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

Examining ADSP-21160 Link Port Backward Compatibility to the ADSP-2106x Link Ports

Overview

This Engineer's Note will discuss the ADSP-21160 link port compatibility to previous SHARC DSPs, along with software and hardware related issues. Software examples for core- and DMA driven reads and writes to an ADSP-21062 will be provided.

General Operation

The ADSP-21160 has six 8-bit wide link ports that provide additional I/O capabilities. These link ports can connect to other DSPs' or peripherals' devices supporting the link port protocol as detailed in the ADSP-21160 data sheet. These bi-directional ports consists of eight data lines (LxDAT7-0), a link clock line (LxCLK), and a link acknowledge line (LxACK). The LxCLK line allows asynchronous data transfers and the LxACK line provides handshaking. When a link port is not enabled, LxDAT7-0, LxCLK and LxACK are three-stated. The ADSP-21160 link ports can be configured to use four data lines for compatibility with older SHARC link ports.

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. A link-port-transmitted word consists of 4 bytes (for a 32-bit word) or 6 bytes (for a 48-bit word). The transmitter asserts the clock (LxCLK) high with each new byte (nibble) of data. The falling edge of LxCLK is used by the receiver to latch the byte (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 6 or 8 bytes). If LxACK is deasserted at that time, the transmitter does not transmit the new word.

The transmitter leaves LxCLK high and continues to drive the first byte if LxACK is deasserted. When LxACK is eventually asserted again, LxCLK goes low and begins transmission of the next word. If the transmit buffer is empty, LxCLK remains low until the buffer is refilled, regardless of the state of LxACK. 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 the processor clock frequency. Link ports have the capability of running at 80 MHz rates accordingly at frequencies up to the same speed as the DSP's internal clock (see timing requirements as detailed in the ADSP-21160 and ADSP-21062 Data sheet, for exceptions and workarounds see ADSP-21160 and ADSP-21062 Hardware Anomaly List), letting each port transfer either 4 or 8 (ADSP-21160) bits of data per internal clock cycle. Note that the ADSP-21160M's internal clock (CK) switches at higher frequencies than the system input clock (CLKIN). The ratio between the DSP's internal clock (CK) frequency and external (CLKIN) clock frequency is programmed with the CLK_CFG3–0 pins, during reset.

To determine switching frequencies for the link ports, divide down the internal clock (CK), using the programmable divider control of each link port (LxCLKD1–0). Calculation of link receiver data setup and hold relative to link clock is required to determine the maximum allowable skew that can be introduced in the transmission path between LxDATA and LxCLK.

The links are designed to drive transmission lines with characteristic impedances of 50 Ohm or greater. 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) is equal to the characteristic impedance of the transmission line. For more information on how to calculate the value of a series termination read the appropriate chapters of any high-speed digital design book.

Link Port Configuration

There are four control registers associated with link port control functions. The SYSCON, LCOM, LAR, and LCTLx registers control the link ports operating modes for the I/O processor. To configure link port operations, these registers should be set in the following order: SYSCON, LAR, LCOM, LCTLx. Before reassigning a link port with the LAR register, disable the link port's assigned buffer with the LCTLx register.

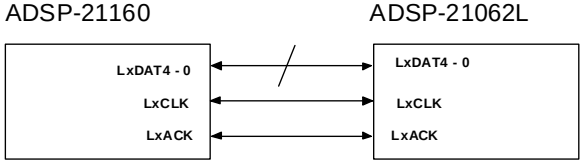
There are six registers, LBUF0-5, that buffer the data flow through the link ports. These registers are independent of the link ports and may be connected to any of the six link ports. The Link Assignment Register (LAR) assigns the link buffer-to-port connections. The link buffers read from or write to internal memory under DMA or processor core control. Each link buffer consists of an external and an internal 48-bit register. When transmitting, the internal register is used to accept core data or DMA data from internal memory.

When receiving, the external register performs the packing and unpacking for the link port, most significant nibble or byte first. These two registers form a two-stage FIFO for the LBUFx buffer. Two writes (32- or 48-bit) can occur to the register by the DMA or the core, before it signals a full condition. Full/empty status for the link buffer FIFOs is given by the LxSTAT bits of the LCOM register. (See Core driven transfer example) This status is cleared for a link buffer when its LxEN enable bit is cleared in the LCTLx register. If a read is attempted from an empty receive buffer, the core stalls (hangs) until the link port completes reception of a word. If a write is attempted to a full transmit buffer, the core stalls until the external device accepts the complete word. The SYSCON register contains the BHD bit, this bit can be set preventing the processor core from detecting a buffer-related stall condition.

Overall there are six bit fields in the Link Port Control Registers LCTLx, controlling for each link buffer appropriate functions. For bit definitions see the ADSP-21160 Hardware Reference Manual.

The LCOM Register contains status bits for each link buffer. For bit definitions see the ADSP-21160 Hardware Reference Manual. Note that the link port error status bit is set when the link port buffer has not received a full word. This means that the error bit would only be cleared on occasions when the link buffer has received a multiple of 8 or 12 nibbles indicating that a full 32 or 48 bit word has been received. If you were to check the status bit at a time when you were not sure that a word has been completely received the error status bit could be set indicating that the transfer is not complete. If you are checking the error status bit at an appropriate time and the bit remains set after a word has been received, it may indicate that there is in fact an error possibly caused by a clock glitch bringing in unintended nibbles.

ADSP-21062 Compatible Mode



ADSP-21160 link ports are logically compatible with ADSP-2106x link ports. However only an ADSP-2106xL (Low Power 3.3V) is electrical compatible. If a 5V device is used a voltage conversion stage must be introduced into the link path. Furthermore the LxDPWID bit in the LCTLx

register must be cleared in order to enable a ADSP-2106x compatible 4-bit data path.

Example: Both DSPs using same CLKin frequency i.e. 33 MHz, ADSP-21160 CK frequency is set to 2xCLKin (CK=66MHz)

ADSP-21160		RECEIVE OCCURS @ IN MHZ	TRANSMIT OCCURS @ IN MHZ
LXCLKD1	LXCLKD0		
0	1	<=66	=66
1	0	<=66	=33
1	1	<=66	=22
0	0	<=66	=16.5
ADSP-21062		RECEIVE OCCURS @ IN MHZ	TRANSMIT OCCURS @ IN MHZ
LCLK2X			
0		<=33	=33
1		>33 & <=66	=66

Theoretically maximum throughput (LCLK=80MHz) in both directions can be obtained by setting the lower LxCLKD bit in the LCTLx register (Selecting a full core-rate transfer frequency) and setting the ADSP-2106x LCLK2x bit in the LCOM register (Transfers at 2x clock rate). Make sure that you met the timing requirements as detailed in the ADSP-21160 and ADSP-21062 data sheets. The LDATA and LCLK signals start at the same time from the source device and reach the destination at the same time since they follow the same length of transmission lines. However, there is a pin to pin skew due to the mismatches in loadings and the transmission line length. For an example calculation on how to calculate the maximum skew allowed between LCLK and LDATA at the receiver link port, see the calculation below. The smallest calculated number is the maximum skew allowed between LCLK and LDATA, in order to ensure reliable bi-directional transfers.
These calculations are worst case; they are made directly from speed specifications. Therefore they include multiple specification guardbands.

ADSP-21160M Receiver Side

ADSP-21062L, LCLK2x=0, t_{LCLK} = 30.3 ns :

Setup Skew = $t_{LCLKTWH}$ (Min) - t_{DLDCH} -t_{SLDCL}
= (t_{LCLK} /2)-1.5 - 2.5 - 2.5 = 8.65 ns

Hold Skew = $t_{LCLKTWL}$ (Min) + t_{HLDCH} -t_{HLDCL}
= (t_{LCLK} /2)-1.0 – 2.5 - 2.5 = 9.15 ns

ADSP-21062L Receiver Side

ADSP-21160M, LxCLKD1=1, t_{LCLK} = 30.3 ns :

Setup Skew = $t_{LCLKTWH}$ (Min) - t_{DLDCH} -t_{SLDCL}
= (t_{LCLK}/2)-1.5 – 6 - 3.0 = 4.65 ns

Hold Skew = $t_{LCLKTWL}$ (Min) + t_{HLDCH} -t_{HLDCL}
= (t_{LCLK} /2)-1.5 – 2.0 – 3.0 = 8.65 ns

Parameter are based on datasheet revision ADSP-21062L REV.C , ADSP-21160 REV.0

Core-Driven Transfer Examples

Code Listings 1.1 to 1.4 shows how the core reads and writes the link buffers, by polling the full or empty status bits. In these examples a ADSP-21160 link port (4) is connected to ADSP-21062L link port (5). The core transfers N values. Received values are stored in a data buffer.

DMA-Driven Transfer Examples

Code Listings 2.1 to 2.4 shows single mode DMA transfers. In these examples a triangle wave of length N is computed and stored in a data buffer. After that the core set up a DMA transfer. By writing the LCTLx register the I/O processor starts transferring the whole block of data. Note that programs should not modify an active DMA channel's bits in the LCTLx register; other than to disable the channel by clearing the LxDEN bit. The I/O processor stores the received block in DM memory.

```
/*-------------------------------------------------------------------------
Code to receive with a ADSP-21160 link port(4) from a ADSP-21062. The core directly
reads N values from the link buffer(LBUF4). The core will hang on the read of
LBUF4 until data is ready to receive. Received values are stored DM Memory.
-------------------------------------------------------------------------*/

#include    "def21160.h"

#define N       1024

/*------------ DM data --------------*/
.section/dm seg_dmda;
/*data to be transmitted to SHARC*/
.VAR dest[N];

/*---- PM interrupt vector code -----*/
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump begin; nop; nop;

/*------ program memory code -------*/
.section/pm seg_pmco;

begin:
             b2=dest;
             l2=N;
             m2=1;

             ustat1 = dm(SYSCON);       /* Clear Buffer Hang Disable     */
             bit clr ustat1 BHD;
             dm(SYSCON) = ustat1;

             r0=0;                      /*clear link control registers*/
             dm(LAR)=r0;
             dm(LCTL1)=r0;

             r0=0x4000;                 /* LBUF4->LPORT4, all other ports are disabled */
             dm(LAR)=r0;
             r0=L4EN | L4CLKD1;         /*LBUF4 enable port, RX, 32bit word size,
                                                    1/2 clock, 4bit data path*/
             dm(LCTL1)=r0;              /*always write LCTLx after LAR*/
             r2=N;

readstat:
             ustat1=dm(LCOM);           /* read Link Port common Control Register */
             bit tst ustat1 L4STAT0 ;   /* test buffer full */
             if not tf jump readstat;   /* test until word is received */
             r2=r2-1;                   /* count received words */

             if GT jump readstat (DB);
             r1=dm(LBUF4);              /* save received words until count expired*/
             dm(i2,m2)=r1;

             /* terminate and wait */
wait1:
jump wait1;
/*-------------------------------------------------------------------------*/
```

Listing 1.1 ADSP-21160 Core-Driven Receive Example

```
/*-------------------------------------------------------------------------
Code to transmit with a ADSP-21160 link port(4) to a ADSP-21062. The core directly
writes N values to the link buffer(LBUF4). The core stops writing words to the
buffer, if the L4STAT1 bit indicates a full condition. In order to transmit to an
ADSP-21160 using 8 bit data width the corresponding LxDPWID must be set.
-------------------------------------------------------------------------*/

#include    "def21160.h"

#define N       1024
```

```
/*------------ DM data --------------*/
.section/dm seg_dmda;


/*---- PM interrupt vector code -----*/
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump begin; nop; nop;


/*------ program memory code -------*/

.section/pm seg_pmco;

begin: r0=0;                                    /*clear link control registers*/
             dm(LAR)=r0;
             dm(LCTL1)=r0;

             ustat1 = dm(SYSCON);               /* Clear Buffer Hang Disable */
             bit clr ustat1 BHD;
             dm(SYSCON) = ustat1;

             r0=0x4000;                        /* LBUF4->LPORT4, all other ports are disabled
                                               */
             dm(LAR)=r0;

             r0=L4EN | L4CLKD1 | L4TRAN;        /*LBUF4 enable port and dma, transmit,
                                                32bit word size, 1/2 clock, 4bit data   path*/
             dm(LCTL1)=r0;                      /*always write LCTLx after LAR*/
             r1=1;
             r2=N;

readstat:
             ustat1=dm(LCOM);                   /* read Link Port common Control Register */
             bit tst ustat1 L4STAT1;
             if tf jump readstat;               /* test until buffer is empty */
             if GT jump readstat (DB);
             dm(LBUF4)=r2;                      /* write value to link port buffer */
             r2=r2-r1;


/* terminate and wait */
wait1:
jump wait1;
/*-------------------------------------------------------------------------*/
```

Listing 1.2 ADSP-21160 Core-Driven Transmit Example

```
/*-------------------------------------------------------------------------
Code to receive with a ADSP-21062 link port(5) from a ADSP-21160. The core directly
reads N values from the link buffer(LBUF2). The core will hang on the read of
LBUF4 until data is ready to receive. Received values are stored DM Memory.
-------------------------------------------------------------------------*/

#include     "def21060.h"
#define N 1024

/*------------ DM data --------------*/
.section/dm seg_dmda;
.var dest[N];

/*---- PM interrupt vector code -----*/
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump start; nop; nop;

/*------ program memory code -------*/
.section/pm seg_pmco;

start: ustat1 = dm(SYSCON);      /* Clear Buffer Hang Disable */
```

```
       bit clr ustat1 0x800;
       dm(SYSCON) = ustat1;

       b1=dest;                    /* setup circular buffer */
       l1=N;
       m1=1;

setup: r0=0x00000000;
       dm(LCTL)=r0;                /* clear link control registers */
       dm(LAR)=r0;

       r0=0x0000000;              /* LCKX2x=0 transfers occurring at CK frequency */
       dm(LCOM)=r0;

       r0=0x3ff7f;                 /* LBUF2 <==> LPORT5, all others disabled */
       dm(LAR)=r0;

       r9=0x00000100;             /* lbuf2 enable, receive, 32-bit word, 4-bit bus width */
       dm(LCTL)=r9;
       r2=N;
readstat:
       ustat1=dm(LCOM);            /* read Link Port common Control Register */
       bit tst ustat1 0x10;        /* test buffer full */
       if not tf jump readstat;    /* test until word is received */
       r2=r2-1;                    /* count received words */
       if GT jump readstat (DB);
       r1=dm(LBUF2);
       dm(i1,m1)=r1;               /* save received words until count expired*/

       /* terminate and wait */
wait1: idle;
       jump wait1;
.ENDSEG;
```

Listing 1.3 ADSP-21062 Core-Driven Receive Example

```
/*-------------------------------------------------------------------------------
Code to transmit with a ADSP-21062 link port(5) to a ADSP-21160. The core directly
writes N values to the link buffer(LBUF2). The core stops writing words to the buffer,
if the L2STAT1 bit indicates a full condition.
-------------------------------------------------------------------------------*/

#include     "def21060.h"

#define N 1024

/*------------ DM data --------------*/
.section/dm seg_dmda;


/*---- PM interrupt vector code -----*/
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump start; nop; nop;

/*------ program memory code -------*/
.section/pm seg_pmco;

start: ustat1 = dm(SYSCON);       /* Clear Buffer Hang Disable */
       bit clr ustat1 0x800;
       dm(SYSCON) = ustat1;

setup: r0=0x00000000;
       dm(LCTL)=r0;                /* clear link control registers */
       dm(LAR)=r0;

       r0=0x0000000;               /* Lclk2x=0 (0x0004000 for Lclk2x=1) */
       dm(LCOM)=r0;

       r0=0x3ff7f;                 /* LBUF2 <==> LPORT5, all others disabled */
       dm(LAR)=r0;
```

```
       r9=0x00000900;            /* lbuf2 enable, transmit, 32-bit word, 4-bit bus width */
       dm(LCTL)=r9;              /* always write LCTLx after LAR */
       r2=N;

readstat:
       ustat1=dm(LCOM);          /* read Link Port common Control Register */
       bit tst ustat1 0x20;
       if tf jump readstat;      /* test until buffer is empty */
       if GT jump readstat (DB);
       dm(LBUF2)=r2;             /* write value to link port buffer */
       r2=r2-1;

/* terminate and wait */

wait1: idle;
       jump wait1;
/*-------------------------------------------------------------------------*/
.ENDSEG;
```

Listing 1.4 ADSP-21062 Core-Driven Transmit Example

```
/*-------------------------------------------------------------------------
Code to receive with a ADSP-21160 link port(4) from a ADSP-21062. The DSP's I/O
processor manages the DMA transfer through the link port. The DMA operation transfers and
stores an entire block of data in DM data memory. In order to receive from a ADSP-21160 using
8 bit data width the corresponding LxDPWID must be set.
-------------------------------------------------------------------------*/

#include    "def21160.h"

#define N       512             /* length of receive data */

/*------------ DM data --------------*/
.section/dm seg_dmda;
.var dest[N];

/*---- PM interrupt vector code -----*/
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump begin; nop; nop;

/*------ program memory code -------*/
.section/pm seg_pmco;


begin: ustat1 = dm(SYSCON);       /* Clear Buffer Hang Disable     */
       bit clr ustat1 BHD;
       dm(SYSCON) = ustat1;

       r0=0;                      /* Clear Link Control Registers */
       dm(LAR)=r0;
       dm(LCTL1)=r0;

                                  /* Set up DMA pointers */
       r0=dest;                   /* Set DMA channel 8 Index Register */
       dm(II8)=r0;

       r0=1;
       dm(IM8)=r0;                /* Set DMA channel 8 Modify Register */

       r0=@dest;
       dm(C8)=r0;                 /* Set DMA channel 8 count Register */

       r0=0x4000;                 /* LBUF4->LPORT4, all other ports are disabled */
       dm(LAR)=r0;


       r0=L4EN | L4DEN | L4CLKD1;/*LBUF4 enable port and DMA, receive, word size, ½
```

```
                                clock, 4bit data path*/
      dm(LCTL1)=r0;             /*always write LCTLx after LAR*/


/* terminate and wait */
wait1:
jump wait1;
/*-------------------------------------------------------------------------*/
```

Listing 2.1 ADSP-21160 DMA-Driven Receive Example

```
/*-------------------------------------------------------------------------
Code to transmit with a ADSP-21160 link port(4) to a ADSP-21062. The DSP's I/O
processor manages the DMA transfer through the link port. The DMA operation transfers an
entire block of data located in DM data memory. In order to transmit to an ADSP-21160
using 8 bit data width the corresponding LxDPWID must be set.
-------------------------------------------------------------------------*/

#include    "def21160.h"

#define N       512        /* length of sample data should be a even number */

/*------------ DM data --------------*/
.section/dm seg_dmda;
.VAR source[N];            /*data to be transmitted to SHARC*/

/*---- PM interrupt vector code -----*/
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump start; nop; nop;

/*------ program memory code -------*/
.section/pm seg_pmco;

/* computes sample data to be transmitted in one DMA transfer block */
start: bit set MODE1 CBUFEN;
       b0=source; b1=b0; i0=source+N/4; i1=source+3*N/4;
       l0=N; l1=N;  m0=1; r0=N/4; r1=-N/4-1;
       /* triangle function period=N, amplitude=N/2 */
       lcntr=N/2, do compute until lce;
       r1=r1+1; dm(i0,m0)=r0;
compute:r0=r0-1, dm(i1,m0)=r1;    /* proper transmission can be easily verified
                                  using the debug window plot function */

setup: ustat1 = dm(SYSCON);       /* Clear Buffer Hang Disable     */
       bit clr ustat1 BHD;
       dm(SYSCON) = ustat1;

       r0=0;               /*clear link control registers*/
       dm(LAR)=r0;
       dm(LCTL1)=r0;

                           /* Set up DMA pointers */
       r0=source;
       dm(II8)=r0;         /* Set DMA channel 8 Index Register */

       r0=1;
       dm(IM8)=r0;         /* Set DMA channel 8 Modify Register */

       r0=@source;
       dm(C8)=r0;          /* Set DMA channel 8 count Register */

       r0=0x4000;
       dm(LAR)=r0;         /* LBUF4->LPORT4, all other ports are disabled */

       r0=0x12C00;         /* LBUF4 enable port and DMA, transmit, 32bit word size, 1/2 clock,
                           4bit data path */
       dm(LCTL1)=r0;       /* always write LCTLx after LAR */

/* terminate and wait */
wait1:
jump wait1;
/*-------------------------------------------------------------------------*/
```

Listing 2.2 ADSP-21160 DMA-Driven Transmit Example

```
/*-------------------------------------------------------------------------
Code to receive with a ADSP-21062 link port(5) from a ADSP-21160. The DSP's I/O
processor manages the DMA transfer through the link port. The DMA operation transfers and
stores an entire block of data in DM data memory.
```

```
-------------------------------------------------------------------------------*/

#include     "def21060.h"

#define N 512              /* length of receive data */

/*------------ DM data --------------*/
.section/dm seg_dmda;
.var dest[N];


/*---- PM interrupt vector code -----*/
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump start; nop; nop;


/*------ program memory code -------*/
.section/pm seg_pmco;

start: ustat1 = dm(SYSCON);
       bit clr ustat1 0x800;
       dm(SYSCON) = ustat1;/* Clear Buffer Hang Disable     */


setup: r0=0;               /* Clear Link Control Registers */
       dm(LCTL)=r0;
       dm(LAR)=r0;

                           /* Set up DMA pointers */
       r0=dest;
       dm(II4)=r0;         /* Set DMA channel 4 Index Register */

       r0=1;
       dm(IM4)=r0;         /* Set DMA channel 4 Modify Register */

       r0=@dest;
       dm(C4)=r0;          /* Set DMA channel 4 count Register */

       r0=0x0000000;       /* receive at clock frequency */
       dm(LCOM)=r0;

       r0=0x3ff7f;
       dm(LAR)=r0;         /* LBUF2 <==> LPORT5, all others disabled */

       r9=0x00000300;
       dm(LCTL)=r9; /* lbuf2 enable, receive, 32-bit word, 4-bit bus width, DMA */


/* terminate and wait */
wait1:
jump wait1;
/*-------------------------------------------------------------------------------*/
```

Listing 2.3 ADSP-21062 DMA-Driven Receive Example

```
/*-------------------------------------------------------------------------------
Code to transmit with a ADSP-21062 link port(5) to a ADSP-21160. The DSP's I/O
processor manages the DMA transfer through the link port. The DMA operation transfers an
entire block of data located in DM data memory.
-------------------------------------------------------------------------------*/

#include     "def21060.h"

#define N 512/* length of sample data should be a even number */

/*------------ DM data --------------*/
.section/dm seg_dmda;
```

```
.VAR source[N];

/*---- PM interrupt vector code -----*/
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump start; nop; nop;

/*------ program memory code -------*/
.section/pm seg_pmco;

/* computes sample data to be transmitted in one DMA transfer block */
start: b0=source; b1=b0; i0=source+N/4; i1=source+3*N/4;
       l0=N; l1=N;   m0=1; r0=N/4; r1=-N/4-1;   /* triangle function period=N,
                                                   amplitude=N/2 */
       lcntr=N/2, do compute until lce;
       r1=r1+1; dm(i0,m0)=r0;
compute:r0=r0-1, dm(i1,m0)=r1;    /* proper transmission can be easily verified
                                  using the debug window plot function */

setup: ustat1 = dm(SYSCON);       /* Clear Buffer Hang Disable     */
       bit clr ustat1 0x800;
       dm(SYSCON) = ustat1;

       r0=0x00000000;             /* Clear Link Control Register */
       dm(LCTL)=r0;
       dm(LAR)=r0;
                                  /* Set up DMA pointers */
       r0=source;                 /* Set DMA channel 4 Index Register */
       dm(II4)=r0;

       r0=1;
       dm(IM4)=r0;                /* Set DMA channel 4 Modify Register */

       r0=@source;                /* Set DMA channel 4 count Register */
       dm(C4)=r0;

       r0=0x0000000;              /* transfer on clock rate, disable 2-D DMA, disable link
                                  pull-down resistor*/
       dm(LCOM)=r0;

       r0=0x3ff7f;                /* LBUF2 <==> LPORT5, all others disabled */
       dm(LAR)=r0;

       r9=0x00000b00;             /* lbuf2 enable, transmit, 32-bit word, 4-bit bus width, DMA
                                  */
       dm(LCTL)=r9;

/* terminate and wait */
wait1: idle;
       jump wait1;
.endseg;
/*-------------------------------------------------------------------------*/
```

Listing 2.4 ADSP-21062 DMA-Driven Transmit Example

References:

ADSP- 21160M SHARC DSP Microcomputer Datasheet REV.0
ADSP- 21062L SHARC DSP Microcomputer Datasheet REV.C
ADSP- 21160 SHARC DSP Hardware Reference (First Edition)
ADSP- 2106x SHARC DSP User's Manual (Second Edition)