

Fast Development with DaVinci On Screen Display (OSD)

Juan Gonzales

Digital Customer Applications Team

ABSTRACT

While On Screen Display (OSD) functionality became prevalent as a cheaper alternative to using buttons/knobs to control television settings, in today's society, it seems like everyday a new gadget comes out which uses OSDs.

Imagine a video phone or set-top application for a minute; both of these applications require video overlaid with some graphics OSD, and may require some blending between video and a graphic (or OSD) plane. In order to come up with what the final screen will look like, engineers often go through several iterations of resizing video windows and rearranging graphic blocks. Each of these iterations comes at the cost of two precious resources: time and money. Now imagine you can draw your graphics in almost any open source or commercially available graphics program and quickly drop your new graphics (or OSD) into the screen to test how it would look before writing a single line of code, allowing you to quickly change your graphics and test how it looks quickly until you have it just right. You do not have to imagine anymore, DaVinci makes this a reality.

This application report will show how easy and fast it can be to test/implement your OSD ideas on Texas Instrument's DaVinci™ platform using a standard open source graphics editor (i.e. GIMP), and the power of the Linux operating system.

This application report contains a tar file that can be downloaded from <http://www.ti.com/lit/zip/SPRAAD7>.

Topic	Page
Trademarks	2
1 Background.....	2
2 Putting it All Together.....	4
3 Conclusion	4
Appendix A OSD Demo	5
Appendix B BMP to RGB565 Conversion Code	10

Trademarks

DaVinci is a trademark of Texas Instruments.

1 Background

This section provides necessary background for understanding how the DaVinci platform makes OSD development so easy and fast.

It is important to understand that Texas Instrument’s DaVinci platform consists of more than just a family of processors, such as TMS320DM6443 and TMS320DM6446; it also includes world-class development tools, software and support. A combination of these components makes DaVinci a faster development platform for efficient and compelling video (and audio) applications.

1.1 About DaVinci OSD

This section focuses on the OSD capabilities of the TMS320DM6443 and TMS320DM6446 processors as these are the DaVinci family processors that are currently available. These processors support a background window color, two video windows, two OSD windows, and a cursor window, in order of ascending priority as shown in [Figure 1](#).

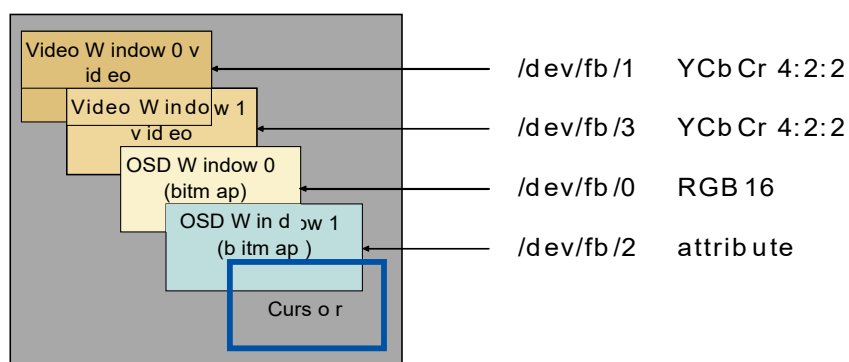


Figure 1. TMS320DM6443/TMS320DM6446 OSD Hierarchy

One unique aspect of the second OSD window (OSDWIN1) is that it can be configured as an attribute window to control the blending (i.e., transparency) between the video windows and the first OSD window (OSDWIN0). Since this “alpha” blending function is the most common use of OSDWIN1, this application report focuses on the configuration where the first OSD window (OSDWIN0) is used to display OSD graphics, and the second OSD window (OSDWIN1) is used as an attribute window to control blending.

The OSD windows are configured to accept either RGB565 or bitmap data. Often, RGB, bitmap, and raw data formats are used interchangeably in the technology industry. These formats are essentially the same if they use the same number of bits per pixel. In DaVinci processors, this is not the case, and thus they provide the option to configure OSD for either RGB565 or bitmap. When the OSD window is configured to receive bitmap data, it uses a color lookup table (CLUT) with a total of 256 entries. This means the maximum color depth of a bitmap pixel is 8-bits (4-bit, 2-bit, and 1-bit color depths are also supported). When the OSD window is configured to receive RGB565 data, CLUT is not required as RGB data from external memory is converted to YCbCr in hardware prior to reaching OSD; RGB565 uses 16-bit per pixel and therefore, can represent 64K colors. Both windows can be configured to accept bitmap data simultaneously; however, only one OSD window can be configured to accept RGB565 data at a time. Therefore, if the second OSD window is used as an attribute window as suggested above, it is preferable to use RGB565 mode on the first OSD window, primarily because it has access to 16-bits (64K colors).

1.2 About Linux, Drivers and Video

Linux has two predominant driver architectures, block drivers and character drivers. Block drivers allow out of order access and can be mounted into the file-system. These include drivers for hard disk drives, external RAM, and compact flash devices just to name a few. Character drivers are read as streams in a FIFO order (e.g. drivers for sound and video). As one can probably guess, OSD functionality is provided by character (video display) drivers.

Another feature of Linux is that character devices can be used in a similar manner to accessing files. This means you can open, read, write, and close these devices just like files. From the Linux command prompt, you can copy (cp), display (cat), and pipe (>>) data to/from another file. This powerful feature allows data to be placed into the OSD window without writing a single line of code.

The DaVinci platform provides access to the video hardware primarily using two Linux drivers, the V4L2 capture driver, and the FBDev display driver.

- **Video For Linux 2 (V4L2)** is a standard, second generation Linux video input driver, which fixed a number of design bugs of the first version.
- **FBDev** is a standard Linux video output driver used to map the frame buffer of a display device, such as DaVinci processors, into user space.

The focus of this application report is on the Linux frame buffer display device (i.e., FBDev driver) because it contains the OSD features. The frame buffer device provides an abstraction for the display hardware. It represents the video output hardware (DaVinci OSD in this case) as a frame buffer device, thus allowing application software to access graphics hardware by “just writing to a buffer”. As mentioned above, the output device buffer is accessed via a special file-like node, usually located in the /dev directory. In this case, the path you are interested in for changing the DaVinci OSD is /dev/fb/0. Using this path, you can modify the display through a well defined interface which includes file-like operations (i.e., open, read, write, close) and device specific commands (ioctl to query/set information about the hardware).

For those new to Linux, they should familiarize themselves with the following copy command below:

```
cp - copy          (e.g. >cp osd.r16 /dev/fb/0          -- copies osd.r16 file to /dev/fb/0 device)
```

1.3 About Graphics Editors

All popular graphic/image editors in the market today use the file extension to determine the proper codec required to decode a file for viewing or editing. There are many widely accepted industry file extensions such as jpg, jpeg, bmp, gif, tiff, and png, which represent their corresponding image formats. All these image formats require some header information to describe the image attributes required by that particular format. The image attributes include items such as height, width, and bits per pixel (bpp). Using the file extension, the graphics utilities can correctly parse the header file as well as the image contents.

As you have learned, TMS320DM6443/TMS320DM6446 processors support two formats of input into their OSD frame buffers with RGB being the most appealing since it provides a greater number of colors. Since RGB data represents raw uncompressed data, there is no additional header information for an image editor to decode. Unfortunately though, there are many flavors of RGB formats, such as RGB24 (8:8:8), RGB16 (5:6:5), and RGB15 (5:5:5). Without some type of header information to tell the image editor the proper format of the data, the editor can only guess. For this reason, most editors do not support RGB files, and the few that do, use “.rgb” file extension to represent only the most common RGB type, (e.g. RGB24). Since the DM6443/6 OSD frame buffer uses RGB16 (5:6:5), there is a disconnect between the hardware and the available graphic editors.

Fortunately, the uncompressed BMP file format available in most editors is essentially some header information followed by bitmap data. Furthermore, bitmap data format and RGB data format are essentially the same so long as you have the same number of bits per pixel. Therefore, it is chosen as the preferred file format for the fast OSD development procedure listed in the next section. To convert from 24-bit BMP (most common) to RGB16 (5:6:5) format -- which is expected by the OSD frame buffer -- a small C program was written and is included in the .tar.gz file accompanying this application note and is also listed in [Appendix B](#).

2 Putting it All Together

So now that you are armed with this knowledge, how do you go about placing graphics into the TMS320DM6443/TMS320DM6446 OSD window without writing a single line of code? As you learned in [Section 1.2](#), by writing a single command at the Linux prompt (`>cp osd.r16 /dev/fb/0`), you can copy the contents of `osd.r16` file into OSD frame buffer, thus displaying it. Where does `osd.r16` file come from? This file contains the image you want to display as your OSD; therefore after designing your OSD in a graphics editor such as GIMP, this file is generated by running the small conversion utility provided in the accompanying `tar.gz` file. But what format should be used to save the file created in GIMP? As you have learned, in [Section 1.3](#), the preferred format used here is BMP.

To summarize the process:

1. Create/revise OSD design in your favorite graphics editor (e.g GIMP) and save in BMP format.
2. Run the small program provided in the accompanying `tar.gz` file to convert to RGB565 format.

```
>./bmpToRgb16 mysod.bmp      (this will generate osd.r16 file)
```

3. Place the converted file into OSD frame buffer to view results

```
> cp osd.r16 /dev/fb/0
```

4. Repeat steps 1 thru 3 until you are satisfied with the final design. Save the `osd.r16` file to be loaded by application source code.

It should be mentioned that the attribute window should not be set to 100% video (all zeros), as this blending level does not allow for OSD graphics to be displayed. For an example on how to run a demo that encompasses the ideas presented in this document on the DaVinci DVEVM, please refer to [Appendix A](#).

3 Conclusion

It is relatively simple to design your OSD graphics on the DaVinci platform. Literally, draw OSD graphics in almost any open source or commercially available graphics program. Quickly drop them onto the screen to test how it looks before writing a single line of code, thus allowing you to quickly change and test until it is just right. This is just one example of how the DaVinci platform can help reduce your time to market and development costs.

Appendix A OSD Demo

The demo described here executes on the TMDXEVM6446 digital video evaluation module (DVEVM) shown in [Figure A-1](#). Although not required, this demo uses a VMware Red Hat Enterprise Linux (RHEL) virtual machine image as the Host. To avoid additional configuration steps, this setup also connects DVEVM to REHL host using router, to take advantage of router's DHCP server capabilities. The following sections cover steps on setting up and running the demo.

A.1 Setting up Demo

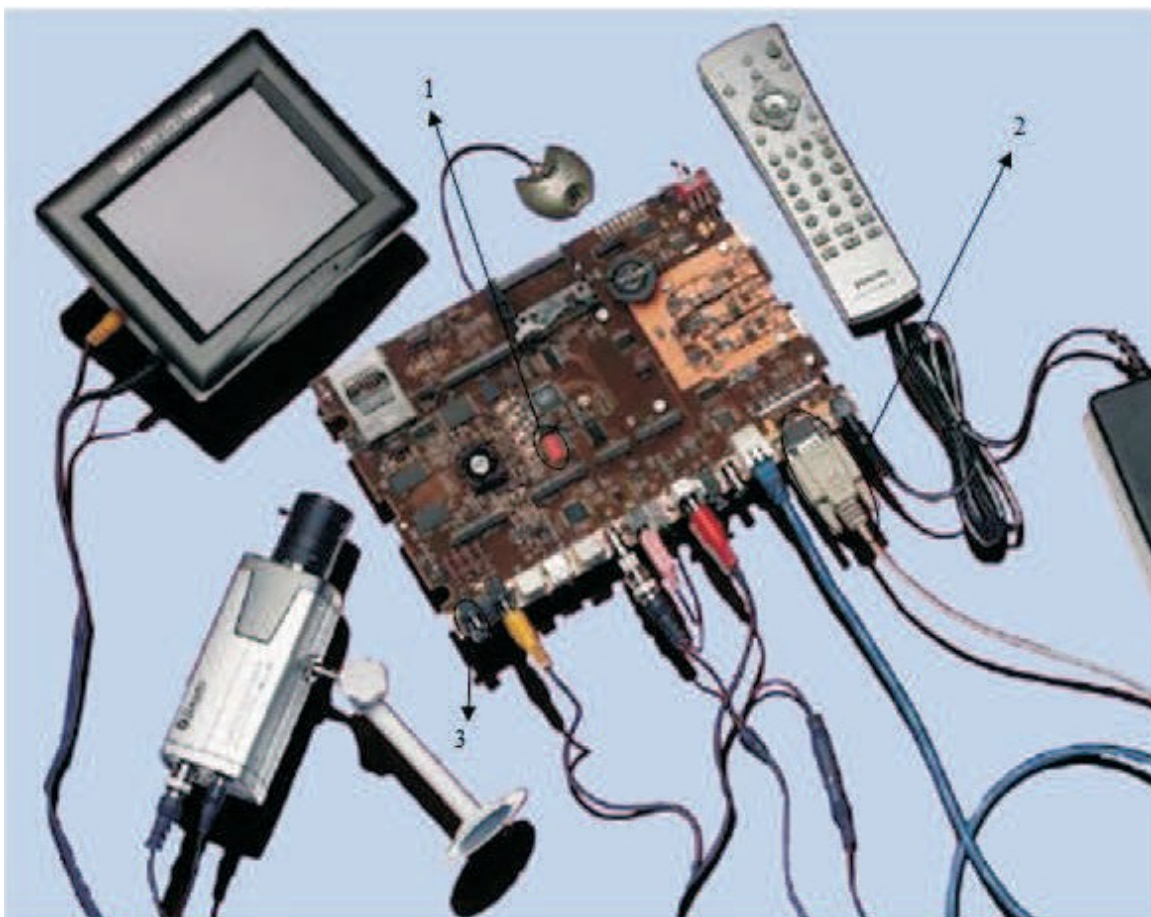


Figure A-1. TMDXEVM6446, DVEVM

1. Set the dip switches in the red block in the center of the DVEVM to the following pattern: 1011111110 (switch #2 and #10 off, all others on). See [Figure A-1](#).
2. Use a serial terminal connection to the DVEVM in 115200 baud 8—1 configuration. You may either use minicom within the VMware image by typing:

```
host # minicom
```

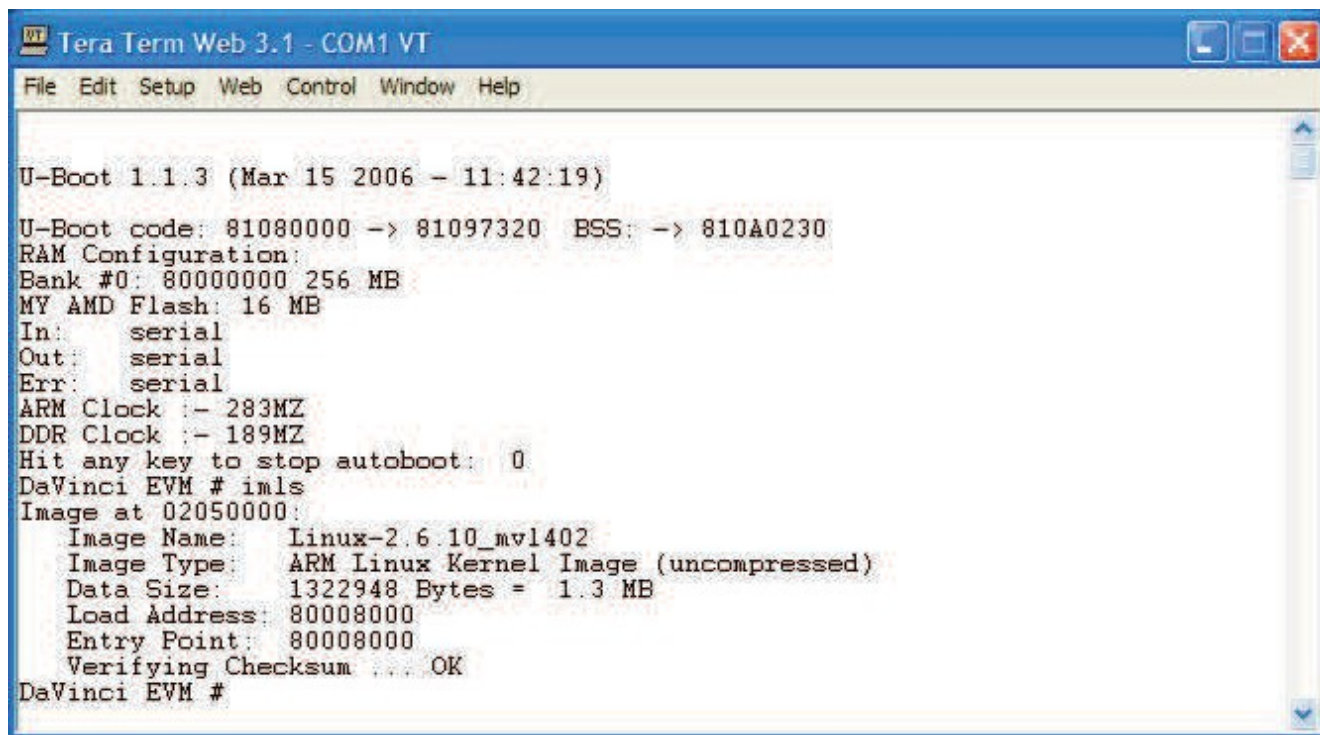
 or terminal emulator such as hyperterminal or terra term in Windows.
3. Cycle power to the DVEVM board. Press any key at the beginning of the u-boot startup sequence (while it is counting down) to interrupt the boot sequence and enter u-boot command mode. If you do not enter the u-boot command after power-cycling the board, check the dipswitch settings listed in step 1) as well as your serial terminal settings from step 2).

Setting up Demo

4. Check that there is a Linux kernel in the board's nor flash at location 0x02050000 using the 'imls' command in u-boot.

```
u-boot # imls
```

You should see output similar to the one shown in [Figure A-2](#).



```

Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help

U-Boot 1.1.3 (Mar 15 2006 - 11:42:19)

U-Boot code: 81080000 -> 81097320 BSS: -> 810A0230
RAM Configuration:
Bank #0: 80000000 256 MB
MY AMD Flash: 16 MB
In: serial
Out: serial
Err: serial
ARM Clock :- 283MZ
DDR Clock :- 189MZ
Hit any key to stop autoboot: 0
DaVinci EVM # imls
Image at 02050000:
  Image Name: Linux-2.6.10_mvl402
  Image Type: ARM Linux Kernel Image (uncompressed)
  Data Size: 1322948 Bytes = 1.3 MB
  Load Address: 80008000
  Entry Point: 80008000
  Verifying Checksum ... OK
DaVinci EVM #

```

Figure A-2. Output as a Result of Invoking u-boot 'imls' Command

5. Set **bootcmd** environment variable (see [Figure A-3](#)).

```
u-boot # setenv bootcmd bootm 0x2050000
```

6. Set **serverip** environment variable using host IP address (see [Figure A-3](#))

```
u-boot # setenv serverip 192.168.1.103.
```

Note: 192.168.1.103 should be substituted by the correct IP address of the host VMware REHL image. To get the IP address, use the following Linux command in the host terminal

```
host # /sbin/ifconfig eth0.
```

7. Set **bootargs** environment variable (see [Figure A-3](#)):

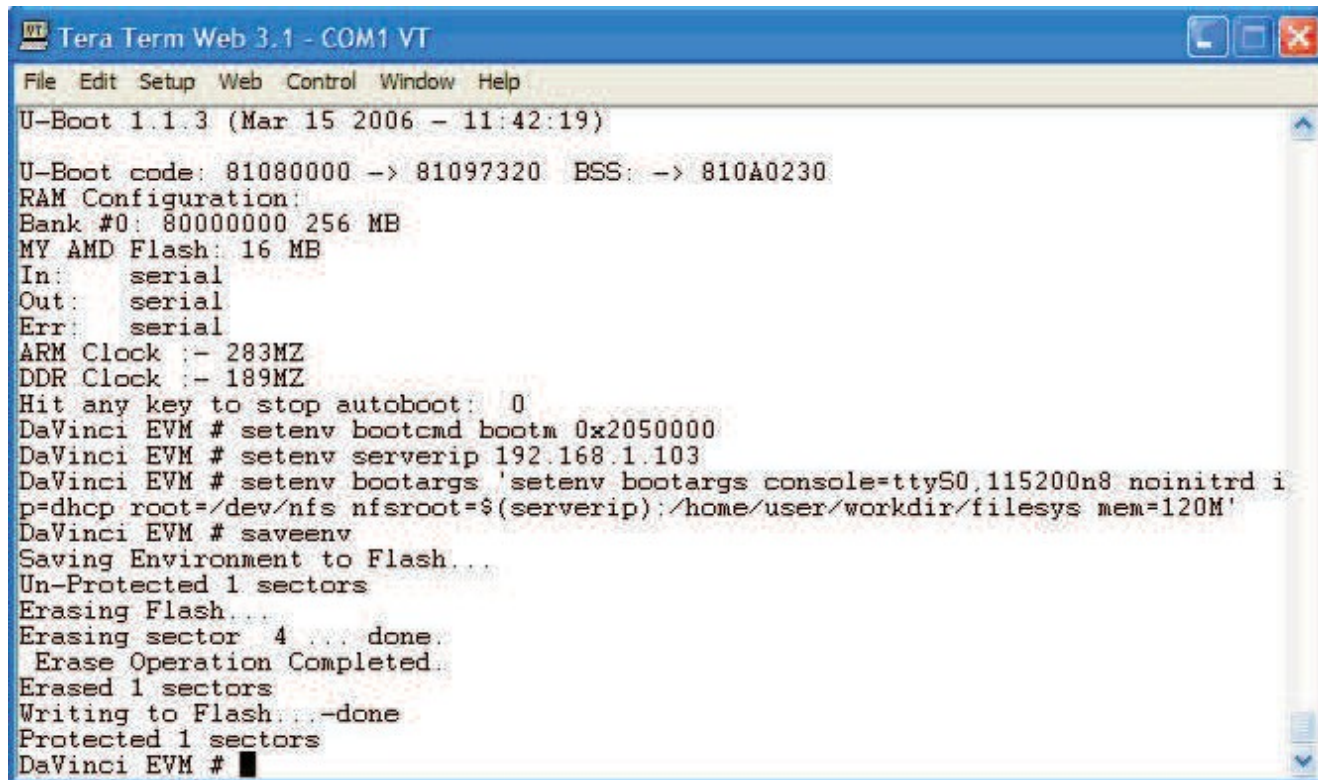
```
u-boot#setenv bootargs 'setenv bootargs console=ttyS0,115200n8
noinitrd ip=dhcp root=/dev/nfs
nfsroot=$(serverip):/home/user/workdir/filesys mem=120M'.
```

Note: The above command should be entered in one line. If you are using a PAL system, the above command should also include the following statement:

```
video=dm64xxfb:output=pal.
```

8. Save u-boot environment variables (see [Figure A-3](#))

```
u-boot # saveenv.
```



```

Tera Term Web 3.1 - COM1 VT
File Edit Setup Web Control Window Help
U-Boot 1.1.3 (Mar 15 2006 - 11:42:19)

U-Boot code: 81080000 -> 81097320 BSS: -> 810A0230
RAM Configuration:
Bank #0: 80000000 256 MB
MY AMD Flash: 16 MB
In: serial
Out: serial
Err: serial
ARM Clock :- 283MZ
DDR Clock :- 189MZ
Hit any key to stop autoboot: 0
DaVinci EVM # setenv bootcmd bootm 0x2050000
DaVinci EVM # setenv serverip 192.168.1.103
DaVinci EVM # setenv bootargs 'setenv bootargs console=ttyS0,115200n8 noinitrd i
p=dhcp root=/dev/nfs nfsroot=$(serverip):/home/user/workdir/filesys mem=120M'
DaVinci EVM # saveenv
Saving Environment to Flash...
Un-Protected 1 sectors
Erasing Flash...
Erasing sector 4 ... done.
Erase Operation Completed..
Erased 1 sectors
Writing to Flash...-done
Protected 1 sectors
DaVinci EVM # █
  
```

Figure A-3. Setting the u-boot Environment Variables

9. Reboot DVEVM and Login as "root" (there is no password).
10. On the host, place the accompanying tar.gz file under '/home/user/workdir/filesys/'. This is the directory that will be NFS mounted per u-boot configuration (see [Figure A-3](#)) and thus visible to DVEVM.
11. Extract demo files by running 'tar' Linux command
 host #: tar -xzvf demo2.tar.gz
 This will create a directory named 'demo2'.
12. Change to 'demo2' directory.
 host # cd demo2
 The contents of this directory include

-bmpToRgb16	(executable binary of BMP to RGB conversion utility)
-bmpToPgb16.c	(source code of conversion utility)
-MakeFile	(Makefile for ease of rebuilding conversion utility)
-osd.r24	(GIMP generated graphics file passed in to utility)
-osd.r16	(file generated by conversion utility)
-allosd.attr	(attribute blending data that allows only OSD visibility)
-allvid.attr	(attribute blending data that allows only video visibility)
-halfhalf.attr	(blending data that allows both video and OSD to be seen)

A.2 Running the Demo

1. On the host terminal, change to the Demo 2 directory
 host # **cd /home/user/seminar/demo2.**
2. Launch gimp (a Linux paint tool) in the host computer.
 host # **gimp.**
3. Draw a picture within gimp.
 - Press ctrl-n or select file->new... to create a new picture.
 - Select a resolution of 720x480 if you are configured for NTSC output and 720x576 if you are configured for PAL output. Keep the default type of RGB (as opposed to grayscale)
 - Press ctrl-s or clicking the > in the top left corner of the image and selecting file->save as...
 - From the “determine file type” pulldown selection, select BMP (bitmap).
 - Though not required, it will add clarity to save your image with the extension “.r24”.



Figure A-4. Drawing a Picture in GIMP

4. Convert your newly saved RGB 24 image to RGB 16 using bmpToRgb16 for instance, if you saved your image as “osd.r24”
 host # **./bmpToRgb16 osd.r24 720 480** for an NTSC system, or
 host # **./bmpToRgb16 osd.r24 720 576** for a PAL system.

Note: The output file will be saved as “osd.r16”.

5. Copy the osd.r16 file into the OSD node using the cp command:
 DVEVM # **cp osd.r16 /dev/fb/0.**

Note: If nothing is displayed, it is probably because the attribute window, which controls blending of video and OSD, is configured for 100% video. The next steps will deal with this.

6. In this step, the blending will be adjusted. If desired, a different program that displays video can be run on the background. There are demos that ship with DEVEM that can be used. This will allow user to see blending effect of OSD as well. To adjust the blending, copy one of the provided attribute window opacity files into the OSD attribute window device.

```
DVEVM # cp halfhalf.attr /dev/fb/2/
```

```
DVEVM # cp allosd.attr /dev/fb/2
```

```
DVEVM #cp allvid.attr /dev/fb/2
```

Appendix B BMP to RGB565 Conversion Code

```

#include <stdio.h>

#define RGB16(red, green, blue) ( ((red >> 3) << 11) | ((green >> 2) << 5) | (blue >> 3))
#define MAX_OSD_WIDTH 720 // Max Davinci OSD width
#define MAX_OSD_HEIGHT 576 // Max Davinci OSD height (for PAL support)
#define MAX_OSD_SIZE MAX_OSD_WIDTH*MAX_OSD_HEIGHT
#define NTSC_OSD_WIDTH 720
#define NTSC_OSD_HEIGHT 480

int main(int argc, char *argv[])
{
    short osdData[MAX_OSD_SIZE];
    FILE *rgb24file;
    FILE *rgb16file;
    char red, green, blue;
    long fileSize;
    int x, y;
    int width=NTSC_OSD_WIDTH, height=NTSC_OSD_HEIGHT; //Default values

    if (argc < 2)
    {
        printf("Usage: %s filename [width][height]\n", argv[0]);
        return -1;
    }
    else if ((argc > 2) && (argc != 4))
    {
        printf("Must specify both width and height or neither\n");
        printf("Usage: %s filename [width][height]\n", argv[0]);
        return -1;
    }
    else if ((argc == 4) && (atoi(argv[2]) > MAX_OSD_WIDTH))
    {
        printf("width cannot exceed %d pixels\n", MAX_OSD_WIDTH);
        return -1;
    }
    else if ((argc==4) && (atoi(argv[3]) > MAX_OSD_HEIGHT))
    {
        printf("height cannot exceed %d pixels\n", MAX_OSD_HEIGHT);
        return -1;
    }
    else if (argc==4)
    {
        //Valid width and Height were entered; therefore override defaults
        width = atoi(argv[2]);
        height = atoi(argv[3]);
    }

    printf("Preparing to convert %s (%d x %d)....\n", argv[1], width, height);

    // Open file in read-binary mode
    rgb24file = fopen(argv[1], "rb");

    if (rgb24file == NULL)
    {
        printf("could not find file %s \n", argv[1]);
        return -1;
    }

    // Get size of file

```

```
fseek(rgb24file, 0, SEEK_END);
fileSize = ftell(rgb24file);
fseek(rgb24file, 0, SEEK_SET);
printf("size %d\n", fileSize);

//Skip BMP header information
fseek(rgb24file, 54, SEEK_SET);
fileSize = fileSize - 54;

//Ensure file size does not exceed Max supported OSD size
if (fileSize > (MAX_OSD_SIZE*3))
{
    printf("This file is too large, maximum supported size is 720x576x3\n");
}
else if (((fileSize % 3) !=0) || (fileSize != (width*height*3)))
{
    printf("this file does not have the size expected \n");
}
else
{
    for( x=0; x < (width*height); ++x)
    {
        fread(&blue, sizeof(char), 1, rgb24file);
        fread(&green, sizeof(char), 1, rgb24file);
        fread(&red, sizeof(char), 1, rgb24file);

        osdData[x] = RGB16(red, green, blue);
    }

    // Open file in read-binary mode
    rgb16file = fopen("osd.r16", "wb");
    for (y= height -1; y >=0; --y)
    {
        for (x=0; x < width; ++x)
        {
            fwrite(&osdData[(width*y) + x], sizeof(short), 1, rgb16file);
        }
    }
    fclose(rgb16file);
}

fclose(rgb24file);
return;
}
```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2019, Texas Instruments Incorporated