Application Report
SPRAAP3A–November 2007

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.ti.com/lit/zip/SPRAAP3.

Contents

List of Figures

List of Tables

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.

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].

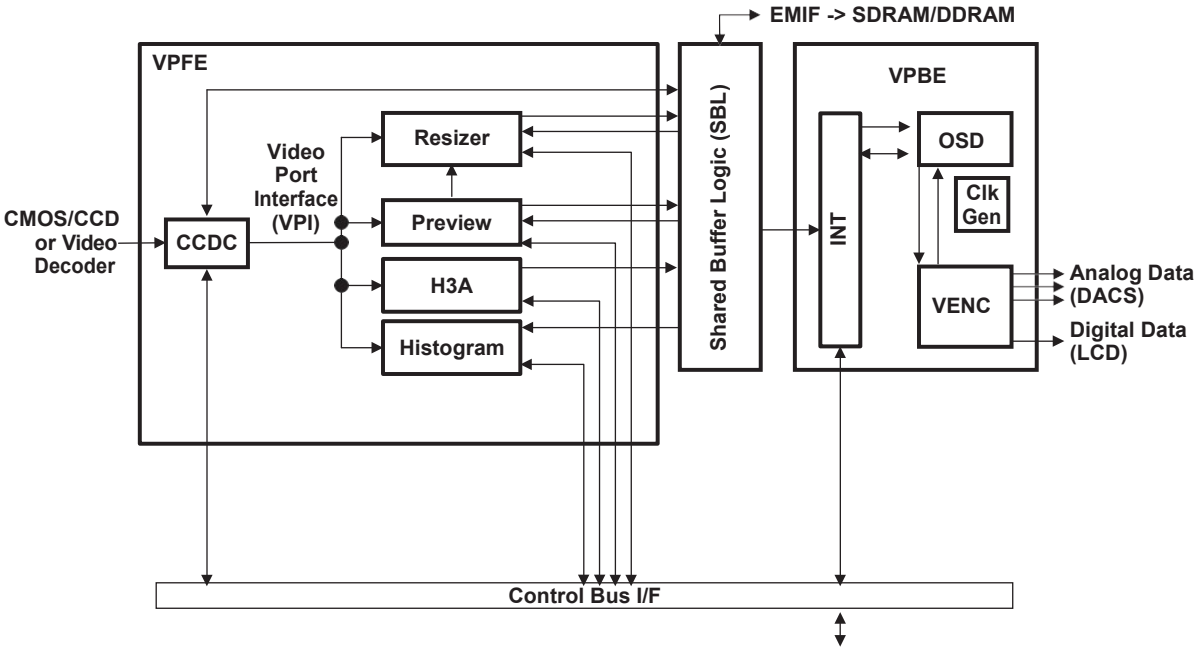

Figure 1. VPSS Block Diagram

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

  - 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 (640×480 to 720×576) 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.

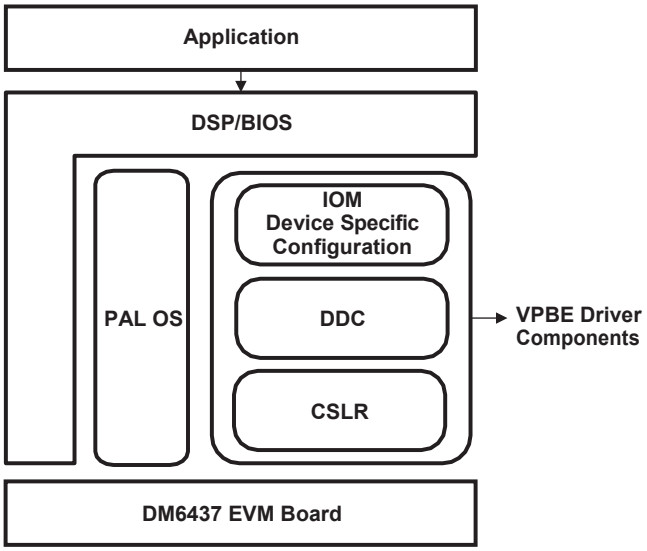

Figure 2. VPBE Driver Structure

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) [4] 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.

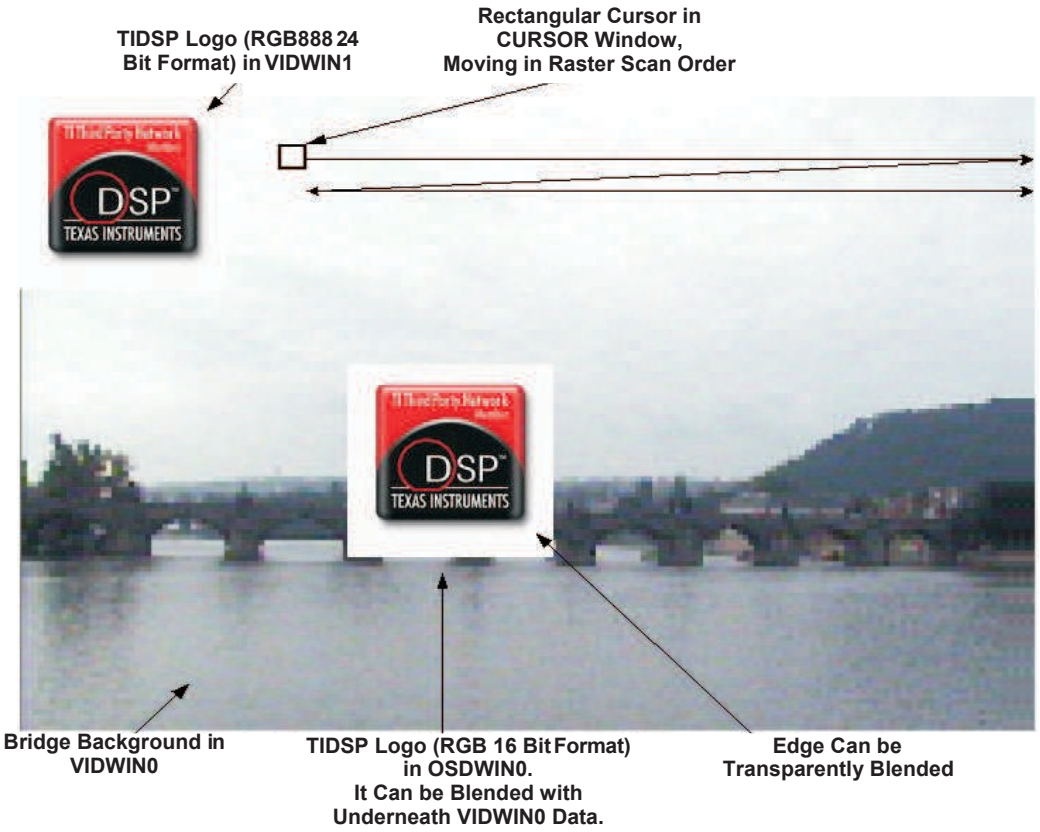

Figure 3. VPBE Example Display

- 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 384×120×30 = 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 *TIDSP146x146RGB888* in file 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 *TIDSP146x146RGB565* in file 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 0×FFFF to blend with the VIDWIN0 data. In this case, the logo edge is transparently blended.
- The rectangular cursor is programmed to display and move in the CURSOR window in raster scan order. Its size is configured to 20×20. Although the CURSOR supports programmable cursor edge, width, and color, the driver fixes the edge width to be one pixel wide and the color to be the entry zero in ROM/RAM CLUT table.

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 *vid0Handle* for VIDWIN0 by calling FVID_create() with configuration parameter *PSP_VPBEOsdConfigParams winParams*. In winParams, bitsPerPixel and colorFormat is set to PSP_VPSS_BITS16 and 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 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 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 rewinds to display the data from the beginning.
   b. Allocate two frame buffers for VIDWIN0 channel by calling 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 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 *vid1Handle* for VIDWIN1 by calling FVID_create() with configuration parameter *PSP_VPBEOsdConfigParams winParams*. In winParams, bitsPerPixel and colorFormat are set to PSP_VPSS_BITS24 and 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 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 FVID_alloc().
   c. Copy the image data to the frame buffer
   d. Transfer the frame buffer to VIDWIN1 by calling 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_VPBEOsdConfigParams 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 = 0×FFFF): Any pixel with value 0×FFFF is completely masked. Since the edge color of the logo is in value 0×FFFF (white color), the edge is not displayed in this option.
      - Option 1 (blending = PSP_VPBE_BLEND2, transparency = TRUE, and transparencyColor = 0×FFFF): Any pixel with value 0×FFFF 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_VPBECursorConfigParams 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_VPBEVencConfigParams vencParams*. Since an external monitor with composite input is used for testing, displayStandard in vencParams is set to PSP_VPBE_DISPLAY_NTSC_INTERLACED_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.

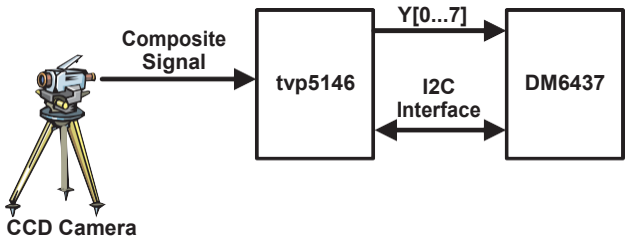

Figure 4. Setup for Capturing YUV422 Data

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 1. VPBE_VPFE_Examples/drivers Contents

Content	Description
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 tvp5146 decoder.
previewer\lib\	• prev_drv_ bios_dbg.lib is the debug version of the previewer driver. • prev_drv_ bios_rel.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_rel.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_rel.lib is the release version of the VPFE driver.
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

Content	Description
lib\	• palos_bios_dbg.lib is the debug version of the palos driver. • palos_ bios_rel.lib is the release version of the palos lib.
inc\	This folder contains all the header files of palos.

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

Content	Description
Debug\	This folder contains the generated code when the example is built with the debug option. • VPBEexample.out is the debug version of the executable code. • VPBEexample.map is the debug version of the memory map file.
Release\	This folder contains the generated code when the example is built with the release option. • VPBEexample.out is the release version of the executable code. • VPBEexample.map is the release version of the memory map file.
images\	images.lib is the library that contains all the image data to be displayed.
src\	This folder contains all the source code of the example. • main.c contains the main function where the pinmux is initialized. • test.c implements the tasks from the steps described in Section 2. • configureCursorWin.c contains the code to configure the CURSOR window. • configureOsdWin0_TIDSP_RGB565.c contains the code to configure OSDWIN0 to display the TI DSP logo in RGB565 (16 bit) format. • configureVidWin0_BRIDGE_YUV422.c contains the code to configure VIDWIN0 to display the bridge image in the YUV422 format as background. • configureVidWin1_TIDSP_RGB888.c contains the code to configure VIDWIN1 to display the TI DSP logo in RGB888 (24 bit) format.
VPBEexample.pjt	This folder contains the Code Composer Studio project file of the example.
VPBEexample.tcf	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.

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

Content	Description
Debug\	This folder contains the generated code when the example is built with the debug option. • VPFEtvp5146Example.out is the debug version of the executable code. • VPFEtvp5146Example.map is the debug version of the memory map file.
Release\	This folder contains the generated code when the example is built with the release option. • VPFEtvp5146Example.out is the release version of the executable code. • VPFEtvp5146Example.map is the release version of the memory map file.
src\	This folder contains all the source code of the example. • main.c contains the main function where the pinmux is initialized. • test.c implements the task realizes the steps described in Section 3. • psp_i2c_interface.c and psp_i2c_interface.h implements the code to communicate to the tvp5146 using the I2C driver. • i2cParams_evmdm6437.c contains the code to initialize the I2C driver. • tvp5146_extDecoder.c contains the code to open, close and dynamically control the tvp5146 decoder.
VPFEtvp5146Example.pjt	This folder contains the Code Composer Studio project file of the example.
VPFEtvp5146Example.tcf	This file is used to configure the DSP/BIOS which is the OS running on the DM643x. • *dm6437_vpbe0.tci* is included to insert the VPBE driver. • *dm6437_i2c0.tci* is included to insert the i2c driver. • *dm6437_vpfe0.tci* is included to insert the VPFE driver.

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)
2. *TMS320DM643x DMP Video Processing Front End (VPFE) User's Guide* (SPRU977).
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 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