COLOR MEMORY MAPPED SCREEN GIOS SKELETON CONFIGURATION GUIDE For CP/M-86 & CCP/M-86

Digital Research

NOVEMBER 9,1983

Contents

WHO IS THIS FOR

This document describes how to implement a color screen GRAPHICS INPUT OUTPUT SYSTEM (gios). The hardware addressed by this skeleton must have the following characteristics. This skeleton has only been implemented on one target device and as such is not as well tested as the monochrome skelton. Note that it has been run through the entire test suite and has run will all DRI gsx based applications. An oem who feels that this skeleton will work on their hardware or that it is close, should contact DRI and we will consider implementing it so as to firm up this skeleton. The kinds of areas not yet addressed in this skeleton are, either two or four bit planes and oems without hardware video lookup tables.

Processor	8088,8086,80186
Display	three bits/pixel bit map addressable by processor.
	processor may access bit map as though it were normal memory. (wait states may be inserted).

The display must use a byte as eight contiguous pixels.

A scan line must be made up of a contiguous sequence of bytes. (bytes may be swapped within words)

The bit planes must be separate sections of memory.

WHAT TOOLS ARE REQUIRED

To build your gios you will require the following tools:

Assembler	Rasm 86 from DRI
Linker	Link 86 from DRI
CP/M 86 Debugger	DDT 86 from DRI
Skeleton	COLMMSK.A86 (you modify)
	COLMMRE.A86
	COLMMOB1.OBJ
	FONT.OBJ
Text editor	<your choice>

HOW TO MODIFY THE SKELETON

In order to simplify the process of building and testing the skeleton, we will do it in several stages.

STAGE 1

Upon completion of this stage your driver will have all it's graphic output primitives and their attribute routines functional. You will need to edit the file `COLMMSK.A86` this is the section which is required to be customized. At the end of the section I will discuss how to modify the monochrome device specific section to be used with the color skeleton. The cp/m version of the monochrome device specific section is particularly flexible in configuring the alphanumeric escape functions. The first task is to modify the equates for your particular hardware. We will now describe the equates in detail and define what value you must place in each.

xresmx

This is the displayable x resolution of your screen. This equate is used to inform the applications programmer how many addressable points in the x axis are available. If you should have more x axis information than can be displayed, then this value should be only that which is displayed.
EXAMPLE: If you have a screen with 640 pixels in the x direction the number returned here is 639.

yresmx

This is the displayable y resolution of your screen. This equate is used to inform the applications programmer how many addressable points in the y axis are available. If you should have more y axis information than can be displayed, then this value should be only that which is displayed.
EXAMPLE: If you have a screen with 480 pixels in the y direction the number returned here is 479.

xsize

This number is the size of the pixel in microns in the x axis. An empirical way to determine the correct value is to start with xsize = 50. Then to draw a box on the screen once your driver is functional useing an equal number of x,y increments. Measure the size of the box and it's ratio is the size in microns in x to y.

ysize

This number is the size of the pixel in microns in the y axis.

graph_plane_1

This is the segment address of the start of the first bit plane.

graph_plane_2

This is the segment address of the start of the second bit plane.

graph_plane_3

This is the segment address of the start of the third bit plane.

plane_size

This is the size of one of the graphics bit planes. It is used only if the scan lines in the bit plane are not sequential. EXAMPLE The IBM P.C. uses 16k or 4000h of memory for it's graphics. It also breaks this memory into two 8k sections with even and odd scan lines coming from alternate sections (segments). Thus the IBM has a plane size of 4000h.

msb_first

This is set to either true or false as required. If the most significant bit of a byte in the bit map is the leftmost on the screen this should be set to TRUE else it is set to FALSE.

byte_swap

This is set to either true or false as required. If leftmost byte of a scan line has an even address then this is FALSE else it is TRUE.

multiseg

This is set to either true or false as required. If sequential scan lines on the screen come from sequential memory then this is FALSE else it is TRUE. EXAMPLE The IBM PC requires a TRUE due to the fact that even and odd scan lines come from memory 2000h bytes apart. And is thus not sequential.

num_segs

If multiseg is true this is the number of segments into which the bitmap is broken. Else it is set to 1.

bytes_line
This is the number of bytes per scan line in the bitmap. On some machines this is different than the displayed number of bytes per line. It is used to calculate how far to the next scan line in memory.

chars_line
This is the number of ascii character columns on the screen when in alphanumeric mode. For an 80 x 24 display this would be an 80.

lines_page
This is the number of ascii character rows on the screen when in alphanumeric mode. For an 80 x 24 display this would be an 24.

last_escape
This is the number of the last escape function implemented. This is typically the same as last_dri_escape but if device specific escapes are added then it will be the number of the last device specific escape.

last_dri_escape
This is the last DRI escape function which is implemented. Typically this is a 19.

mouse
This returns either a true or false. If a mouse driver is to be included this should be set to TRUE else it is set to false.

curwtx
This sets 1/2 the size of the cross hair cursor in the x direction. This and curwty should form a square cursor.

curwty
This sets 1/2 the size of the cross hair cursor in the y direction. This and curwtx should form a square cursor.

status_line
This enables code which will turn off the status line in graphics mode and on in alpha mode. Some machines will require this such as the IBM P.C.

- Now four routines must be written or modified to allow the gios to be partially functional. They are each outlined below.

escfn2:
This is an escape function used to put the display into graphics mode. It is called at OPEN WORKSTATION time and whenever going into graphics mode. It should init the display to graphics mode then call clearmem. If a mouse is present it should call mouse_function to initialize the mouse.

concat:
This is used to calculate the actual physical address in memory from an x,y coordinate. The sample code given will work with little change on many systems. If the y resolution is greater than 256 then a 16 bit multiply is required and one should exercise caution since the ax,dx pairs are used as the result of this operation. The entry and exit registers used and the values to be returned in them should not be altered.

init_lut

This routine is called at open workstation and is used to initialize a hardware video lookup table. It is also responsible for seting the realized table for each color index. These values are used by the inquire color representation function. Please refer to the sample driver for sample code which performs these functions.

load_lut

This routine is called by the set color representation function. It is responsible for loading the requested color index in a hardware lookup table with the closest color to the requested value. It will then load the realized table with the realized color.

- The driver may now begin testing. To build the driver use `rasm86` to assemble the file you just edited. A file will automatically be included into this which contains many highly optimized routines (`colmmre.a86`). After obtaining no errors during assembly, link the output of this assembly with monommob.obj and font.obj to create a device driver. After setting up an assign.sys file with your driver as the only device and numbered 01. Type `GRAPHICS` then try running testgios.cmd. This is done by typing `RUN TESTGIOS`. You should recieve the first screen which is all graphics output. If problems arise debugging is done quite simply. After runing `GRAPHICS` an address is printed on the screen indicating where the driver was loaded. With `ddt86` you can look at this area and the bytes at 3 and 4 are the segment address for the code. Since your modified file was linked first it is the entry into your gios. When debugging the gios the first instuction you will see is a jump into the main body of the code (`monommob.obj`). This was done so as to simplify debugging. To breakpoint your routine make a listing of your module after it is assembled and the offset in the listing will be where you will place your breakpoint in the gios.

Using Monochrome Device Specific Sections

To use the monochrome device specific section to start with, you will need to add and remove some routines from it. You must first remove the following external declarations.

- plncol
- wrmode
- style
- txtcol
- txtclx
- bakcol

You must next modify the clear memory routine to clear all three bitplanes and not use the variable bakcol. The bitplanes should be cleared using 0's. You must next add routines called init_lut and load_lut. Init_lut is called to initialize any hardware lookup table and to initialize the color realized table. Please refer to the sample color device specific routines for an example. Load_lut is called when a set color representation command is passed to the driver. It's function is to modify the specified index in the hardware lookup table. Refer to the color device specific file for an example of this.

STAGE 2

- Upon completion of this stage your driver will have all it's alphanumeric escapes functional. Most of the escapes are table driven and should be very simple to customize. The escape table is found at the end of the file, the format is character count followed by the characters for the function. We will discuss each routine in detail each of these routines are the alphanumeric escapes discussed in the GSX Version 1.X programmers guide.

Escape	Description
escfn1:	NO MODIFICATION REQUIRED This returns the number of character rows and columns.
escfn3:	This routine initializes the display into alpha numeric mode. It is called at CLOSE WORKSTATION time, and any time the display is to be used in alpha mode. If a mouse is connected and is interrupt driven then the mouse should be de initialized by calling the subroutine mouse_function.
escfn4:	This routine moves the alpha cursor up one character row. If at top of screen no action occurs. To customize modify the data following cur_up in the data segment.
escfn5:	This routine moves the alpha cursor down one character row. If at bottom of screen no action occurs. To customize modify the data following cur_down in the data segment.
escfn6:	This routine moves the alpha cursor right one character column. If at right edge of screen no action occurs. To customize modify the data following cur_right in the data segment.
escfn7:	This routine moves the alpha cursor left one character column. If at left edge of screen no action occurs. To customize modify the data following cur_left in the data segment.

escfn8:	This routine homes the cursor to the upper left corner of the screen. To customize modify the data following home_cur in the data segment.
escfn9:	This routine erases from the current alpha cursor location to the end of screen. The cursor location is unchanged at the end of the routine. To customize modify the data following erase_to_eop in the data segment.
escf10:	This routine erases from the current alpha cursor location to the end of the line. The cursor location is unchanged at the end of the routine. To customize modify the data following erase_to_eol in the data segment.
escf11:	This routine moves the cursor to the specified x,y location. The x,y location is converted to ascii characters and is output as the cur_position string.
escf12:	This routine outputs text to the alpha numeric display. The length of the string in contained in contrl (4) and the string is in intin. Note that only one character is contained in each word of intin. Also the charac output should have the current attribute of escape functions 13,14 in effect.
escf13:	This routine turns on a reverse video attribute for all subsequent characters output through escape function 11. Note that the attribute used must be able to be set at any character location and must not take up a character location. To customize modify the data following reverse_on in the data segment.
escf14:	This routine turns off a reverse video attribute for all subsequent characters output through escape function 11. To customize modify the data following

	reverse_off in the data segment.
escf15:	This routine returns the current x,y cursor address in the array intout (1), intout (2). Note for those displays not able to return the cursor position, returning a 1,1 is acceptable.
escf16:	NO MODIFICATION REQUIRED This routine returns the status of whether a mouse, joystick, data tablet, etc is connected. This is done automatically by the true/false you placed in (mouse).
escf17:	This routine would copy the screen image to a hardcopy device. This is very device specific and has been rarely implemented. THIS ROUTINE IS NOT REQUIRED FOR ANY DRI APPLICA TIONS SOFTWARE.
escf18:	NO MODIFICATION REQUIRED This routine turns on the graphic cross hair cursor. This code is automatically working when your graphics output functions are working.
escf19:	NO MODIFICATION REQUIRED This routine turns off the graphic cross hair cursor. This code is automatically working when your graphics output functions are working.

- The driver may now be tested. You must assemble the new driver using rasm86 and link86. To test the driver RUN TESTGIOS. The second and third screens will test out the alpha numeric escapes. All functions should now work except locator, string, choice and valuator input.

STAGE 3

Upon completion of this stage your driver will be completely functional. We will discuss each routine in detail each of these routines are the input functions. Each is written from a sampled point of view. This means that whenever the routine is called it tests to see if a key is available if not it returns to the caller with the appropriate status set. The parameter passing conventions are well defined in the header to each function. The routines which need to be written or modified are described below.

get_loc_key

This routine returns deltax, deltay or character and or status informa- tion. In addition if a mouse is connected this is a

convienent spot for it's information to be return ed. There is a conditional assembly at the head of this module allowing a mouse to be connected. To facili- tate a mouse being connected this should be left intact. A mouse may be simply be added by modifying the mouse driver and setting the mouse switch to true. The get_loc_key routine returns three possible conditions. (1) no keys were pressed where al=0 is returned. (2) a key was pressed but it was not a key to be used for locator movement, it is returned in ah and al = 1. (3) a key was pressed which is to be used for locator movement. The delta x,delta y values can be looked up in a table found in the data section called loc_tbl. This table uses equates found at the front of the file to define the ammount of deltax,deltay to be returned for key presses. We suggest that shifted and unshifted arrow keys be used for locator. Shifted should move in 1 pixel increments and unshifted should move in large increments. In addition a home key should home.

valuator:

This routine returns a delta value or character and or status informa- tion. The routine returns three possible conditions. (1) no keys were pressed where al=0 is returned. (2) a key was pressed but it was not a key to be used for valuator movement, it is returned in ah and al = 1. (3) a key was pressed which is to be used for valuator movement. The delta value can be looked up in a table found in the data section called val_tbl. This table uses equates found at the front of the file to define the ammount of delta to be returned for key presses. We suggest that shifted and unshifted arrow keys be used for valuator. Shifted should move in increments of 1 and unshifted should move in large increments.

choice:

This routine returns a choice value and status information. The routine returns two possible conditions. (1) no keys were pressed where al=0 is returned. (2) a key was pressed which is to be used for a choice. The choice keys are typically function keys and can be tested arithmetically rather than by a lookup table. If the key pressed was a choice key it's number is returned in the range bx = 1 - maximum number of choice keys and al = 1. If not a valid key then al=0.

get_char:

This routine returns key and or status information. The routine returns two possible conditions.

1. no keys were pressed where al=0 is returned.
2. a key was pressed and it is returned in BX and al=1.

- # ADDING A MOUSE

To add a mouse to the gios you have completed you will modify a skeleton mouse driver. You will need to familiarize yourself with the type of mouse you wish to implement. If the mouse is connected to the computer via RS232 then you will also need to be aware of how to initialize the RS232 channel's baud rate, data bits/char, stop bits, parity and the mechanism for generating and acknowleging an interrupt from the port. The skeleton mouse driver was implemented on an IBM P.C.

- for a MOUSE SYSTEMS MOUSE. This section will describe how to modify the skeleton provided for your system to use a MOUSE SYSTEMS MOUSE. If another mouse with a different protocall than this is desired to be implemented it should not be a difficult task. DRI will be implementing other skeleton's for mice not compatible with the MOUSE SYSTEMS protocall. Check with the technical representative from DRI in your locale for the availability of other mice skeletons. We will discuss the routines which will need to be modified for you to implement the mouse on your computer.

  mouse_init:
  This routine initializes several status bytes. This code should be left intact. It next initializes the baud rate and other uart parameters. This must be modified for your particular system. The baud rate should be set to 1200, with 8 data bits, no parity and 1 stop bit. Next the code initializes the interrupt vector location. This is easily modified by seting the equate mouse_int_vector_offset to the value 4*intlvl where intlvl is the level of interrupt the uart is connected to. Next the code must enable the recieve interrupt and if required the interrupt controller may require initialization of the interrupt mask.

  mouse_deinit:
  This routine disables the interrupt from the mouse. It also should put the old interrupt vector back in memory if the operating system is expecting to recieve interrupts from that channel. This is very implementation specific and the only thing that MUST be done is to disable the interrupts on the chanel so that when other gios's are loaded no spurious interrupts are generated.

  mouse_int_vector:
  This is the interrupt time handler for the mouse. There are two area`s which may need to be modified. The first is how you read the byte from the communications port this is typically done by a simple IN instruction with the port being equated to recieve_port. The second area where modification may be required is at mouse_int_exit where interrupts from the comm port should be ensured to be reenabled and the interrupt controller may require and end of interrupt command.

  mouse
  This equate must be set to TRUE for the mouse to be functional. The mouse driver will now be included into your skeleton driver at assembly time. The number of locator devices will automatically be updated now that you have your mouse installed.