

```

/*****
*
*                               V D A C C . C
*
**-----**
* Task                        : Demonstrates programming of the DAC color
*                             register in 256 color graphics mode of the
*                             VGA card. This program requires the V3240CA.ASM
*                             assembly language module.
**-----**
* Author                      : MICHAEL TISCHER
* Developed on                : 01/02/91
* Last update                 : 01/14/91
**-----**
* Memory model                : SMALL
**-----**
* (MICROSOFT C)
* Compilation                 : CL /AS /c /W0 vdacc.c
*                             LINK vdacc v3240ca;
**-----**
* (BORLAND TURBO C)
* Compilation                 : Create a project file using the following:
*                             vdacc.c
*                             v3240ca.obj
**-----**
* Call                        : vdacc
**-----**
* Info                        : Message "Structure passed by value ..." in
*                             Turbo C is not an error!
*****/

```

```

#include <dos.h>
#include <stdarg.h>
#include <stdlib.h>

```

```

#include <stdio.h>
#include <conio.h>

/*-- Type declarations -----*/
typedef unsigned char BYTE;                                /* We create a BYTE */

typedef union {
    struct { BYTE Red, Green, Blue; } b;
    BYTE RGB[3];
} DACREG;
typedef DACREG DACARRAY[256];                             /* Complete DAC table */

/*-- External references to the assembler routines -----*/

extern void init320400( void );
extern void setpix( int x, int y, unsigned char pcolor);
extern BYTE getpix( int x, int y );
extern void setpage( BYTE page );
extern void showpage( BYTE page );
extern void far * getfontptr( void );

/*-- Constants -----*/

#define MAXX 319                                           /* Maximum X- and Y-coordinates */
#define MAXY 399

#define BWIDTH 10                                          /* Width of a color block in pixels */
#define BHEIGHT 20                                         /* Height of a color block in pixels */
#define SPACING 2                                          /* Distance between the blocks */
#define TOWIDTH ( 16 * BWIDTH + ( 15 * SPACING ) )      /* Total width */
#define TOHEIGHT ( 16 * BHEIGHT + ( 15 * SPACING ) )    /* Total height */

```

```

#define STARTX  ( MAXX - TOWIDTH ) / 2 /* Upper left corner of block */
#define STARTY  ( MAXY - TOHEIGHT ) / 2

/*****
*   IsVga: Determines whether a VGA card is installed.
*   -----
*   Input   : None
*   Output  : 0, if no VGA exists, otherwise < 0
*****/

BYTE IsVga( void )
{
    union REGS Regs;          /* Processor registers for Interrupt call */

    Regs.x.ax = 0x1a00;        /* Function 1AH applies to VGA only */
    int86( 0x10, &Regs, &Regs );
    return (BYTE) ( Regs.h.al == 0x1a ); /* Is the function available? */
}

/*****
*   PrintChar : Writes a character to the screen while in graphic mode.*
*   -----
*   Input   :   THECHAR = Character to be written
*               x, y     = X- and Y-coordinates of upper-left corner
*               FG        = Foreground color
*               BK        = Background color
*   Info    : Character is created in an 8x8 matrix, based on the
*               8x8 ROM font.
*****/

void PrintChar( char thechar, int x, int y, BYTE fg, BYTE bk )
{

```

```

typedef BYTE FDEF[256][8];                                /* Font definition */
typedef FDEF far *TPTR;                                    /* Pointer to font */

BYTE          i, k,                                       /* Loop counter */
              BMask;                                     /* Bit mask for character design */

static TPTR fptr = (TPTR) 0;                             /* Pointer to font in ROM */

if ( fptr == (TPTR) 0 )                                  /* Pointer to font already set? */
    fptr = getfontptr();  /* No --> Use the assembly function to load */

/*- Generate character, pixel by pixel -----*/

if ( bk == 255 )                                          /* Drawing transparent characters? */
    for ( i = 0; i < 8; ++i )  /* Yes --> Set foreground pixels only */
    {
        BMask = (*fptr)[thechar][i];  /* Get bit pattern for one line */
        for ( k = 0; k < 8; ++k, BMask <= 1 )  /* Execute columns */
            if ( BMask & 128 )  /* Pixel set? */
                setpix( x+k, y+i, fg );  /* Yes */
    }
else  /* No consider background as well */
    for ( i = 0; i < 8; ++i )  /* Execute lines */
    {
        BMask = (*fptr)[thechar][i];  /* Get bit pattern for one line */
        for ( k = 0; k < 8; ++k, BMask <= 1 )  /* Execute columns */
            setpix( x+k, y+i, ( BMask & 128 ) ? fg : bk );
    }
}

/*****
*   Line: Draws a line based on the Bresenham algorithm
*

```

```

**-----**
*   Input   : X1, Y1 = Starting coordinates (0 - ...)           *
*             X2, Y2 = Ending coordinates                       *
*             LPCOL = Color of line pixels                      *
*****/

/*-- Function for swapping two integer variables -----*/

void SwapInt( int *i1, int *i2 )
{
    int dummy;

    dummy = *i2;
    *i2   = *i1;
    *i1   = dummy;
}

/*-- Main part of function -----*/

void Line( int x1, int y1, int x2, int y2, BYTE lpcol )
{
    int d, dx, dy,
        aincr, bincr,
        xincr, yincr,
        x, y;

    if ( abs(x2-x1) < abs(y2-y1) )           /* X- or Y-axis overflow? */
    {                                           /* Check Y-axis */
        if ( y1 > y2 )                         /* y1 > y2? */
        {
            SwapInt( &x1, &x2 );               /* Yes --> Swap X1 with Y1 */
            SwapInt( &y1, &y2 );               /*      and Y1 with Y2 */
        }
    }
}

```

```

}

xincr = ( x2 > x1 ) ? 1 : -1;          /* Set X-axis increment */

dy = y2 - y1;
dx = abs( x2-x1 );
d = 2 * dx - dy;
aincr = 2 * (dx - dy);
bincr = 2 * dx;
x = x1;
y = y1;

setpix( x, y, lpcol );                  /* Set first pixel */
for (y=y1+1; y<= y2; ++y )             /* Execute line on Y-axes */
{
    if ( d >= 0 )
    {
        x += xincr;
        d += aincr;
    }
    else
        d += bincr;
    setpix(x, y, lpcol);
}
}
else                                     /* Check X-axes */
{
    if ( x1 > x2 )                       /* x1 > x2? */
    {
        SwapInt( &x1, &x2 );            /* Yes --> Swap X1 with X2 */
        SwapInt( &y1, &y2 );            /*      and Y1 with Y2 */
    }
}

```

```

yincr = ( y2 > y1 ) ? 1 : -1;          /* Set Y-axis increment */

dx = x2 - x1;
dy = abs( y2-y1 );
d  = 2 * dy - dx;
aincr = 2 * (dy - dx);
bincr = 2 * dy;
x = x1;
y = y1;

setpix(x, y, lpcol);                  /* Set first pixel */
for (x=x1+1; x<=x2; ++x )             /* Execute line on X-axes */
{
    if ( d >= 0 )
    {
        y += yincr;
        d += aincr;
    }
    else
        d += bincr;
    setpix(x, y, lpcol);
}
}

/*****
* GrfxPrintf: Displays a formatted string on the graphic screen.      *
* Input      : X, Y          = Starting coordinates (0 - ...)          *
*              FG            = Foreground color                        *
*              BK            = Background color (255 = transparent)    *
*              STRING        = String with format information           *
*****/

```

```

*          ...      = Arguments similar to printf          *
*****/

void GrafPrintf( int x, int y, BYTE fg, BYTE bk, char * string, ... )
{
    va_list parameter;          /* Parameter list for VA... Macros */
    char stngbuf[255],          /* Buffer for formatted string */
        *cp;

    va_start( parameter, string );          /* Convert parameter */
    vsprintf( stngbuf, string, parameter ); /* Format */
    for ( cp = stngbuf; *cp; ++cp, x+= 8 ) /* Formatted string */
        PrintChar( *cp, x, y, fg, bk );    /* Display using PrintChar */
}

/*****
*   GetDac: Gets the contents of a specific number of DAC registers   *
**-----**
*   Input   : FIRST = Number of first DAC register (0-255)           *
*             NUM   = Number of DAC register                         *
*             BUFP  = Pointer to the buffer, in which the contents    *
*                   of the DAC are to be loaded. It must be a DACREG *
*                   type variable or an array of this type.          *
*   Info    : The passed buffer must have three reserved for DAC     *
*             register, in which the red, green and blue parts of    *
*             each color are recorded.                                *
*****/

void GetDac( int First, int Num, void far *BufP )
{
    union REGS Regs;          /* Processor register for interrupt call */
    struct SREGS SRegs;       /* Segment register */

```



```

Regs.x.ax = 0x1017;          /* Function and sub-function no. */
Regs.x.bx = First;          /* Number of the first DAC register */
Regs.x.cx = Num;            /* Number of register to be loaded */
Regs.x.dx = FP_OFF( BufP );
SRegs.es = FP_SEG( BufP );   /* Load pointer to buffer */
int86x( 0x10, &Regs, &Regs, &SRegs ); /* Call BIOS video interrupt */
}

```

```

/*****
*   SetDac: Loads a specific number of DAC registers
**-----**
*   Input   : FIRST = Number of the first DAC register (0-255)
*             NUM    = Number of the DAC register
*             BUFP    = Pointer to the buffer, from which various
*                   DAC registers are to be loaded. It must be
*                   DACREG type variable or an array of this type.
*   Info    : See GetDac()
*****/

```

```

void SetDac( int First, int Num, void far *BufP )
{
    union REGS Regs;          /* Processor registers for interrupt call */
    struct SREGS SRegs;       /* Segment register */

    Regs.x.ax = 0x1012;       /* Function and sub-function no. */
    Regs.x.bx = First;        /* Number of the first DAC register */
    Regs.x.cx = Num;          /* Number of register to be loaded */
    Regs.x.dx = FP_OFF( BufP );
    SRegs.es = FP_SEG( BufP ); /* Load pointer to the buffer */
    int86x( 0x10, &Regs, &Regs, &SRegs ); /* Call BIOS video interrupt */
}

```

```

/*****
*   PrintDac: Displays the contents of a DAC register on the screen   *
*   and sets the color in DAC register 255                           *
**-----**
*   Input   : DREG   = DAC register                                   *
*             NO      = The number of this register                  *
*             COLOR   = Display color                                *
*****/

```

```

void PrintDac( DACREG DReg, BYTE No, BYTE Color )
{
    SetDac( 255, 1, &DReg );           /* Color in DAC register 255 */
    GrafPrintf( 60, MAXY-10, Color, 0, "DAC:%3d R:%2d G:%2d B:%2d",
               No, DReg.b.Red, DReg.b.Green, DReg.b.Blue);
}

```

```

/*****
*   Frame   : Draws a frame around one of the color boxes           *
**-----**
*   Input   : X       = X-coordinates of the color box (0-15)      *
*             Y       = Y-coordinates of the color box (0-15)      *
*             COLOR   = Color of the frame                          *
*   Info    : The line thickness is one pixel, regardless of      *
*             the distance of the color boxes.                     *
*****/

```

```

void Surround( int x, int y, BYTE Color )
{
    int sx, sy,           /* Upper-left corner of frame */
        ex, ey;          /* Lower-right corner of frame */
}

```



```

* Demo: Demonstrates the programming of the DAC register and the      *
* color model of the VGA card                                         *
**-----**
* Input      : None                                                  *
*****/

```

```

void Demo( void )
{
    int      x,  y,
             ix, jx,
             iy, jy,
             k,  f;
    char      ch;
    DACARRAY dacbuf;
    DACREG    DReg;

    /* Loop counter */
    /* Key */
    /* Array with complete DAC table */
    /* The current DAC register */

    /*-- Generate screen -----*/

    setpage( 0 );
    showpage( 0 );
    GetDac( 0, 255, dacbuf );

    /* Process page 0 */
    /* Display page 0 */
    /* Load complete DAC table */

    GrafPrintf( 10, 0, 255, 0,
                "VDACC - (c) 1991, 1992 by M. Tischer" );

    /*-- Make the block out of 16x16 color boxes -----*/

    iy = STARTY;
    jy = STARTY + BHEIGHT - 1;
    f = 0;
    for ( y = 0; y < 16; ++y )
    {
        /* Starting point on the screen */
        /* Execute the 16 block lines */
    }
}

```

```

ix = STARTX;
jx = STARTX + BWIDTH - 1;
for ( x = 0; x < 16; ++x )          /* Execute the 16 block columns */
{
    for ( k = iy; k <= jy; ++k )    /* Make block from single lines */
        Line( ix, k, jx, k, (BYTE) f );
    ix += BWIDTH + SPACING;          /* Next block right */
    jx += BWIDTH + SPACING;
    ++f;                             /* Increment color for the next block */
}
iy += BHEIGHT + SPACING;            /* Starting pos. for next */
jy += BHEIGHT + SPACING;            /* block line */
}

/*-- Read user input and respond to it -----*/

ix = 0;                             /* Start in upper-left corner with color 0 */
iy = 0;
jx = 0;
jy = 0;
k = 0;
GetDac( 0, 1, &DReg );              /* Get color 0 */
Surround( 0, 0, 15 );               /* Draw frame around color box */
PrintDac( DReg, 0, 15 );            /* and display contents */
do
{
    ch = (char) getch();              /* Read key */
    if ( ch )                         /* Not an extended key? */
        switch ( ch )                /* No, evaluate */
        {
            case 'r' : ChangeDacReg( &DReg, (BYTE) k, 0, +1 ); /* r = Red + */
                        break;

```

```

    case 'g' : ChangeDacReg( &DReg, (BYTE) k, 1, +1 ); /* g = Green +*/
                break;
    case 'b' : ChangeDacReg( &DReg, (BYTE) k, 2, +1 ); /* b = Blue +*/
                break;
    case 'R' : ChangeDacReg( &DReg, (BYTE) k, 0, -1 ); /* R = Red - */
                break;
    case 'G' : ChangeDacReg( &DReg, (BYTE) k, 1, -1 ); /* G = Green -*/
                break;
    case 'B' : ChangeDacReg( &DReg, (BYTE) k, 2, -1 ); /* B = Blue -*/
                break;
    case ' ' : { /* Space = Set original value */
                DReg = dacbuf[ k ];
                ChangeDacReg( &DReg, (BYTE) k, 1, 0 );
                break;
            }
}
else /* Extended keyboard code */
switch ( getch() )
{
    case 72 : if ( iy == 0 ) /* Cursor up */
                jy = 15;
                else
                jy = iy - 1;
                break;

    case 80 : if ( iy == 15 ) /* Cursor down */
                jy = 0;
                else
                jy = iy + 1;
                break;

    case 75 : if ( ix == 0 ) /* Cursor left */

```



```

{
    union REGS regs;

    if ( IsVga() )
    {
        /* VGA card installed? */
        /* Yes, go ahead */
        init320400();
        /* Initialize graphics mode */
        Demo();
        regs.x.ax = 0x0003;
        /* Shift into text mode */
        int86( 0x10, &regs, &regs );
    }
    else
        printf( "VDACC - (c) 1991 by MICHAEL TISCHER\n\nAttention! This "\
                "program requires a VGA card.\n\n" );
}

```