Behavior Research Methods, Instruments, & Computers
1991, 23 (3), 430-436

Independent control of dual video subsystems on the IBM PC and compatibles

KEVIN J. HAWLEY
University of Utah, Salt Lake City, Utah

Two methods for generating video output on multiple video monitors are described. The first method involves splitting the signal from a single video adaptor card so that multiple video monitors may be attached. Although this method is simple and relatively cost-effective, it is limited to adaptors that produce only digital video signals, thus precluding the use of VGA systems and composite displays. The second method involves the installation and programming of dual video adaptors: Two sample programs, which control a secondary adaptor by means of either BIOS routines or direct commands via C code, are described. Although more complex, this second method allows output to each display to be controlled independently. Furthermore, output to one screen may consist of graphics information while output to the second screen consists of text. Together, both methods can be used to create an experimental system composed of multiple data-collection stations and an independent experimenter console.

With most experimental generation and data-collection programs for the IBM PC and compatibles, a single video subsystem is employed for the presentation of visual stimuli (e.g., CEDATS—Eamon, 1982; APT PC—Poltrock & Foltz, 1988; MEL—Schneider, 1988).[1] In general, this limitation precludes the use of the video monitor for other tasks such as the output of diagnostic messages to the experimenter. During the course of an experimental session, however, it is often desirable to monitor the progress of a subject through the experimental task. Current methods for monitoring a subject's performance in these situations range from the installation of microphones, videocameras, or one-way mirrors in subject testing areas to the stationing of an experimental assistant either behind or beside the subject during the experimental session. These methods are awkward and imprecise at best, and they may bias a subject's responses. An alternative solution involves the use of multiple video monitors with the experimental system. Although multiple-display capability is built in to the Macintosh II series of computers, this capability can also be achieved on IBM PC computers and fully compatible clones. In this paper, I will describe two methods for using a single IBM PC to control multiple video outputs.

The first method for achieving video output to more than one video monitor involves splitting the output signal from the video adaptor. This allows more than one video monitor to be connected to a single video adaptor card, thus producing an exact replica of the information being displayed to a subject. The advantage of this method is that it is simple (signal splitters, or Y connectors, can be purchased or assembled from components available at most electronics shops) and relatively cost-effective (only the additional monitor need be purchased). Furthermore, several monitors can potentially be driven by a single adaptor card, allowing several subjects to view a stimulus display independently, with no loss of resolution.[2]

However, there are two important disadvantages to this method. First, since all of the information that appears on one screen also appears on the other, there is no way to generate custom messages or output for the experimenter that will not also be seen by the subject. Thus, although this method provides a convenient way either to display the same stimuli to multiple subjects or to simply monitor a subject's progress through an experimental task, it does not allow the system to display information that an experimenter may wish to keep from the subject's eyes. Second, the loss of signal strength that results from the attachment of multiple video monitors to a single video adaptor means that this method can only be used with adaptors that generate digital video signals. Since digitally driven monitors need only recognize the presence or absence of a video signal, the loss of signal strength that occurs when the output from the adaptor card is split does not affect video output until the signal strength drops below some minimum (e.g., 1 mV). In contrast, in video systems in which analog signals are employed, the strength of the signal is typically used to determine the color and intensity of a video character. Attenuation of these signals can lead to changes in these display characteristics. Since composite video displays, as well as VGA and MCGA adaptors, all employ analog video

This work was supported in part by a University of Utah Graduate Research Fellowship to the author and by an Air Force Office of Scientific Research grant (89-0275) to William Johnston at the University of Utah. I would like to thank Thayne Cooper of Unisys Corporation for his tutelage in understanding some of the mysteries of IBM video subsystems, as well as Bill Johnston, Jim Farnham, and two anonymous reviewers for their comments on earlier drafts of this article. Correspondence should be addressed to Kevin Hawley, Department of Psychology, University of Utah, Salt Lake City, UT 84112.

Table 1
Allowable Combinations of Video Subsystems on the IBM PC and Compatibles

	MDA	CGA	MCGA	EGA	VGA
MDA		×	×	×	×
CGA	×			×	
MCGA	×			×	×
EGA	×	×	×		
VGA	×		×		

Note—Although commercially available, the MCGA is generally employed only in the PS/2 Models 25 and 30.

signals, this method is not suitable for these types of subsystems.

A second method for achieving video output to more than one monitor involves the installation and programming of dual video adaptors. Not only does this method allow for the independent control of information displayed on the attached monitors, but the type of information presented (i.e., graphics vs. text) can be independent. Furthermore, since the video output signals generated by the installed adaptor cards are not altered en route to the monitor, the use of composite displays or VGA and MCGA cards is not precluded. In the section that follows, I will describe the logic underlying such a system and provide program listings (written in Turbo C Version 1.5) for independently controlling two video adaptors. For expository purposes, in the present paper I will focus on achieving dual video output with a monochrome display adaptor (MDA) in conjunction with a graphics adaptor (e.g., CGA, EGA, or VGA). However, in principle it is possible to combine several other types of displays within a single system by using the described techniques. A listing of feasible combinations for the most common video adaptors is provided in Table 1. Information for programming these other types of video adaptors can be found in the *Programmer's Guide to PC & PS/2 Video Systems* (Wilton, 1987).

PROGRAMMING A VIDEO ADAPTOR

Hardware Parameters and BIOS Routines

Independent output to dual video subsystems is made possible by the different I/O ports and video buffer mappings used by the MDA and standard graphics adaptors. Specifically, the MDA's port addresses range from 3B0h through 3BFh, and its video buffer (i.e., the area of computer memory where the information being displayed on the screen is actually held) is contained between B000:0000 and B000:FFFF.[3] In contrast, the I/O ports for graphics adaptors range from 3D0h through 3DFh, and their video buffers start at B800:0000 when they are used in text mode. These differences mean that program instructions can be written to the MDA without affecting the state of the graphics card. Furthermore, since there is no overlap in their video buffers, information can be written to one adaptor without its appearing on the other.

When the IBM PC is initially booted, a default video subsystem and mode are defined by internal switch settings on the motherboard. In most cases, these default settings instruct the system to use the highest available text mode on the installed adaptors (e.g., 80 × 25 full-color text on CGA, EGA, and VGA systems). A record of the default video subsystem and its display mode is stored in a global variable at 0040:0010 (called EQUIP_FLAG in IBM's technical manuals). To make use of a secondary video subsystem, one must initialize the secondary adaptor into some video mode, and one must write output destined for that subsystem to the appropriate video buffer. There are two ways to do this. The first involves using BIOS video routines to select the active video controller and its display mode (for the second way, see the next section below). An example of how to use these routines to switch video output between the MDA and CGA is provided in Listing 1. All video BIOS routines are accessed by loading their function number into register AH and then executing interrupt 10h. In Turbo C, this can be done by using either the int86( ) or int86x( ) functions.[4] Output can be switched between the MDA and a graphics card by changing the active video subsystem information stored in EQUIP_FLAG and executing the BIOS function that sets the video mode (function 0h). In Listing 1, these tasks are performed by the function SetMode( ).

The advantage to using BIOS routines to select the active video controller is that these routines tend to be highly portable; programs written for one machine will usually run on another. There are several disadvantages, however. First, BIOS routines tend to be relatively slow. As a result, the use of these routines can substantially decrease performance, particularly for programs displaying graphics images or producing animation effects. Second, changing the video mode by using function 0h clears the display buffer, thus erasing the contents of the screen, each time it is called. On EGA and VGA systems, this can be avoided by setting bit 7 in register AL before calling interrupt 10h. However, for other systems this limitation can prove unacceptable. Third, since some video adaptor cards are capable of emulating the MDA (e.g., EGA and VGA systems), the use of BIOS commands to change the video mode may not always result in a transfer of control to the alternate video card. The exact outcome tends to vary with the manufacturer of the video system. Detailed information regarding the use of BIOS routines in general, and the video functions in particular, can be found in *IBM ROM BIOS* (Duncan, 1988) and *Programmer's Guide to PC & PS/2 Video Systems* (Wilton, 1987), respectively.

Direct Programming via C Functions

The second way to initialize a secondary video adaptor and redirect output to its video buffer is to initialize and write to the adaptor directly. This method is employed in the PsyExper experimental generation system (Hawley, 1991). Two functions for accomplishing this by using an MDA are described in Listing 2. The first of these functions, InitMono( ), programs the MDA's CRT controller (CRTC) to output data in 80 × 25 monochrome text mode (in practice, this is the only mode available on the MDA,

and the result is equivalent to using the BIOS commands described in Listing 1, except that the alternate adaptor card is not disabled). This is accomplished by loading the value 29h into the MDA's Mode Control register, which is located at 3B8h. This value instructs the controller to enable the adaptor, turn on the screen, and enable the blinking attribute for displayed characters by setting bits 0, 3, and 5, respectively. The remaining commands consist of pairs of outportb( ) calls, the first of which specifies the particular CRTC register to be written to, and the second of which contains the actual value to be placed in that register. In all, 16 different registers are written to, defining the height and width of the screen, the location of the video buffer, and the timing parameters described by a single screen refresh cycle.

The second function in Listing 2, mvtomono( ), writes a string of characters to a particular *x*-,*y*- coordinate of the monochrome screen. The function works by creating a pointer to the start of the MDA's video buffer (B000:0000),[5] computing an offset into that buffer defined by the *x*-,*y*- screen coordinates, and then moving the individual characters of the string that is to be displayed to their proper locations. When used in conjunction with InitMono( ), this function allows independent output to the monochrome screen while standard Turbo C output functions are used to display information on the graphics screen.

There are several advantages to using such functions. First, since the secondary adaptor is initialized without BIOS commands, two independent adaptors can be active simultaneously. Second, since the video buffers are not cleared with every switch between displays, the contents of one screen will not be erased when information needs to be written to the second. Third, since the MDA is initialized without disturbing the state of the primary adaptor, the mode for the primary adaptor can be established and controlled independently, through the use of standard C functions. One implication of this is that output on the primary adaptor can be set to either text or graphics mode without disturbing the MDA. Examples of this are provided in Listing 3. As noted above, although the examples given here assume the use of an MDA as the secondary video subsystem, the techniques described by these functions can be used for other video subsystem combinations as well.

There are a few disadvantages to this last technique. First, since functions such as InitMono( ) are written for particular video adaptors, a programmer must either write initialization functions for all types of adaptors that may be used, or verify that the adaptors contained on the executing system can be handled by the existing functions. In the present examples, the MDA has been employed partially because it is relatively inexpensive to add as a secondary video subsystem, but also because it can only be programmed in one mode, thus ensuring its correct functioning when it is installed with a compatible primary system (see Table 1). Second, although there is no overlap in the video buffers of the MDA and other graphics cards when the latter are employed in text mode, this is not necessarily true when graphics modes are used. In general, although the CGA and MCGA subsystems retain their functional separation from the MDA in graphics mode, the same is not true of EGA and VGA subsystems. In the latter cases, the location of the video buffer is contained between A000:0000 and B800:7FFF. These locations overlap those employed by the MDA. To avoid any conflict, it is necessary to avoid writing any information destined for display on the graphics subsystem to the area of RAM used by the MDA. Since the information constituting one full graphics display rarely occupies all of video RAM, it is usually possible to segregate output to the two subsystems. In general, segregating output between the two systems means never writing to the video page defined by the address boundaries B000:0000 through B000:FFFF when an EGA or VGA system is in graphics mode. At worst, this limitation means that video paging cannot be employed when either the EGA or the VGA is set in its highest graphics mode.

Summary and Conclusions

In the present paper, I have described two general techniques for achieving multiple video outputs with a single IBM PC or compatible. The first of these methods involves splitting the output signal from a single video adaptor between two attached monitors. Although this method allows the generation of multiple displays that contain identical information, the individual displays cannot be customized. The second method involves the installation and programming of dual video adaptors. I have described two ways of programming, involving the use of either BIOS functions or direct commands to the video adaptor. Although the BIOS functions tend to be more portable, the enhanced flexibility provided by the direct programming commands seems to make them a better choice for independent control of the video output on a secondary video subsystem.

Finally, although this paper has primarily been focused on techniques for generating flexible video output to multiple monitors, it should be noted that these techniques can be combined to create a multiple-subject testing station without the need for purchasing an expensive networking system and its corresponding controller. Specifically, while splitting the output from a single adaptor can be used to create several identical subject display screens, the reprogramming of a secondary adaptor can be used to create an independent experimenter's console. Alternatively, instead of an independent experimenter's console, the second technique can be used to create an interactive testing station in which the responses of 1 subject are used to independently alter the information displayed to a 2nd subject. In short, the techniques above, either singly or in combination, provide a high degree of flexibility in the control of the output of visually presented stimuli during an experiment.

REFERENCES

DUNCAN, R. (1988). *IBM ROM BIOS*. Redmond, WA: Microsoft Press.

EAMON, D. B. (1982). CEDATS: A cognitive experimental design and testing system. *Behavior Research Methods, Instruments, & Computers*, **14**, 142-145.

HAWLEY, K. J. (1991). PsyExper: Another experimental generation system for the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **23**, 155-159.

POLTROCK, S. E., & FOLTZ, G. S. (1988). APT PC and APT II: Experiment development systems for the IBM PC and Apple II. *Behavior Research Methods, Instruments, & Computers*, **20**, 201-205.

SCHNEIDER, W. (1988). Micro Experimental Laboratory: An integrated system for IBM PC compatibles. *Behavior Research Methods, Instruments, & Computers*, **20**, 206-217.

WILTON, R. (1987). *Programmer's guide to PC & PS/2 video systems*. Redmond, WA: Microsoft Press.

NOTES

1. All assessments of commercially available packages were made on the basis of the information contained in the published summaries of these systems. Although the author is unaware of any updates to these systems along the lines discussed in the present paper, the interested reader should refer to the manuals for these packages for more information about their capabilities and limitations.

2. The exact number of monitors that may be attached to a single adaptor card varies with the strength of the signal generated by the card and the minimal signal strength required by the monitor. As a rule, although most video adaptors generate signals of about 1 V (peak to peak), video monitors will typically respond to much weaker signals (e.g., 1 mV). Thus, it is usually possible to drive at least two monitors with most commercially available adaptor cards and monitors. It is important to note that video signals may be substantially attenuated when they are transmitted over long distances (i.e., over video cables longer than 40-50 ft). With an adaptor card generating a 1-V signal and a monitor requiring a 1-mV signal, the maximum cable length at which the monitor will still respond is approximately 140 ft. However, when this signal is split, this distance drops to approximately 70 ft; with 4 monitors, it is 35 ft, and so forth. Since most commercial vendors do not sell cables longer than 12 ft, the loss of signal strength due to cable resistance is usually negligible. Nevertheless, when one uses monitors that require large peak signals, signal degradation due to cable length may become a significant factor.

3. All port addresses and input values are given in hexidecimal (base 16) notation. All RAM addresses are specified as a segment:offset number.

4. Although all of the functions described in this paper are present in Turbo C Version 1.5, one may have to substitute function names if one uses a different compiler. For example, the Turbo C outportb( ) function (which writes a single byte to an I/O port) would have to be changed to outp( ) under Microsoft C.

5. Care must be taken to ensure that the starting address defined in the variable *scnptr* agrees with the starting buffer addresses defined in registers 0Ch and 0Dh in the function InitMono( ). Mismatches between these two references will cause the characters of the string that is to be displayed to appear at a location other than those defined by the *x*,*y* screen coordinates.

Listing 1

```
/* code to toggle video output between MDA  */
/* and CGA using BIOS video routines        */

#define CGAmode 0x03           /* mode # for 80 x 25 full color text  */
#define MDAmode 0x07           /* mode # for 80 x 25 monochrome text  */
#define CGAbits 0x20           /* sets bit 5 of Equip_Flag 80x25 color*/
#define MDAbits 0x30           /* sets bits 4 & 5 - monochrome        */
#define ClrBits 0xcf           /* clears bits 4&5 before setting      */
#define Equip_Flag 0x00400010  /* location of Equip_Flag variable     */

#include <dos.h>
#include <stdio.h>

main()
{
  /* direct video output to the MDA  */
  SetMode(MDAmode);
  printf("This should appear on the monochrome screen\n");

  /* direct video output to the CGA  */
  SetMode(CGAmode);
  printf("and this should appear on the CGA screen\n");
}

/*********************************************/

SetMode(mode)
int mode;
{
  union REGS in,out;           /* register variables used to call */
                               /* interrputs                      */
  int E_Vals,                  /* holds values from Equip_Flag    */
  *flgptr;                     /* pointer to Equip_Flag           */

  flgptr = (int *)Equip_Flag;  /* direct pointer to Equip_Flag    */
  E_Vals = *flgptr;            /* store values                    */
```

```
    /* test if bits 4 & 5 set (i.e., monochrome active) and if new  */
    /* mode is not monochrome                                       */

    if (((E_Vals & MDAbits) == MDAbits) && (mode !=7)
      {
      E_Vals = (E_Vals & ClrBits);        /* clear bits 4 & 5        */
      *flgptr = (E_Vals | CGAbits);       /* activate graphics card  */
      }
    else
      {
      E_Vals = (E_Vals & ClrBits);        /* clear bits 4 & 5        */
      *flgptr = (E_Vals | MDA bits);      /* activate monochrome card*/
      }
     in.h.al = mode;                  /* load video mode in AL       */
     in.h.ah = 0x00;                  /* load function #0h in AH     */
     int86(0x10,&in,&out);            /* execute interrupt #10h      */
   }                                  /* end SetMode()               */
```

Listing 2

```
#define scnstr 0xb0000000          /* RAM address for MDA buffer      */

InitMono()
{
  int i;                           /* loop counter                    */
  int *scnptr;                     /* pointer to video buffer         */
  scnptr = (int *) scnstr;         /* set pointer to location of video */
                                   /* RAM                             */
  outportb(0x3b8,0x29);            /* set video controller to enable  */
                                   /* the adaptor, turn on the screen, */
                                   /* and enable blinking             */
/***************    initialize MDA control registers ***************** */
  outportb(0x3b4,0x0);             /* set horizontal total            */
  outportb(0x3b5,97);
  outportb(0x3b4,0x1);             /* set horizontal displayed        */
  outportb(0x3b5,80);
  outportb(0x3b4,0x2);             /* set horizontal sync position    */
  outportb(0x3b5,82);
  outportb(0x3b4,0x3);             /* set horizontal sync width       */
  outportb(0x3b5,15);
  outportb(0x3b4,0x4);             /* set vertical total              */
  outportb(0x3b5,25);
  outportb(0x3b4,0x5);             /* set vertical total adjust       */
  outportb(0x3b5,6);
  outportb(0x3b4,0x6);             /* set vertical displayed          */
  outportb(0x3b5,25);
  outportb(0x3b4,0x7);             /* set vertical sync position      */
  outportb(0x3b5,25);
  outportb(0x3b4,0x8);             /* set interlace mode              */
  outportb(0x3b5,2);
  outportb(0x3b4,0x9);             /* set max scan line address       */
  outportb(0x3b5,13);
  outportb(0x3b4,0xa);             /* set cursor start address        */
  outportb(0x3b5,11);
  outportb(0x3b4,0xb);             /* set cursor end address          */
  outportb(0x3b5,12);
  outportb(0x3b4,0xc);             /* next 2 calls set high & low     */
  outportb(0x3b5,0);               /* addresses for video buffer.     */
  outportb(0x3b4,0xd);             /* These MUST agree w/the address  */
  outportb(0x3b5,0);               /* defined in scnstr.              */
```

```
  outportb(0x3b4,0xe);          /* Last 2 calls define the size of  */
  outportb(0x3b5,0);            /* the cursor.  Here the cursor is  */
  outportb(0x3b4,0xf);          /* turned off.                      */
  outportb(0x3b5,0);

  for (i=0; i < 8*1024; i++)
    {
    *scnptr=0x720;              /* clear screen to spaces           */
    scnptr++;
    }
}                               /* end InitMono()                   */

/*********************************************************/

mvtomono(strng,row,col)
char strng[];                   /* input character string           */
int row,col;                    /* x,y screen coordinates           */
{
  int i;                        /* loop counter                     */
  int *scnptr;                  /* pointer to video buffer          */

  scnptr = (int *) scnstr;      /* set pointer to video buffer      */
  i = (row * 80) + col;         /* compute offset                   */
  scnptr = scnptr + i;          /* move pointer                     */
  for (i=0; strng[i] !=0; i++)  /* move strng to video buffer and   */
    *scnptr++=(strng[i] | 0x700); /* set attributes for each character*/
}
```

Listing 3

```
/* following programs use InitMono() and mvtomono() to display */
/* data on either a CGA or MDA screen, where the CGA is in     */
/* either text or graphics mode                                */

/* Program 1: set CGA in text mode   */

#include <stdio.h>
#include <dos.h>

main()
{
  InitMono();                   /* call only once at start of prog. */

  printf("This should appear on the color/graphics screen\n");
  mvtomono("and this should appear on the monochrome screen...",12,3);
}

/*********************************************************************/
/* Program 2: set CGA in graphics mode */

#include <stdio.h>
#include <dos.h>
#include <graphics.h>

main()
{
  int g_driver =1;              /* load CGA drivers                 */
  g_mode = 4;                   /* 640 x 200 b/w graphics mode      */
```

```
  InitMono();                    /* initialize monochrome screen     */
  initgraph(&g_driver,&g_mode);  /* set CGA in graphics mode         */

  outtextxy(3,5,"This should appear on the color/graphics screen");
  mvtomono("and this should appear on the monochrome screen...",12,3);
  closegraph();                  /* return CGA to previous mode      */
}
```

(Manuscript received January 16, 1991;
revision accepted for publication April 12, 1991.)