Helios Support for 2-D Block Moves

Perihelion Software Technical Report No. 9

Andy Evans

June 1989

Perihelion Software Limited
The Maltings
Charlton Road
Shepton Mallet
Somerset
BA4 5QE
England
Telephone +44 749 4203
Fax. +44 749 4977

Contents

1 bytblt()

This note describes the Helios routine, bytblt(), which is used to copy a two dimensional array of bytes from one area of memory to another. Note that this function is an interface to the 2-dimensional block move instructions which are implemented on the T800 processor. An emulation of the simpler of these instructions has been provided to allow this function to be used with a T414.

The syntax of this routine is:

```
void bytblt(source,dest,sourceix,destix,sstride,dstride,length,width,op)
```

where the arguments have the following types and meanings:

Argument	Type	Meaning
source	byte*	pointer to source array
dest	byte*	pointer to destination array
sourceix	word	index into source array
destix	word	index into destination array
sstride	word	stride of source array
dstride	word	stride of destination array
length	word	number of rows to copy
width	word	width of each row
op	word	operation code

1.1 Operation codes

The final argument to bytblt() is an operation code which can be one of three integer numbers: 0, 1 or 2. Each of these opcodes maps onto one of the move instructions which are implemented by the transputer. If an illegal opcode is specified then this routine returns without doing anything. If you are using a T414, you should always specify opcode 0.

If an opcode of 0 is specified all the bytes within the specified region are copied. A value of 1 causes all non-zero bytes to be copied; this can be used to overlay an image onto another image. Similarly, a value of 2 can be specified so that all zero bytes are copied; this can be used to mask-out a shape from an image.

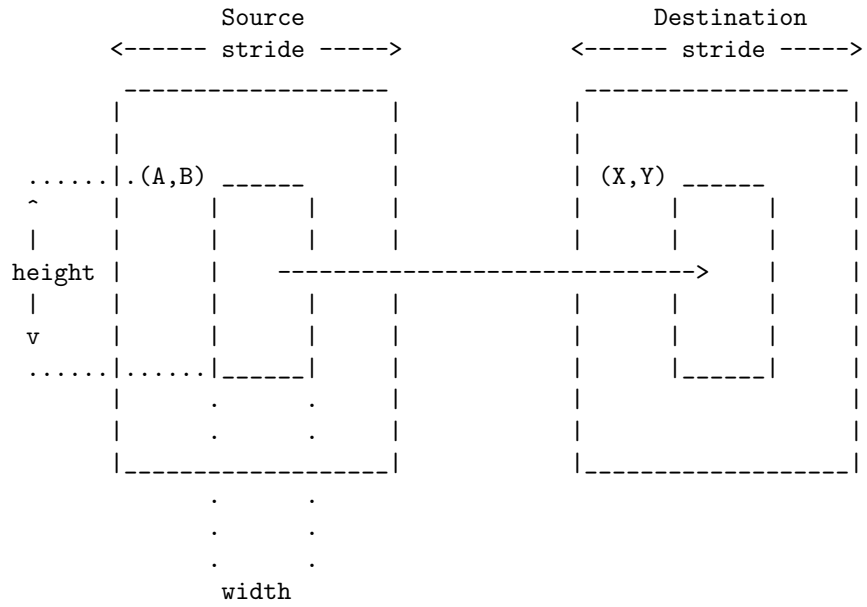
1.2 Source/Destination Index

The source and destination indexes specify the offset of the 2-dimensional array of bytes within the source or destination block of memory. In the diagram shown below, this is represented by the points (x,y) or (a,b). The value which is passed as an argument to bytblt() is calculated by adding the X and Y coordinates of this point and then multiplying the result by the stride; that is:

```
sourceix = A + B * source_stride
destix   = X + Y * destination_stride
```

1.3 Graphical Representation

The following diagram will help to clarify the meaning of the arguments which bytblt() expects.



1.4 Useful Tricks

The previous illustration showed that bytblt() can be used to copy a block of data from one area of memory to another. By adjusting some of the arguments it is possible to perform various tricks. To understand how these tricks work, consider the algorithm which the transputer's block move instructions use.

```
source pointer = pointer to first byte to be copied
destination pointer = pointer to the destination

for i = 1 to length
{
   copy width bytes from source pointer to destination pointer
   add source stride to source pointer
   add destination stride to destination pointer
}
```

The following sections show two examples of useful operations: repeated copy of a single byte into a block of memory, and repeated copy of an array of bytes.

1.4.1 Repeated Copy of Single Byte

The following illustration shows a single byte which has been copied into a contiguous block of memory. To achieve this using bytblt(), specify a width of 1, a source stride of 0, and a destination stride of 1. The length is then interpreted as the number of times that the byte will be copied.

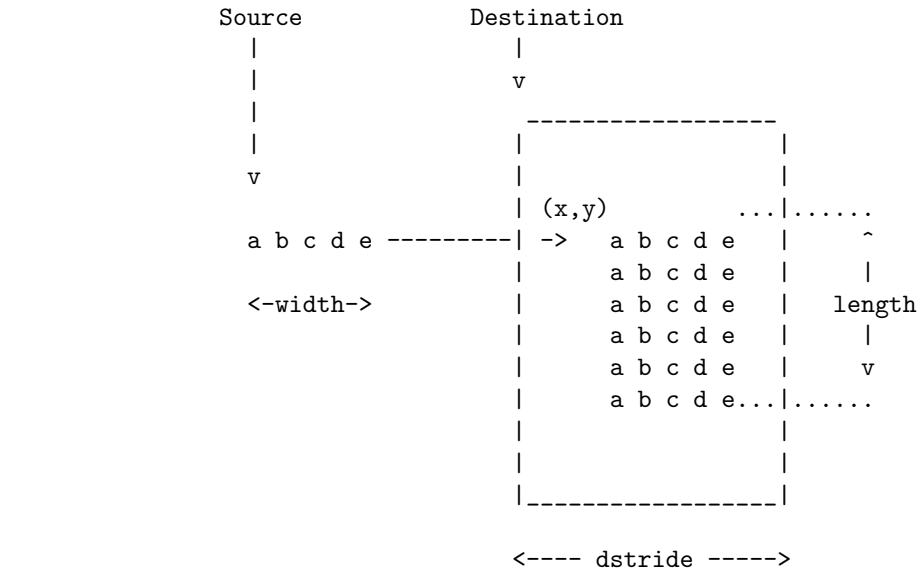
```
Source     Destination

           <----- length ------>
|a|        |a|a|a|a|a|a| ... |a|
```

If *source* and *destination* are pointers to the byte and the destination array respectively, then this operation is achieved by a statement of the form:

```
bytblt(source,dest,0,0,0,1,length,1,0)
```

1.4.2 Repeated Copy of an Array of Bytes

The following illustration shows an array of bytes, *abcde*, which have been copied into an array within the destination block. To achieve this using bytblt(), specify a source stride of 0, and use the length field to state the number of times which the source array is to be copied.



This operation is achieved by a statement of the form:

```
bytblt(source,dest,0,destix,0,dstride,length,width,opcode)
```

2 Use of bytblt(); An Example

The following listing shows a program that was written for use with the Atari Transputer Workstation. It draws 20,000 randomly-coloured boxes on the screen at randomly selected positions. Note the two calls of bytblt() in the function, clear_screen(). The first repeatedly copies a single byte into an array, and the second repeatedly copies this array into the video RAM; this causes the screen to be cleared.

```
/* Program for use with Atari Transputer Workstation. */
/* TJK, February 1988. */
/* Copyright (c) 1988, Perihelion Software Ltd. ALL rights reserved. */

#include <helios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <attrib.h>
#include <nonansi.h>

#define STRIDE 2048
```

```
char *array = (char *)(MinInt * 0x70);

/* Simple window structure */
struct window {
    int x,y,width,height;
} window;

void boxes(void);

extern void Delay(long);
extern void exit(int);
extern void bytblt(char *,char *,WORD,WORD,WORD,WORD,WORD,WORD,WORD);

#define MAP0 0x00000000L
#define MAP1 0x00200000L
#define MAP2 0x00400000L
#define MAP3 0x00600000L

char *screen_base;
char *savedscreen;

WORD *vd453wraddraddr = (WORD *)0x42d00000;
WORD *vd453wrcoladdr  = (WORD *)0x42d02000;

void set_colour(WORD red,WORD green,WORD blue)
{
    *vd453wrcolsddr = red;
    *vd453wrcoladdr = green;
    *vd453wrcoladdr = blue;
}

void setup_clut(UBYTE *buffer)
/* set up the colour lookup table */
{
    UBYTE   r,g,b;
    WORD    i;

    *vd453wraddraddr = 0L;

    for(i=0; i<256; i++)
    {
        r = *buffer++;
        g = *buffer++;
        b = *buffer++;
        set_colour(r,g,b);
    }
}
```

```
/* Set the colour lookup table for specific index entry */
void set_clu(WORD index,WORD r,WORD g,WORD b)
{
    *vd453wraddraddr = index;
    set_colour(r,g,b);
}

/* Reset the colour lookup table to something sensible */
void reset_clut()
{
    int i;
    int max      = 255;
    int half_max = 255 / 2;

    set_clu(0,max,max,max);            /* white */
    set_ctu(1,0L,0L,0L);               /* black */
    set_clu(2,max,0L,0L);              /* red */
    set_clu(3,0L,max,0L;               /* green */
    set_clu(4,0L,0L,max);              /* blue */
    set_clu(5,0L,max,max);             /* cyan */
    set_clu(6,max,max,0L);             /* yellow */
    set_clu(7,max,0L,max);             /* magenta */
    set_clu(8,half_max,max,max);       /* light cyan */
    set_clu(9,0L,0L,0L);               /* black */
    set_clu(10,max,half_max,half_max); /* light red */
    set_clu(11,half_max,max,half_max); /* light green */
    set_clu(12,half_max,half_max,max); /* light blue */
    set_clu(13,half_max,max,max);      /* light cyan */
    set_clu(14,max,max,half_max);      /* light yellow */
    set_clu(15,max,half_max,max);      /* light magenta */

    set_clu(255,0,0,0);
    set_clu(254,max,max,max);
    set_clu(253,0,max,max);
    set_clu(252,max,0,max);
    set_clu(251,max,max,0);
    set_clu(250,max,0,0);
    set_clu(249,0,0,max);
    set_clu(248,0,msx,0);
    set_clu(247,0,0,0);
    set_clu(246,max,max,max);
    set_clu(245,0,half_max,half_max);
    set_clu(244,half_max,0,half_max);
    set_clu(243,half_max,half_max,0);
    set_clu(242,half_max,0,0);
    set_clu(241,0,0,half_max);
    set_clu(240,0,half_max,0);

    for (i=16;i<240;i++)
```

```
    {
        set_clu(i,half_max + i,half_max + 1,half_max + 1);
    }
}

void clear_screen()
{
    char ptr = 0;

    bytblt(&ptr,array,0,0,0,1,window.width,1,0);

    bytblt(array,screen_base,0,0,0,STRIDE,
           window.height,window.width,0);
}

void save_screen()
{
    savedscreen = Malloc(window.width * window.height);

    if (savedscreen==0)
    {
        printf("Not enough memory\n");
        exit(1);
    }

    bytblt(screen_base,savedscreen,0,0,STRIDE,window.width,
           window.height,window.width,0);
}

void restore_screen()
{
    bytblt(savedscreen,screen_base,0,0,wirdow.width,STRIDE,
           window.height,window.width,0);
    Free(savedscreen);
}

int main()
{
    #define INITIATOR            0x9B
    #define WINDOW_SIZE_REPORT   0x76
    #define DEFAULT_SIZE         100
    #define WRITE_TIMEOUT        (1 * OneSec)

    char buffer[100],buf[2];
    int i;
    Attributes attr;

    #define get_char() (Read(Heliosno(stdin),buf,1,WRITE_TIMEOUT),*buf)
```

```
setvbuf(stdin,NULL,_IONBF,0);

if (GetAttributes(Heliosno(stdout),&attr) < 0)
{
    printf("Unable to get window attributes.\n");
    exit(1);
}

AddAttribute(&attr,ConsoteRawOutput);

if (SetAttributes(Heliosno(stdout),&attr) < 0)
{
    printf("Unable to set window attributes.\n");
    exit(1);
}

sprintf(buffer,"%c%c",INITIATOR,WINDOW_SIZE_REPORT);

/* new protocol says that CSI's must be atomic - */
/* hence should flush stdout.                    */

fflush(stdout);

Write(Heliosno(stdout),buffer,2,WRITE_TIMEOUT);

if (get_char() != INITIATOR)
{
    IOdebug("bad reply received to window size request\n");

    window.height = window.width = DEFAULT_SIZE;
}
else
{
    i = 0;

    while ((buffer[i++] = get_char()) != ';');

    *(buffer + i) = '\0';

    window.height = atoi(buffer);

    i = 0;

    while ((buffer[i++] = get_char()) != 'R');

    *(buffer + i) = '\0';

    window.width = atoi(buffer);
}
```

```
    window.x = 0;

    window.y = 0;

    screen_base = (char *)(MAP1 + (window.y * STRIDE) + window.x);

    save_screen();

    boxes();

    reset_clut();
    restore_screen();

    return 0;
}

void setup_colour()
{
    WORD i,bias;

    *vd453wraddraddr = 0L;
    set colour(~0,~0,~0);
    bias = 147;

    for(i=1; i <= 36; i++)
    {
        set_colour(bias + i*3,0,0);
        set_colour(0,bias + i*3,0);
        set_colour(0,0,bias + i*3);
        set_colour(bias + i*3,bias + i*3,0);
        set_colour(bias + i*3,0,bias + i*3);
        set_cotour(0,bias + i*3,bias + i*3);
        set_colour(bias + i*3,bias + i*3,bias + i*3);
    }
    set_colour(~0,0,0);
    set_colour(0,0,~0);
}

static unsigned long next = 1;

WORD randno(void)
{
    next = next * 1103515245 + 12345;

    return ((WORD)((next >> 16)));
}

void boxes(void)
```

```
{
    void gen_box(void);

    setup_colour();

    clear_screen();

    gen_box();
}

void draw_box(WORD x,WORD y,WORD width,WORD height,WORD colour)
{
    char *ptr;

    if (y + height >= window.height) return;
    if (x + width >= window.width) return;

    width &= ~3;
    height &= ~3;

    bytblt((char *)&colour,array,0,0,0,1,width,1,0);

    ptr = screen_base;
    ptr += STRIDE * y + x;
    ptr += 3;
    ptr &= ~3;

    bytblt(array,ptr,0,0,0,STRIDE,height,width,0;
}

void gen_box(void) /* put a square on the screen */
{
    WORD i,x,y;
    WORD width,height,colour;
    WORD randno(void);
    void draw_box(WORD,WORD,WORD,WORD,WORD);

    for (i=0;i<=20000;i++)
    {
        x = randno() & 0x3ff;
        y = randno() % window.height;
        width = (randno() & 0xff) + 70;
        height = (randno() & 0xff) + 70;
        colour = randno() & 0xff;
        draw_box(x,y,width,height,colour);
    }
}
```