BITBLT Examples: NS32CG16 and NS32FX16

National Semiconductor
Application Note 634
July 1990

1.0 INTRODUCTION

The NS32CG16 has several instructions that automate the process of performing a BIT aligned BLock Transfer (BITBLT). In this application note, the parameter setup for these instructions will be discussed. The NS32FX16 is pin compatible with the NS32CG16 but can run at frequencies up to 25 MHz.

For additional information on Series 32000® programming, please see the Series 32000 Programmer's Reference Manual. For additional information on NS32CG16-specific instructions, please see the NS32CG16 Programmer's Reference Manual Supplement.

2.0 DESCRIPTION

A BITBLT is a very common technique used in graphics routines to copy a rectangular area of one image to another. Various effects may be produced by different logical operations being performed between the source and destination data, and the NS32CG16 implements many of these operations.

Conventions

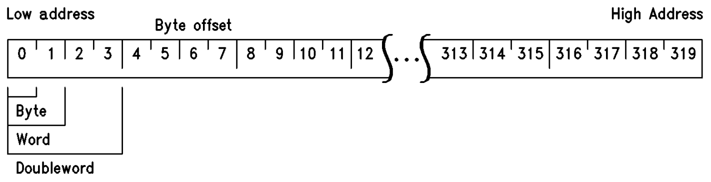
This document assumes the displayed image is in a specific format in memory. The information below explains the terminology and directional assumptions, as well as byte and bit ordering conventions in Series 32000.

Figure 1 presents one scan line of a standard 8½ inch by 11 inch page on a 300 Dot Per Inch (DPI) laser printer, in the portrait orientation (8½ inches wide, 11 inches high). There are 3300 such scan lines on each page. The start of the second scan line on the page would be at byte offset 320 decimal, 140 hex. Since the first scan line is at the top of the page, successive scan lines proceed down the page.

All Series 32000 processors have 32-bit internal data paths, with a "natural" size of 32 bits, or 4 bytes. The least significant byte is stored at the lowest address. Referring to the previous example, writing a byte of hex A5 to address zero would result in address zero containing hex A5. Writing a word of hex A55A to address zero would result in address zero containing hex 5A, and address 1 containing hex A5. Writing a doubleword of hex FFA55A00 to address zero would result in address zero containing hex 00, address 1 containing hex 5A, address 2 containing hex A5, and 3 containing hex FF.

Series 32000 does **not** have an alignment restriction in that data of byte, word or doubleword size need not reside at an even memory address. The Bus Interface Unit, internal to all Series 32000 processors, requests multiple bus transfers as required, aligning the data automatically.

The bit offset is equally consistent. Bit ordering is always least significant to most significant bit. In *Figure 1,* bit zero of byte zero would be the first pixel imaged on the page. Bit one would be the next pixel, bit two the next, and so on. Bit 2549 would be the last pixel imaged on the page in the horizontal direction, since 8½ inches × 300 DPI yields a width of 2,550 dots, or pixels. Bit 2549 is contained within byte 318, at bit position 5. Bit Addressing and Byte Addressing with a byte address and a bit offset are available in Series 32000. *Figure 2* is an expansion of the first three bytes of a scan line, showing the bit addressing, as it would appear on the page printer or graphics screen.



TL/EE/10493–1

FIGURE 1

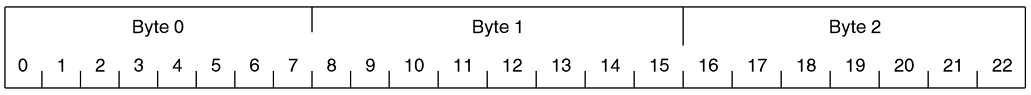


FIGURE 2

To clarify these conventions further, the following example illustrates how a line 1 dot high and 10 dots wide is drawn. This line appears on scan line one, starting at the ninth pixel or bit position 8. This will result in hex FF in address one, and hex 03 in address two. This is referred to as the horizontal direction.

The width of the memory for an image is referred to as the image warp (sometimes referred to as raster, or pitch). The warp of the page printer image in the previous example is 320 decimal (140 hex) bytes, or 2560 bits. Note that the image width is actually 2550 on this sample page printer at 300 DPI, since 8½ inches × 300 DPI yields a width of 2,550 dots. The width is rounded up to 2560 bits (320 bytes) to make memory addressing simpler in a typical hardware design.

When the warp is known, vertical lines can be drawn. A vertical line 10 dots high and 1 dot wide starting at the first line, ninth pixel, with a warp of 320 (140 hex) and a base address of 0 would result in addresses 1 (1 hex), 321 (141 hex), 641 (281 hex) . . . 2881 (B41 hex) each containing 01 hex.

To summarize, for portrait applications, the "top left" pixel is bit 0. The "top right" pixel is bit 2,549. The "bottom left" pixel is bit 8,445,440. The "bottom right" pixel is bit 8,447,989. To calculate x,y bit positions on the page, the formula:

$$\text{Bit offset} = (y \times 2560) + x$$

may be used, where y is the scan line number ranging from 0 to 3299, for the sample 8½ by 11 inch page, and x is the pixel displacement across the page from the left hand edge.

More information on Series 32000 conventions can be found in the Series 32000 Programmer's Reference Manual. See Section 3.5 in the Instruction Set Reference Manual for a complete discussion of bit instructions and bit addressing.

All shifts in this document are in the increasing bit/byte direction. This is a Series 32000 Left Shift. However, visually on a page, this is a left-to-right shift. Zeros are inserted in the least significant bits.

All rotates in this document are in the increasing bit/byte direction. This is a Series 32000 Left Rotate. However, visually on a page, this is a left-to-right shift. Data from the most significant bit position will be transferred to the least significant bit position.

Numbers preceeded by 0x are in hexadecimal. 0x1AC would be decimal 428.

Sample BITBLT

In this example, we will set up the parameters for a BITBLT operation from (28, 36) to (123, 76) (see *Figure 3*), with an area of 36 x 36.

The first problem is how to describe the area to be copied. One of the best ways to do this is to supply an (x, y) coordinate pair for the upper left corner of the source and destination, and a width and height for the copy operation (see *Figure 3*). From this information, each of the parameters of the BITBLT operation can be calculated. As an example, take the 36 x 36 area starting at (28, 36), and move it to (123, 76).

The first step is to translate the image coordinates into physical addresses. This can be done by multiplying the y coordinate by the image warp, or length of each scan line. In our example, the warp is 2560 pixels. Since the source image starts at y=36, so the actual source image begins at bit offset 92188 (28 + 36 × 2560) from the beginning of the image space. The destination starts at y=76, so the actual destination image will begin at bit offset 194683 (123 + 76 × 2560) from the beginning of the image space. Adding this to the physical starting address of the page gives the actual starting address, as follows.

The base of the source address can be determined by taking the source bit offset, and dividing by 8 to obtain the offset in bytes. This value can then be added to the physical memory image address, which in our example is 0x8000. The source base address is therefore 0x8000 + (28 + 36 × 2560)/8, or 0xAD02.

The base address of the destination can be determined much in the same way as the base address of the source, substituting the destination bit offset computation. The destination base address is therefore 0x8000 + (28 + 36 × 2560)/8, or 0xDF0E. This value may be altered by the shift computation, as will be seen.

The BITBLT functions in the NS32CG16 operate on 16 bits of data (a word) at once, to optimize speed. Since the source is not aligned on bit 0 of an address, we must mask the unwanted data from the source in order to extract only the bits we are interested in. Each left-hand edge of the source is on bit 12. This can be determined by taking the x value of the source (28), and logically ANDing it with 15. Since we DO NOT want bits 0–11 of the source, our first mask will be 1111 0000 0000 0000 (0xf000).

The second mask can be determined by taking the source x value, adding the width, subtracting one, and logically ANDing with 15. The end of the source, therefore, is on bit 15. We require all of this word, so the second (or ending) mask will be 1111 1111 1111 1111 (0xffff).

The masks are only required on the first and last words of a line. The reason for this is that all words in the middle would have a mask of 1111 1111 1111 1111 (0xffff), since all bits are required. Because of this fact, there is no need to compute an "inner" mask value.

We know that the image is 36 bits wide, and 36 lines high. From this information we can determine that the source is three words wide (a word is 16 bits, and the source cannot be contained in two words in this example).

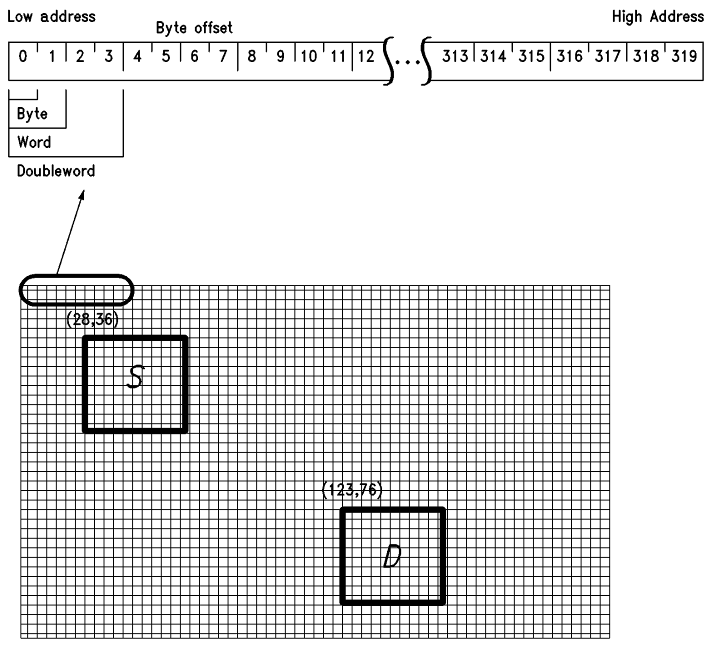
The next item we need to determine is the shift amount. This can be found by subtracting the source x value from the destination x value. The best way to do this is to logically AND the source and destination x values with 15, to determine the bit offset within the word. In this example, the source bit offset is 12, and the destination bit offset is 11. Since the destination bit offset is less than the source bit offset, the resultant shift value will be −1. In the NS32CG16, only positive shift values are permitted, so we "back up" one word on the destination, and add 16 to the shift amount, giving a shift of 15 bits. This will make the destination base address 0xDF0C.

The adjusted source warp, required by the BITBLT instructions, would be calculated as (Source warp in bytes − 2 × (width in words − 1)). Our warp for both the source and destination is the same, 2560 bits or 320 bytes. The source warp would be (320 − 2 × (3−1)) = 316. Since the source and destination images are in the same image space, the source and destination warps are the same. The adjusted source and destination warps are both 316 for this example.

Once all the calculations are complete, we can convert to physical parameters to the BITBLT instruction, and perform the BITBLT.

If we assume that the memory image starts at address 0x8000, we can now pass all the parameters, as follows:

Base address of source	=	0xAD02
Base address of dest.	=	0xDF0C
Shift value in bits	=	15
Height in lines	=	36
First mask	=	0xF000
Second mask	=	0xFFFF
Adjusted source warp	=	316
Adjusted dest. warp	=	316
Width in words	=	3



TL/EE/10493–2

FIGURE 3

3.0 IMPLEMENTATION

This example may be turned into a general-case program. A sample of this follows, written in the C language.

```
#define IMAGEADDR 0x8000        /* Image starting address */
#define SOURCEWARP 2560         /* Source warp, in bits */
#define DESTWARP 2560           /* Destination warp, in bits */


unsigned int MASK1[ ] = {        /* First mask table */
        0xffff,0xfffe,0xfffc,0xfff8,
        0xfff0,0xffe0,0xffc0,0xff80,
        0xff00,0xfe00,0xfc00,0xf800,
        0xf000,0xe000,0xc000,0x8000,
        0x0000 };
unsigned int MASK2[ ] = {        /* Second mask table */
        0x0001,0x0003,0x0007,0x000f,
        0x001f,0x003f,0x007f,0x00ff,
        0x01ff,0x03ff,0x07ff,0x0fff,
        0x1fff,0x3fff,0x7fff,0xffff,
        0x0000 };
/*
        Preprocessor for BITBLT function
        Inputs:
                sx - source x value (column)
                sy - source y value (row)
                dx - destination x value
                dy - destination y value
                wx - width of image to transfer (in bits)
                wy - height of image to transfer (in lines)
        Outputs:
                Calculates all required values for NS32CG16
                internal BITBLT functions
*/
preproc(sx,sy,dx,dy,wx,wy)
int sx,sy,dx,dy,wx,wy;
{
        int srcadd,dstadd,width,height;
        unsigned int mask1,mask2;
        int shift,adswarp,addwarp;
        /* calculate the word-aligned source address */
        srcadd = (((sy * SOURCEWARP) + sx)/16) * 2 + IMAGEADDR;
        /* calculate the word-aligned destination address */
        dstadd = (((dy * DESTWARP) + dx)/16) * 2 + IMAGEADDR;
        /* Calculate the first and last masks */
        mask1 = MASK1[sx & 15];
        mask2 = MASK2[(sx + wx - 1) & 15];
        /* Calculate the width in words. It is not permissible
        for the width in bits or words to be zero.
        */
        width = ((wx+15)/16) ;
        if (width == 1)       /* If the width is 1, combine masks */
                mask1 |= mask2;
        shift = (dx & 15) - (sx & 15);  /* Calculate the shift */
        if (shift < 0) {      /* If the shift is negative */
                shift += 16;  /* Add 16 to the shift amount */
                dstadd -=  2; /* and decrement the dest. addr */
        }
        height = wy;     /* Height computation is simply the passed height */
        /* Calculate the adjusted source and destination warps */
        adswarp = SOURCEWARP/8 - 2 * (width - 1);
        addwarp = DESTWARP/8 - 2 * (width - 1);
    BBfunc(srcadd,dstadd,shift,wy,mask1,mask2,adswarp,addwarp,width);
}
```

This example program will generate all required values for the BBOR, BBXOR, BBAND, BBSTOD and BBFOR instructions. It assumes non-overlapping source and destination images, and a top-to-bottom, left-to-right BITBLT operation. The BBOR, BBXOR, BBAND and BBSTOD instructions are capable of full 4-direction BITBLTs, required for overlapping images, and the BBFOR is capable of a 2-direction BITBLT.

To implement a bottom-to-top BITBLT with any of the BITBLT functions, simply adjust the source and destination starting address to the last line of the BITBLT, and negate the source and destination warps.

To implement a right-to-left BITBLT operation with the BBOR, BBXOR, BBAND or BBSTOD instructions, use the "-d" option of the instruction, and adjust the source and destination warps as follows:

```
        /* Calculate the adjusted source and destination warps */
        adswarp = SOURCEWARP/8 + 2 * (width - 1);
        addwarp = DESTWARP/8 + 2 * (width - 1);
```

3.1 Implementation for Non-Impact Printers

Non-impact printers have a slightly different requirement for BITBLT functions. Generally the function required is a left-to-right, top-to-bottom, logical OR BITBLT. Source and destination are not generally in the same memory space, and the source and destination warps are not equal. Since the source is usually a character bit-image, and the character image is typically left margin aligned, no first mask is required. This is why the BBFOR (BitBlt Fast OR) instruction was created. Based on this information, we can create an optimized version of the preprocessor for non-impact printers.

```
#define IMAGEADDR 0x8000      /* Image starting address */
#define SOURCEWARP 2560       /* Source warp, in bits */
#define DESTWARP 2560         /* Destination warp, in bits */
unsigned int MASK2[] = {      /* Second mask table */
        0x0001,0x0003,0x0007,0x000f,
        0x001f,0x003f,0x007f,0x00ff,
        0x01ff,0x03ff,0x07ff,0x0fff,
        0x1fff,0x3fff,0x7fff,0xffff,
        0x0000 };
/*
        Preprocessor for Non-Impact Printer BITBLT function
        Inputs:
                saddr - source character byte address
                swarp - warp of source character (in bytes)
                swidth - width of character (in bits)
                sheight - height of character (in lines)
                dx - destination x value
                dy - destination y value
        Outputs:
                Calculates all required values for NS32CG16
                internal BITBLT functions
*/
nipproc(saddr,swarp,swidth,sheight,dx,dy)
unsigned char *saddr;
int swidth,sheight,dx,dy;
{
        int srcadd,dstadd,width;
        unsigned int mask1,mask2;
        int shift,adswarp,addwarp;
        srcadd = saddr;
        dstadd = (((dy * DESTWARP) + dx)/16) * 2 + IMAGEADDR;
        /* Calculate the first and last masks */
        mask1 = 0xffff;         /* First mask is always 1's */
        mask2 = MASK2[swidth & 15];
        /* Calculate the width in words. It is not permissible
           for the width in bits or words to be zero.
        /*
        width = ((swidth+15)/16) ;
        if (width == 1)         /* If the width is 1, combine masks */
                mask1 &= mask2;
        shift = (dx & 15);      /* Calculate the shift */
        /* Calculate the adjusted source and destination warps */
        adswarp = swarp - 2 * (width - 1);
        addwarp = DESTWARP/8 - 2 * (width - 1);
    BBfunc(srcadd,dstadd,shift,sheight,mask1,mask2,adswarp,addwarp,width);
}
```

Converting this code to assembly language, we can show a very high performance preprocessor, as follows.

```
# Preprocessor for non-impact BITBLT function
# Inputs:
#       r0 - source character byte address
#       r1 - destination y value
#       r2 - destination x value
#       r3 - source height (in lines)
#       r4 - source width (in bits)
#       r5 - source warp in bytes
#
        .set  DESTWARP,2560
        .set  IMAGADDR,0x8000
nipproc:
        indexd  r1,$(DESTWARP-1),r2     # obtain bit index
        lshd    $3,r1                   # divide by 8 and
        andd    $0xfffffffe,r1          # mask to get word offset
        addr    IMAGADDR(r1),r1         # add image address
        addr    0xf,r7                  # get mask value in r7
        andd    r7,r2                   # mask dest. x to get shift
        movd    r5,r6                   # move source warp to r6
        movd    r4,r5                   # move width to temp
        andd    r7,r5                   # mask to low order bits
        movzwd  MASK2[r5:w],r5          # get second mask from table
        addd    r7,r4                   # round width to word multiple
        lshd    $4,r4                   # r4 is now width in words
        addr    -1(r4),r7               # get width-1 in r7
        subd    r7,r6                   # adjust source warp
        subd    r7,r6                   # to swarp - 2*(width-1)
        addd    r7,r7                   # double r7
        negd    r7,r7                   # negate it (-2*(width-1))
        addd    $(DESTWARP/8),r7        # add destination warp
        movw    r4,tos                  # place width on stack
        cmpqw   $1,r4                   # is width 1?
        addr    0xffff,r4               # assume not, get 1st mask
        beq     width1                  # it is a width of 1, branch

        cmpb    r2,$0                   # set L flag for optimization
        bbfor                           # do BBFOR
        cmpqw   0,tos                   # unstack, and
        ret     $0                      # return

width1: andd    r5,r4                   # combine masks
        cmpb    r2,$0                   # set L flag for optimization
        bbfor                           # do BBFOR
        cmpqw   0,tos                   # unstack, and
        ret     $0                      # return
```

This document shows average BITBLT performance of the NS32CG16 instructions, on a no wait system.

BITBLT Performance

The performance of the BITBLT (Bit Aligned Block Transfer) routines can be measured by imaging 3,500 characters with each of the BITBLT functions. The time includes pre-processor overhead for each BITBLT operations, and assumes a 32-bit wide by 54 line high character. The pre-processor overhead is 287 clock cycles. Both source and destination warp values are included, and a separate time for the BITWT instruction is shown, assuming no source warp. The average time is shown, including a shift of up to 8 bits.

The EXTBLT instructions shows a width of 3, since three words of destination are required to store two words (32 bits) of shifted source data.

Time to Image 3,500 Characters

Function	10 MHz	15 MHz
BITWT (No src Warp)	1.15 sec	0.77 sec
BITWT (src Warp)	1.23 sec	0.82 sec
BBFOR	1.52 sec	1.01 sec
BBOR	2.14 sec	1.42 sec
BBXOR	2.14 sec	1.42 sec
BBAND	2.21 sec	1.48 sec
BBSTOD	3.34 sec	2.22 sec
EXTBLT (Preread)	1.21 sec	0.80 sec
EXTBLT (No Preread)	1.09 sec	0.73 sec

4.0 SUMMARY

While the BITBLT function parameters may seem complex, a simple relationship exists between the desired operation values and the BITBLT functions.

BITBLT Examples: NS32CG16 and NS32FX16

AN-634

Lit. # 100634

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

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

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

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

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

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