                 */



/*_______________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=0x3f1ff;
      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                         */

      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 irptl 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=0x3f1ff;
      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                     */

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

      bit set imask LP2I;     /* Enable Link buffer 2 interrupt              */
      bit set mode1 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;
```