

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://ftp.analog.com>, WEB: www.analog.com/dsp

Copyright 1999, Analog Devices, Inc. All rights reserved. Analog Devices assumes no responsibility for customer product design or the use or application of customers' products or for any infringements of patents or rights of others which may result from Analog Devices assistance. All trademarks and logos are property of their respective holders. Information furnished by Analog Devices Applications and Development Tools Engineers is believed to be accurate and reliable, however no responsibility is assumed by Analog Devices regarding the technical accuracy of the content provided in all Analog Devices' Engineer-to-Engineer Notes.

BDMA Usage on 100 pin ADSP-218x DSPs configured for IDMA use.

The 100 pin versions of the ADSP-218x processors can be configured to either fully support the IDMA interface of a memory interface. When these components are configured to utilize the IDMA interface, the memory interface is only partially functional. This allows only limited usage for the memory interfaces. The BDMA interface can still be used in this mode, albeit in a somewhat limited way.

This document will show how to set up BDMA in IDMA mode with the following features.

- 20 Bits of BMDA address space.
- Approx. 8 cycles overhead per DM or PM word transferred in blocks of up to 256 DM words or 170 PM words.
- Supports both reading and writing of BM.

This document will show example source code demonstrating the use of BDMA on the ADSP-2185 in IDMA mode. This code is shown at the end of this document.

In this mode of operation the processor only normally provides 1 address bit. This does not make it impossible to use the BDMA functions. Up to 20 bits of address can be generated for use with the BDMA function. The approach used in this document creates an address for BDMA in the following way:

Pin name	fl2	fl1	fl0	pf7:0	d23:16	a0
Number of bits	3			8	8	1

This creates an address of up to 20 bits which can address 1 Megabyte. The different sections of the address need to be updated manually, but a trick in setting up the transfers allows the d23:16 section of the address to automatically be updated. By setting the BEAD register to all ones for the bits which are not used causes an automatic carry into the BMPAGE register. This allows the one, two, or three transfers needed to transfer an internal memory word to be done as a single BDMA transfer.

The example code is structured to set up a block transfer of up to 256 words for DM and 170 words for PM. The blocks are transferred at a rate of one DM or PM word per BDMA interrupt. The interrupt routine automatically sets up the next word for transfer. The block size is limited due to the fact that the code shown does not update either the pf7:0 pins or the fl2:0 pins. The problem is basically that a single DM or PM word which would cross a 512 byte boundary set by the number of bits automatically updated by the BDMA

interface would require the update the other address fields. There is a tradeoff between supporting larger blocks and overhead. This example shows a bias toward limiting overhead at a cost of additional limitations.

The code has three parts, the main program, a block transfer setup subroutine, and the BDMA interrupt handler. The main program creates a small DM buffer full of data and then uses the block transfer subroutine to copy it out to BM. It then reads it back from the BM to a small buffer in DM. It then copies the first 100 words of PM (itself) to BM. The main program polls the number of remaining words in each transfer to determine when to continue. The main program would be available do other tasks at these waiting points.

The block transfer subroutine "hokeyb" is used to set up the BMDA control registers and the address bits to cause the requested transfer.

The BDMA interrupt routine does the necessary adjustments to the control words, decrements the block count, and triggers the next transfer. It could also call a user BDMA subroutine when the block is finished.

```
{ -----
```

```
Show an example of BDMA on IDMA configured part
```

```
uses address bus made up of:  
    fl0, pf7:0, d23:16, a0
```

```
transfers multiple dm or pm words to/from bm with  
one word transferred per BDMA interrupt. This version  
does not automatically update pf7:0 or fl0.
```

```
Date:    May 16, 1997
```

```
----- }
```

```
.module/ram/abs=0 hokey;
```

```
.var/dm/ram hbwords;
```

```
jump START; nop; nop; nop;    {0x0000 reset}  
rti; nop; nop; nop;          {0x0004 irq2}  
rti; nop; nop; nop;          {0x0008 irq11}  
rti; nop; nop; nop;          {0x000c irq10}  
rti; nop; nop; nop;          {0x0010 sp0tx}  
rti; nop; nop; nop;          {0x0014 sp0rx}  
rti; nop; nop; nop;          {0x0018 irqe}  
jump BDMAINT; nop; nop; nop; {0x001c bmda}  
rti; nop; nop; nop;          {0x0020 spltx/nirq1}
```

```

    rti; nop; nop; nop;          {0x0024 splrx/nirq0}
    rti; nop; nop; nop;          {0x0028 timer}
    rti; nop; nop; nop;          {0x002c pwd}

#define ENABDMA 0x8
#define CLRBDMA 0x8

START:
{ Set wait states to 0 for PM and 0 for DM }

    icntl=0x03;

    ax0=CLRBDMA;      {clear bdma interrupt if left over from booting}
    ifc=ax0;

    ax0=0x1bff; {1 bdma waitstate, cms normal, pfs=output}
    dm(0x3fe6)=ax0;

    imask=ENABDMA; {unmask bdma interrupt}

{main code begins here}
    i0=0x3000;
    m0=1;
    l0=0;

    my0=0x4321;      {initialize dm with a pattern}
    cntr=24;
    do loopy until ce;
    mx0=cntr;
    mr=mx0*my0(ss);
loopy:    dm(i0,m0)=mr0;

    { set up first transfer (write of dm to bm)}
    ax0=0x3000;      {set biad}
    dm(0x3fe1)=ax0;

    ax0=24;          {number of words}
    dm(hbwords)=ax0;

    ay0=0x05;        {direction and mode}
    si=0x0;          {address lsb's}
    ax0=0x0000;      {address msb's}

    call hokeyb;     {do the transfer}

```

```

wait0:
    idle;                {wait until transfer complete}
    ar=dm(hbwords);
    ar=pass ar;
    if ne jump wait0;

{ set up second transfer read the buffer back to internal dm}
    ax0=0x3100;          {set biad}
    dm(0x3fe1)=ax0;

    ax0=24;              {number of words}
    dm(hbwords)=ax0;

    ay0=0x01;           {direction and mode}
    si=0x0;              {address lsb's}
    ax0=0x0000;         {address msb's}

    call hokeyb;

wait1:
    idle;                {wait until transfer complete}
    ar=dm(hbwords);
    ar=pass ar;
    if ne jump wait1;

{set up third transfer (write pm to bm)}
    ax0=0x0;             {set biad}
    dm(0x3fe1)=ax0;

    ax0=100;            {number of words}
    dm(hbwords)=ax0;

    ay0=0x07;           {direction and mode}
    si=512;             {address lsb's}
    ax0=0x0000;         {address msb's}

    call hokeyb;

wait2:
    idle;                {wait until transfer complete}
    ar=dm(hbwords);
    ar=pass ar;
    if ne jump wait2;

```

END: jump END;

```
{ Hokeyb is a routine which sets up bdma for the "hokey"  
  bdma feature on the 100 pin 218x components. Its calling  
  convention is as shown:
```

```
    biad: set to internal address  
    dm(hbwords): number of words to transfer  
  
    ay0: bdir,btype for transfer (3bits, other bits 0)  
    si: 16 lsb's of starting address  
    ax0: remaining address bits (right justified)
```

Restrictions:

- specified transfer must not cross a 512 byte block.
- this example code does not set fl,fl2
- this example code uses the secondary registers during the service of the bdma interrupt.

```
}  
hokeyb:  
  { build bdma control word }  
  sr = lshift si by 7 (lo); {line up bits for bmpage}  
  
  ar = tstbit 7 of sr0;      {check a0 bit}  
  ar = 0xffff;              {set bead to all 1's}  
  if eq ar = clrbit 0 of ar; {set bit 0 to be lsb}  
  dm(0x3fe2)=ar;            {write bead register}  
  ar = clrbit 7 of sr0;     {lose the a0 bit}  
  ar = ar OR ay0;           {add in control bits}  
  dm(0x3fe3)=ar;           {write bdma control reg}  
  
  sr = lshift si by -9 (lo); {get rid of 9 used bits}  
  si = ax0;  
  sr = sr or lshift si by 7 (lo); {full remaining address}  
  
  dm(0x3fe5)=sr0;          {put out the 8 bits on the pfs}  
  ar = tstbit 8 of sr0;    {put out one bit on the fl0 pin}  
  if ne set fl0;  
  if eq reset fl0;  
  ar=1;  
  dm(0x3fe4)=ar;          {set count, start bdma}  
  rts;
```

BDMAINT:

```
  { we got or sent one word, now set up the next one }  
  ena sec_reg;           {use alternate registers}
```

```

ay0=dm(0x3fe2);           {reset the bits in bead so carry}
ar=0x3ffe;                {updates bmpage}
ar=ar OR ay0;
dm(0x3fe2)=ar;           {set bead}

ar=dm(hbwords);          {decrement word count}
ar=ar-1;
dm(hbwords)=ar;
if eq rti;                {none left, either return}
                           {without starting next fetch}
                           {or call user routine...}
{if eq call userbdma;}    {call user bdma routine }
                           {which ends with an rti }

ar=1;                     {count not zero, get next}
dm(0x3fe4)=ar;           {word}
rti;

.endmod;

```