Giaccess

Read/Write sound chip registers

Class: XBIOS

Category: Sound Functions

SYNOPSIS

```
#include <osbind.h>

val=Giaccess(data,reg);

short val;          value of register
short data;         data to write into register
short reg;          register to get/set
```

DESCRIPTION

The Giaccess function is used access the ST sound chip. The register to consider is passed in reg and the new data value to be loaded passed in data. If reg has bit 7 clear (i.e. ANDed with 0x7f) then the setting of the register is not changed and the current value returned. The legal values for reg are:

0 1	Channel A frequency
2 3	Channel B frequency
4 5	Channel C frequency
6	Noise period
7	Enable flags
10	Channel A amplitude
11	Channel B amplitude
12	Channel C amplitude
13 14	Envelope period
15	Envelope shape

RETURNS

The function returns the new value of the register in val.

Ikbdws

Write string to keyboard processor

Class: XBIOS

Category: IKBD I/O

SYNOPSIS

```
#include <osbind.h>

Ikbdws(count,buf);

short count;        number of bytes to write-1
const char *buf;    pointer to characters to write
```

DESCRIPTION

The Ikbdws function is used to write a string to the IKBD. count-1 characters are written from a buffer at buf.

SEE

Iorec, Initmous

Initmous

Set mouse mode and packet handler

Class: XBIOS

Category: IKBD I/O

SYNOPSIS

```
#include <osbind.h>

Initmous(mode,param,hand);

short mode;              new mouse mode
void *param;             mouse mode parameter block
void (*hand)(void);      mouse packet handler
```

DESCRIPTION

Initmous is used to change the way the mouse movements are interpreted by the system. The mouse is capable of operating in several modes, the value of mode sets which one is to be used:

Value	Meaning
0	Disable mouse.
1	Enable relative mouse mode, i.e. report the position changes to the packet handler.
2	Enable absolute mouse mode, i.e. always report an absolute mouse position to the packet handler.
4	Enable mouse keycode mode, i.e. never send motion packets, but pretend that a cursor key was pressed.

If the mouse is being placed into relative or keycode mode, param should point to a structure of the form:

```
struct param
{
  char topmode;
  char buttons;
  char xparam;
  char yparam;
};
```

The topmode element can have two values; 0 indicates that Y=0 occurs at the bottom of the screen; 1 indicates that Y=0 occurs at the top of the screen.

buttons allows the button reporting state to be changed. If bit 2 is set then the mouse buttons act like normal keys, otherwise they are reported as packets to the handler. Bits 0 and 1 (when set) cause the absolute mouse position to be reported on pressing and/or on releasing a mouse button respectively.

xparam and yparam change the way the X and Y position information is reported. They have different meanings for each of the three mouse modes:

Mode	Meaning
Relative	Mouse threshold, the number of mouse 'clicks' between relative position reports.
Absolute	Mouse scaling factor, the number of 'clicks' to give a single step in the absolute position.
Keycode	Mouse delta factor, the number of 'clicks' before reporting a left/right/up/down cursor motion.

In mouse absolute mode the param structure is extended so that it has the form:

```
struct param
{
  char topmode;
  char buttons;
  char xparam;
  char yparam;
  short xmax;
  short ymax;
  short xinitial;
  short yinitial;
};
```

xmax and ymax specify the maximum X and Y positions that the mouse may be allowed to move to, whilst xinitial and yinital give the position at which the mouse should be placed.

hand points to a mouse packet handler which will be called when mouse packets become available. Note that in keycode mode you need not supply a handler.

SEE

Ikbdws, Kbdvbase

CAVEATS

If you are using the AES or VDI then changing the mode of the mouse from the relative mode required for their operation will stop them from functioning correctly.

Iorec

Find serial device I/O structure

Class: XBIOS

Category: MFP Configuration

SYNOPSIS

```
#include <osbind.h>

base=Iorec(dev);

void *base;              base of I/O record
short dev;               serial device
```

DESCRIPTION

Iorec is used to obtain the base of the system data structure for one of the serial devices. The parameter dev gives the device:

Value	Device
0	RS-232
1	Keyboard
2	MIDI

The structure returned has the form:

```
struct iorec
{
  char *ibuf;          pointer to buffer
  short ibufsiz;       size of buffer
  short ibufhd;        head index
  short ibuftl;        tail index
  short ibuflow;       low-water mark
  short ibufhi;        high-water mark
};
```

If the structure requested was the for the RS-232 port then a second structure follows the first giving the RS-232 output buffer structure.

SEE

Midiws, Bconout, Bcostat, Bconin, Bconstat, Rsconf

Jenabint, Jdisint

Enable/Disable 68901 interrupt

Class: XBIOS

Category: MFP Configuration

SYNOPSIS

```
#include <osbind.h>

Jdisint(intno);  disable MFP interrupt
Jenabint(intno); enable MFP interrupt

short intno;          interrupt to manipulate
```

DESCRIPTION

The Jenabint and Jdisint functions enable and disable respectively interrupt intno on the 68901. This function is most often with Mfpint to enable or disable interrupts after changing the handler. The values for intno are as described under Mfpint.

SEE

Mfpint

Kbdvbase — Obtain system IKBD/MIDI dispatch handler

Class: XBIOS *Category: IKBD/MIDI I/O*

SYNOPSIS

```
#include <osbind.h>

base=Kbdvbase();

void (*volatile *base)(void); pointer to structure
```

DESCRIPTION

The Kbdvbase function obtains a pointer to the system structure used for dispatching MFP ACIA interrupts, so that you may patch into these if you wish. The Kbdvbase structure has the form:

```
struct kbdvecs
{
  void  (*midivec)(void);        MIDI-input
  void  (*vkbderr)(void);        keyboard error
  void  (*vmiderr)(void);        MIDI error
  void  (*statvec)(void);        IKBD status packet
  void  (*mousevec)(void);       mouse packet
  void  (*clockvec)(void);       clock packet
  void  (*joyvec)(void);         joystick packet
  void  (*midisys)(void);        system MIDI vector
  void  (*ikbdsys)(void);        system IKBD vector
  char  ikbdstate;               IKBD packet state
};
```

These vectors are used by the system for the following purposes:

midivec	MIDI input, by default a character is available in D0, which is then buffered into an lorec structure.
vkbderr vmiderr	Keyboard and MIDI overrun handler.
statvec mousevec clockvec joyvec	IKBD status, mouse, clock and joystick packet handlers. These routines are passed a pointer to the received packet in A0.
midisys ikbdsys	Low-level MIDI and IKBD packet handlers. These routines are called initially and parse the status of the MFP before calling the appropriate sub-function.

If you replace any of the handlers you should either call the old handler or return via an RTS instruction.

RETURNS

As noted above.

SEE

Mfpint

Kbrate

Get/Set the keyboard repeat rate and delay

Class: XBIOS *Category: Keyboard Configuration*

SYNOPSIS

```
#include  <osbind.h>

old=Kbrate(delay,rate);

short  old;          packed old delay and repeat rate
short  delay;        initial delay before repeat starts
short  rate;         new repeat rate
```

DESCRIPTION

Kbrate is used to change the keyboard repeat rate and the initial delay before repeating starts. delay gives the time (in 50Hz system ticks) before the key starts repeating, whilst rate gives the rate at which the key is to repeat. If a parameter is -1 then the current value is not changed.

RETURNS

A packed word is returned giving the old key repeat and delay rates. The initial delay is in the high byte of old, whilst the repeat rate is in the low byte.

Keytbl

Change keyboard translation tables

Class: XBIOS *Category: Keyboard Configuration*

SYNOPSIS

```
#include  <osbind.h>

ktab=Keytbl(normal,shift,caps);

char  **ktab;          keyboard translation vector
const  char  *normal;  un-shifted translation table
const  char  *shift;   shifted translation table
const  char  *caps;    CAPS-lock translation table
```

DESCRIPTION

Keytbl is used to change the mapping from keyboard scan codes to key-presses. Note that *all* keyboards return identical scan-codes for keys in the same place, but it is these translation tables, which give the ASCII value for the legend marked on a key, that are used to internationalise a keyboard.

The normal, shift and caps pointers should point a arrays of 128 characters which map scan-codes into ASCII codes when the appropriate key is depressed. If a scan-code does not have an ASCII representation the value returned is 0.

If you do not wish to change one of the translation tables the value (char *)-1 should be passed.

RETURNS

Keytbl returns in ktab a pointer to the structure in which all three tables are held:

```
struct  keytab
{
  char  *unshift;     /*  normal  table  */
  char  *shift;       /*  shifted  table  */
  char  *capslock;    /*  CAPS-lock  table  */
};
```

SEE

Bioskeys

Logbase

Find base of current drawing area

Class: XBIOS

Category: Graphics Configuration

SYNOPSIS

```
#include <osbind.h>

base=Logbase();

void *base;        base of logical screen
```

DESCRIPTION

Logbase returns a pointer to the base of the logical screen (i.e. the one onto which any drawing by the GEM VDI is done).

Do not confuse the physical and logical screens. The physical screen is that displayed, whilst the logical screen is the one onto which drawing occurs. Normally these will be the same but this is not required.

RETURNS

The function returns the base of the logical screen.

SEE

Physbase, Setscreen

Mfpint

Set MFP interrupt handler

Class: XBIOS

Category: MFP Configuration

SYNOPSIS

```
#include <osbind.h>

Mfpint(num,hand);

short num;              interrupt number to change
void (*hand)(void)      new interrupt handler
```

DESCRIPTION

Mfpint is used to change one of the multi-function peripheral adaptor (MFP) vectors. The vector to change is given by num, which has values:

Vector	Function
0	Parallel port
1	RS-232 Data Carrier Detect
2	RS-232 Clear-To-Send
3	BitBlt complete
4	RS-232 baud rate generator (Timer D)
5	200Hz System clock (Timer C)
6	Keyboard/MIDI
7	Floppy and Hard disk
8	Horizontal Blank (Timer B)
9	RS-232 transmit error
10	RS-232 transmit buffer empty
11	RS-232 receive error
12	RS-232 receive buffer full
13	DMA sound (Timer A)
14	RS-232 ring indicator
15	Mono monitor detect/DMA sound complete

The new interrupt handler is passed in hand. Note that installing a handler does not enable an interrupt this must be done separately via Jenabint.

SEE

Setexc, Jenabint, Jdisint

CAVEATS

The old MFP interrupt handler is discarded and so cannot subsequently be restored.

Note that the DMA sound option is only implemented on the Atari STE.

Midiws

Write string to MIDI port

Class: XBIOS *Category: MIDI I/O*

SYNOPSIS

```
#include <osbind.h>

Midiws(count,buf);

short count;        number of bytes to write-1
const char *buf;    pointer to characters to write
```

DESCRIPTION

The Midiws function is used to write a string to the MIDI port. count-1 characters are written from a buffer at buf.

SEE

lorec

Ongibit, Offgibit

Atomically set/reset port A bit

Class: XBIOS

Category: Miscellaneous Functions

SYNOPSIS

```
#include <osbind.h>

Ongibit(onmask);
Offgibit(offmask);

short onmask;       mask of bits to set
short offmask;      mask of bits to clear
```

DESCRIPTION

Ongibit and Offgibit are used to atomically set and reset bits on the sound chip port A. This atomic access is *essential* as the BIOS often modifies these bits under interrupt control. For Ongibit, onmask contains a 1 in every bit position which is to be set and a 0 in every position which is to be unchanged. By comparison the Offgibit offmask contains a 1 in every bit position which is to be unchanged and a 0 in every position which is to be reset.

The bits in these masks are used for the following purposes:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	General Purpose Output	Centronics Strobe	RS-232 DTR	RS-232 RTS	Floppy 1 Select	Floppy 0 Select	Floppy Side Select

SEE

Rsconf, Floprd, Flopwr

Physbase

Find base of current screen display

Class: XBIOS

Category: Graphics Configuration

SYNOPSIS

```
#include <osbind.h>

base=Physbase();

void *base;       base of physical screen
```

DESCRIPTION

Physbase returns a pointer to the base of the physical screen (i.e. the one actually displayed).

Do not confuse the physical and logical screens. The physical screen is that displayed, whilst the logical screen is the one onto which drawing occurs. Normally these will be the same but this is not required.

RETURNS

The function returns the base of the physical screen.

SEE

Logbase, Setscreen

Protobt

Build prototype boot sector

Class: XBIOS

Category: Miscellaneous Functions

SYNOPSIS

```
#include <osbind.h>

Protobt(buf,serial,type,exec);

void *buf;      512 byte prototype buffer
long serial;    serial number
short type;     disk type
short exec;     executable status of boot sector
```

DESCRIPTION

The Protobt function is used to build a boot sector for freshly formatted floppies. buf should point to a 512 byte buffer into which the sector will be built. This should contain any boot sector code you require.

serial gives the serial number to use for the disk. Note that the BIOS uses the serial number to distinguish floppies so if you give disks identical serial numbers they may become damaged. If serial has the value -1 then the current serial number in the boot sector is unchanged, otherwise if it has a value ≥0x01000000 then a random serial number is computed and used.

type specifies the disk type to construct it may have the values:

0	40 tracks, single sided (180K)
1	40 tracks, double sided (360K)
2	80 tracks, single sided (360K)
3	80 tracks, double sided (720K)
-1	Do not change type information

exec specifies whether the resulting sector is to executable. If exec is 0 the sector is made non-executable, 1 it is made executable and -1 the executable/non-executable status is preserved.

SEE

Flopfmt

Prtblk

Print bitmap

Class: XBIOS

Category: Printer Functions

SYNOPSIS

```
#include <osbind.h>

status=Prtblk(blk);

short status        error status
void *blk;          pointer to prtarg structure
```

DESCRIPTION

Prtblk is the general ST bitmap print utility. blk should point to a structure of the form:

```
struct prtarg
{
  char *blkptr;             block pointer
  unsigned short offset;    bit offset
  unsigned short width;     width
  unsigned short height;    height
  unsigned short left;      left leader
  unsigned short right;     right trailer
  unsigned short srcres;    source resolution
  unsigned short dstres;    destination resolution
  unsigned short *colpal;   colour palette
  unsigned short type;      printer type
  unsigned short port;      printer port
  char *masks;              halftone masks
};
```

The blkptr member points to the base of a bitmap to print, or to a string in text mode. offset gives the offset of the first bit to printed from the base of blkptr. height gives the height of the bitmap in pixels, or is 0 to indicate that this is a text mode usage. width gives the bitmap pixel width or a count of the number of characters to print in text mode. left and right specify the number of pixels to be skipped at the left and right hand edges when moving between lines.

type may have 1 of 4 values indicating the type of printer. The current values are:

0	Monochrome Atari printer
1	Colour Atari printer
2	Monochrome Daisy-wheel
3	Monochrome Epson Compatible

srcres gives the source resolution using the same values as Getrez. dstres gives the printer resolution and is 0 for draft mode and 1 for final mode. colpal points to a list of the colour palette settings. port gives the port to use, 0 for parallel, 1 for serial. masks points to a set of half-tone masks to use when mapping colours onto printer colours, or NULL to use the default masks.

Note that the system global _prt_cnt should be set to 1 prior to calling this function to ensure that the user cannot hit Alt-Help.

RETURNS

Prtblk returns zero if the printing was completed successfully or a negative error code.

```
/* Emulate the Scrdmp() command */
#include <osbind.h>
#include <stdlib.h>
#include <linea.h>

enum {MONO_ATARI, COLOUR_ATARI, DAISY, EPSON};

int lock(void)
{
  *(short *)0x4ee=1;  /* lock out Alt-Help */
}

int main(void)
{
  static struct
  {
    char *blkptr;
    unsigned short offset,width,height,left,right;
    unsigned short srcres,dstres,*colpal,type,port;
    char *masks;
  } prt;
  short palette[16],conf;
  register int i;

  conf=Setprt(-1);
  prt.blkptr=Physbase();   /* dump physical screen */
  if (conf&1)
    abort();               /* can't do daisywheels */
  else if (conf&4)
    prt.type=EPSON;
  else if (conf&2)
    prt.type=COLOUR_ATARI;
  else
    prt.type=MONO_ATARI;
  for (i=16; i--; )
    palette[i]=Setcolor(i,-1);
  prt.colpal=palette;      /* get palette */
  prt.port=(conf&16)>>4;   /* port */
  prt.srcres=Getrez();     /* screen resolution */
  prt.dstres=(conf&8)>>3;  /* printer resolution */
  linea0();                /* init Line-A for _MAX */
  prt.width=V_X_MAX;       /* find screen width */
  prt.height=V_Y_MAX;      /* and height */
  Supexec(lock);           /* enable Alt-Help */
  return Prtblk(&prt);     /* and dump */
}
```


Puntaes

Discard AES

Class: XBIOS *Category: Miscellaneous Functions*

SYNOPSIS

```
#include <osbind.h>

Puntaes();
```

DESCRIPTION

Puntaes is used to throw away the AES and any memory it occupies. Note that this function will only work for RAM-loaded TOS.

Random — Obtain random number

Class: XBIOS *Category: Miscellaneous Functions*

SYNOPSIS

```
#include <osbind.h>

rand=Random();

long rand;                    system random value
```

DESCRIPTION

Rand is the system random number generator and is normally used when obtaining serial numbers for freshly formatted floppies.

RETURNS

Random returns a 24 bit random number. Note that the algorithm used gives an *exact* 50% distribution for bit 0 and so this function should be used with care.

SEE

Protobt

Rsconf — Configure RS-232 communications port

Class: XBIOS *Category: MFP Configuration*

SYNOPSIS

```
#include <osbind.h>

save=Rsconf(speed,flow,ucr,rsr,tsr,scr);

unsigned long old;      old 68901 configuration
short speed;            new RS-232 speed request
short flow;             flow control mode
short ucr;              USART control register
short rsr;              receive status register
short tsr;              transmit status register
short scr;              synchronous character register
```

DESCRIPTION

Rsconf is used to configure the RS-232 communications interface. The speed parameter gives the requested speed:

Value	Baud Rate	Value	Baud Rate
0	19200	8	600
1	9600	9	300
2	4800	10	200
3	3600	11	150
4	2400	12	134
5	2000	13	110
6	1800	14	75
7	1200	15	50

flow allows the flow control method to be adjusted. The values are:

Value	Method
0	No flow control (default)
1	XON/XOFF (^S/^Q)
2	RTS/CTS
3	XON/XOFF and RTS/CTS

ucr sets the USART control register, the low byte only is used:

Bit 7	Bits 6-5	Bits 4-3	Bit 2	Bit 1	Bit 0
CLK/16	00-8 bits per word 01-7 bits per word 10-6 bits per word 11-5 bits per word	00-No Start/Stop 01-1 Start,1 Stop 10-1 Start,1½Stop 11-1 Start, 2 Stop	Parity	Use odd parity	Unused

rsr sets the receiver status register, the low byte only is used:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Buffer full	Overrun error	Parity error	Frame error	Break detect	Match busy	Sync strip	Receiver enable

tsr sets the transmit status register, the low byte only is used:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Buffer empty	Underrun error	Parity error	Frame error	Break detect	Match busy	Sync strip	Receiver enable

scr sets the synchronous character register, the low byte only is used and gives the character that will be searched for when an underrun error occurs in synchronous mode.

If any of the parameters has the value -1 then it is ignored and the current setting is unchanged.

RETURNS

Rsconf returns the old 68901 settings in a long word with the old ucr, rsr, tsr and scr packed from high to low in that order.

SEE

Bconout, Bcostat, Bconin, Bconstat, Ioctl

Scrdmp

Copy screen to printer

Class: XBIOS

Category: Printer Functions

SYNOPSIS

```
#include <osbind.h>

Scrdmp();
```

DESCRIPTION

This function dumps the screen to the printer in the same form as with the Alt-Help key.

SEE

Prtblk, v_hardcopy

Setcolor

Set display palette

Class: XBIOS

Category: Graphics Configuration

SYNOPSIS

```
#include <osbind.h>

old=Setcolor(num,new);

short old;  old BCD colour value
short num;  logical colour number to modify
short new;  new BCD colour value
```

DESCRIPTION

Setcolor is used to change the mapping from logical to physical colours. Colour values are stored in a BCD manner with the least-significant bit replacing the most-significant bit. A physical colour is packed in the following manner:

bits 15-12	bits 11-8 (Red)				bits 7-4 (Green)				bits 3-0 (Blue)			
Unused	R0	R3	R2	R1	G0	G3	G2	G1	B0	B3	B2	B1

R0 represents the least-significant bit of the red component of the colour, R3 the most-significant. Similarly, G0-G3 give the green component and B0-B3 the blue component.

Note that the peculiar packing method is to ensure backward compatibility from the Atari STE to the Atari ST, hence bits R0, G0 and B0 are not used on the ST.

The logical colour to change is passed in num, and the new packed colour in new. If new has the value -1 then the colour is not changed.

RETURNS

Setcolor returns the old BCD value for the logical colour.

SEE

Setpalette

Setpalette

Set display palette

Class: XBIOS

Category: Graphics Configuration

SYNOPSIS

```
#include <osbind.h>

Setpalette(palette);

short *palette;  pointer to screen palette
```

DESCRIPTION

Setpalette is used to reset the screen palette. For the current screen modes palette should point to an array of 16 words giving the BCD representations of the required screen colours.

Note that the palette assignment does not occur until the next vertical blank so a call to this routine should be followed by one to Vsync to ensure that the memory used by palette cannot be re-allocated before the new palette is installed.

SEE

Setcolor

CAVEATS

This function was spelt Setpallete (sic) in the original Atari bindings; both versions are included in the osbind.h file.

Setprt

Set/Get printer configuration

Class: XBIOS

Category: Printer Functions

SYNOPSIS

```
#include <osbind.h>

old=Setprt(new);

short old;        old configuration word
short new;        new configuration word
```

DESCRIPTION

Setprt is used to get or set the printer configuration. The configuration is changed to the value of new, currently 6 bits are defined in this:

Bit Number	Meaning when clear	Meaning when set
0	Dot matrix	Daisy wheel
1	Monochrome	Colour
2	Atari mode	"Epson" compatible
3	Preview mode	Final mode
4	Parallel port	RS-232 port
5	Continuous	Single sheet

Other bits should be preserved for future compatibility. In order to read the current status the value -1 may be used for new, in which case the configuration is not changed and the current configuration returned.

RETURNS

Setprt returns the old printer configuration word.

SEE

Scrdmp, Prtblk

CAVEATS

Beware of some older documentation which lists bit 1 as being set for mono and clear for colour, the bit should be *clear* for mono and *set* for colour.

Setscreen

Set screen parameters

Class: XBIOS

Category: Graphics Configuration

SYNOPSIS

```
#include <osbind.h>

Setscreen(log,phys,mode);

void *phys;       pointer to new physical screen base
void *log;        pointer to new logical screen base
short mode;       new screen mode request
```

DESCRIPTION

The Setscreen call is used to change the current screen mode, physical screen base and/or logical screen base. If any of the parameters is negative (e.g. -1) then that parameter is unchanged as a result of this call.

phys specifies a new physical screen base. This takes effect immediately (not at the next vertical blank as mentioned in older documentation), and as such you should be aware that screen 'flicker' may result. Note that on the Atari ST this *must* be on a 256 byte boundary. On the Atari STE this limitation has been relaxed and phys need only be word aligned.

log specifies a new logical screen base. It is onto this screen that all drawing is done. Note that it is recommended that after changing the logical screen base the (logical) screen be cleared to ensure that all pointers used internally by the VDI are correctly initialised.

mode specifies a new screen resolution. This parameter has the same values as those returned by Getrez. When mode is positive (i.e. the resolution is changed) the screen is automatically cleared and the internal state of the VT52 emulator reset.

SEE

Getrez

CAVEATS

This function does not inform the AES of a resolution change and so cannot be used once the AES has been initialised, unless you no longer require its services.

Ssbrk

Reserve system memory

Class: XBIOS

Category: Memory Allocation

SYNOPSIS

```
#include <osbind.h>

base=Ssbrk(len);

void *base;          base of memory allocated
short len;           amount of memory required
```

DESCRIPTION

Ssbrk was provided on the very first STs to provide a way of reserving system memory before the OS was loaded from disk. It is no longer implemented or required.

Supexec

Execute function in supervisor mode

Class: XBIOS

Category: Miscellaneous Functions

SYNOPSIS

```
#include <osbind.h>

val=Supexec(func);

long func();          function to call
```

DESCRIPTION

Supexec is used to call the named function in supervisor mode. The function should be careful if it wishes to call the BIOS or XBIOS since these are only re-entrant to three levels.

The value returned from the func is passed back as the return value from Supexec.

RETURNS

As noted above.

Vsync

Wait for vertical sync to occur

Class: XBIOS

Category: Miscellaneous Functions

SYNOPSIS

```
#include <osbind.h>

Vsync();
```

DESCRIPTION

Vsync is used to wait for a vertical blank to occur. It is often used to prevent 'flicker' when drawing graphics or to ensure that vertical blank driven objects are complete before being re-used (e.g. Setpalette).

Xbtimer

Configure MFP timer

Class: XBIOS

Category: MFP Configuration

SYNOPSIS

```
#include <osbind.h>

Xbtimer(timer,ctrl,data,hand);

short  timer;          number of timer to change
short  ctrl;           value to place in control
                       register
short  data;           value for timer data register
void  (*hand)(void);   pointer to interrupt handler
```

DESCRIPTION

Xbtimer allows the 68901 timers to be setup. The timer to change is passed in timer and has a value 0-3 indicating timer A, B, C or D. control is placed in the control register of the timer. data is placed in the data register of the timer. The interrupt handler for the timer is pointed to hand. The allocation of the timers is:

Timer	Usage
A	DMA sound counter
B	HBlank counter
C	200Hz System timer
D	RS-232 baud rate generator

SEE

Mfpint

7 Line-A Library

This section describes the Line-A library supplied with the Lattice C compiler. To access the facilities of the Line-A you should #include the file linea.h into your program.

The Line-A emulator provides the graphics primitives which are used by the VDI. The Line-A interface is in general inconsistent, difficult to use and completely non-portable. The name Line-A comes from the special 68000 instructions used to access the routines, which have the top nybble set to 'A'.

Before any of the Line-A routines may be used the linea0 function must be called to initialise the structures used by the bindings. All access to the Line-A routines is through a parameter block in which the input variables are placed, prior to executing the function, with a second data block made available for configuration and interrogation of the screen device layout.

The sixteen functions available in the Line-A are:

linea0	Initialise Line-A data structure
linea1	Plot single pixel
linea2	Get pixel value
linea3	Draw arbitrary line
linea4	Draw horizontal line
linea5	Render a filled rectangle
linea6	Render a line of a filled polygon
linea7	Perform a BITBLiT
linea8	Render bit-mapped character on screen
linea9	Show mouse cursor
lineaa	Hide mouse cursor
lineab	Transform mouse cursor
lineac	Remove user sprite
linead	Render user sprite
lineae	Copy raster form
lineaf	Flood fill area

linea0

Initialise Line-A data structure

Class: Line-A

Category: Initialisation

SYNOPSIS

```
#include <linea.h>

data=linea0();

struct la_data *data;      pointer to Line-A structure

extern LINEA_INFO la_info;
```

DESCRIPTION

linea0 is used to initialise the structure used when interrogating the Line-A data structures and calling the Line-A routines. It fills in the external structure la_info, containing the items:

```
typedef struct linea_info
{
  long li_d0;                  linea data structure
  struct la_data *li_a0;       linea data structure
  struct la_font **li_a1;      system font vector
  long (*li_a2)();             linea function vector
} LINEA_INFO;
```

li_d0 and li_a0 both point to the middle of the Line-A structures. Positive offsets from them are input parameters to Line-A commands, whilst negative offsets give the configuration and status information. The positive offset structure is:

```
typedef struct la_data
{
  short ld_vplanes;      number of bit planes
  short ld_vwrap;        number of bytes/video line
  short *ld_contrl;      pointer to CONTRL array
  short *ld_intin;       pointer to INTIN array
  short *ld_ptsin;       pointer to PTSIN array
  short *ld_intout;      pointer to INTOUT array
  short *ld_ptsout;      pointer to PTSOUT array
  short ld_colbit[4];    colour bit-plane[i] value
  short ld_lstlin;       draw last pixel flag
  short ld_lnmask;       line-style mask
  short ld_wmode;        writing mode
  short ld_x1;           X1 coordinate
  short ld_y1;           Y1 coordinate
  short ld_x2;           X2 coordinate
  short ld_y2;           Y2 coordinate
  short *ld_patptr;      fill pattern pointer
  short ld_patmsk;       fill pattern mask
  short ld_mfill;        multi-plane fill flag
  short ld_clip;         clipping flag
  short ld_xmincl;       minimum X clipping value
  short ld_ymincl;       minimum Y clipping value
  short ld_xmaxcl;       maximum X clipping value
  short ld_ymaxcl;       maximum Y clipping value
  short ld_xdda;         accumulator for textblt dda
  short ld_ddainc;       fixed point scale factor
  short ld_scaldir;      scale direction flag
  short ld_mono;         current font is monospaced
  short ld_srcx;         X coord of character in font
  short ld_srcy;         Y coord of character in font
  short ld_dstx;         X coord of character on screen
  short ld_dsty;         Y coord of character on screen
  short ld_delx;         width of character
  short ld_dely;         height of character
  void *ld_fbase;        pointer to start of font form
  short ld_fwidth;       width of font form
  short ld_style;        textblt special effects flags
  short ld_litemsk;      lightening mask
  short ld_skewmsk;      skewing mask
  short ld_weight;       thickening factor
  short ld_roff;         skew offset above baseline
  short ld_loff;         skew offset below baseline
  short ld_scale;        scaling flag
  short ld_chup;         character rotation angle
  short ld_textfg;       text foreground colour
  void *ld_scrtchp;      word-aligned effects buffer
  short ld_scrpt2;       offset to scaling buffer
  short ld_textbg;       text background colour
  short ld_copytran;     copy raster form type flag
  int (*ld_seedabort)(void);
                         seedfill abort detect
} LA_DATA;
```

The negative offset structure is:

```
typedef struct la_ext
{
  long ld_resvd1;
  struct la_font *ld_cur_font;
                              pointer to current font
                              header
  short ld_resvd2[23];
  short ld_m_pos_hx;          mouse x hot spot
  short ld_m_pos_hy;          mouse y hot spot
  short ld_m_planes;          writing mode for mouse
  short ld_m_cdb_bg;          mouse background colour
  short ld_m_cdb_fg;          mouse foreground colour
  short ld_mask_form[32];     mouse mask and form
  short ld_inq_tab[45];       vq_extnd information
  short ld_dev_tab[45];       v_opnwk information
  short ld_gcurx;             current mouse x position
  short ld_gcury;             current mouse x position
  short ld_m_hid_ct;          mouse hide count
  short ld_mouse_bt;          mouse button status
  short ld_req_col[3][16];    internal vq_color lookup
  short ld_siz_tab[15];       current text, line and
                              marker sizes
  short ld_resvd3;
  short ld_resvd4;
  short *ld_cur_work;         current vwork attributes
  struct la_font *ld_def_font;
                              default font header
  struct la_font *ld_font_ring[4];
                              vdi font ring
  short ld_font_count;        number of fonts in font
                              ring
```

```
    short  ld_resvd5[45];
    unsigned  char  ld_cur_ms_stat;
                                    mouse  status
    char  ld_resvd6;
    short  ld_v_hid_cnt;            cursor  hide  count
    short  ld_cur_x;                mouse  x  position
    short  ld_cur_y;                mouse  y  position
    char  ld_cur_flag;              mouse  draw  status
    char  ld_mouse_flag;            mouse  processing  enabled
    long  ld_resvd7;
    short  ld_v_sav_xy[2];          saved  cursor  xy  position
    short  ld_save_len;             height  of  saved  form
    short  *ld_save_addr;           screen  address  of  saved
                                    form
    short  ld_save_stat;            save  status
    long  ld_save_area[4][16];      form  save  area
    void  (*ld_user_tim)();         user  timer  vector
    void  (*ld_next_tim)();         next  timer  vector
    void  (*ld_user_but)();         user  button  vector
    void  (*ld_user_cur)();         user  cursor  vector
    void  (*ld_user_mot)();         user  motion  vector
    short  ld_cel_ht;               cell  height
    short  ld_cel_mx;               max  x  cells
    short  ld_cel_my;               max  y  cells
    short  ld_cel_wr;               displacement  to  next
                                    vertical  cell
    short  ld_col_bg;               background  colour  index
    short  ld_col_fg;               foreground  colour  index
    void  *ld_cur_ad;               cursor  address
    short  ld_cur_off;              offset  to  first  cell
    short  ld_cur_xy[2];            cursor  xy  position
    char  ld_cur_cnt;               cursor  flash  period
    char  ld_cur_tim;               cursor  flash  countdown
    void  *ld_fnt_ad;               address  of  font  data
    short  ld_fnt_nd;               last  ade  in  font
    short  ld_fnt_st;               first  ade  in  font
    short  ld_fnt_wr;               font  form  width
    short  ld_x_max;                horizontal  pixel
                                    resolution
    void  *ld_off_ad;               pointer  to  font  offset
                                    table
    short  ld_status;               cursor  status
    short  ld_y_max;                vertical  pixel  resolution
    short  ld_bytes_lin;            width  of  destination  form
} LA_EXT;
```

Note that this structure may be accessed using ((LA_EXT *)la_info.li_a0-1)->ld..

The remaining structure members in linea_info are; li_a1 which points to a NULL terminated array of system fonts, *currently* three fonts are available. li_a2 points to an array of the 16 Line-A entry points so that you may remove the Line-A handler overhead and call them directly. If you do this be aware that some of the functions *must* be run in supervisor mode and that the registers they destroy is *completely* undefined.

To simplify access to these variables macros are provided to perform all the indirections. These macros are named on a variant of the structure names, so that to gain access to, for instance, ld_y_max you may simply use V_Y_MAX, or to access one of the positive structures, e.g. ld_patptr, simply PATPTR.

You should be aware that the CONTRL, INTIN, PTSIN, INTOUT and PTSOUT are inherited from the last user, hence if a process has terminated these arrays may point to non-allocated memory. If you need to use these arrays you should ensure that you have *either* allocated a VDI virtual workstation, *or* have placed pointers to your own private arrays in these elements.

The system fonts use the same format as GDOS fonts, a structure of the form:

```
typedef  struct  la_font
{
  short  font_id;                   face  identifier
  short  font_size;                 font  size  in  points
  char  font_name[32];              face  name
  short  font_low_ade;              lowest  ASCII  value
  short  font_hi_ade;               highest  ASCII  value
  short  font_top_dst;              top  line  distance
  short  font_ascent_dst;           ascent  line  distance
  short  font_half_dst;             half  line  distance
  short  font_descent_dst;          descent  line  distance
  short  font_bottom_dist;          bottom  line  distance
  short  font_fatest;               widest  char  in  font
  short  font_fat_cell;             widest  char  cell  in  font
  short  font_left_off;             left  offset
  short  font_right_off;            right  offset
  short  font_thickening;           pixels  to  widen  chars
  short  font_underline;            underline  pixel  width
  short  font_lightening;           lightening  mask
  short  font_skewing;              skewing  mask
  short  font_flags;                flags
  short  *font_horiz_off;           pointer  to  HOT
  short  *font_char_off;            pointer  to  COT
  void  *font_data;                 pointer  to  font  form
  short  font_width;                font  width
  short  font_height;               font  height
  struct  la_font  *font_next;      pointer  to  next  font
} LA_FONT;
```

Most fields in the LA_FONT structure are self-explanatory with reference to v_gtext, the other fields are:

font_thickening	Number of pixels to increase each horizontal pixel run by to achieve a **bold** font.
font_underline	Number of pixels in the underline effect.
font_lightening	Mask used when removing pixels to create a 'disabled' character. This normally has the value 0x5555, indicating that alternate pixels should be dropped.
font_skewing	Mask used when creating *skewed* characters. This mask is considered rotated vertically, and then for each row that has the skew mask set the pixel row is shifted right by one pixel . The usual value is 0x5555, giving a skew of 26.6°.

font_flags	This consists of a bitmap giving flags for this font: Bit — Meaning (When set) 0 — Font is default system font 1 — Horizontal offset table present 2 — Font is in Motorola format 3 — Font is monospaced Note that *all* fonts which are in memory will be in Motorola format.
font_horiz_off	Pointer to horizontal offset table (HOT). This is an array of short integers with the most significant (signed) byte giving the left offset (i.e. added *prior* to printing) and the least significant (signed) byte giving the right offset (i.e. added *after* to printing). This can be useful for kerning or accented use. Note that the VDI output functions do not support horizontal offset tables correctly, and the Line-A routines are the only way to use them successfully.
font_char_off	Pointer to character offset table (COT). This is an array of shorts giving the 'X' co-ordinate of each character in the font within the form. Note that the first element is for the first character in the set and not 0, hence you must subtract font_low_ade before indexing into this array.

SEE

v_opnwk, v_opnvwk

linea1

Plot single pixel

Class: Line-A

Category: Pixel Manipulation

SYNOPSIS

```
#include <linea.h>

linea1();

putpixel(x,y,colour);

INTIN[0]=colour;          colour of pixel to plot
PTSIN[0]=X;               X co-ordinate of pixel
PTSIN[1]=Y;               Y co-ordinate of pixel
```

DESCRIPTION

linea1 plots single pixels on screen. INTIN(0) holds the colour to give the pixel, PTSIN(0) and PTSIN(1) hold the required X and Y co-ordinates.

The putpixel macro is provided in linea.h to simplify the use of this function and takes parameters x, y and colour.

SEE

v_pmarker, v_pline, linea2, linea3, linea4

CAVEATS

This function pays no regard to any clipping rectangle installed in the Line-A input array.

linea2 — Get pixel value

Class: Line-A

Category: Pixel Manipulation

SYNOPSIS

```
#include <linea.h>

colour=linea2();

colour=getpixel(x,y);

short colour;                  colour of pixel
PTSIN[0]=X;                    X co-ordinate of pixel
PTSIN[1]=Y;                    Y co-ordinate of pixel
```

DESCRIPTION

linea2 obtains the colour value of a single pixels on screen. PTSIN(0) and PTSIN(1) hold the required X and Y co-ordinates, and the current value of the pixel is returned in colour.

The getpixel macro is provided in linea.h to simplify the use of this function and takes parameters (x, y) returning colour.

SEE

v_get_pixel, linea1

CAVEATS

This function pays no regard to any clipping rectangle installed in the Line-A input array.

linea3 — Draw arbitrary line

Class: Line-A

Category: Line Drawing

SYNOPSIS

```
#include <linea.h>

linea3();

X1=x1;                         starting X co-ordinate
Y1=y1;                         starting Y co-ordinate
X2=x2;                         ending X co-ordinate
Y2=y2;                         ending Y co-ordinate
COLBIT0=colour;                value for bit plane 0
COLBIT1=colour>>1;             value for bit plane 1
COLBIT2=colour>>2;             value for bit plane 2
COLBIT3=colour>>3;             value for bit plane 3
LNMASK=style;                  line pattern mask.
WMODE=mode;                    writing mode.
LSTLIN=last;                   draw last pixel flag
```

DESCRIPTION

linea3 draws a line between points (X1,Y1) and (X2,Y2). The colour to use is split into bits and provided in the COLBIT elements. LNMASK gives the bit pattern to use when drawing the line, whilst the drawing mode is given by WMODE. The values for WMODE (which are the VDI MD_... modes -1) are:

0	Replace mode; the new data replaces the old.
1	Transparent mode only affects pixels where the pixel is already set.
2	Exclusive OR mode.
3	Reverse transparent mode only affects pixels where the source pixel is not set.

LSTLIN is used when drawing lines in XOR mode and normally is -1 indicating that the last point in the line is to be omitted, or if 0 the point is plotted.

SEE

linea1, linea4, v_pline, vswr_mode

CAVEATS

This function pays no regard to any clipping rectangle installed in the Line-A input array.

linea4

Draw horizontal line

Class: Line-A

Category: Line Drawing

SYNOPSIS

```
#include <linea.h>

linea4();

X1=x1;                  starting X co-ordinate
X2=x2;                  ending X co-ordinate
Y1=y;                   Y co-ordinate
COLBIT0=colour;         value for bit plane 0
COLBIT1=colour>>1;      value for bit plane 1
COLBIT2=colour>>2;      value for bit plane 2
COLBIT3=colour>>3;      value for bit plane 3
WMODE=mode;             writing mode
PATPTR=pattern;         pointer to fill pattern
PATMSK=index;           pattern count
MFILL=flag;             multi plane fill flag
```

DESCRIPTION

linea4 draws a horizontal line between points (X1,Y1) and (X2,Y1). The colour to use is split into bits and provided in the COLBIT elements. PATPTR points to an array of PATMSK+1 line patterns. The pattern chosen for a particular line segment is then a function of Y1 and PATMASK. If MFILL is zero then the writing mode WMODE is used as described under linea3.

When MFILL is non-zero the value of WMODE is ignored and the planes are simply filled with the bits in COLBITs.

SEE

v_pline, linea1, linea3, linea5

CAVEATS

This function pays no regard to any clipping rectangle installed in the Line-A input array.

linea5

Render a filled rectangle

Class: Line-A

Category: Area Filling

SYNOPSIS

```
#include <linea.h>

linea5();

X1=x1;                  left X co-ordinate
Y1=y1;                  top Y co-ordinate
X2=x2;                  right X co-ordinate
Y2=y2;                  bottom Y co-ordinate
COLBIT0=colour;         value for bit plane 0
COLBIT1=colour>>1;      value for bit plane 1
COLBIT2=colour>>2;      value for bit plane 2
COLBIT3=colour>>3;      value for bit plane 3
WMODE=mode;             writing mode
PATPTR=pattern;         pointer to fill pattern
PATMSK=index;           pattern count
MFILL=flag;             multi plane fill flag
CLIP=state;             clipping flag
XMINCL=x1clip;          left edge X clipping
YMINCL=y1clip;          top edge Y clipping
XMAXCL=x2clip;          right edge X clipping
YMAXCL=y2clip;          bottom edge Y clipping
```

DESCRIPTION

linea5 draws a filled rectangle with upper left corner (X1,Y1) and lower right corner (X2,Y1).

The COLBIT, PATPTR, PATMSK, MFILL and WMODE parameters are as described under linea4. Note that the PATPTR value is identical to that of linea4 which is used as the primitive for this function.

An optional clipping rectangle may be specified with this function; it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). To enable clipping CLIP should be set to 1, or disabled by setting CLIP to 0.

SEE

linea1, linea4, v_bar, v_recfl

linea6

Render a line of a filled polygon

Class: Line-A

Category: Line Drawing

SYNOPSIS

```
#include <linea.h>

linea6();

PTSIN[]=...;                array of vertices
CONTRL[1]=n;                number of vertices
Y1=y1;                      line to draw
COLBIT0=colour;             value for bit plane 0
COLBIT1=colour>>1;          value for bit plane 1
COLBIT2=colour>>2;          value for bit plane 2
COLBIT3=colour>>3;          value for bit plane 3
WMODE=mode;                 writing mode
PATPTR=pattern;             pointer to fill pattern
PATMSK=index;               pattern count
MFILL=flag;                 multi plane fill flag
CLIP=state;                 clipping flag
XMINCL=x1clip;              left edge X clipping
YMINCL=y1clip;              top edge Y clipping
XMAXCL=x2clip;              right edge X clipping
YMAXCL=y2clip;              bottom edge Y clipping
```

DESCRIPTION

linea6 draws one line of a filled polygon. The polygon is specified as an array of vertices in PTSIN, with the number of vertices in CONTRL(1). Note that the first vertex must be repeated as the last vertex, but this extra vertex is not included in the vertex count. The line drawn as a result of this function is Y1.

The COLBIT, PATPTR, PATMSK, MFILL and WMODE parameters are as described under linea4. Note that the PATPTR value is identical to that of linea4 which is used as the primitive for this function.

An optional clipping rectangle may be specified with this function; it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). To enable clipping CLIP should be set to 1, or disabled by setting CLIP to 0.

SEE

linea1, linea4, v_fillarea

CAVEATS

This function only performs the fill line selection correctly when the fill pattern height is an exact power of 2. Also you must ensure that the PTSIN array is large enough for your requirements, otherwise the system may crash mysteriously.

EXAMPLE

```
/*
 * draw a simple polygon filled with a single plane
 * pattern
 */

#include <linea.h>

int main(void)
{
  short pts[]={160,100,0,50,319,199,319,50,160,100};
  short contrl[2];
  short pattern[]=
  {
    0x0940,    /* 0000100101000000 */
    0x0940,    /* 0000100101000000 */
    0x0f40,    /* 0000111101000000 */
    0x0940,    /* 0000100101000000 */
    0x0940,    /* 0000100101000000 */
    0x0000,    /* 0000000000000000 */
    0x64dc,    /* 0110010011011100 */
    0x8a88,    /* 1000101010001000 */
    0xcac8,    /* 1100101011001000 */
    0x2a88,    /* 0010101010001000 */
    0xa488,    /* 1100100100001000 */
    0x0000,    /* 0000000000000000 */
    0x0000,    /* 0000000000000000 */
    0x0000,    /* 0000000000000000 */
    0x0000,    /* 0000000000000000 */
    0x0000,    /* 0000000000000000 */
  };
  register int i;

  linea0();                 /* initialise */
  PTSIN=pts;                /* setup ptsin */
  CONTRL=contrl;            /* and contrl */
  contrl[1]=sizeof(pts)/(sizeof(short)*2)-1;
  COLBIT0=1;                /* use all bit planes */
  COLBIT1=1;
  COLBIT2=1;
  COLBIT3=1;
  WMODE=0;                  /* replace mode */
  PATPTR=pattern;           /* set up pattern pointer */
  PATMSK=sizeof(pattern)/sizeof(short)-1;
  MFILL=0;                  /* no multi-plane fill */
  CLIP=0;                   /* no clipping */

  Y1=0;                     /* step over all lines used */
  for (i=0; i<200; i++)
  {
    linea6();               /* render one line */
    Y1++;                   /* move to next line */
  }
  return 0;
}
```

linea7

Perform a BITBLiT

Class: Line-A

Category: BITBLiT Functions

SYNOPSIS

```
#include <linea.h>

linea7(blit);

LA_BLIT *blit;    pointer to blit structure
```

DESCRIPTION

linea7 is the system BITBLiT primitive (**bit block transfer**), and is unusual in that it does not use the input array. The function is passed a pointer to an LA_BLIT structure, which has the form:

```
typedef struct la_blk
{
  short  bl_xmin;        minimum x
  short  bl_ymin;        minimum y
  short  *bl_form;       word aligned memory form
  short  bl_nxwd;        offset to next word in line
  short  bl_nxln;        offset to next line in plane
  short  bl_nxpl;        offset to next plane
} LA_BLK;

typedef struct la_blit
{
  short  bb_b_wd;        width of block in pixels
  short  bb_b_ht;        height of block in pixels
  short  bb_plane_ct;    number of planes
  short  bb_fg_col;      foreground colour
  short  bb_bg_col;      background colour
  char   bb_op_tab[4];   fg/bg logic table
  struct la_blk bb_s;    source info block
  struct la_blk bb_d;    destination info block
  short  *bb_p_addr;     pattern buffer address
  short  bb_p_nxln;      offset to next pattern line
  short  bb_p_nxpl;      offset to next pattern plane
  short  bb_p_mask;      pattern index mask
  char   bb_fill[24];    work space
} LA_BLIT;
```

The function performs a blit from a source to a destination form. The source form has a top left corner (bb_s.bl_xmin, bb_s.bl_ymin) with width and height bb_b_wd and bb_b_ht respectively. bb_plane_ct bit planes are then transferred to the destination form with top left corner (bb_d.bl_xmin, bb_d.bl_ymin). Note that the algorithm employed deals successfully with overlapping forms.

The remaining parameters of the source and destination form definitions (bb_s.bl_... and bb_d.bl_...) are the pointer to the base of the form bl_form, bl_nxwd, the offset to the next word in the same plane (i.e. skipping the interleaved planes), bl_nxln a count of the number of bytes in one line of the form and finally bl_nxpl, the offset to the next plane from the start of one plane, thus allowing blitting from a linear form in memory to the interleaved plane structure of the ST display.

As the planes are transferred by the blit operation, a logical operation is performed on the bits. The operations are a generalisation of those performed for the VDI vro_cpyfm routine. The logic table consist of 4 bytes, indexed by considering the value of the bits in foreground and background colours, bb_fg_col and bb_bg_col. The logic operation used for a particular bit plane is obtained by considering bb_op_tab(bb_fg_col * 2 + bb_bg_col). The logical operations are identical to those discussed under vro_cpyfm (S_AND_D etc.).

The final variant available with linea7 allows a pattern to be ANDed into the source prior to being combined with the destination. To enable the pattern integration, bb_p_addr should point to an array of patterns, similar to those used for linea4. Note that if you do not require the pattern facility you should set b_p_addr to NULL. p_nxln and p_nxpl are used identically to bl_nxln and bl_nxpl, discussed above for forms, but apply instead to the pattern 'form'. Note that p_nxln must be an exact power of two. p_mask is used with p_nxln to mask the appropriate part of the source. If p_nxlen has a value of 1<<n (i.e. an exact power of two), then the value for p_mask is (p_nxln/2-1)<<n.

The remaining 24 bytes of the LA_BLIT structure, bb_fill, are used internally by the blit algorithm.

SEE

lineae, vro_cpyfm, vrt_cpyfm

CAVEATS

This call makes almost no checks as to the validity of what is being attempted, so great care should be taken when using it as it is very easy to disrupt the machine without due care.

This function pays no regard to any clipping rectangle installed in the Line-A input array.

EXAMPLE

```
/*
 * blit the top left of the screen to the bottom
 * right
 */

#include  <linea.h>
#include  <vdi.h>
#include  <string.h>
#include  <osbind.h>
#include  <stddef.h>

int  main(void)
{
  LA_BLIT  blt;

  linea0();

  blt.bb_b_wd=V_X_MAX/2-1;      /* blit half screen */
  blt.bb_b_ht=V_Y_MAX/2-1;
  blt.bb_plane_ct=VPLANES;      /* number of planes */
  blt.bb_fg_col=1;              /* maintain colours */
  blt.bb_bg_col=1;
  memset(blt.bb_op_tab,S_OR_D,sizeof(blt.bb_op_tab));
  blt.bb_s.bl_xmin=blt.bb_s.bl_ymin=0;
  blt.bb_s.bl_form=blt.bb_d.bl_form=Logbase();
  blt.bb_s.bl_nxwd=blt.bb_d.bl_nxwd=1<<VPLANES;
  blt.bb_s.bl_nxln=blt.bb_d.bl_nxln=VWRAP;
  blt.bb_s.bl_nxpl=blt.bb_d.bl_nxpl=2;
  blt.bb_p_addr=NULL;           /* no pattern */
  blt.bb_s.bl_xmin=blt.bb_s.bl_ymin=0;
  blt.bb_d.bl_xmin=V_X_MAX/2;
  blt.bb_d.bl_ymin=V_Y_MAX/2;

  linea7(&blt);                 /* blit it */
  return  0;
}
```


linea8

Render bit-mapped character on screen

Class: Line-A

Category: BITBLiT Functions

SYNOPSIS

```
#include  <linea.h>

linea8();

FBASE=font;          base of font form
FWIDTH=width;        width of font form
SRCX=ch;             X co-ordinate of character in
                     font form
SRCY=0;              Y co-ordinate of character in
                     font form
DELX=w;              width of character
DELY=h;              height of character
DSTX=x;              X co-ordinate to plot character
                     on screen
DSTY=y;              Y co-ordinate to plot character
                     on screen
TEXTFG=fgcol;        foreground colour
TEXTBG=bgcol;        background colour
STYLE=effect;        text effect
LITEMASK=lmask;      lightening mask
SKEWMASK=smask;      skewing mask
WEIGHT=thick;        thickening width
ROFF=roff;           skewing offset above baseline
LOFF=loff;           skewing offset below baseline
SCALE=enable;        enable scaling
XDDA=0x8000;         scaling variable
DDAINC=factor;       scaling factor
SCALDIR=dir;         scaling direction
CHUP=angle;          rotation angle
MONO=monoflag;       mono-spaced flag
SCRTCHP=buffer;      work buffer
SCRPT2=offset;       scaling offset into buffer
WMODE=mode;          writing mode
CLIP=state;          clipping flag.
XMINCL=x1clip;       left edge X clipping
YMINCL=y1clip;       top edge Y clipping
XMAXCL=x2clip;       right edge X clipping
YMAXCL=y2clip;       bottom edge Y clipping.
```

DESCRIPTION

linea8 is used for rendering bit-mapped fonts on screen. A character from the font at pixel offset (SRCX, SRCY) with width and height DELX and DELY is transferred to the destination (DSTX, DSTY). TEXTFG and TEXTBG give the foreground and background colours which should be used when rendering.

To enable font scaling SCALE is made non-zero, and the direction of scaling put in SCALDIR; 0 for down, otherwise up. When using scaling the fixed point variable XDDA should be initialised to 0.5 (0x8000 in the representation used), and the DDA scaling factor set up. If the final size required is final and the actual font size is actual then for scaling up DDAINC should be set to 0x100 * (final - actual)/actual, else for scaling down 0x100 * final/actual.

The effects applied to the font may be set via the STYLE bitmap:

Bit	Effect
0	**Thicken**
1	'Lighten'
2	*Skew*
3	Underline (in-operative)
4	Outline

To rotate the text CHUP may be set to the number of degrees required times 10, in the same way as vst_rotation. MONO should be set to 1 for mono-spaced fonts or non-zero for proportional fonts. SCRTCHP should be set to a word aligned scratchpad area in which text special effects are rendered. The size of this buffer should be twice the size of the largest character which may result. SCRPT2 gives an offset into the SCRTCHP buffer which is used when scaling fonts. Like the SCRTCHP buffer it should have space for twice the largest character which may result.

When rendering the text into the destination form the writing mode WMODE is used as described under linea3, however this is extended from the normal set of four modes and any of the BITBLiT modes may be used (S_AND_D etc.), by adding 4 to the normal value.

The remaining variables which must be set are normally copied directly from the font header.

An optional clipping rectangle may be specified with this function, it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). To enable clipping CLIP should be set to 1, or disabled by setting CLIP to 0.

SEE

linea7, v_gtext, v_justified, vst_rotation

EXAMPLE

```
/*
 * write out text in all styles, not rotated or
 * scaled
 */
#include <linea.h>
int main(void)
{
  register int i;

  linea0();
  for (i=0; i<0x20; i++)
  {
    register const char *s="Hello World";
    register char c;

    /* set up initial screen X co-ordinate */
    DSTX=0;
    if (i>=0x10)
      DSTX=V_X_MAX/2;
    while (c=*s++)
    {
      short x[500];

      TEXTFG=1;          /* colours */
      TEXTBG=0;
      STYLE=i;           /* and style */
      /* compute source position and height */
      c-=V_DEF_FONT->font_low_ade;
      SRCX=V_DEF_FONT->font_char_off[c];
      SRCY=0;
      DELX=V_DEF_FONT->font_char_off[c+1]-SRCX;
      DELY=V_DEF_FONT->font_height;
      FBASE=V_DEF_FONT->font_data;
      FWIDTH=V_DEF_FONT->font_width;
      /* copy masks and effects */
      LITEMSK=V_DEF_FONT->font_lightening;
      SKEWMSK=V_DEF_FONT->font_skewing;
      WEIGHT=V_DEF_FONT->font_thickening;
      /* offsets for skewed text */
      if (STYLE & 1<<2)     /* skewed */
      {
        ROFF=V_DEF_FONT->font_right_off;
        LOFF=V_DEF_FONT->font_left_off;
      }
      else
        ROFF=LOFF=0;
      SCALE=0;
      XDDA=0x8000;          /* initialise anyway */
      DDAINC=256;
      SCALDIR=0;
      CHUP=0;
      MONO=0;
      SCRTCHP=x;
      SCRPT2=sizeof(x)/2;
      CLIP=0;
      DSTY=(i&0xf)*V_DEF_FONT->font_height;
      /* no need to redo DSTX as linea8 does it */
      linea8();
    }
  }
}
```

linea9

Show mouse cursor

Class: Line-A

Category: Sprite Manipulation

SYNOPSIS

```
#include <linea.h>
linea9();
INTIN[0]=force;  zero to force mouse to show
showmouse(force);
```

DESCRIPTION

linea9 is identical to the VDI call v_show_c and is used to decrease the mouse hide depth. It takes a single parameter in INTIN(0), which if zero forces the mouse hide depth counter to be reset and the mouse displayed regardless; if it is non-zero then the hide depth is reduced by one and the mouse displayed if the hide depth becomes zero.

The showmouse macro is provided to simplify the interface and takes a single parameter force, as described above.

SEE

lineaa, v_show_c, v_dspcur, graf_mouse

lineaa

Hide mouse cursor

Class: Line-A

Category: Sprite Manipulation

SYNOPSIS

```
#include <linea.h>
lineaa();
hidemouse();
```

DESCRIPTION

lineaa (and the equivalent name hidemouse) is identical to the VDI call v_hide_c and is used to increase the mouse hide depth. When the hide depth is non-zero the mouse cursor is not displayed.

SEE

linea9, v_hide_c, v_rmcur, graf_mouse

lineab — Transform mouse cursor

Class: Line-A *Category: Sprite Manipulation*

SYNOPSIS

```
#include <linea.h>

lineab();
```

DESCRIPTION

lineab is identical is used to change the form of the mouse cursor, in an identical manner to vsc_form. A pointer to an LA_SPRITE structure is placed in INTIN(0-1) giving the new form:

```
typedef struct la_sprite
{
  short ls_xhot;        X hot spot offset
  short ls_yhot;        Y hot spot offset
  short ls_form;        1 for VDI, -1 for XOR
  short ls_bgcol;       background colour index
  short ls_fgcol;       foreground colour index
  short ls_image[32];   interleaved image
} LA_SPRITE;
```

The image is stored in image/mask interleaved form. The first word in the ls_image array gives the mask, the second the data, the third the mask, etc. The ls_form value gives the way the mouse is rendered on screen. For both VDI and XOR modes, most combinations are identical:

Foreground	Background	Colour plotted
0	0	Destination
0	1	Background
1	0	Foreground (VDI mode) Inverse destination (XOR mode)
1	1	Foreground

To save the old mouse form before changing it the old form should be copied from V_MASK_FORM (note that the full LA_SPRITE structure for the mouse cursor starts at V_M_POS_HX). Also when changing the mouse form you should disable drawing of the mouse by setting V_MOUSE_FLAG to 0 and restore it afterwards. This ensures that 'droppings' do not occur.

SEE

linead, vsc_form, graf_mouse

lineac — Remove user sprite

Class: Line-A *Category: Sprite Manipulation*

SYNOPSIS

```
#include <linea.h>

lineac(save);

void *save;   pointer to sprite save area
```

DESCRIPTION

lineac is used to remove a sprite previously drawn using linead. A pointer to the sprite save area is passed and the screen restored from this.

SEE

linead, linea9, lineaa

linead — Render user sprite

Class: Line-A *Category: Sprite Manipulation*

SYNOPSIS

```
#include <linea.h>

linead(x,y,sprite,save);

int x;                  x position for sprite
int y;                  y position for sprite
LA_SPRITE *sprite;      pointer to sprite definition
void *save;             pointer to sprite save area
```

DESCRIPTION

linead is used to render a user defined sprite. The position to plot the sprite at is passed in x and y, and the sprite definition in sprite. sprite is a pointer to an LA_SPRITE structure, discussed previously under lineab.

The save area is used to keep a copy of the screen area corrupted by the sprite. It shares the first 5 fields of the LA_SPRITE structure, but must have room for the image from all screen bitplanes. Hence it should have a size of 10+VPLANES*64 bytes.

SEE

lineac, lineab

lineae — Copy raster form

Class: Line-A *Category: BITBLiT Functions*

SYNOPSIS

```
#include <linea.h>

lineae();

INTIN[0]=wr_mode;     logic operation to perform
CONTRL[7-8]=src;      source memory form definition
                      block
CONTRL[9-10]=dest;    destination memory form
                      definition block
INTIN[1]=one_col;     colour index for 1s in the data
INTIN[2]=zer_col;     colour index for 0s in the data
PTSIN[0]=llx1;        lower-left X of first rectangle
PTSIN[1]=lly1;        lower-left Y of first rectangle
PTSIN[2]=urx1;        upper-right X of first rectangle
PTSIN[3]=ury1;        upper-right Y of first rectangle
PTSIN[4]=llx2;        lower-left X of second rectangle
PTSIN[5]=lly2;        lower-left Y of second rectangle
PTSIN[6]=urx2;        upper-right X of second rectangle
PTSIN[7]=ury2;        upper-right Y of second rectangle
COPYTRAN=mode;        opaque/transparent mode
CLIP=state;           clipping flag.
XMINCL=x1clip;        left edge X clipping
YMINCL=y1clip;        top edge Y clipping
XMAXCL=x2clip;        right edge X clipping
YMAXCL=y2clip;        bottom edge Y clipping.
```

DESCRIPTION

lineae is the VDI raster copy primitive and performs the equivalent of both vrt_cpyfm and vro_cpyfm. Referring to the description of vrt_cpyfm and vro_cpyfm, the parameters discussed there are placed in the arrays as noted above. To perform a vro_cpyfm, COPYTRAN should be set to 0, or 1 to perform a vrt_cpyfm.

When COPYTRAN is 1, one_col and zer_col should be provided to give the colours for ones and zeroes respectively, as discussed under vrt_cpyfm.

An optional clipping rectangle may be specified with this function; it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). To enable clipping CLIP should be set to 1, or disabled by setting CLIP to 0.

SEE

linea7, vro_cpyfm, vrt_cpyfm

lineaf

Flood fill area

Class: Line-A

Category: Area Filling

SYNOPSIS

```
#include <linea.h>

lineaf();

INTIN[0]=colour;     colour to search for
PTSIN[0]=x;          x co-ordinate of start point
PTSIN[1]=y;          y co-ordinate of start point
WMODE=mode;          writing mode
PATPTR=pattern;      pointer to fill pattern
PATMSK=index;        pattern count
MFILL=flag;          multi plane fill flag
CLIP=state;          clipping flag.
XMINCL=x1clip;       left edge X clipping
YMINCL=y1clip;       top edge Y clipping
XMAXCL=x2clip;       right edge X clipping
YMAXCL=y2clip;       bottom edge Y clipping
SEEDABORT=fn;        abort fill pointer
```

DESCRIPTION

lineaf is used to flood fill an area (often called seed fill), in an identical manner to v_contourfill. The x and y parameters of v_countourfill are passed in PTSIN(0) and PTSIN(0), whilst the boundary colour is passed in INTIN(0).

The PATPTR, PATMSK, MFILL and WMODE parameters are as described under linea4. Note that the COLBIT values are not passed to this function, instead the current workstation fill colour attribute is used, hence this function must always be used with an open workstation.

A clipping rectangle *must* be specified with this function, it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). Note that the clipping flag CLIP is ignored, clipping is always performed.

SEEDABORT is a function called after plotting every line, and is used to abort the fill. If the function called returns 0 the flood fill continues, otherwise it is aborted.

SEE

linea4, v_countourfill

CAVEATS

This function does not evaluate the COLBIT values for its drawing colour and the colour used is that of the current workstation, hence a workstation must be opened by the application. The function is still however of great use as it allows an abort function to be specified so that a user may abort an incorrect or 'leaking' fill.

EXAMPLE

```
/*
 * draw a circle on screen and then seed-fill it
 */
#include <linea.h>
#include <vdi.h>
#include <aes.h>

short __saveds sab(void)
{
  return V_MOUSE_BT;  /* stop when a button pressed */
}

int main(void)
{
  short v_handle,junk;
  short pattern[]={
    0x0940,    /* 0000100101000000 */
    0x0f40,    /* 0000111101000000 */
    0x0940,    /* 0000100101000000 */
    0x64dc,    /* 0110010011011100 */
    0x8a88,    /* 1000101010001000 */
    0xcac8,    /* 1100101011001000 */
    0x2a88,    /* 0010101010001000 */
    0xa488,    /* 1100010010001000 */
  };

  appl_init();
  v_handle=graf_handle(&junk,&junk,&junk,&junk);
  linea0();
  hidemouse();
  vs_clip(v_handle,0,NULL);
  vswr_mode(v_handle,MD_REPLACE);
  vsf_color(v_handle,BLACK);
  vsf_interior(v_handle,FIS_HOLLOW);
  vsf_perimeter(v_handle,1);
  v_circle(v_handle,V_X_MAX/2,V_Y_MAX/2,V_Y_MAX/2);
  PTSIN[0]=V_X_MAX/2;
  PTSIN[1]=V_Y_MAX/2;
  INTIN[0]=-1;
  XMINCL=YMINCL=0;
  XMAXCL=V_X_MAX;
  YMAXCL=V_Y_MAX;
  SEEDABORT=sab;
  WMODE=MD_REPLACE;
  PATPTR=pattern;
  PATMSK=sizeof(pattern)/sizeof(short)-1;
  MFILL=0;
  lineaf();
  showmouse(1);
  return appl_exit();
}
```

Index