



Videophone Development Platform Programmer Guide

Version 1.37

Wintech Digital Systems Technology Corporation
<http://www.wintechdigital.com>

Preface

Read this First

About This Manual

This Programmer's Guide aims to give you a comprehensive technical overview of the VDP embedded software with the hope that it will help you evaluate the VDP and eventually enable you to start developing your own Videophone.

How to Use This Manual

Chapter 1---Introduction
Chapter 2---Getting Started
Chapter 3---Development Platform Framework
Chapter 4---Programming
Chapter 5---API Reference
Chapter 6---Device Driver Development
Chapter 7---Flash Programming

Related Documentation

VDP Quick Start
VDP User's Guide
VDP Programmer's Guide
VDP Technical Reference

Revision Information

Tables 1: Revision Information

Date	Revision	Description
12/25/04	V1.0	Creation
02/20/05	V1.1	<p>Section 2.7, 2.8 added on.</p> <p>Move section 4.4 out to create chapter 5.</p> <p>Figure 3-5, 13-15 added on.</p> <p>Table 1 added on.</p>
11/22/05	V1.33	<p>Section 2.7, 3.3.7, 5.5, 6.3 modified.</p> <p>Figure 5, 13, 14, 15 modified.</p> <p>Table 1-5 modified</p>
04/29/06	V1.36	<p>Section 5.4 modified.</p> <p>Figure 14 modified.</p> <p>Table 1-3,5 modified</p>

Contents

READ THIS FIRST	I
<i>ABOUT THIS MANUAL.....</i>	<i>I</i>
<i>HOW TO USE THIS MANUAL</i>	<i>I</i>
<i>RELATED DOCUMENTATION FROM TEXAS INSTRUMENTS.....</i>	<i>I</i>
<i>REVISION INFORMATION.....</i>	<i>I</i>
CONTENTS	1
FIGURES	3
TABLES	4
1 INTRODUCTION.....	5
1.1 VDP APPLICATION DEVELOPMENT PLATFORM ARCHITECTURE	5
1.2 SOFTWARE ARCHITECTURE	6
1.3 GENERAL VIDEOPHONE OPERATION (CALL FLOW)	6
2 GETTING STARTED.....	8
2.1 CONNECT YOUR VDP.....	8
2.1.1 Installation Requirements	8
2.1.2 Installation Steps.....	8
2.2 INSTALLING CODE COMPOSER STUDIO	9
2.3 INSTALLING VDP DM643 CD-ROM	9
2.4 APPLYING CODE COMPOSER PATCHES.....	9
2.5 CONFIGURING THE JTAG EMULATOR	10
2.5.1 Installing Emulation Drivers.....	11
2.5.2 Import Configuration	11
2.6 STARTING CODE COMPOSER STUDIO	12
2.7 VDP DIRECTORY STRUCTURE	12
2.8 VDP SOURCE FILES	14
2.9 BUILDING THE PROJECT	17

3 DEVELOPMENT PLATFORM FRAMEWORK	18
3.1 OVERVIEW	18
3.2 ARCHITECTURE	18
3.3 TASK DESCRIPTIONS	19
3.3.1 Audio Capture.....	19
3.3.2 Audio Playback.....	21
3.3.3 Video Capture	22
3.3.4 Video Display	22
3.3.5 Video Encoder	23
3.3.6 Video Decoderk.....	25
3.3.7 Quality of Service (QoS)	27
3.3.8 Call Control and Management (CM)	27
3.3.9 Networking Task	28
3.3.10 User Interface Task.....	28
4 PROGRAMMING	28
4.1 OVERVIEW	28
4.2 SOFTWARE DIRECTORY.....	28
4.3 PROGRAMMING MODE DESCRIPTION	30
4.3.1 Introduction	30
4.3.2 Global System State Variables.....	30
4.3.3 Message	33
4.3.4 System Status Reporting.....	35
5 API REFERENCE.....	35
5.1 CALL MESSAGE TRANSFER FUNCTIONS	38
5.2 KEY MAP FUNCTION	39
5.3 MENU DESIGN FUNCTIONS	40
5.4 DIALING TONE GENERATION FUNCTIONS.....	42
5.5 CONFIGURATION	43
6 DEVICE DRIVER DEVELOPMENT	59
6.1 OVERVIEW	59
6.2 DRIVER DIRECTORY	59
6.3 KEYPAD DRIVER.....	60

6.4 AUDIO PORT DRIVER.....	67
6.5 VIDEO PORT DRIVER.....	67
6.6 ETHERNET DRIVER	69
6.7 SERIAL PORT DRIVER	69
7 FLASH PROGRAMMING	70
7.1 OVERVIEW	70
7.2 MANUALLY FLASH UPDATE	71
7.3 ON-LINE FLASH UPGRADE	71

Figures

Figure 1	Software Architecture	6
Figure 2	General Videophone Operation (call flow)	7
Figure 3	Installing Code Composer Studio	9
Figure 4	Installing emulation drivers	11
Figure 5	VDP software directory structure.....	12
Figure 6	Software architecture block diagram	19
Figure 7	Audio Capture State	20
Figure 8	Audio Playback State	21
Figure 9	Video Capture State	22
Figure 10	Video Display State.....	23
Figure 11	The video encoding task.....	24
Figure 12	The video decoding task	26
Figure 13	Project view	29
Figure 14	Structure of CFG.....	31
Figure 15	Driver Directory.....	60

Tables

Table 1	VDP source files.....	17
Table 2	CFG Structure.....	32
Table 3	Messages	35
Table 4	System Status.....	35
Table 5	APIs	38
Table 6	Audio Type	43
Table 7	Key Assignments	61
Table 8	Flash Memory Map	70

1 Introduction

This Programmer's Guide for the VDP Developer's is mainly a programming API reference guide. It describes the various API functions provided by the VDP libraries and it intends to assist the development of videophone applications.

1.1 VDP Application Development Platform Architecture

The VDP applicative development platform is an application-oriented reference design for video and voice over IP (V2oIP) applications. The VDP itself is a ready-to-use videophone, and with only some minor modifications, you can build your own videophone end products.

A general description of the call flow for the VDP based video phone is shown in Figure 2 below.

With VDP, various hardware and software are integrated and carefully tuned to provide you with a complete and readily operational V2oIP application. These include:

- A TI DM643 based media processing system
- Dial Keypad and associated MCU circuits handling user inputs
- LCD panel and associated driving circuits
- Network interface
- Video encoders and decoders: H.263, H.264
- Audio codecs: G.711U, G.723
- Acoustic Echo Canceller G.167
- A DSP/BIOS based call management stack, with compliance to H.323 protocol
- DTMF signal tone generator
- User interface software

To build your own video phone product, you could:

- Modify the user interface software and make it your style
- Modify the DTMF generation software and create your signal tones
- Adapt the LCD panel and associated driver circuits to your design
- Adapt the keypad and associated circuits to your design
- Adapt the DM643 based media processing system to your design

Carefully follow the guidelines described in this document when making your modifications. With our proven and powerful software tools, you will find the Guide quite easy and impressive.

1.2 Software Architecture

The VDP software is actually an embedded software running on the TI DM643 digital signal processor. The software uses RF-5 framework to integrate various video and audio encoders and decoders as well as a H.323 stack based on DSP/BIOS. The general architecture is shown below:

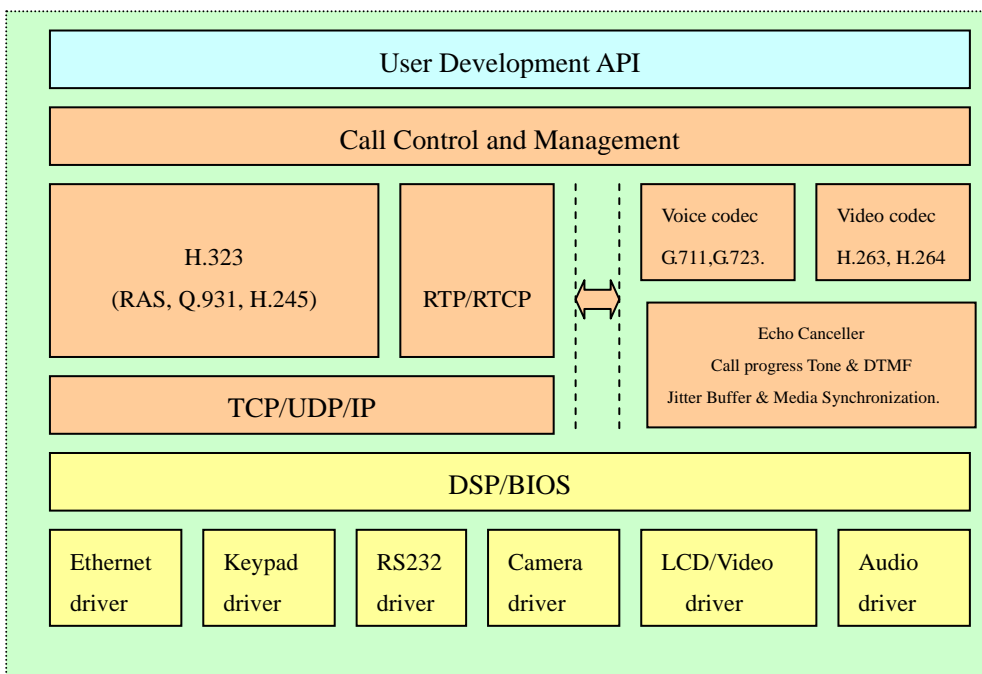


Figure 1 Software Architecture

1.3 General Videophone Operation (Call Flow)

The complete call procedure in H.323 protocol is consisted of four phrases

- Phase A Call Setup
- Phase B Initial communication and capability exchange
- Phase C Establishments of Audiovisual communication
- Phase D Call termination

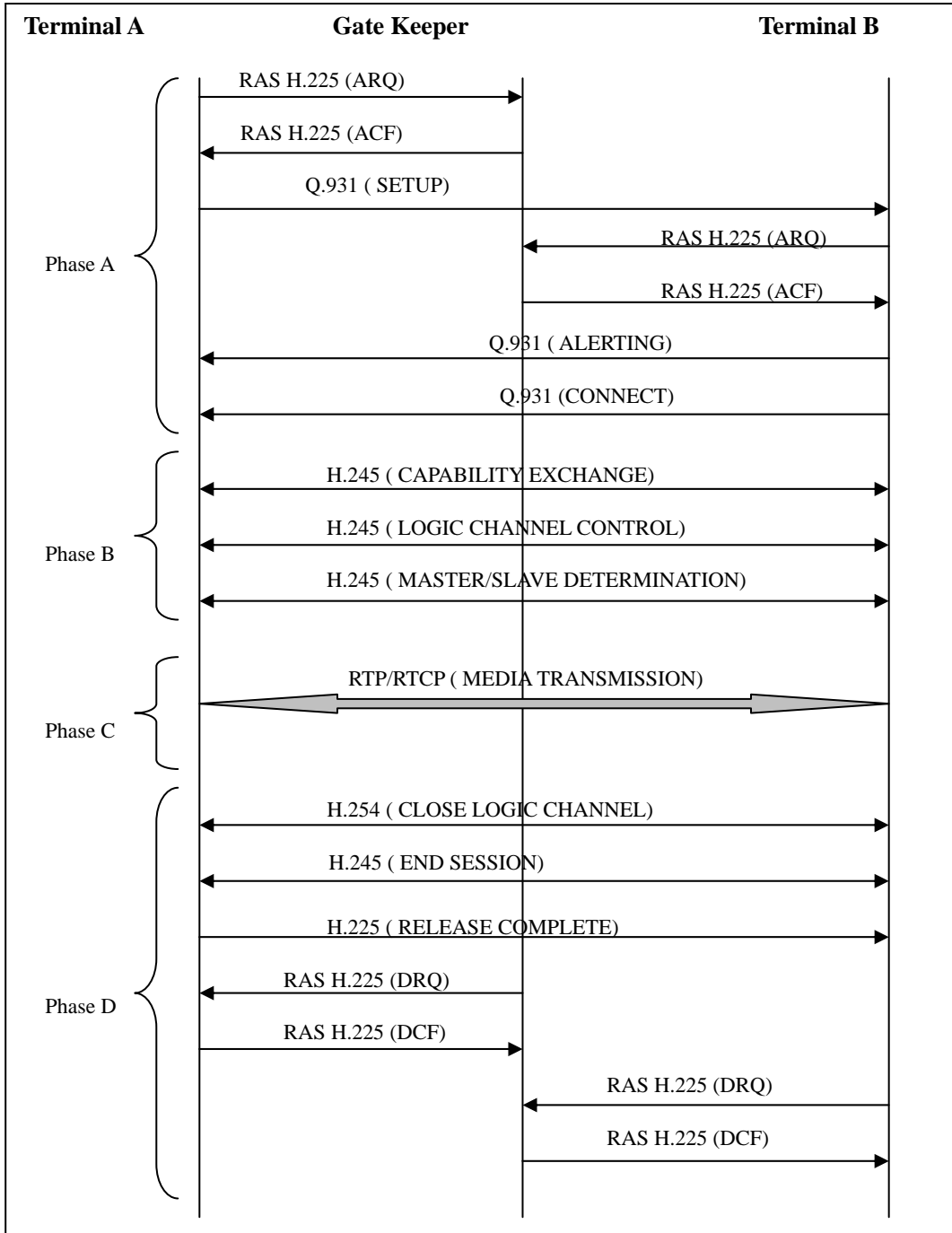


Figure 2 General Videophone Operation (call flow)

2 Getting Started

2.1 Connect your VDP

2.1.1 Installation Requirements

The following is a list of hardware/software necessary to install the VDP:

- TMDSVDP64X-2 VDP Main Board
- TMDSVDP64X-2 VDP Camera/Screen panel
- Handset
- 12V DC power adapter with power cord
- Ethernet switch with power adapter
- The cables:
 - ✓ Video-In cable (Camera/Screen panel to main board)
 - ✓ Video-Out cable (main board to Camera/Screen panel)
 - ✓ 10/100 BASE-T category-5 Ethernet cable
 - ✓ Auxiliary power cord (main board to Camera/Screen panel)
- Emulator with port (510 or 560)
- Computer

2.1.2 Installation Steps

- Step 1** Connect signal cables, except power, refer to VDP User's Guide
 - Step 2** Plug the power cable into the VDP units, refer to VDP User's Guide
 - Step 3** Connect JTAG to emulator via port, refer to Emulator User's Guide
 - Step 4** Connect Ethernet switch's power adapter to an electrical power outlet
 - Step 5** Connect the 12V DC power adapter cord to an electrical power outlet
- When the VDP units are properly connected and powered up, the four green activity LEDs on the main board would flash to indicate initialization. In a few seconds, the results of initialization will be displayed on the screen for a few seconds; then the video loop back will be on.

2.2 Installing Code Composer Studio

The VDP is designed to work with Code Composer Studio (CCS) version 2.21 which is an upgrade to CCS 2.20 that supports newer C6000 family DSPs. CCS 2.21 is installed in two steps. For one thing, You must install CCS 2.20 for the C6000, then run the CCS 2.21 patch..

The patch can upgrade the original version of BIOS and CSL to a new version and you will not be able to produce code with the original version of BIOS and CSL.

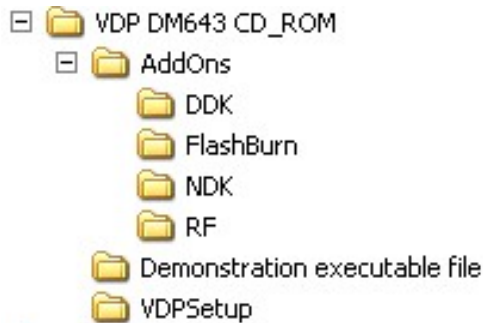


Figure 3 Installing Code Composer Studio

Note:

The default installation directory for CCS is c:\ti. If you install in a different directory you must make sure all of the patches and drivers get installed into the same directory as the CCS install. This document uses the keyword TI_DIR to denote the CCS install directory.

2.3 Installing VDP DM643 CD-ROM

After the proper version of Code Composer is installed, the next step is to add the board specific examples and documentation. To do this, run VDPSetup.exe on the VDP DM643 CD-ROM\VDP Setup. The installer will ask for a target directory, you would then use the directory that you installed CCS 2.21 into (TI_DIR) so the examples will be placed correctly.

2.4 Applying Code Composer Patches

The examples installed in Step 2 allow for easy evaluation and basic code

development with the VDP. However, the example relies on TI software components outside of the basic development environment. You must install the patches associated with these components now to get a full development environment. These patches are normally available through Code Composer's on-line Update Advisor or on the web at <http://dspvillage.ti.com> but have been included in the AddOns directory of the VDP DM643 CD-ROM for your convenience.

The following plugins, updates, and patches are found on the CD-ROM:

DDK

The DDK is TI's Device Driver Development Kit. The DDK is needed to rebuild or develop code but not to run pre-compiled code. The DDK should be installed in TI_DIR.

FlashBurn

FlashBurn is a Code Composer plug-in that allows you to program the contents of the on-board Flash memory. It is typically used to store bootable programs (the EVM can boot out of Flash) and data to be used to configure the on-board OSD FPGA.

NDK

The NDK is TI's TCP/IP Network Developer's Kit. It is included in library form for easy TCP/IP based network communication using the on-chip Ethernet interface. The NDK should be installed as TI_DIR\C6000\NDK. In this system, you must use NDK 1.71, which has been included in the VDP DM643 CD-ROM.

Reference Frameworks

The Reference Frameworks are a set of TI developed codes that act as software reference designs for common applications such as video transfer through a codec. The Reference Frameworks should be installed in TI_DIR.

2.5 Configuring the JTAG Emulator

The DM643 is based on a C64xx 1.1 DSP core. The core revision number is significant since each version of the driver only works with its intended silicon core version.

2.5.1 Installing Emulation Drivers

Users with Wintech Digital TDS510 PCI and USB emulators should install drivers that support the 1.1 core. Users with Wintech Digital TDS560 PCI and USB emulators receive emulation driver updates when the CCS 2.21 patch is applied.

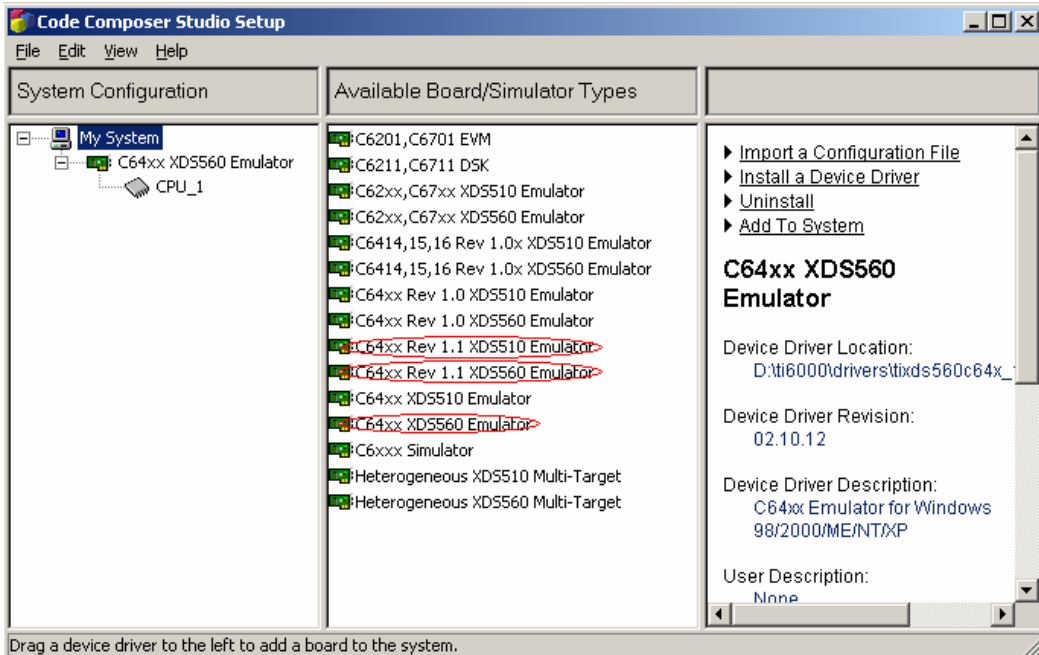


Figure 4 Installing emulation drivers

For more details, refer to Wintech Digital's *Emulator User's Guide*.

2.5.2 Import Configuration

When the correct emulation drivers are installed, start Code Composer Setup from its icon on your desktop. The Import Configuration screen should come up. If it does not, open it manually using File Import.

Code Composer Setup configures Code Composer for a specific chip/emulator configuration. Select the C64xx family under the filter options and import the configuration that matches your emulator. Do make sure you save your configuration before exiting Code Composer Setup.

2.5.3 Advanced Users

If you choose to manually set up your board, use a single C64xx device for the processor configuration and the board specific GEL file `TI_DIR\vdp\gel\VDPDM643.gel` as your GEL file.

2.6 Starting Code Composer Studio

Now that your tools are set up, you can start running and debugging with Code Composer. Make sure your emulator is connected and your board is powered by the included 12V power supply. Launch Code Composer by double-clicking on its icon:



After a short delay, the CCS Integrated Development Environment (IDE) will appear. If you are new to Code Composer, you should launch the Code Composer help (using the Help menu option) to become familiar with its basic usage.

2.7 VDP Directory Structure

Once installed, a new VDP directory will be accessible in `TI_DIR`. The VDP directory will look as follows:

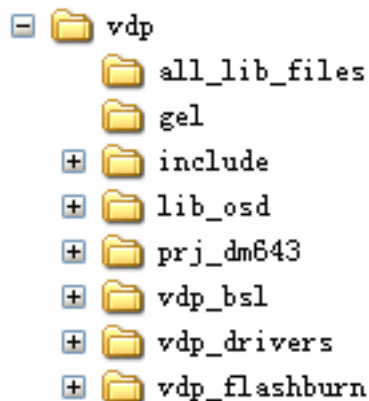


Figure 5 VDP software directory structure

Below is the top level directory structure:

<VDP>

< all_lib_files >	Lib directory
< gel >	Gel file of platform
< include >	Include files
< lib_osd >	Lib project of on screen display
< prj_dm643 >	Project directory
< vdp_bsl >	Lib project of VDP BSL
< vdp_drivers >	Source code to drivers and examples
< vdp_flashburn >	Flashburn files

Below is the directory structure under the folder *prj_dm643* and a short description of each folder.

<VDP\prj_dm643 >	Video development platform user's source code
-----< audio >	Source code of audio functions
-----<bin>	Output directory
-----<include>	Include files
-----< main >	Source code of main and initialization functions
-----< NetWork >	Source code of networking functions
-----< QoS >	Source code of the QoS (Quality of Service) functions
-----< settings >	Source code of audio and video settings
-----< timer >	Source code of timer
-----< uart >	Source code of UART functions
-----< ui >	Source code of user interface functions
-----< video >	Source code of video functions

Below is the directory structure under the folder *vdp_drivers*, and a short description of each folder.

<VDP\vdp_drivers>	
-----<audio>	Source code of audio drivers
-----<examples>	Source code of driver examples
-----<include>	Include files
-----<uart>	Source code of UART drivers

-----<vport> Source code of video drivers

Below is the directory structure under the folder *all_lib_files* and a short description of each folder.

<VDP\all_lib_files>

-----c6x1x_edma_mcaspl.l64	McASP driver (Audio)
-----dm643_edma_aic23.l64	AIC23 driver (Audio)
-----vdpvport.l64	Video port driver
-----dm643.lib	System Library
-----G723_DM643.lib	G.723 library
-----h263_dec_720_480_cat.lib	H.263 decoder library
-----h263_enc_720_480_cat.lib	H.263 encoder library
-----UB_Live_H264BP_Dec.l64	H.264 decoder library
-----UB_Live_H264BP_DM643_Enc.l64	H.264 encoder library
-----UB_Live_ImageTools_DM643.l64	Image utilities library
-----h323_core.lib	Lib to H.323 call stack
-----h323_hjgl.lib	Lib to H.323 call stack
-----h323_low.lib	Lib to H.323 call stack
-----h323_rtp.lib	Lib to H.323 call stack
-----vdp643bsl.lib	VDP BSL
-----wtosd.lib	OSD library
-----uart.l64	Serial Interface Library

2.8 VDP Source Files

This section gives a brief description of the source files contained in the VDP software package.

Location	Source file name	Description
VDP\prj_dm643\audio	tskAudioCapture.c	Audio Capture
	tskAudioPlayer.c	Audio Player
	tskAudioEncoder.c	Audio Encoder
	tskAudioDecoder.c	Audio Decoder
	ring_buffer.c	Ring buffer file
VDP\prj_dm643\main	appmain.c	Main Function

	appResources.c	Time Test Initialization
	dm643init.c	Initialization
VDP/prj_dm643\network	network_main.c	Network Main Function
VDP/prj_dm643\network\cgi	cgiparse.c	Basic CGI Functions
VDP/prj_dm643\network\cgi	cgiparsem.c	Multipart CGI Parsing for "multipart/form-data" forms
VDP/prj_dm643\ network\config_web	config_web.c	web page functions
VDP/prj_dm643\ network\console	console.c	Example router console
VDP/prj_dm643\QoS	qos_jitter.c	Packet Assemble, Disassemble and Sequencing
	qos_media.c	video and audio device manager
	qos_pool_mbx.c	buffer pool and mail box control
	qos_powerdown.c	powerdown control
	qos_receive_buf.c	equilibrate packet losing rate and rectify order
	qos_tsk.c	Data Flow Control
VDP\prj_dm643\settings	dm643_codec_devParams.c	Audio Parameter Configuration
	dm643_vcapparamsCIF.c	Video Capture Parameter Configuration
	dm643_vdisparamsNTSC.c	Video Player Parameter Configuration
	dm643_vdisparamsPAL.c	
VDP\prj_dm643\uart	uart.c	Serial Interface
VDP\prj_dm643\ui	keypad.c	Keyboard Driver

	t_flash.c	functions of add flash pictures
	tskui.c	Process Management
	ui_cfg.c	Parameter Configure Interface
	vdp_api.c	User Interface
	vdp_menudis.c	Menu Configuration
	vdp_menuset.c	Menu Configuration
	vdp_tone.c	Signal Sounds
	vdp_tone_ring_music.c	the data of audio ringback music
VDP\prj_dm643\ui\flash_ring	flash_ring_1.c	the 1st picture of ring flash
	flash_ring_2.c	the 2nd picture of ring flash
	flash_ring_3.c	the 3th picture of ring flash
	flash_ring_4.c	the 4th picture of ring flash
	flash_ring_5.c	the 5th picture of ring flash
	flash_ring_6.c	the 6th picture of ring flash
VDP\prj_dm643\video	tskVideoCapture.c	Video Capture
	tskVideoPlayer.c	Video Player
	tskVideoEncoder.c	Video Encoder
	tskVideoDecoder.c	Video Decoder
	tskVideoEncoder_control.c	Video Encoder param
	handfree.c	the picture of handfree icon
	media_263.c	the picture of H.263 icon
	media_264.c	the picture of H.264 icon

media_711.c	the picture of G.711U icon
media_723.c	the picture of G.723 icon
mute.c	the picture of audio mute icon
netlink.c	the picture of network link status icon
pip.c	functions of picture in picture
pip_sa.sa	functions of picture in picture
privacy.c	the picture of video privacy icon
t_media_choice.c	functions of media choice picture

Table 1 VDP source files

2.9 Building the project

After installed, you can Build the project. The following steps are for re-building an individual project:

- Step 1. Start Code Composer Studio on your PC.
- Step 2. Select Project-> Open to open the project vphone_dm643.pjt to be rebuilt from the TI_DIR\vdp\prj_dm643 folder.
- Step 3. Select the configuration that needs to be rebuilt from the configuration drop-down list box. The default configuration is Debug.
- Step 4. Select Project -> Build or Project -> Rebuild All to re-build the project for the selected configuration.
- Step 5. Select File -> Load Program..., select vphone.out in TI_DIR\ vdp\prj_dm643\bin to load the Demonstration executable file to run.
- Step 6. Select Debug -> Run or press F5 to run the project.

3 Development Platform Framework

3.1 Overview

This chapter describes the system architecture. In terms of module architecture, it includes: 1) capture and player module, 2) call manager module, 3) codec module, 4) user interface module. In terms of data flow architecture, it includes: 1) data flow of video, 2) data flow of audio, 3) data flow of call control.

3.2 Architecture

The overall software task architecture of VDP is shown in Figure3 below. The architecture uses RF-5 framework and integrates various video and audio encoders and decoders such as H.263 encoder/decoder, H.264 encoder/decoder, G.711U encoder/decoder and G.723 encoder/decoder, etc. A H.323 stack based on DSP/BIOS is also incorporated into the framework. The TSK-based feature and scalable channel management of RF-5 framework make it possible for so many processing tasks and algorithms to be integrated into the VDP software.

The data flow of the software architecture follows the following sequence:

Stage 1: The call management and user interface tasks manage the call flow status. On receiving an incoming call or initiating an outgoing call, the media encoding and decoding tasks are created dynamically.

When a call is established, the following stages take place:

Stage 2: A video frame is captured from the CCD camera and an audio frame is captured from the microphone.

Stage 3: The acquired video and audio data are fed to the encoder task and video/audio encoder encodes corresponding input frame.

Stage 4: The generated bit-stream is passed to the H.323 stack and sent through the network interface to the remote end.

Stage 5: The network interface receives the incoming bit-stream sent by the remote end.

Stage 6: The incoming bit-stream is decoded and re-framed as video bit-frame and audio bit-frame.

Stage 7: The video and audio bit-frame is passed to the video decoder and audio decoder respectively.

Stage 8: The video and audio decoder decodes the bit-frame and outputs the decoded frames.

Stage 9: The decoded video frame is then displayed on the output device and the decoded audio frame is played through the speaker.

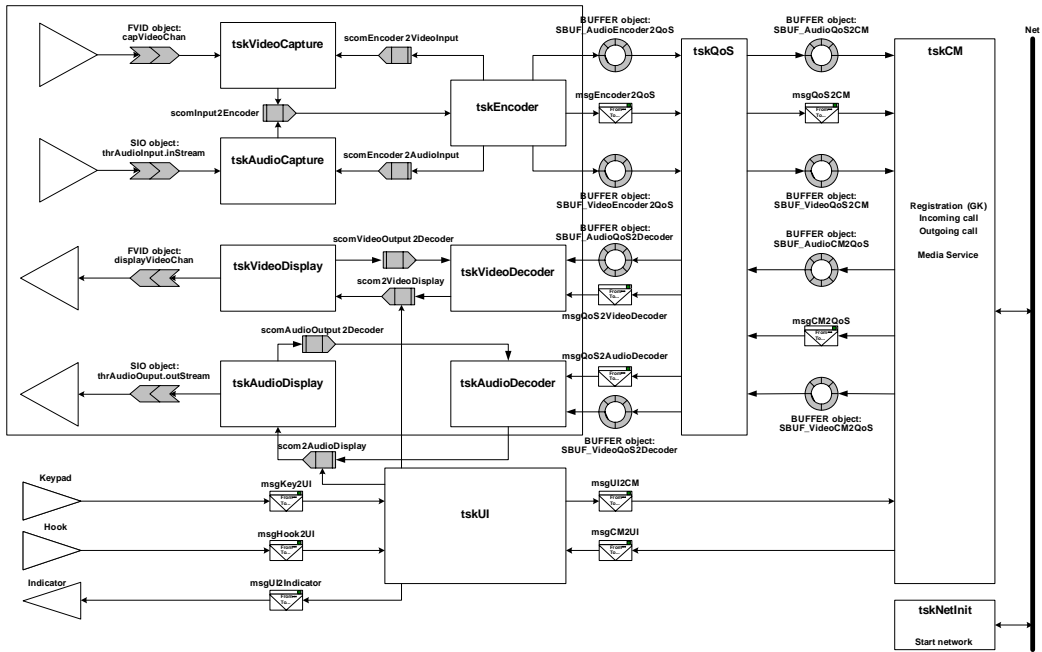


Figure 6 Software architecture block diagram

3.3 Task Descriptions

3.3.1 Audio Capture

The audio gathering task starts up automatically when the system starts; its status is controlled mainly by tskQoS. This task reads the data gathered from AIC23 driver, separates the data from left and right channel, call a coding module to process with a specific codec according to the arrangement of calling management module. The coded data will be sent to the buffer pool SBUF_AudioEncoder2QoS.

Figure 7 is the state diagram of this task.

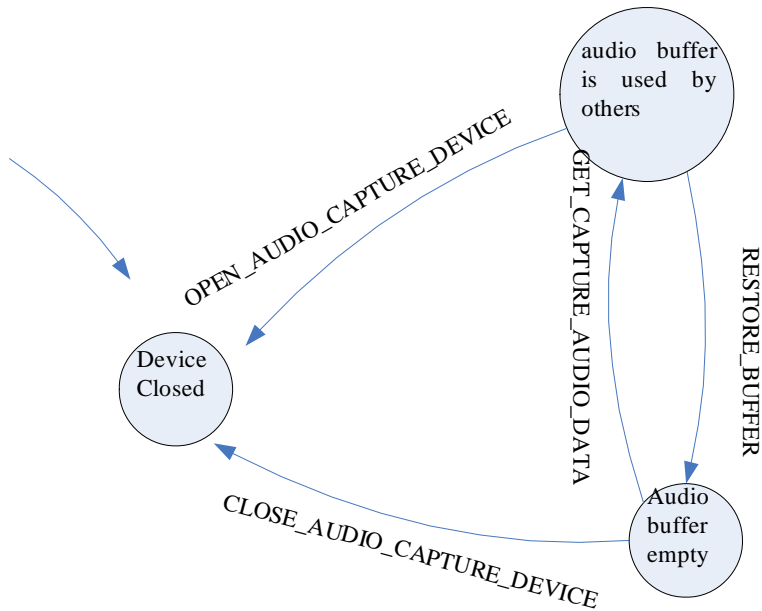


Figure 7 Audio Capture State

Audio gathering task exchange data with AIC23 driver is different from video gathering task. Audio gathering task's buffer is provided by application while video gathering task's buffer is allocated by driver when initialized.

The audio driver buffers are defined as:

```

Static Audio Sample
buf_for_audio_driver[DRIVER_BUFFER_COUNT][AUDIO_CAPTURE_CHANNELS_COUNT*
AUDIO_SAMPLE_MAX_FRAMELEN];
  
```

These buffers are passed to the audio device driver as the audio driver is initiated.

A separate buffer is defined for the audio encoder that compresses the data:

```

Static Audio Sample
buf_for_encoder[AUDIO_CAPTURE_CHANNELS_COUNT][AUDIO_SAMPLE_MAX_FRAMELEN];
  
```

The data from the driver is first copied to this buffer and then passed on to the encoder task.

3.3.2 Audio Playback

The audio playback task first calls the audio decoding function to decode the received bit-stream into audio data, and then forwards the audio data to the AIC23 device driver for audio playback. The scheme is describe in the Figure 8.

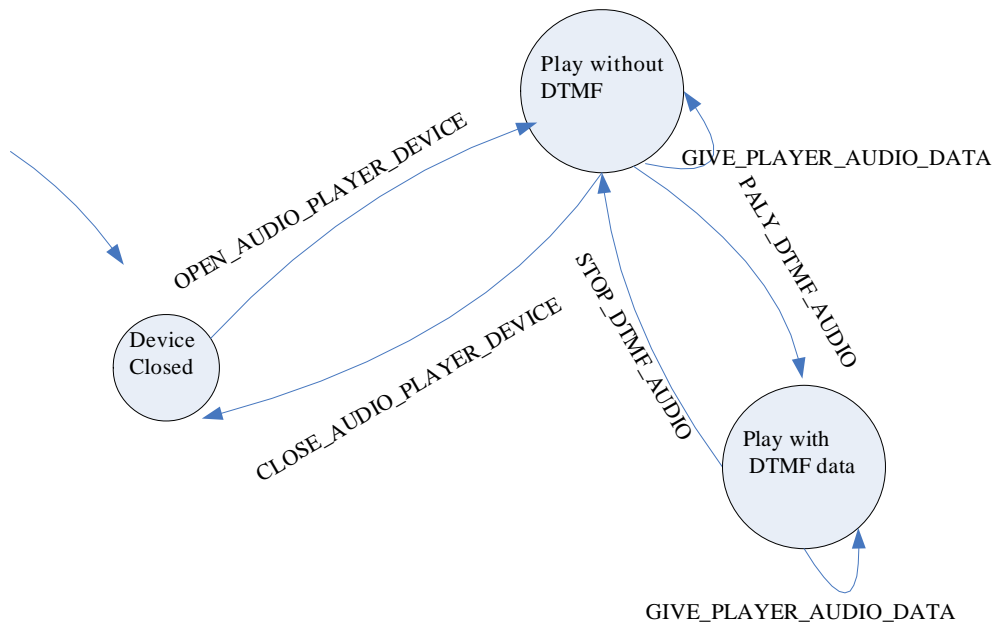


Figure 8 Audio Playback State

As same as audio capture module, the audio playing application also need to prepare some of buffers for audio device driver. It is defined in the file.

```

Static Audio Sample
player_buf_for_driver[ PLAYER_DRIVER_BUFFER_COUNT ][ USE_AUDIO_ADC_COUNT*
AUDIO_SAMPLE_MAX_FRAMELEN ];
  
```

The several buffers are transferred to audio device driver as audio device is

initiated.

3.3.3 Video Capture

The video capture task receives data from the video port device driver, after re-sampling, these data are forwarded to the video encoding task. The scheme is described in Figure 9.

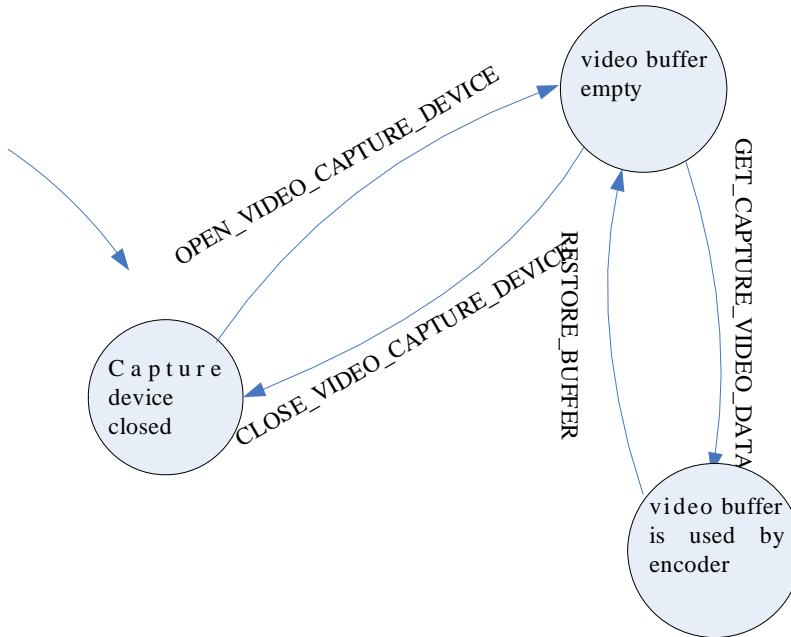


Figure 9 Video Capture State

The video capture module has something different from audio capture module. The process of encoding is done in an independent task other than video capture task. The two tasks exchange data through SCOM module.

3.3.4 Video Display

The video display task first calls the video decoding function to decode the received bit stream into video data, and then forwards the video data to the device driver for video playback. The scheme is described in Figure 7.

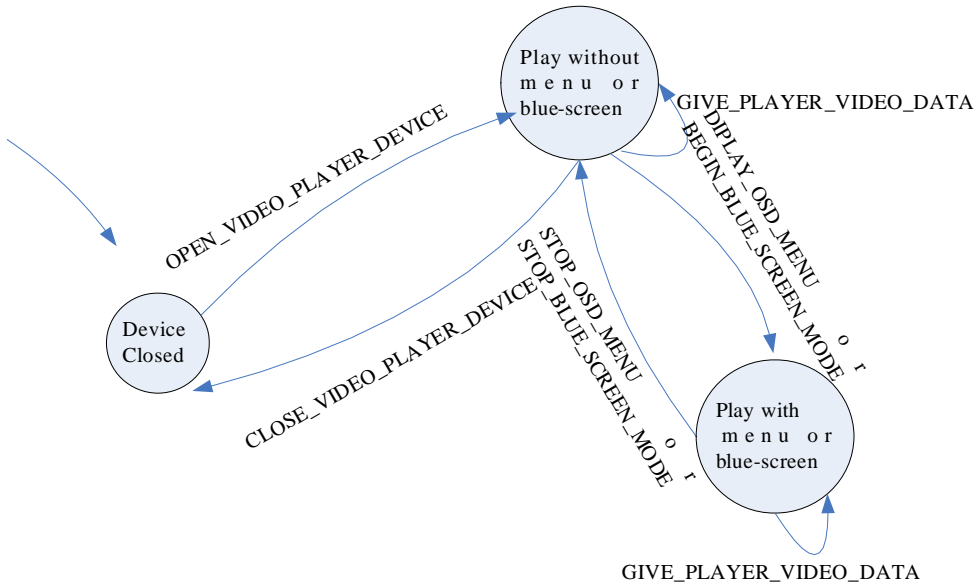


Figure 10 Video Display State

3.3.5 Video Encoder

The video data is taken from video capture task through two SCOM objects. One object's name is `scomVideoCapture2VideoEncoder`; the other is `scomVideoEncoder2VideoCapture`. According to the different encoder type, the corresponding encoder encodes the video data. The encoded data are placed in the pool buffer `SBUF_VideoEncoder2QoS`. Any other task can take encoded data from this pool in certain system state and send them to net or local loop back.

The flow chart of this task is described in Figure 11.

Video encoder task
tskVideoEncoder.c

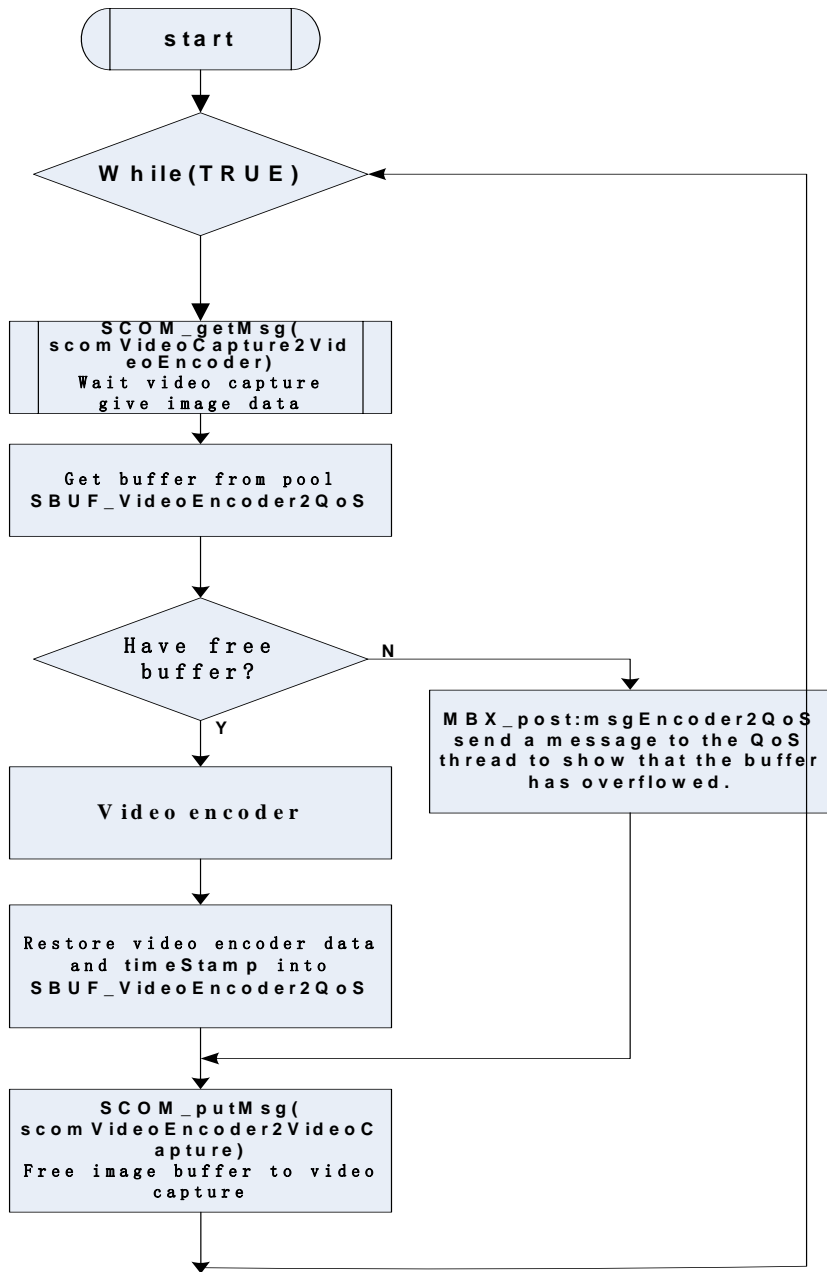


Figure 11 The video encoding task

3.3.6 Video Decoder

The encoded video data are taken from the buffer pool SBUF_VideoQoS2Decoder and decoded by the corresponding decoder. After decoding, the video data will be sent to video display task. The decoding task and display task exchange data through two SCOM_objects. One object's name is scomVideoPlayer2VideoDecoder, the other is scomVideoDecoder2VideoPlayer. The task is illustrated in Figure 12.

Video decoder task
tskVideoDecoderRun

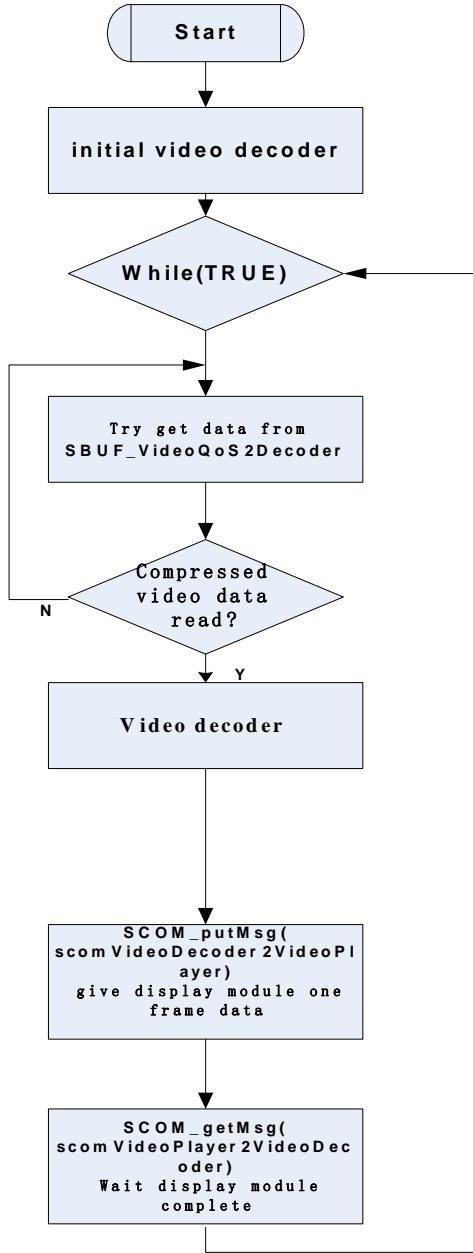


Figure 12 The video decoding task

3.3.7 Quality of Service (QoS)

The QoS task controls the media and CODEC tasks. When powered up, QoS task will open all the tasks except the network correlative tasks, as follows:

- Open audio capture task
- Open audio player task
- Open video capture task
- Open video player task
- Open video encoder task
- Open video decoder task
- Open user interface task

This task also manages the system status:

`STATUS_CONNECTED`, `STATUS_LOOPBACK`, `STATUS_POWERDOWN`

Normally, the system status is `STATUS_LOOPBACK`. But when the call is established, it will change to `STATUS_CONNECTED`.

At link status, audio and video frames will be split to packets and be sent to QOS2CM pools. The task CM will send the received packets to CM2QOS pools. So the QOS task can compose the packets to audio and video frames, and allow decoder tasks decode them.

If necessary, the QOS task can close and reopen every task; for example, it will close and reopen audio encoder and decoder tasks for the sake of changing G.711 CODEC to G.723 CODEC.

Note: only at `STATUS_LOOPBACK` status can menu and ring be played.

3.3.8 Call Control and Management (CM)

The Call Control and Management task manages an H.323 stack. It controls the call flow and sends system status messages to the QOS task. User Interface task controls it by MBX. See 4.4.

At normal mode, it can call by IP address. At GK mode, it can call by IP address and telephone number.

It's call flow will look as follows:

Q931->H.254->channel connect

When audio and video channels are established, the CM will get audio and video

data from QOS2CM pools, pack them with the given protocol header, then send them to network, receive packets from network, unpack and send them to CM2QOS pools.

3.3.9 Networking Task

This task is the network's main task. It configures and opens the NDK. Also, it opens the H.323 call manage task, because the call manage task is based on NDK.

Normally, it works at either DHCP mode or DHCP mode, which can be configured by user interface.

About more NDK, please see NDK 1.71 documents.

3.3.10 User Interface Task

This task is the most important task for users. It provides user's interface for the system, which includes configuring interface, user's interface to CM, user's interface to OSD, user's interface to audio play, and providing an example of a completed call flow.

4 Programming

4.1 Overview

This chapter describes all the information that will be used when you implement software using the VDP (Videophone Development Platform).

4.2 Software directory

The graph below describes the content of VDP directory that contains the VDP project, its source code, its library file and examples of simple projects that show you how to use the VDP.

Project view

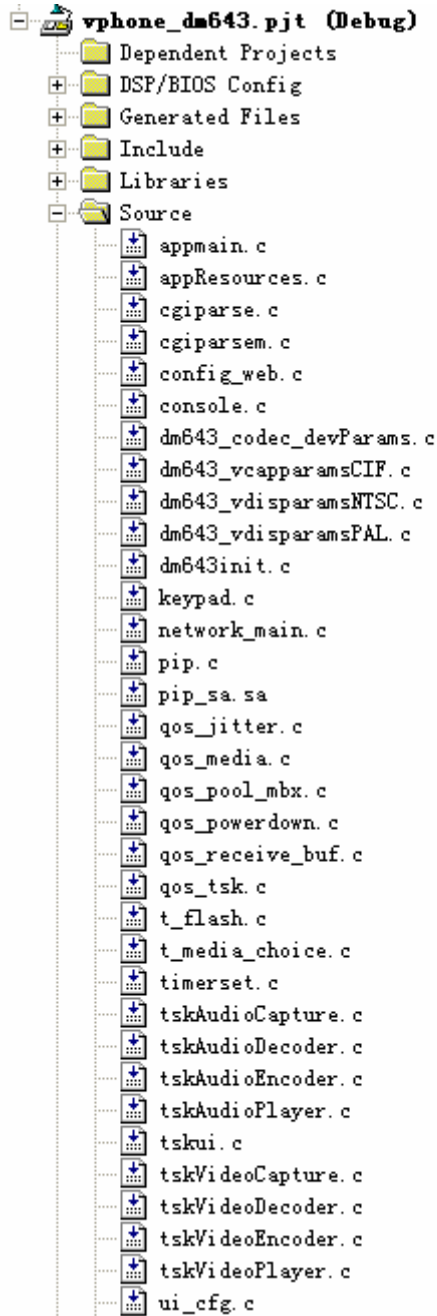


Figure 13 Project view

4.3 Programming Mode Description

4.3.1 Introduction

The VDP provide a convenient programming mode by which you can build the excellent V2oIP(voice and video over IP)system as wished.

4.3.2 Global System State Variables

The VDP uses global variables to save the global information. UiCfg is a global variable and is used to save global information needed in the VDP. The variable uiCfg is a structure of CFG as defined below.

```

typedef struct
{
    int          flag;
    char         FTPServer[URLLEN];
    BOOL        useDHCP;
    char         LocalIPAddr[IPADDRESSLEN];
    char         LocalIPMask[IPADDRESSLEN];
    char         GatewayIP[IPADDRESSLEN];
    char         DNSServer[IPADDRESSLEN];
    BOOL        UseGK;
    char         GKAddr[IPADDRESSLEN];
    char         GKUserName[USERNAMELEN];
    char         GKTelNum[TELNUMLEN];
    MEDIA        AudioType;
    MEDIA        VideoType;
    VIDEO_SIZE   VideoSize;
    VIDEO_FORMAT VideoFormat;
    VPORT_TYPE   VportType;
    Uns          VideoBitRate;
    char         FTPPassWord[USERNAMELEN];
    BOOL        usePPPoE;
    char         PPPOEUserName[30];
    char         PPPOEPassWord[30];
    BOOL        GKAutoScan;
    char         GKPassWord[USERNAMELEN];
    char         FTPUserName[USERNAMELEN];
    char         FTPRoute[URLLEN];
    int          FrameCount;
    BOOL        AutoAnswer;
} CFG;

```

Figure 14 Structure of CFG

The fields of the CFG structure are described in table 2 below.

Member of CFG structure	Description
flag	Information of this structure
FTPServer[IPADDRESSLEN]	FTP server address
useDHCP	DHCP is enable or disable
LocalIPAddr[IPADDRESSLEN]	Local IP address
LocalIPMask[IPADDRESSLEN]	Local IP mask
GatewayIP[IPADDRESSLEN]	Gateway address
DNSServer[IPADDRESSLEN]	DNS server address
UseGK	GK is enable or disable
GKAddr[IPADDRESSLEN]	GK IP address
GKUserName[USERNAMELEN]	GK user name
GKTelNum[TELNUMLEN]	GK telephone number
AudioType	Audio CODEC type
VideoType	Video CODEC type
VideoSize	Current video resolution
VideoFormat	Current video standard
VportType	Video port type
VideoBitRate	Current video bit rate
FTPPassWord[USERNAMELEN]	FTP server password
usePPPoE	PPPoE is enable or disable
PPPOEUserName[30]	PPPoE user name
PPPOEPassWord[30]	PPPoE password
GKAutoScan	Gate keeper autoscan is enable or disable
GKPassWord[USERNAMELEN]	gate keeper password
FTPUserName[USERNAMELEN]	Gate keeper user name
FTPRoute[URLLEN]	FTP server router
FrameCount	capture frame count
AutoAnswer	Auto answer is enable or disable

Table 2 CFG Structure

4.3.3 Message

The VDP takes the DSP/BIOS mailbox mechanism and defines some of messages to realize thread communication. The necessary message is defined in the table below.

MESSAGE	DESCRIPTION
KEY2UI_OFFHOOK	Notify the hook key is not pressed
KEY2UI_ONHOOK	Notify the hook key is pressed
KEY2UI_KEYPRESSED	Notify the key is pressed
KEY2UI_CONFIG	Notify the config key is pressed
KEY2UI_UNDEFINEKEY	Notify the undefined key is pressed
KEY2UI_INF	Notify the F1 key is pressed
USER_EVENT_OFFHOOK	Notify the off hook event
USER_EVENT_ONHOOK	Notify the on hook event
USER_EVENT_CALL_OUT	Notify the confirm call out event
USER_EVENT_BEGIN_MENU	Notify the begin menu event
USER_EVENT_END_MENU	Notify the end menu event
CALL2CM_ACCEPTCALL	Accept an incoming call
CALL2CM_DROPCALL	Drop a call
CALL2CM_NEWCALL	Make an outgoing call
CALL2CM_ENDCALL	End an incoming or connecting call
CM_EVENT_INCOMINGCALL	Have an incoming call
CM_EVENT_RINGBACK	Call ring back
CM_EVENT_CONNECTED	Call connecting is successful
CM_EVENT_CLOSE	Call stack has been closed
CM_EVENT_TIMEOUT	Connecting time out
CM_EVENT_GATEWAY_REG_FAIL	register gateway failed
CM_EVENT_GATEWAY_REG_OK	register gateway ok
CM_EVENT_GATEWAY_REMOTE_UNREG	remote not register gateway
CM_EVENT_AUDIO_CODER_UNMATCH	Audio CODEC does not match each other
CM_EVENT_VIDEO_CODER_UNMATCH	Video CODEC does not match each other
CM2QOS_AUDIO_DISCONNECTED	audio disconnected, stop exchange

	audio stream
CM2QOS_VIDEO_DISCONNECTED	video disconnected, stop exchange video stream
CM2QOS_AUDIO_CONNECTED	audio connected, start exchange audio stream
CM2QOS_VIDEO_CONNECTED	video connected, start exchange video stream
OPEN_AUDIO_CAPTURE_DEVICE	open audio capture device
OPEN_VIDEO_CAPTURE_DEVICE	open video capture device
CLOSE_AUDIO_CAPTURE_DEVICE	close audio capture device
CLOSE_VIDEO_CAPTURE_DEVICE	close video capture device
OPEN_AUDIO_PLAYER_DEVICE	open audio player device
OPEN_VIDEO_PLAYER_DEVICE	open audio player device
CLOSE_AUDIO_PLAYER_DEVICE	close audio player device
CLOSE_VIDEO_PLAYER_DEVICE	close video player device
RESTORE_BUFFER	give buffer back
POWERDOWN_DEVICE	Power down the special device
WAKEUP_DEVICE	wake up the special device
GET_CAPTURE_AUDIO_DATA	get audio capture data mode
GET_CAPTURE_VIDEO_DATA	get video capture data mode
GIVE_PLAYER_AUDIO_DATA	play audio data mode
GIVE_PLAYER_VIDEO_DATA	play video data mode
DIPLAY_OSD_MENU	display on screen menu
DIPLAY_OSD_MENU_ACK	display on screen menu ack
STOP_OSD_MENU	stop on screen menu
STOP_OSD_MENU_ACK	stop on screen menu ack
PALY_DTMF_AUDIO	play DTMF voice
PALY_DTMF_AUDIO_ACK	play DTMF voice ack
STOP_DTMF_AUDIO	stop DTMF voice
STOP_DTMF_AUDIO_ACK	stop DTMF voice ack
START_MUTE_AUDIO	play audio mute music
START_MUTE_AUDIO_ACK	play audio mute music ack
STOP_MUTE_AUDIO	stop audio mute music

STOP_MUTE_AUDIO_ACK	stop audio mute music ack
BEGIN_RING_FLASH_MODE	begin play ring flash
STOP_RING_FLASH_MODE	end play ring flash
SYSTEM_POWERDOWN	system powerdown message
SYSTEM_WAKEUP	system wake up message
FORCE_I_FRAME	Force I frame message

Table 3 Messages

4.3.4 System Status Reporting

System may be in different status when receiving corresponding message.

Different status are described in the following table. The message is described in the section 4.3.3. Programmers can use VDP_PostMessage() and VDP_GetMessage() to transmit messages between user and system. These two functions will be described later.

SYSTEM STATUS	DESCRIPTION
CALL_ST_POWERDOWN	System is in powerdown status
CALL_ST_READY	System is in ready status
CALL_ST_DIAL	Dialing status
CALL_ST_RINGING	System is in ringing status
CALL_ST_CALLING	System is in calling status
CALL_ST_RINGBACK	System is in ringback status
CALL_ST_CONNECTING	System is in connecting status
CALL_ST_TALKING	System is in talking status
CALL_ST_WAITINGCMEND	Waiting end call
CALL_ST_WAITINGHOOKON	Call has finished
CALL_ST_ERROR	Error status
CALL_ST_MENU	System is in system configuration status
CALL_ST_SYSINF	System is in system configuration status

Table 4 System Status

5 API Reference

The VDP provides a series of application programming interfaces to help

programmers build an appropriate V2oIP application. These APIs include user interface functions and system configuration functions that are listed in the following table.

VDP_PostMessage()	VDP_GetMessage()
VDP_StartOSD()	VDP_StopOSD()
VDP_KeyMapCbk()	VDP_SignalTone()
VDP_StartRing()	VDP_StopRing()
CFG_init()	CFG_uninit()
CFG_loadDefault()	CFG_setFTPServer()
CFG_setFTPPassWord()	CFG_setFTPUserName()
CFG_setFTPRoute()	CFG_setEnablePPPoE()
CFG_setPPPoEUserName()	CFG_setPPPoEPassWord()
CFG_loadParam()	CFG_saveParam();
CFG_setEnableDHCP()	CFG_setIPAddress()
CFG_setIPMask()	CFG_setGateWay()
CFG_setDNS()	CFG_setEnableGK()
CFG_setGKAutoScan()	CFG_setGKPassWord()
CFG_setGKAddress()	CFG_setGKUserName()
CFG_setGKTelNum()	CFG_setAudioCoder()
CFG_setVideoCoder()	CFG_setVPort()
CFG_setVideoSize()	CFG_setVideoFormat()
CFG_setVBR()	CFG_setFrameCount()
CFG_setOptAutoAnswer()	CFG_getOptAutoAnswer()
CFG_getFTPServer()	CFG_getFTPRoute()
CFG_getFTPUserName()	CFG_getFTPPassWord()
CFG_getEnableDHCP()	CFG_getEnablePPPoE()
CFG_getPPPoEUserName()	CFG_getPPPoEPassWord()
CFG_getIPAddress()	CFG_getIPMask()
CFG_getGateWay()	CFG_getDNS()
CFG_getEnableGK()	CFG_getGKAutoScan()
CFG_getGKAddress()	CFG_getGKUserName()
CFG_getGKTelNum()	CFG_getGKPassWord()
CFG_getAudioCoder()	CFG_getVideoCoder()
CFG_getFrameCount()	CFG_getVPort()
CFG_getVideoSize()	CFG_getVideoFormat()
CFG_getVBR()	VDP_getVersion()

Table 5 APIs

5.1 Call Message Transfer Functions

The user interface function offers call message transfer, keypad map, menu design, and dialing tone generation functionalities.

The VDP provides two functions to implement call message transfer. Both functions use the DSP/BIOS message box mechanism.

Function

Bool VDP_PostMessage(MBX_msg* pMsg, Uns timeout)

Parameters

MBX_msg* pMsg	message pointer
Uns timeout	return after this many system clock ticks

Return value

Bool TRUE if successful, FALSE if timeout

Description

VDP_PostMessage() posts a call message from a user to the system. If timeout is SYS_FOREVER, the task remains suspended until the transmitted call message has been taken away from the system queue. If timeout is 0, VDP_PostMessage() returns immediately. Otherwise, the task is suspended for the timeout system clock to tick. The actual time of task suspension can be up to 1 system clock tick less than timeout, due to granularity in system timekeeping.

Function

Bool VDP_GetMessage(MBX_msg* pMsg, Uns timeout);

Parameters

MBX_msg* pMsg	message pointer
Uns timeout	return after this many system clock ticks

Return value

Bool TRUE if successful, FALSE if timeout

Description

VDP_GetMessage() gets a call message from the system queue. If timeout is SYS_FOREVER, the task remains suspended until the call message is fetched. If timeout is 0, VDP_GetMessage() returns immediately. Otherwise, VDP_GetMessage() suspends the execution of the current task until the call message is fetched or the timeout expires. The actual time of task suspension can be up to 1 system clock tick less than timeout due to granularity in system timekeeping.

Apart from the two functions described above, pMsg, which is a MBX_msg type pointer, is used to transfer a call message between the user and the system. The MBX_msg type is defined as follows.

```
typedef struct MBX_msg {
    MBX_Handle sender;

    LgUns cmd;           /* Message code */
    LgUns arg1;         /* 1st message argument */
    LgUns arg2;         /* 2nd message argument */
    LgUns arg3;         /* 3rd message argument */
    LgUns arg4;         /* 4th message argument */
} MBX_msg;
```

The sender is a mailbox handle. The cmd member of this structure represents the call message that will be transferred between the user and the system. The rest members of this structure contain information attached to the call message. The particular type of call message is described in section 4.3.3.

5.2 Key Map Function

The VDP offers a callback function to realize the key map of the virtual value and the true value. The programmer can realize different maps through changing the callback function.

Function

void VDP_KeyMapCbk(Uint16 key)

Parameters

Uint16 key: key value returned from system

Return value

void

Description

VDP_KeyMapCbk() is a callback function. User can rewrite this function according to actual need.

5.3 Menu Design Functions

The VDP offers two functions to realize visual user interface design. The programmer can make different menus and labels according to different needs by calling both functions.

Function

void VDP_StartOSD(struct OSDUTIL_Display *pDisplay)

Parameters

struct OSDUTIL_Display *pDisplay pDisplay is a display handle that contains all the necessary information when drawing menus or labels.

Return value

void

Description

VDP_StartOSD() is used to draw menus and labels. Programmer can make various draw menus and labels by setting the corresponding input parameters.

Function

void VDP_StopOSD(struct OSDUTIL_Display *pDisplay);

Parameters

struct OSDUTIL_Display *pDisplay pDisplay is a display handle that contains all the necessary information when drawing menus or labels.

Return value

void

Description

VDP_StopOSD() stops to show menus and labels and releases resources allocated by VDP_StartOSD().

In the above two functions, the input parameter pDisplay is a OSDUTIL_Display type pointer. The OSDUTIL_Display structure is defined below.

```
struct OSDUTIL_DisplayItem{
    Int x;
    Int y;
    Int isEdit;
    char Caption[50];
    char HotKey;
    struct OSDUTIL_Display *pSub_Display;
    void (*callback_function)(char);
};

struct OSDUTIL_Display{
    char Name[8];
    Bool isMenu;
    Int itemCount;
    Int Place;
    Int SelectedNumber;
    struct OSDUTIL_DisplayItem *pItem;
    Uint8 *disHandle;
    struct OSDUTIL_Display *pParent_Display;
};
```

5.4 Dialing Tone Generation Functions

The VDP offers a series of API functions to generate various signal tones.

Function

void VDP_StartRing(void)

Parameters

None

Return value

void

Description

Start to play incoming ring

Function

void VDP_StopRing(void)

Parameters

None

Return value

void

Description

Stop to play incoming ring

Function

void VDP_SignalTone(SIGNAL signalTone, unsigned char key)

Parameters

SIGNAL signalTone: type of signal to be generated;

unsigned char key: only when the parameter signalTone is VP_Audio_DTMF, is the parameter key effective, which then represents the user-pressed key value.

Return value

Return: void

Description

VDP_SignalTone() can generate various signal tones according to the different input parameters. The input parameter signalTone is a variable of SINGAL type. The SINGAL type is defined below.

```

typedef enum
{
    VP_Audio_NONE = 0,
    VP_Audio_DIAL,
    VP_Audio_DTMF,
    VP_Audio_RINGBACK,
    VP_Audio_BUSY,
    VP_Audio_STOP,
} SIGNAL;

```

Enumerating type	Generated tone
VP_Audio_NONE	silence
VP_Audio_DIAL	dialing tone
VP_Audio_DTMF	key tones
VP_Audio_RINGBACK	ring back
VP_Audio_BUSY	busy tone
VP_Audio_STOP	stop

Table 6 Audio Type

5.5 Configuration

The VDP provides a series of functions to save and load the system parameters that include a local IP address, audio and video CODEC, video bit rate, video format, etc.

Function

void CFG_init()

Parameters

void

Return value

void

Description

CFG_init() initiates system configuration information and must be called at system

initiation.

Function

void CFG_uninit();

Parameters

void

Return value

void

Description

CFG_uninit() uninstall system configuration.

Function

void CFG_loadDefault();

Parameters

void

Return value

void

Description

CFG_loadDefault() load default system configuration parameters.

Function

void CFG_loadParam();

Parameters

void

Return value

void

Description

CFG_loadParam() loads saved system configuration parameters from ROM.

Function

void CFG_saveParam();

Parameters

void

Return value

void

Description

CFG_saveParam() saves changed system configuration parameters to ROM.

Function

void CFG_setFTPServer(const char* ftpServer);

Parameters

const char* ftpServer

Return value

void

Description

CFG_setFTPServer() is used to configure the FTP address from which the user can update the program.

Function

void CFG_setFTPPassWord(const char* PassWord);

Parameters

const char* PassWord

Return value

void

Description

CFG_setFTPPassWord () is used to configure the password of FTP Server from which the user can update the program.

Function

void CFG_setFTPUserName(const char* UserName);

Parameters

const char* UserName

Return value

void

Description

CFG_setFTPUserName () is used to configure the user name of FTP Server from which the user can update the program.

Function

```
void CFG_setFTPRoute(const char* s);
```

Parameters

```
const char* s
```

Return value

```
void
```

Description

CFG_setFTPRoute () is used to configure the route of FTP Server from which the user can update the program.

Function

```
void CFG_setEnablePPPoE (BOOL b);
```

Parameters

```
BOOL b b = TRUE, enable PPPoE; b = FALSE, disable PPPoE
```

Return value

```
void
```

Description

CFG_setEnablePPPoE() set the flag value that determines if PPPoE is enabled.

Function

```
void CFG_setPPPoEUserName(char* str);
```

Parameters

```
char* str
```

Return value

```
void
```

Description

CFG_setPPPoEUserName () is used to configure the user name of PPPoE.

Function

```
void CFG_setPPPoEPassword(char* str);
```

Parameters

```
char* str
```

Return value

```
void
```

Description

CFG_setPPPoEPassword () is used to configure the password of PPPoE.

Function

```
void CFG_setEnableDHCP(BOOL b);
```

Parameters

BOOL b b = TRUE, enable DHCP; b = FALSE, disable DHCP

Return value

void

Description

CFG_setEnableDHCP() set the flag value that determines if DHCP is enabled.

Function

```
void CFG_setIPAddress(const char* IPAddress);
```

Parameters

const char* IPAddress

Return value

void

Description

CFG_setIPAddress() sets local IP address.

Function

```
void CFG_setIPMask(const char* IPMask);
```

Parameters

const char* IPMask

Return value

void

Description

CFG_setIPMask() sets local IP mask.

Function

```
void CFG_setGateWay(const char* GatewayAddress);
```

Parameters

const char* GatewayAddress

Return value

void

Description

CFG_setGateWay() sets local IP gateway address.

Function

```
void CFG_setDNS(const char* DNSAddress);
```

Parameters

const char* DNSAddress

Return value

void

Description

CFG_setDNS() sets local DNS address.

Function

```
void CFG_setEnableGK(BOOL b);
```

Parameters

BOOL b b = TRUE, enable GK; b = FALSE, disable GK

Return value

void

Description

CFG_setEnableGK() sets the flag value that determine if GK is enable.

Function

```
void CFG_setGKAutoScan(BOOL b);
```

Parameters

BOOL b b = TRUE, enable auto scan; b = FALSE, disable auto scan

Return value

void

Description

CFG_setGKAutoScan () sets the flag value that determine if gate keeper auto scan is enable.

Function

```
void CFG_setGKAddress(const char* GKAddress);
```

Parameters

const char* GKAddress

Return value

void

Description

CFG_setGKAddress() sets local GK address.

Function

```
void CFG_setGKUserName(const char* GKUserName);
```

Parameters

const char* GKUserName

Return value

void

Description

CFG_setGKUserName() sets local GK user name.

Function

```
void CFG_setGKPassWord(const char* GKPassWord);
```

Parameters

const char* GKPassWord

Return value

void

Description

CFG_CFG_setGKPassWord () sets local GK password.

Function

```
void CFG_setGKTelNum(const char* GKTelnum);
```

Parameters

const char* GKTelnum

Return value

void

Description

CFG_setGKTelNum() sets local GK telephone number.

Function

```
void CFG_setAudioCoder(MEDIA audio_type);
```

Parameters

MEDIA audio_type

Return value

void

Description

CFG_setAudioCoder() sets current audio CODEC type.

Function

```
void CFG_setVideoCoder(MEDIA video_type);
```

Parameters

MEDIA video_type

Return value

void

Description

CFG_setVideoCoder() sets current video CODEC type.

Function

```
void CFG_setVPort(VPORT_TYPE vport_type);
```

Parameters

VPORT_TYPE vport_type

Return value

void

Description

CFG_setVPort() sets video port.

Function

```
void CFG_setVideoSize(VIDEO_SIZE video_size);
```

Parameters

VIDEO_SIZE video_size

Return value

void

Description

CFG_setVideoSize() sets the current video resolution.

Function

```
void CFG_setVideoFormat(VIDEO_FORMAT video_format);
```

Parameters

VIDEO_FORMAT video_format

Return value

void

Description

CFG_setVideoFormat() sets the current video standard.

Function

```
void CFG_setVBR(Uns vbr);
```

Parameters

Uns vbr

Return value

void

Description

CFG_setVBR() sets the current video bit rate.

Function

```
void CFG_setFrameCount(int count);
```

Parameters

Uns count

Return value

void

Description

CFG_CFG_setFrameCount() sets the current frame count.

Function

```
void CFG_getFTPServer(char* ftpServer);
```

Parameters

char* ftpServer

Return value

void

Description

CFG_setFTPServer() is used to get the FTP address from which user can update program.

Function

```
void CFG_getFTPRoute(char* s);
```

Parameters

char* s

Return value

void

Description

CFG_ CFG_getFTPRoute() is used to get the FTP route from which user can update program.

Function

```
void CFG_ CFG_getFTPUserName(char* UserName);
```

Parameters

char* UserName

Return value

void

Description

CFG_ CFG_getFTPUserName () is used to get the FTP user name from which user can update program.

Function

```
void CFG_getFTPPassWord(char* PassWord);
```

Parameters

char* PassWord

Return value

void

Description

CFG_ CFG_getFTPPassWord () is used to get the FTP password from which user can

update program.

Function

BOOL CFG_getEnableDHCP();

Parameters

void

Return value

BOOL TRUE, DHCP is enabled; FALSE, DHCP is disabled.

Description

CFG_getEnableDHCP() gets the flag value that determine if DHCP is enable.

Function

BOOL CFG_getEnablePPPoE();

Parameters

void

Return value

BOOL TRUE, PPPoE is enabled; FALSE, PPPoE is disabled.

Description

CFG_getEnablePPPoE() gets the flag value that determine if PPPoE is enable.

Function

void CFG_CFG_getPPPoEUserName(char* str);

Parameters

char* str

Return value

void

Description

CFG_CFG_getPPPoEUserName () is used to get the PPPoE user name.

Function

void CFG_getPPPoEPassWord(char* str);

Parameters

char* str

Return value

void

Description

CFG_CFG_getPPPoEPassword() is used to get the PPPoE password.

Function

void CFG_getIPAddress(char* IPAddress);

Parameters

char* IPAddress

Return value

void

Description

CFG_getIPAddress() returns local IP address.

Function

void CFG_getIPMask(char* IPMask);

Parameters

char* IPMask

Return value

void

Description

CFG_getIPMask() returns local IP mask.

Function

void CFG_getGateWay(char* GatewayAddress);

Parameters

char* GatewayAddress

Return value

void

Description

CFG_getGateWay() returns local IP gateway address.

Function

void CFG_getDNS(char* DNSAddress);

Parameters

char* DNSAddress

Return value

void

Description

CFG_getDNS() returns local DNS address.

Function

BOOL CFG_getEnableGK();

Parameters

void

Return value

BOOL TRUE, enable GK; FALSE, disable GK

Description

CFG_getEnableGK() returns the flag value that determines if GK is enabled.

Function

BOOL CFG_getGKAutoScan ();

Parameters

void

Return value

BOOL TRUE, enable auto scan; FALSE, disable auto scan

Description

CFG_ CFG_getGKAutoScan() returns the flag value that determines if geta keeper suto scan is enabled.

Function

void CFG_getGKAddress(char* GKAddress);

Parameters

char* GKAddress

Return value

void

Description

CFG_getGKAddress() returns local GK address.

Function

```
void CFG_getGKUserName(char* GKUserName);
```

Parameters

```
char* GKUserName
```

Return value

```
void
```

Description

CFG_getGKUserName() returns local GK user name.

Function

```
void CFG_getGKPassWord(char* GKPassWord);
```

Parameters

```
char* GKPassWord
```

Return value

```
void
```

Description

CFG_CFG_getGKPassWord () returns local GK password.

Function

```
void CFG_getGKTelNum(char* GKTelmun);
```

Parameters

```
char* GKTelnum
```

Return value

```
void
```

Description

CFG_getGKTelNum() returns the local GK telephone number.

Function

```
MEDIA CFG_getAudioCoder();
```

Parameters

```
void
```

Return value

```
MEDIA
```

Description

CFG_getAudioCoder() returns the current audio CODEC type.

Function

MEDIA CFG_getVideoCoder();

Parameters

void

Return value

MEDIA

Description

CFG_getVideoCoder() returns the current video CODEC type.

Function

VIDEO_SIZE CFG_getVideoSize();

Parameters

void

Return value

VIDEO_SIZE

Description

CFG_getVideoSize() returns the current video resolution.

Function

VIDEO_FORMAT CFG_getVideoFormat();

Parameters

void

Return value

VIDEO_FORMAT

Description

CFG_getVideoFormat() returns the current video standard.

Function

VPORT_TYPE CFG_getVPort();

Parameters

void

Return value

VPORT_TYPE

Description

CFG_getVPort() returns the video port.

Function

```
Uns CFG_getVBR();
```

Parameters

void

Return value

Uns

Description

CFG_getVBR() returns the current video bit rate.

Function

```
int CFG_getFrameCount ();
```

Parameters

void

Return value

int

Description

CFG_getFrameCount () returns the current frame rate.

Function

```
BOOL VDP_getVersion(char* str)
```

Parameters

char* str

Return Value

TRUE if successful; FALSE if failed.

Description

CFG_getVersion() returns to get the current version.

Function

```
void CFG_setOptAutoAnswer(BOOL flag)
```

Parameters

BOOL flag

Return Value

Uns

Description

CFG_setOptAutoAnswer() determines if auto answer is enable.

Function

BOOL CFG_getOptAutoAnswer()

Parameters

Uns

Return Value

TRUE if auto answer is enabled; FALSE if auto answer is disabled.

Description

CFG_getOptAutoAnswer() inquires if auto answer is enable.

6 Device Driver Development

6.1 Overview

VDP's drivers mainly make reference to EVMDM642's drivers by using DSP/BIOS Driver Developer's Kit (DDK) for development. The DDK is designed to simplify the development of device drivers for peripherals presented on TMS320 DSPs and their associated evaluation boards. The DDK offers a number of complete device drivers for peripherals such as codecs, UARTs, PCI controllers, and serial ports. These drivers are provided in both binary and source code formats. The availability of source code makes it straightforward to port the driver to custom board configurations by using the same peripherals.

6.2 Driver Directory

The graph below describes the content of the driver directory in the VDP project folder.

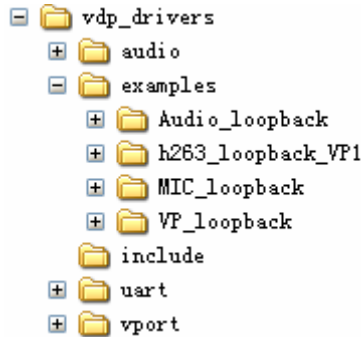


Figure 15 Driver Directory

In audio, uart and vport folder, there are drivers of audio, UART and Video respectively. In examples folder, some simple examples are provided which include audio loopback, H.263 loopback, MIC loopback and video loopback. Some head files that are used for driver only are also included in the folder .

6.3 Keypad Driver

The keypad is controlled by a microprocessor, and output pin of microprocessor is connected with GPIO of DSP. When a key is pressed, microprocessor will send key assignments to GPIO, and send an interrupt to DSP synchronously. When DSP received an interrupt of keypad, DSP will read information of pressed key from GPIO in corresponding interrupt service routine. Commonly, the key assignments that microprocessor send is not in accordance with the value that the developer is looking for. So, it needs a conversion between the two values. The function map_key() realizes the conversion in the following example. Mailbox, named mbx_keyboard, is used to communicate information between keypad driver and application function. Definition of Mailbox structure has been described in section 5.1. When interrupt is generated, the mbx_keyboard will send two messages: conversion value and key state.

The following table shows the conversion of key assignments.

Key	key assignments	Conversion value	Key state
1	1	'1'	KEY2UI_KEYPRESSED
2	2	'2'	KEY2UI_KEYPRESSED
3	3	'3'	KEY2UI_KEYPRESSED

4	4	'4'	KEY2UI_KEYPRESSED
5	5	'5'	KEY2UI_KEYPRESSED
6	6	'6'	KEY2UI_KEYPRESSED
7	7	'7'	KEY2UI_KEYPRESSED
8	8	'8'	KEY2UI_KEYPRESSED
9	9	'9'	KEY2UI_KEYPRESSED
0	10	'0'	KEY2UI_KEYPRESSED
*	11	'.'	KEY2UI_KEYPRESSED
#	12	'#'	KEY2UI_KEYPRESSED
F1:ON	13	13	KEY2UI_INF
F2:ON	14	14	KEY2UI_UNDEFINEKEY
F3:ON	15	15	KEY2UI_UNDEFINEKEY
Enter	16	16	KEY2UI_KEYPRESSED
Esc	17	27	KEY2UI_KEYPRESSED
Cfg	18	18	KEY2UI_CONFIG
Back Space	19	26	KEY2UI_KEYPRESSED
HOOK:OFF	20	20	HOOK2UI_OFFHOOK
HOOK:ON	21	21	HOOK2UI_ONHOOK
F1:OFF	22	22	KEY2UI_INF
F2:OFF	23	23	KEY2UI_UNDEFINEKEY
F3:OFF	24	24	KEY2UI_UNDEFINEKEY

Table 7 Key Assignments

The following code shows the source code of the keypad's driver. You can also find the code in file "keypad.c".

```

/* key mapping */
void map_key(Uint16 key)
{
    MBX_msg txMsg;
    switch(key)
    {
        case 1:
            {
                txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=key+0x30;
            }
            break;
        case 2:
            {
                txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=key+0x30;
                txMsg.arg2=13;
            }
            break;
        case 3:
            {
                txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=key+0x30;
            }
            break;
        case 4:
            {
                txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=key+0x30;
                txMsg.arg2=15;
                txMsg.arg3='-';
            }
            break;
        case 5:
            {
                txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=key+0x30;
            }
            break;
        case 6:
            {
                txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=key+0x30;
                txMsg.arg2=16;
                txMsg.arg3='+';
            }
            break;
    }
}

```

```
        case 7:
        {   txMsg.cmd=KEY2UI_KEYPRESSED;
            txMsg.arg1=key+0x30;
        }
        break;
case 8:
        {   txMsg.cmd=KEY2UI_KEYPRESSED;
            txMsg.arg1=key+0x30;
            txMsg.arg2=14;
        }
        break;
case 9:
        {   txMsg.cmd=KEY2UI_KEYPRESSED;
            txMsg.arg1=key+0x30;
        }
        break;
case 10:
        {   txMsg.cmd=KEY2UI_KEYPRESSED;
            txMsg.arg1='0';
        }
        break;
case 11:
        {   txMsg.cmd=KEY2UI_KEYPRESSED;
            txMsg.arg1='.';
        }
        break;
case 12:
        {   txMsg.cmd=KEY2UI_KEYPRESSED;
            txMsg.arg1='#';
        }
        break;
```

```

        case 13:
            {   txMsg.cmd=KEY2UI_INF;
                txMsg.arg1=key;           //F1 ON
            }
            break;
case 14:                                     //F2 ON
case 15:                                     //F3 ON
            {   txMsg.cmd=KEY2UI_UNDEFINEKEY;
                txMsg.arg1=key;
            }
            break;
case 16:
            {   txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=0x10; // Enter
            }
            break;
case 17:
            {   txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=0x1b; // ESC
            }
            break;
case 18:
            {   txMsg.cmd=KEY2UI_CONFIG;
                txMsg.arg1=key; //nemu
            }
            break;
case 19:
            {   txMsg.cmd=KEY2UI_KEYPRESSED;
                txMsg.arg1=0x1a; // SUB
            }
            break;

```

```
case 20:
    { txMsg.cmd=KEY2UI_OFFHOOK;

        }
    break;
case 21:
    { txMsg.cmd=KEY2UI_ONHOOK;

        }
    break;
case 22:
    { txMsg.cmd=KEY2UI_INF;
      txMsg.arg1=key;
        }
    break;
case 23:
case 24:
    { txMsg.cmd=KEY2UI_UNDEFINEKEY;
      txMsg.arg1=key;
        }
    break;
default: break;
}
```

```

/* keyboard initialization */
void keyboard_init()
{
    HWI_dispatchPlug(IRQ_EVT_EXTINT7, (Fxn)keyISR, -1, NULL);
    IRQ_reset(IRQ_EVT_EXTINT7);

    GPEN =GPEN | 0xDf00;
    GPDIR =GPDIR |(0xC000);
    GPDIR =GPDIR &(~0x1f00);
    IRQ_enable(IRQ_EVT_EXTINT7);
}

```

```

/* Keyboard interrupt */
void keyISR(int arg)
{
    Uint16 keyval;

    keyval=((GPVAL & 0x1f00) >> 8);

    map_key(keyval);
}

```

6.4 Audio Port Driver

VDP's audio port driver makes reference to AIC23 codec device driver of TM320DM642 EVM. This device driver is written in compliance with the DSP/BIOS IOM device driver model and uses the generic TMS320C6x1x EDMA McASP driver to transfer samples to and from the serial port. The codec specific portion of the mini-driver inherits the features of the generic TMS320C6x1x EDMA McASP driver. It uses two codec specific functions, `mbBindDev()` and `mdCreateChan()`, to complete the VDP and AIC23 specific setup. These functions then call `mbBindDev()` and `mdCreateChan()` in the generic driver to complete generic portions of the driver initialization. The only thing the codec-specific part does is to set up the codec and leaves the transfers of samples to the generic device driver. The fact that this device driver uses the generic device driver is hidden from the user in all aspects except that the generic device driver library has to be linked with the application.

The function `mdBindDev()` is responsible for configuring the codec through the control channel based on the `EVMDM642_EDMA_AIC23_DevParams` structure. It does some basic setup, then calls a function called `AIC23_setParams()` function in `aic23.c` to do most of the real configuration work. The function `mdCreateChan()` generates the EDMA configuration used for the data transfers.

For details on this device driver, see the application note *A DSP/BIOS AIC23 Codec Device Driver for the TMS320DM642EVM (SPRA922)* and the application note *A DSP/BIOS EDMA McASP Device Driver for TSM320C6x1x DSPs (SPRA870)*.

6.5 Video Port Driver

VDP is different from DM642 EVM, VDP's video capture and video display adopts TI TVP5150A video decoder and Phillips SAA7104 video encoder respectively. The design of the video capture and display mini-drivers can be found in *The TMS320DM642 Video Port Mini-driver (SPRA918A)*. These device drivers are compliant with the DSP/BIOS IOM device driver model. The DSP's EDMA is used to transfer data between memory and the VDP Video Port. To maximize code reuse and streamline the integration process, both drivers are designed to have two distinctive parts: the generic part and the board specific part. The external device control interface (EDC) is defined to bind these two parts together in a plug-and-play manner.

Features:

- Multi-instance (can handle multiple video ports simultaneously).
- Capture driver supports the following modes:
 - Single-channel 8/10-bit BT.656 mode with embedded or external sync
 - Dual-channel 8/10 bit BT.656 mode with embedded sync
 - Single-channel 16/20 bit Y/C mode with embedded or external sync
- Display driver supports the following modes:
 - 8/10-bit BT.656 mode with embedded or external sync.
 - 16/20-bit Y/C mode with embedded or external sync for output formats such as high-definition 480p, 720p and 1080i
 - 8/10/16/20 raw mode with $\frac{3}{4}$ unpacking, for output formats such as 8/16/24-bit RGB
- Supports enable/disable of video port global interrupt on all defined video port events
- Drivers allocate video frame buffers at initialization time based on configuration parameters passed in by the application
- External Control Interface for seamless integration with different video encoder or decoder devices

To use the capture or display device driver, a device entry must be added and configured in the DSP/BIOS configuration tool. Refer to the *DSP/BIOS Device Driver Developer's Guide* (literature number SPRU616) for more information on how to use the DSP/BIOS configuration tool to configure device drivers. The following are the device configuration settings required to use the capture driver:

- **Init function:** N/A, not used by this driver
- **Function table ptr:** `_VPORTCAP_Fxns`
- **Function table type:** `IOM_Fxns`
- **Device id:** 0 or 2 for VDP:, specify which video port to use
- **Device params ptr:** An optional pointer to an object of type `VPORT_PortParams` as defined in the header file `vport.h`. This pointer will point to a device parameter structure. Setting this pointer to `NULL` requires that an additional `FVID_control` call made from the application to initialize the video port. The parameter structure is described below. An example of this structure is the `_EVM643_vCapParamsNTSCPort` that is defined in the `evm643_vcapparamsNTSC.c` file for NTSC format video capture.
- **Device global data ptr:** N/A, not used by this driver

The following are the device configuration settings required to use the display driver:

- **Init function:** N/A, not used by this driver
- **Function table ptr:** _VPORTDIS_Fxns
- **Function table type:** IOM_Fxns
- **Device id:** 1 for VDP: specify which video port is in use
- **Device params ptr:** Same as for the capture driver
- **Device global data ptr:** N/A, not used by this driver

Please refer to *The TMS320DM642 Video Port Mini-driver (SPRA918A)* and *TMS320C64x DSP Video Port/ VCXO Interpolated Control (VIC) Port Reference Guide* (literature number SPRU629) for a better understanding of the video port driver.

6.6 Ethernet Driver

VDP's Ethernet driver use TMS320C6000 TCP/IP Network Developer's Kit (NDK), version 1.71. NDK is a platform for development and demonstration of network enabled applications on the TMS320C6000 DSP family.

The NDK includes demonstration software showcasing C6000 DSP capabilities across a range of network-enabled applications. In addition, the NDK serves as a rapid prototype platform for the development of network and packet processing applications, or to add network connectivity to the existing DSP applications for communications, configuration and control.

Using the software and hardware components provided in the NDK, developers can quickly move from development concepts to working implementations attached to the network..

For more details on this device driver, see the application note of NDK.

6.7 Serial Port Driver

VDP's serial port driver makes reference to UART device driver of TM320DM642 EVM. The use of the software UART is simulated on a DSP's McBSP channel. These drivers were written in conformance to the DSP/BIOS IOM device driver model and APIs. For details on this device driver, see the application note *DSP/BIOS Hardware and Software UART Device Drivers (SPRA882A)*.

7 Flash Programming

7.1 Overview

The DM643 has 4 megabytes of Flash memory mapped into the lower portion of the CE1 space. The CE1 space is configured as 8 bits wide on the VDP and the Flash memory is 8 bits wide. The memory address space available in CE1 space is smaller than the size of the Flash so the FPGA is used to create 3 extended page address lines. These extended address lines are addressable via the EPLD Flash Base Register and default to 000 binary at Reset. The flash memory map is shown in the table below.

Address Range	Page Number	Size (byte)	Usage
0X90000000-0X9007FFFF	Page 0	1K	Bootloader
		511K	Reserved
	Page 1	512K	User's DSP application
	Page 2	512K	
	Page 3	512K	
	Page 4	512K	
	Page 5	512K	
	Page 6	512K	Software for online upgrade
	Page 7	512K	

Table 8 Flash Memory Map

When the DM643 comes out of reset, the 1Kbyte of data which contain bootloader is copied to the beginning (at address 0) of the DM643's internal memory from beginning of Flash memory (CE1, address 0x90000000) and then run it. When bootloader is executed, it copies user's DSP application (from page 1 to page 5) or online upgrade software (from page 6 to page 7) to properly area and run it, the state of upgrade switch decides which program should be copied by the bootloader should copy.

7.2 Manually Flash Update

If user needs to update flash memory by manual, they must burn all of three codes to correct address of the flash memory shown above by the Flashburn utility.

Before burning your application in flash memory, you have to convert your application .out to .hex. For that, you can use hex6x utility. For more details on the hex utility, please see the Code Composer help under Code Generation Tools → Hex Conversion Utility.

7.3 On-line Flash Upgrade

When bootloader copy online upgrade software (from page 6 to page 7) to run, only user's DSP application in Flash (from page 1 to page 5) is re-freshened.