How to Use the VPBE and VPFE Driver on TMS320DM643x Devices

Zhengting He

ABSTRACT
This application report describes how to use the existing video processing back end (VPBE) and video processing front end (VPFE) driver on the TMS320DM643x devices. The VPBE block is comprised of the on-screen display (OSD) and the video encoder (VENC) modules. The VPFE is comprised of five modules: charge-coupled device (CCD) controller (CCDC), resizer, preview, hardware 3A statistic generator (H3A) and histogram; the driver only covers the CCDC. Programming other modules inside the VPFE is supported by other specific drivers. For more detailed information on the drivers not described in this application report, see the device-specific documents.

The VPBE and VPFE drivers are included in the DM6437 digital video software development kit (DVSDK) which can be downloaded from www.ti.com/davincisoftwareupdates by registered customers.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/spraap3.zip

Contents
1 Introduction .................................................................................................................. 1
2 VPBE Driver Usage Example .................................................................................... 5
3 VPFE Driver Usage Example: Capturing YUV422 Data From Video Decoder .... 8
4 Package Usage Guide .............................................................................................. 9
5 References ............................................................................................................... 12

List of Figures
1 VPSS Block Diagram ............................................................................................... 2
2 VPBE Driver Structure ............................................................................................ 4
3 VPBE Example Display ........................................................................................... 5
4 Setup for Capturing YUV422 Data .......................................................................... 8

List of Tables
1 VPBE_VPFE_Examples/drivers Contents .............................................................. 9
2 VPBE_VPFE_Examples/Pal_os Contents ................................................................. 10
3 VPBE_VPFE_Examples/VPBE_example Contents ................................................ 10
4 VPBE_VPFE_Examples/VPFE_tvp5146_example Contents .................................. 11

1 Introduction
Two examples are provided with this application report. The first example demonstrates the usage of the VPBE driver. The second example demonstrates how to use the VPFE driver to capture YUV422 data from a video decoder.

The DSP/BIOS, Code Composer Studio are trademarks of Texas Instruments. Microsoft, Windows are registered trademarks of Microsoft Corporation in the United States and/or other countries.
Introduction

This document does not explain the details of the VPBE and VPFE hardware although some important concepts are explained along with the examples.

This document does not explain the details of the VPBE and VPFE driver Application Programming Interfaces (APIs). For more detailed information regarding the VPBE/VPFE driver API, see the VPBE/VPFE Driver User's Guide provided along with the driver. This application report serves as an enhancement to the VPBE/VPFE User's Guide by providing additional real usage examples. For more detailed information on the VPBE and VPFE, see the device-specific user's guides.

1.1 Introduction to VPBE and VPFE

This section explains important technique concepts of the VPBE and VPFE hardware in the TMS320DM643x devices. For more detailed information on the VPBE, see the TMS320DM643x DMP Video Processing Back End (VPBE) User’s Guide (SPRU952) [1]. For more detailed information on the VPFE, see the TMS320DM643x DMP Video Processing Front End (VPFE) User’s Guide (SPRU977) [2].

Figure 1 shows the video processing subsystem (VPSS). The VPSS provides an input interface (VPFE) for external imaging peripherals such as image sensors, video decoders, etc., and an output interface (VPBE) for display devices, such as analog SDTV displays, digital LCD panels, HDTV video encoders, etc. There is a set of common buffer memory and direct memory access (DMA) controls to ensure efficient use of the DDR2 memory controller burst bandwidth in addition to these peripherals. The shared buffer logic/memory acts as the primary source or sink to all of the VPFE and VPBE modules that are either requesting or transferring data to/from the DDR2 memory controller.

The OSD and VENC modules provide a powerful and flexible back-end display interface.

- The OSD graphic accelerator manages the display data in various formats for several types of hardware display windows; it also handles the blending of display windows into a single display frame, which the VENC module then outputs. The OSD is programmed via control and parameter registers. The OSD supports the following primary features:
  - Support for two video and two OSD windows that can be displayed simultaneously (VIDWIN0/VIDWIN1 and OSDWIN0/OSDWIN1)
  - Separate control for each window, (i.e., enable/disable), programmable window size and position, external memory address and offset registers, etc.
  - Support for 2x and 4x zoom in both the horizontal and vertical direction for each window
  - OSDWIN1 can be configured as an attribute window for OSDWIN0

Figure 1. VPSS Block Diagram
- Ability to select either field/frame mode for the windows (interlaced/progressive)
- Eight-step blending process between the OSD and video windows
- Transparency support for the OSD and video data (blending between bitmap and video only for pixels matching the background color)
- Ability to resize from video graphics adapter (VGA) to National Television System Committee (NTSC)/PAL (640x480 to 720x576) for each window
- The OSD window (either one, but not both at the same time) is capable of red, green, blue (RGB) data in 16-bit format instead of bitmap data in YCbCr format (restricted to a maximum of 8 bits).
- The OSD bitmap data width is selectable between 1 bit, 2 bits, 4 bits, or 8 bits.
- Each OSD window supports 16 entries for the bitmap (to index into a 256 entry RAM/ROM CLUT table).
- Indirect support for 24-bit RGB input data that is converted to 16-bit YCbCr video window data
- Programmable color palette with the ability to select between a RAM/ROM table with support for 256 colors
- Selectable width, height, and color of the cursor
- The display priority is: Rectangular-Cursor → OSDWIN1 → OSDWIN0 → VIDWIN1 → VIDWIN0 → background color

- The VENC converts the display frame from the OSD into the correctly formatted, desired output signals (data, clocks, sync, etc.) required to interface to the display devices. The VENC consists of three primary sub-blocks:
  - The analog video encoder that generates the signaling to interface to NTSC/PAL television displays
  - The digital LCD controller that supports interfaces to various digital LCD display formats and standard digital YUV outputs to interface to high-definition video encoders and/or DVI/HDMI interface devices
  - Timing generator to generate the specific timing required for analog video output and various digital video output modes

The VPFE is comprised of the CCDC, preview engine image pipe (IPIPE), H3A, resizer, and histogram blocks. Together, these modules provide a powerful and flexible front-end interface; the existing VPFE driver is only designed to support CCDC. Programming other VPFE modules is supported by other specific drivers that are not described in this document.

- The CCDC is responsible for accepting raw (unprocessed) image/video data from a sensor (CMOS or CCD). In addition, the CCDC can accept YUV video data in numerous formats, typically from video decoder devices. In the case of raw inputs, the CCD controller output requires additional image processing to convert the raw input image to the final processed image. This additional image processing can be done either on-the-fly in the preview engine or in the software. The CCDC is programmed via control and parameter registers. The CCDC module supports the following features:
  - Conventional Bayer pattern sensor formats
  - Generates HD/VD timing signals and field ID to an external timing generator or synchronizes to the external timing generator
  - Supports progressive and interlaced sensors (hardware support for up to two fields)
  - Supports up to 75-MHZ sensor clocks
  - Supports REC656/CCIR-656 standard (YCbCr 422 format, either 8-bit or 16-bit)
  - Supports YCbCr 422 format, (either 8 or 16 bit) with discrete H and VSYNC signals
  - Supports up to 16-bit input
  - Generates optical black clamping signals
  - Supports shutter signal control
  - Supports digital clamping and black level compensation
  - Supports 10-bit to 8-bit A-law compression
  - Supports a low-pass filter prior to writing to SDRAM. If this filter is enabled, two pixels each in the left and right edges of each line are cropped from the output.
  - Generates 8-bit to 16-bit output (8-bits wide allows for 50% saving in storage area)
  - Supports downsampling via programmable culling patterns
  - Ability to control output to the DDR2 memory controller via an external write enable signal
  - Supports up to 32K pixels (image size) in both the horizontal and vertical directions
1.2 Introduction to VPBE Driver

This section gives a brief introduction of the VPBE driver. For more detailed information regarding this driver, see the DM643x DSP/BIOS Video Processing Back End (VPBE) IOM Driver User's Guide (SPRUF56) [3] provided with the VPBE driver code.

Figure 2 shows the software structure of the VPBE driver that includes the following sub-components:

- The DSP/BIOS™ software kernel foundation is the real-time operating system which provides the runtime environment for the driver.
- IOM serves as the common DSP/BIOS driver interface to the application.
- CSLR is the register abstraction of the hardware.
- DDC is the operating system (OS) independent part of driver core.
- PALOS serves as the DSP/BIOS abstraction layer through which the VPBE driver can request the OS service.
- The driver was tested on the DM6437 evaluation module (EVM) board hardware, although it should also work on other DM643x based hardware.

The VPBE driver supports the following significant features:

- Individual channels for each video/OSD window (VIDWIN0/VIDWIN1 and OSDWIN0/OSDWIN1), VENC, and CURSOR
- Synchronous driver that operates in interrupt mode only.
- Flipping of multiple frame buffers for seamless capture and display of video
- Easy to maintain and re-target to new platforms. To port the VPBE driver to a different device, simply change the device-specific configuration parameters. To port the VPBE driver to a different OS, simply modify the PAL OS layer to adapt the target OS.
1.3 Introduction to VPFE Driver

This section gives a brief introduction of the VPFE driver. For more detailed information on this driver, see the TMS320DM643x DSP/BIOS Video Processing Front End (VPFE) IOM Driver User's Guide (SPRUF58) provided with the VPFE driver code.

The VPFE driver has the same software structure as the VPBE driver, as shown in Figure 2. The VPFE driver supports the following significant features:

- CCDC channel
- Synchronous driver that operates in interrupt mode only.
- Flipping of multiple frame buffers for seamless capture and display of video
- Easy to maintain and re-target to new platforms

2 VPBE Driver Usage Example

This section provides an example that demonstrates how to use the VPBE windows and lists some of the important features supported by the VPBE. The example displays the following images as shown in Figure 3.

The bridge video data is displayed in VIDWIN0 as background. The video data is stored as an array named BRIDGE30f in file images.lib. It consists of 30 frames of YUV422 data in 180×120 resolution. It is upscaled 4x both horizontally and vertically to the NTSC D1 resolution (720×480). Since the background image does not always require high quality, the upscaling feature allows a reduced amount of data to be stored in the memory but still keeps NTSC D1 resolution. The size of the valid data for each row is 360 bytes. Since the VPBE hardware requires the pitch to be 32-byte aligned, 24 bytes were added after each row to make the row size 384 bytes. Thus, the total size of array BRIDGE30f is $384 \times 120 \times 30 = 1,382,400$ bytes.
The first TI DSP logo in RGB888 (24 bitmap) format is displayed in VIDWIN1 at the upper left corner. The image is stored as an array named \texttt{TIDSP146x146RGB888} in files images.lib. Its size is 146×146 with the valid data for each row being 146×3 = 438 bytes. Since the VPBE hardware requires the pitch to be 32-byte aligned, 10 bytes were added after each row to make the row size 448 bytes. Thus, the total size of array TIDSP146×146RGB888 is 448×146 = 65,408 bytes.

The second TI DSP logo in RGB565 (16 bitmap) format is displayed in OSDWIN0 in the center. The image is stored as an array named \texttt{TIDSP146x146RGB565} in files images.lib. Its size is 146×146 with the valid data for each row being 146×2 = 292 bytes. Since the VPBE hardware requires the pitch to be 32-byte aligned, 28 bytes were added after each row to make the row size 320 bytes. Thus, the total size of array TIDSP146×146RGB565 is 320×146 = 46,720 bytes. The whole logo image can be partially blended with the underneath VIDWIN0 data, using the blending feature of the OSD module. It is also possible to only blend the pixels with a certain value in the logo with the underneath VIDWIN0 data. The example is programmed to allow all the pixels with the value 0xFFFF to blend with the VIDWIN0 data. In this case, the logo edge is transparently blended.

The following steps are used to configure the VPBE driver to display the data described above.

1. Configure VIDWIN0 to display the bridge video data.
   a. Create channel \texttt{vid0Handle} for VIDWIN0 by calling \texttt{FVID_create()} with configuration parameter \texttt{PSP_VPBE0sConfigParams winParams}. In \texttt{winParams}, \texttt{bitsPerPixel} and \texttt{colorFormat} is set to \texttt{PSP_VPSS\_BITS16} and \texttt{PSP\_VPBE\_YCbCr422}, respectively, so that the driver knows the input data is in YUV422 format. The window size is set to 720×480, and hScaling and vScaling are both set to \texttt{PSP\_VPBE\_ZOOM\_4X} so that the original 180×120 frame can be the upscaled NTSC D1 resolution. The pitch is set to 384 since the input video data is 32-byte aligned per row. A callback is also provided in \texttt{winParams}. Since it is called a field every time by the driver, half of an interlaced frame is sent to output, it is used to count the number of frames that have been displayed. When all 30 frames have been displayed, the example renews to display the data from the beginning.
   b. Allocate two frame buffers for VIDWIN0 channel by calling \texttt{FVID\_alloc()} twice.
   c. Copy the two beginning frames to the frame buffers
   d. Transfer the frame buffer, which contains the first bridge frame to be displayed, to VIDWIN0 by calling \texttt{FVID\_queue()}. Once VENC is configured, the frame can be displayed immediately.

2. Configure VIDWIN1 to display the TIDSP logo image in RGB888 format.
   a. Create channel \texttt{vid1Handle} for VIDWIN1 by calling \texttt{FVID\_create()} with configuration parameter \texttt{PSP\_VPBE0sConfigParams winParams}. In \texttt{winParams}, \texttt{bitsPerPixel} and \texttt{colorFormat} are set to \texttt{PSP\_VPSS\_BITS24} and \texttt{PSP\_VPBE\_RGB\_888}, respectively, so that the driver knows the input data is in 24-bitmap format. The window size is set to 146×146, and hScaling and vScaling are set to \texttt{PSP\_VPBE\_ZOOM\_IDENTITY} without zooming. The pitch is set to 448 since the input image data is made 32-byte aligned per row. Since the image does not need to be updated, no callback needs to be provided in this case.
   b. Allocate a frame buffer for VIDWIN1 channel by calling \texttt{FVID\_alloc()}.
   c. Copy the image data to the frame buffer
   d. Transfer the frame buffer to VIDWIN1 by calling \texttt{FVID\_queue()}. Once VENC is configured, the logo can be displayed immediately.
3. Configure OSDWIN0 to display the TIDSP logo image in RGB565 format.
   a. Create channel osd0Handle for OSDWIN0 by calling FVID_create() with configuration parameter 
PSP_VPBE OSDConfigParams winParams. In winParams, bitsPerPixel and colorFormat are set to 
PSP_VPSS_BITS16 and PSP_VPBE_RGB565, respectively, so that the driver knows the input 
data is in 16-bitmap format. The window size is set to 146×146, and hScaling and vScaling are set 
to PSP_VPBE_ZOOM_IDENTITY without zooming. The pitch is set to 320 since the input image 
data is made 32-byte aligned per row. Since the image does not need to be updated, no callback 
needs to be provided in this case.
   b. Three blending options are provided by the example. Depending on your input, one of the following 
configurations is enabled.
      • Option 0 (blending = PSP_VPBE_BLEND0, transparency = TRUE, and transparencyColor = 
        0xFFFF): Any pixel with value 0xFFFF is completely masked. Since the edge color of the logo 
is in value 0xFFFF (white color), the edge is not displayed in this option.
      • Option 1 (blending = PSP_VPBE_BLEND2, transparency = TRUE, and transparencyColor = 
        0xFFFF): Any pixel with value 0xFFFF is partially blended with underneath VIDWIN0 data. 
      (The blending ratio is set to 2/8 (25%). In this case, the logo edge is partially blended with VIDWIN0 
data.
      • Option 2 (blending = PSP_VPBE_BLEND4, transparency = FALSE): The whole logo image is 
        partially blended with underneath VIDWIN0 data. The blending ratio is set to 4/8 (50%).
   c. Allocate a frame buffer for OSDWIN0 channel by calling FVID_alloc().
   d. Copy the image data to the frame buffers.
   e. Transfer the frame buffer to OSDWIN0 by calling FVID_queue(). Once VENC is configured, the 
frame can be displayed immediately.
4. Configure CURSOR to display the rectangular cursor.
   a. Create channel cursorHandle for CURSOR by calling FVID_create() with configuration parameter 
PSP_VPBE CursorConfigParams cursorParams. In cursorParams, the cursor size is set to 20×20. 
The ROM CLUT table is selected by setting CursorCLUTS to PSP_VPBE_CLUTSOURCE_ROM.
5. Configure VENC module.
   a. Create channel vencHandle for VENC by calling FVID_create() with configuration parameter 
PSP_VPBE VencConfigParams vencParams. Since an external monitor with composite input is 
used for testing, displayStandard in vencParams is set to 
PSP_VPBE DISPLAY NTSC INTERLACE COMPOSITE.
6. Start the loop for a specified number of times to display all the provided video and image data. Inside 
the loop:
   a. Transfer the new frame buffer, which contains the next bridge frame to be displayed, to VIDWIN0 
by calling FVID_queue().
   b. Receive a frame buffer back from VIDWIN0 by calling FVID_dequeue(). The VPBE driver returns 
the frame buffer back if the data in the frame has been displayed. Fill the returned frame buffer with 
the next bridge frame to be displayed.
   c. Directly program the CURXP and CURYP register to make the cursor move to the next position in 
the raster scan order. This step is necessary because the current VPBE driver does not support 
dynamically changing the cursor position.
7. Once the program exits the loop, free all the channels and frame buffers.
   a. Call FVID_free() twice to free the two frame buffers for VIDWIN0. Call FVID_delete() to free the 
VIDWIN0 channel.
   b. Call FVID_free() to free the frame buffer for VIDWIN1. Call FVID_delete() to free the VIDWIN1 
channel.
   c. Call FVID_free() to free the frame buffer for OSDWIN0. Call FVID_delete() to free the OSDWIN0 
channel.
   d. Call FVID_delete() to free the CURSOR channel.
3 VPFE Driver Usage Example: Capturing YUV422 Data From Video Decoder

This section explains how to use the VPFE driver to capture the YUV422 data from a video decoder. Figure 4 shows the setup. A CCD camera is connected to the tvp5146 which is a video decoder decoding the analog composite signal into the digital YUV422 data. The YUV422 data is sent to the DM6437 processor through the 8-bit data bus. Between the tvp5146 and the DM6437, there is an inter-integrated circuit (I2C) bus through which the DM6437 acts as a master to configure the tvp5146.

The provided example demonstrates how to use the VPFE driver to configure the tvp5146 decoder and capture the video input data. The captured data is also displayed on the monitor using the VPBE driver. The CCD camera outputs data in NTSC D1 resolution. However, the CCDC module in the VPFE has enough flexibility to capture any frame size no larger than NTSC D1.

The following steps are used to configure the VPFE driver to capture and display the data.

1. Create `ccdcHandle` as the CCDC channel by calling `FVID_create()` and passing parameter `ccdcParams`. In `ccdcParams`, `dataFlow` is set to `PSP_VPFE_CCDC_YCBCR_8`, `ffMode` is set to `PSP_VPSS_FRAME_MODE`, `height` and `width` are set to 480 and 720, respectively, so that the CCDC module is configured to capture the interlaced NTSC D1 frame. The following three functions related to `tvp5146` decoder are also passed to the VPFE driver in `ccdcParams`.
   a. `PSP_VPFE_TVP5146_Open()` is called when the CCDC channel is created to initialize the I2C interface between the DM6437 and the `tvp5146` decoder.
   b. `PSP_VPFE_TVP5146_Close()` is called when the CCDC channel is deleted to de-initialize the I2C interface.
   c. `PSP_VPFE_TVP5146_Control()` is called when a specific control command to configure the setting of the `tvp5146` is issued.

2. Configure the `tvp5146` by calling `FVID_control()` and passing parameter `tvp5146Params`. The `tvp5146` decoder is configured to capture composite signal in auto mode. Since most configuration parameters of the `tvp5146` are set to optimal for typical usage by default, this is probably the only place you are required to configure the `tvp5146`.

3. Allocate two frame buffers for the CCDC module by calling `FVID Alloc()`. Enqueue the buffers to the driver by calling `FVID_queue()`.  

4. Create `vid0Handle` as the VIDWIN0 channel in VPBE by calling `VID_create()` and passing parameter `vid0Params` which configures the VIDWIN0 to display the captured data in YUV422 format and NTSC D1 resolution.

5. Allocate two frame buffers for VIDWIN0 by calling `FVID Alloc()`. Enqueue the buffers to the driver by calling `FVID_queue()`.  

6. Create `vencHandle` as the VENC channel in VPBE by calling `VID_create()` and passing parameter `vencParams` which configures the VENC module to display composite signals.

7. Start the loop for a specified number of times to capture and display the input video signal. Inside the loop:
   a. Get a captured video frame and also transfer an empty frame buffer to the CCDC channel by calling `FVID_exchange()`. The function returns after the CCDC module has captured a new frame.
   b. Transfer the frame buffer that contains the captured data to VIDWIN0 and receive a frame buffer back by calling `FVID_exchange()`. The function call returns after the data in the frame is displayed.
8. Once the program exits the loop, free all the channels and frame buffers.
   a. Call FVID_free() twice to free the two frame buffers for VIDWIN0. Call FVID_delete() to free the VIDWIN0 channel.
   b. Call FVID_free() twice to free the two frame buffers for CCDC. Call FVID_delete() to free the CCDC channel.
   c. Call FVID_delete() to free the VENC channel.

4 Package Usage Guide

This section describes how to run and re-compile the provided examples.

4.1 Hardware and Software Requirement

The following equipments and software are needed to compile and execute the provided example code.

- A PC with Microsoft® Windows® XP pre-installed
- Texas Instruments Code Composer Studio™ 3.3 software being installed on PC
- A TMS320C6437 EVM board
- A CCD camera that captures video data at NTSC D1 resolution and has composite interface
- A monitor with composite interface

4.2 Package Contents

The provided package is compressed as a zip file, spraap3.zip. Unzipping the package to the PC creates a folder called VPBE_VPFE_Examples. There are 5 sub-folders in the VPBE_VPFE_Examples: CSL_inc, drivers, Pal_os, VPBE_example, and VPFE_tvp5146_example.

CSL_inc contains the chip support library (CSL) header files of all the peripherals on the TMS320DM643x. It provides an abstraction layer for accessing all the device registers. In the VPBE example, it is used to directly program the CURXP and CURYP registers to set the CURSOR position.

drivers contains various peripheral libraries used in the provided examples. The content of sub-folder drivers is listed in Table 1.

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
</table>
| i2c\lib | • i2c_drv_bios_dbg.lib is the debug version of the I2C driver library.  
         | • i2c_drv_bios_dbg.lib is the release version of the I2C driver library. |
| i2c\inc\psp_i2cApi.h | This is the header file to define the DDC layer API of the I2C driver. The provided VPFE example uses the API to communicate to the i2cDrv API |
| previewer\lib | • prev_driv_bios_dbg.lib is the debug version of the previewer driver.  
                 | • prev_driv_bios_dbg.lib is the release version of the previewer driver.  
                 | The VPFE driver calls the previewer driver to disable the previewer when the CCDC is being initialized. |
| VPBE\lib | • vpbe_drv_bios_dbg.lib is the debug version of the VPBE driver.  
         | • vpbe_drv_bios_dbg.lib is the release version of the VPBE driver. |
| VPFE\lib | • vpfe_drv_bios_dbg.lib is the debug version of the VPFE driver.  
         | • vpfe_drv_bios_dbg.lib is the release version of the VPFE driver. |
| CSL_inc | This folder contains all the header files common to the PSP driver. |
Pal_os is the implementation of the OS abstraction layer accessed by the drivers. The content of sub-folder Pal_os is listed in Table 2.

Table 2. VPBE_VPFE_Examples/Pal_os Contents

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib\</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• palos_bios_dbg.lib is the debug version of the palos driver.</td>
</tr>
<tr>
<td></td>
<td>• palos_bios_rel.lib is the release version of the palos lib.</td>
</tr>
<tr>
<td>inc\</td>
<td>This folder contains all the header files of palos.</td>
</tr>
</tbody>
</table>

The VPBE_example folder contains the example described in Section 2. The contents are listed in Table 3.

Table 3. VPBE_VPFE_Examples/VPBE_example Contents

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug\</td>
<td>This folder contains the generated code when the example is built with the debug option.</td>
</tr>
<tr>
<td></td>
<td>• VPBEexample.out is the debug version of the executable code.</td>
</tr>
<tr>
<td></td>
<td>• VPBEexample.map is the debug version of the memory map file.</td>
</tr>
<tr>
<td>Release| This folder contains the generated code when the example is built with the release option.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• VPBEexample.out is the release version of the executable code.</td>
</tr>
<tr>
<td></td>
<td>• VPBEexample.map is the release version of the memory map file.</td>
</tr>
<tr>
<td>images\</td>
<td>images.lib is the library that contains all the image data to be displayed.</td>
</tr>
<tr>
<td>src\</td>
<td>This folder contains all the source code of the example.</td>
</tr>
<tr>
<td></td>
<td>• main.c contains the main function where the pinmux is initialized.</td>
</tr>
<tr>
<td></td>
<td>• test.c implements the tasks from the steps described in Section 2.</td>
</tr>
<tr>
<td></td>
<td>• configureCursorWin.c contains the code to configure the CURSOR window.</td>
</tr>
<tr>
<td></td>
<td>• configureOsdWin0_TIDSP_RGB565.c contains the code to configure OSDWIN0 to display the TI DSP logo in RGB565 (16 bit) format.</td>
</tr>
<tr>
<td></td>
<td>• configureVidWin0_BRIDGE_YUV422.c contains the code to configure VIDWIN0 to display the bridge image in the YUV422 format as background.</td>
</tr>
<tr>
<td></td>
<td>• configureVidWin1_TIDSP_RGB888.c contains the code to configure VIDWIN1 to display the TI DSP logo in RGB888 (24 bit) format.</td>
</tr>
<tr>
<td>VPBEexample.pjt</td>
<td>This folder contains the Code Composer Studio project file of the example.</td>
</tr>
<tr>
<td>VPBEexample.tcf</td>
<td>This file is used to configure the DSP/BIOS which is the OS running on the DM643x. dm6437_vpbe0.tci is included by VPBEexample.tcf to insert in the VPBE driver.</td>
</tr>
</tbody>
</table>
The **VPFE_tvp5146_example** folder contains the example described in **Section 3**. The contents are listed in **Table 4**.

**Table 4. VPBE_VPFE_Examples/VPFE_tvp5146_example Contents**

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug\</td>
<td>This folder contains the generated code when the example is built with the debug option.</td>
</tr>
<tr>
<td></td>
<td>• VPFEtvp5146Example.out is the debug version of the executable code.</td>
</tr>
<tr>
<td></td>
<td>• VPFEtvp5146Example.map is the debug version of the memory map file.</td>
</tr>
<tr>
<td>Release\</td>
<td>This folder contains the generated code when the example is built with the release option.</td>
</tr>
<tr>
<td></td>
<td>• VPFEtvp5146Example.out is the release version of the executable code.</td>
</tr>
<tr>
<td></td>
<td>• VPFEtvp5146Example.map is the release version of the memory map file.</td>
</tr>
<tr>
<td>src\</td>
<td>This folder contains all the source code of the example.</td>
</tr>
<tr>
<td></td>
<td>• main.c contains the main function where the pinmux is initialized.</td>
</tr>
<tr>
<td></td>
<td>• test.c implements the task realizes the steps described in <strong>Section 3</strong>.</td>
</tr>
<tr>
<td></td>
<td>• psp_i2c_interface.c and psp_i2c_interface.h implements the code to communicate to the tvp5146 using the I2C driver.</td>
</tr>
<tr>
<td></td>
<td>• i2cParams_evmdm6437.c contains the code to initialize the I2C driver.</td>
</tr>
<tr>
<td></td>
<td>• tvp5146_extDecoder.c contains the code to open, close and dynamically control the tvp5146 decoder.</td>
</tr>
<tr>
<td>VPFEtvp5146Example.pjt</td>
<td>This folder contains the Code Composer Studio project file of the example.</td>
</tr>
<tr>
<td>VPFEtvp5146Example.tcf</td>
<td>This file is used to configure the DSP/BIOS which is the OS running on the DM643x.</td>
</tr>
<tr>
<td></td>
<td>• dm6437_vpbe0.tci is included to insert the VPBE driver.</td>
</tr>
<tr>
<td></td>
<td>• dm6437_i2c0.tci is included to insert the i2c driver.</td>
</tr>
<tr>
<td></td>
<td>• dm6437_vpf0.tci is included to insert the VPFE driver.</td>
</tr>
</tbody>
</table>

### 4.3 Running the VPBE Example

This section demonstrates the steps to run VPBE examples.

#### 4.3.1 How to Run the VPBE Example

The following steps show how to run the VPBE example described in **Section 2**.

1. Connect the monitor to the DM6437 EVM board using a composite cable; the connection port on the board is **DAC D**. Power on the board.
2. Start the Code Composer Studio and open project file VPBEexample.pjt.
3. Load file **Debug/VPBEexample.out** (**Release/VPBEexample.out**) to the board to run the debug (release) version of the example. Loading a file to the board can be accomplished by clicking the Code Composer Studio menu **File → Load Program**.
4. Reset the CPU by clicking the Code Composer Studio menu **Debug → Reset CPU**.
5. Restart the program by clicking the Code Composer Studio menu **Debug → Restart**.
6. Select the **F5** key to run the program. The images that will display on the monitor are shown in **Figure 3**.
4.3.2 How to Run the VPFE Example Capturing YUV422 Data Using TVP5146
The following steps demonstrate how to run the example described in Section 3.
1. Connect the monitor to the DM6437 EVM board using a composite cable; the connection port on the board is DAC D.
2. Connect the CCD camera to the VIDEO IN port of the DM6437 board. Power on the board.
3. Start the Code Composer Studio and open project file VPFEtvp5146Example.pjt.
4. Load file Debug/VPFEtvp5146Example.out (Release/VPFEtvp5146Example.out) to the board to run the debug (release) version of the example. Loading a file to the board can be accomplished by clicking the Code Composer Studio menu File → Load Program.
5. Reset the CPU by clicking the Code Composer Studio menu Debug → Reset CPU.
6. Restart the program by clicking the Code Composer Studio menu Debug → Restart.
7. Select the F5 key to run the program. The captured video will be displayed on the monitor.

4.4 Compiling the VPBE/VPFE Examples
This section demonstrates the steps to compile VPBE/VPFE examples.

4.4.1 How to Compile the VPBE Example
The following steps demonstrate how to compile the VPBE example described in Section 2.
1. Start the Code Composer Studio and open project file VPBEexample.pjt.
2. Select Debug (Release) in the Code Composer Studio configuration window to compile the debug (release) version of the example. Click the Code Composer Studio menu Project → Build to build the code.

4.4.2 How to Compile the VPFE Example Capturing YUV422 Data Using TVP5146
The following steps demonstrate how to compile the example described in Section 3.
1. Start the Code Composer Studio and open project file VPFEtvp5146Example.pjt.
2. Select Debug (Release) in the Code Composer Studio configuration window to compile the debug (release) version of the example. Click the Code Composer Studio menu Project → Build to build the code.

5 References
1. TMS320DM643x DMP Video Processing Back End (VPBE) User's Guide (SPRU952)
3. DM643x DSP/BIOS Video Processing Back End (VPBE) IOM Driver User's Guide (SPRUF56)
4. TMS320DM643x DSP/BIOS Video Processing Front End (VPFE) IOM Driver User's Guide (SPRUF58)
IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI’s standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or “enhanced plastic.” Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer’s risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

<table>
<thead>
<tr>
<th>Products</th>
<th>Applications</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplifiers</td>
<td>Audio</td>
<td><a href="http://www.ti.com/audio">www.ti.com/audio</a></td>
</tr>
<tr>
<td>Data Converters</td>
<td>Automotive</td>
<td><a href="http://www.ti.com/automotive">www.ti.com/automotive</a></td>
</tr>
<tr>
<td>DSP</td>
<td>Broadband</td>
<td><a href="http://www.ti.com/broadband">www.ti.com/broadband</a></td>
</tr>
<tr>
<td>Interface</td>
<td>Digital Control</td>
<td><a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a></td>
</tr>
<tr>
<td>Logic</td>
<td>Military</td>
<td><a href="http://www.ti.com/military">www.ti.com/military</a></td>
</tr>
<tr>
<td>Power Mgmt</td>
<td>Optical Networking</td>
<td><a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a></td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>Security</td>
<td><a href="http://www.ti.com/security">www.ti.com/security</a></td>
</tr>
<tr>
<td>RFID</td>
<td>Telephony</td>
<td><a href="http://www.ti.com/telephony">www.ti.com/telephony</a></td>
</tr>
<tr>
<td>Low Power Wireless</td>
<td>Video &amp; Imaging</td>
<td><a href="http://www.ti.com/video">www.ti.com/video</a></td>
</tr>
<tr>
<td></td>
<td>Wireless</td>
<td><a href="http://www.ti.com/wireless">www.ti.com/wireless</a></td>
</tr>
</tbody>
</table>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2007, Texas Instruments Incorporated