VoverlayHD SDK User Manual

For

Writing Software to Real-time Overlay PC-Generated High-Definition Text, Graphics & Video On High-Definition SDI & HDMI Video Input

Version 1.0.0.0

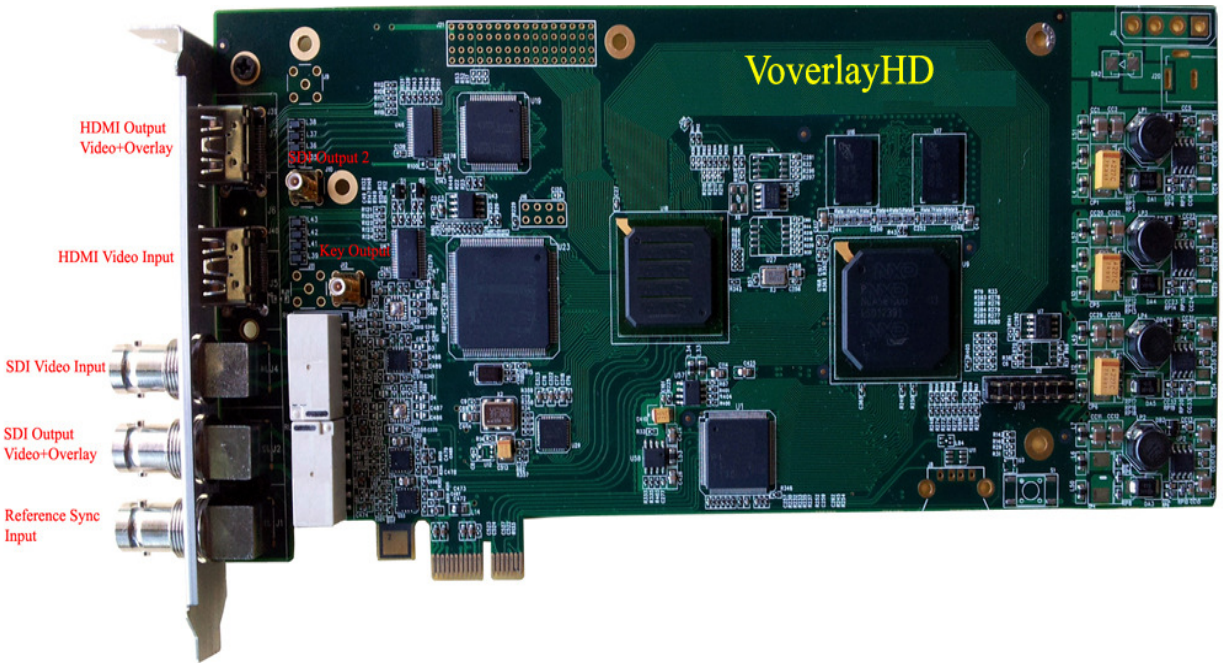


Table of Contents

1. Introduction

VoverlayHD SDK (Software Development Kit) is for rapidly developing application software using **VoverlayHD** High-Definition Overlay PCI-Express card. **VoverlayHD SDK** shields the software developers from coding complicated hardware interfacing by supplying highly integrated and easy to use C-styled function calls. Using **VoverlayHD SDK**, software developers of any experience level can quickly write application software to control one or more **VoverlayHD** cards on the same PC to output PC-generated HD/SD colour text, graphics and video onto external HD TV or HD video input devices, using C++, C# , VB or other programming languages.

The main functions of the **VoverlayHD** PCIe card include Realtime overlaying PC-generated colour text, graphics and video onto HD or SD video input signal, and Realtime displaying these text, graphics & video on its High-Definition SDI and HDMI video output ports together with (or without) the input signal.

VoverlayHD PCIe card mixes (overlays, superimposes) the graphics with live video using on-board dedicated hardware ICs without involving host PC's CPU/GPU, allowing application software to directly read and write on-board 32-bit per pixel(ARGB) memory locations whose contents will instantly mix with the incoming video signal, and output this mixing result at the video output ports connected to HD TV, Bluray-recorder, HD camcorder, HD video capture card, etc. With such a straightforward memory-mapped graphical overlay mechanism, application software can utilise the full power of the MS Window's GDI/GDI+, Direct3D etc programming interfaces to generate un-limited graphical objects to appear on external HD video devices instantly, without any other hardware/software processing.

Multiple VoverlayHD cards (maximum 16) can be installed and programmed on the same PC to output different text/graphics/video contents onto multiple external devices/ports simultaneously.

VoverlayHD SDK works on 32-bit/64-bit **MS Windows XP**, **Vista**, **Windows 7** & **Windows 8** OS.

2. The VoverlayHD Card Working Principle

Several important factors make it possible to deliver PC-generated High-Definition text/graphics/video onto the SDI & HDMI video output ports of the **VoverlayHD** card:

2.1 Overlay Memory:

This is an on-board (the VoverlayHD card) memory area of 1920 X 1080 X 4 bytes that holds the PC generated graphical data (colour pixels) to display on the output video ports mixed with the input video signal. Every 4-byte of memory in this area represent an overlay colour pixel: the highest byte represents the "Alpha Channel"(See below), the second highest byte represents the Red colour, the third highest byte the Green colour, and the lowest byte represents the Blue colour. The 1920 X 1080 X4 memory is for displaying one frame of 1920X1080-Pixel Resolution HD video signal. When the input video signal resolution is lower than 1920X1080-Pixels, e.g. 1280X720-Pixels or 720X480-Pixels, a smaller portion of this overlay memory area is used.

2.2 Alpha Value:

As mentioned above, every pixel representing colour to be overlaid onto incoming video signal has an "Alpha" byte: its values control how much visibility the overlaid graphical pixel has in comparison with the video signal pixel underneath it (at the same X/Y position) carried by the external video input: an Alpha value 255 (hex 0xFF) means the PC-generated pixel is completely in front of the underneath external video pixel, while an Alpha value 0 means the PC-generated pixel is completely invisible (the

external video signal pixel is fully exposed). Any Alpha value between 254 and 1 will see some degree of mixing PC-generated overlay pixel with external incoming video carried pixel: bigger Alpha value means more visibility for the PC-generated overlay pixel.

2.3 Video Input and Output Ports:

VoverlayHD card uses its SDI or HDMI video input port to accept external video signal, mixing them in realtime with PC-generated text/graphics/video overlay data, then send the mixed result to the video output ports (simultaneously on the output SDI and HDMI ports) immediately, thus displaying a Video + Overlay signal at any external TV etc devices connected at the output ports. Furthermore, the output ports can be configured as to output video only, overlay only, or video + overlay by a simple SDK function call (SetMixMode).

To display one graphical pixel generated by PC onto a (X,Y) location on the video output frame of the **VoverlayHD** card, the application software needs to copy 4-byte colour data to the (X,Y) location of the Overlay Memory Area: the highest byte is the Alpha value, and next 3 bytes represent the RGB colour of the overlay pixel. Writing a zero Alpha value to a pixel's overlay position will clear its overlay content therefore fully exposing the input video (if any) at that position.

3. VoverlayHD.DLL and VoverlayHD SDK

The **VoverlayHD SDK** is based on a dynamic linking library **VoverlayHD.DLL,** which supports all the **VoverlayHD SDK** function calls the application software might use. **VoverlayHD.DLL** and its associated files as listed at the end of this manual need to be installed on the target PC running your application software.

4. VoverlayHD SDK Function List

4.1 Card Start and Stop Functions

-- bool **VOVERLAYHD_Initial**(
UINT uCardNo=0, BOOL bClearScreen= true, PVOID *cardRAM = 0);

Function: Initialize one **VoverlayHD Card** in the PC.

Parameters:

uCardNo: Card number, from 0 to 15

bClearScreen: True to clear all overlay output on all output ports during the init process

cardRAM: If not supplied as NULL, on successful return this will point to the card memory representing overlay contents with every 4-Bytes for one pixel's ARGB value: highest byte is Alpha, 2nd highest byte is Red, 3rd-highest is Green and lowest byte is Blue. If mapping this variable to a 4-byte long integer 2-dimentional array then each array member's location will decide the location of the pixel on the video frame, e.g. for a 1920X1080-Pixel video frame mapping this **cardRAM** to a 1920X1080 integer array then each array member (x, y) will represent the pixel (x, y)'s overlay ARGB value. Application software can safely read from or write to these memory locations: writing to a pixel's location will make the ARGB value to appear instantly on output video, reading from a pixle's location will get the ARGB value of that location(a 0x00000000 means no overlay).

Writing a zero ARGB value to a pixel position will clear its overlay thus making the video at that position fully visible.

Note 1. This **cardRAM** value will become invalid once **VOVERLAYHD_Close** is called**.**
Note 2. When input video mode is **progressive** such as 1280X720P50 or 1920X1080P60 etc. there will be a double memory area for each input frame used alternatively, so writing overlay contents for progressive video signals should be done twice to avoid flickering: to write ARGB value to pixel position (x, y) the position (x + video-width, y + video-height) should also be written with the same ARGB value, e.g. when input is 1280X720P writing the same ARGB to position (120, 500) and (120+1280, 500+720) should be done together. If these two positions are written with different ARGB values then the pixel will have a flickering effect: the two overlay values will appear alternatively on video surface.
When input signal is **interlaced (non-progressive)** such as 1920X1080i or 720X480i then this double memory area strategy is not used by the **VoverlayHD** card.

Return Value: True if successful
Note: This function must be called first and return successfully before a VoverlayHD Card and most of the other SDK functions can be used.

-- bool **VOVERLAYHD_InitialEvent**(UINT uCardNo, LPCTSTR szEvent);
Function: Assign an event name to an initialized **VoverlayHD Card**, Return non-zero for success.
Parameters:
uCardNo: Card number, from 0 to 15
szEvent: Event name, can be any string, usually the words "Global Event" + Card Number.
Return Value: True if successful
Note: At least one call to this function must follow the **VOVERLAYHD_Initial()** call to the same card before any other calls can be used on that card.

-- bool **VOVERLAYHD_Close**(UINT uCardNo);
Function: De-initialize a **VoverlayHD Card** to release resources allocated to it. Return true for success.
Parameters:
uCardNo: Card number, from 0 to 15
Note: Before exiting software this function must be called for every VoverlayHD Card that has called **VOVERLAYHD_Initial**, or memory leak will happen**.**
Return Value: True if successful.

-- void * **VOVERLAYHD_GetEvent**(UINT uCardNo, LPCTSTR szEvent);
Function: Return the event name that was created by calling **VOVERLAYHD_InitialEvent**.
Parameters:
uCardNo: Card number, from 0 to 15
szEvent: Event name, as being used in **VOVERLAYHD_InitialEvent.**
Return Value: If successful return the event name.

-- bool **VOVERLAYHD_IsCardInited**(UINT uCardNo);
Function: Test if a card has been initiated properly.
Parameters:
uCardNo: Card number, from 0 to 15
Return Value: True if this card has been initialised.

4.2 Card Configuration Functions

When a **VoverlayHD** card is configured using functions described in this section, the configuration will be remembered by the card so next time when it is started, event after a PC shutdown or loss of power, the previously set configurations will still be used.

-- bool **VOVERLAYHD_GetSignalStatus**(UINT uCardNo , ULONG &stat);

Function: Test if there are video signals at the video input sockets

Parameters:

uCardNo: Card number, from 0 to 15

stat: On success, **stat** bit 1(lowest) indicates if **SDI Input** has signal, **stat** bit 2 indicates if **HDMI Input** has signal.
If any of these input has signal their corresponding bit in **stat** will be 1, otherwise their corresponding bit in **stat** will be 0.

Return Value: True for success.

-- bool **VOVERLAYHD_SetVideoMode**(UINT uCardNo, ULONG mode);

Function: Set the incoming video mode at the currently selected input video port SDI or HDMI

Parameters:

uCardNo: Card number, from 0 to 15

mode: 0x00 ~ 0x12 as defined below (**P** means **P**rogressive, **I** means **I**nterlaced)

```
#define VIDEO_MODE_1920_1080_50I        (0x00)
#define VIDEO_MODE_720_576_50I          (0x01)
#define VIDEO_MODE_1920_1080_60I        (0x02)
#define VIDEO_MODE_720_480_60I          (0x03)
#define VIDEO_MODE_1920_1080_24P        (0x04)
#define VIDEO_MODE_1920_1080_25P        (0x05)
#define VIDEO_MODE_1920_1080_30P        (0x06)
#define VIDEO_MODE_1280_720_50P         (0x07)
#define VIDEO_MODE_1280_720_60P         (0x08)
#define VIDEO_MODE_1920_1080_5994I      (0x09)
#define VIDEO_MODE_1920_1080_2397P      (0x0A)
#define VIDEO_MODE_1920_1080_2997P      (0x0B)
#define VIDEO_MODE_1280_720_5994P       (0x0C)
#define VIDEO_MODE_1280_720_25P         (0x0D)
#define VIDEO_MODE_1280_720_2997P       (0x0E)
#define VIDEO_MODE_1280_720_30P         (0x0F)
#define VIDEO_MODE_1920_1080_50P        (0x10)
#define VIDEO_MODE_1920_1080_60P        (0x11)
#define VIDEO_MODE_1920_1080_5994P      (0x12)
```

Return Value: true for success.

Note 1: The **mode** set by this function must match the actual input signal mode to make the card to function properly, the card cannot automatically detect and configure itself to match a random input signal mode.

Note 2: The **mode** set here will also be the card's **output** mode, i.e., video input and output mode will be the same for the card.

Note 3: The **mode** set by this function will not take effect until the VoverlayHD card is reset: When the card has been initialized, this means calling function sequence **VOVERLAYHD_Close(), _Initial()** and **_InitialEvent()**.

-- ULONG **VOVERLAYHD_GetVideoMode**(UINT uCardNo);
Function: Retrieve the current incoming video mode as set by **VOVERLAYHD_SetVideoMode**
Parameters:
uCardNo: Card number, from 0 to 15
Return Value: As **mode** parameter defined in **VOVERLAYHD_SetVideoMode**
Note: This function merely returns the **mode** as set by **VOVERLAYHD_SetVideoMode**, it cannot automatically detect arbitrary input signal mode.

-- bool **VOVERLAYHD_GetVideoSize**(UINT uCardNo, ULONG &width, ULONG &height);
Function: Retrieve the current video frame width and height as set by **VOVERLAYHD_SetVideoMode** function.
Parameters:
uCardNo: Card number, from 0 to 15
width: Current video frame input/output width in pixels
height: Current video frame input/output height in pixels
Return Value: True for success.

-- bool **VOVERLAYHD_SetSyncMode**(UINT uCardNo, ULONG mode);
Function: Set Sync mode (the clock used to generate timing for incoming video signal)
Parameters:
uCardNo: Card number, from 0 to 15
mode: **0** – Internal Sync (the clock on-board VoverlayHD Card)
1 – External Sync (the clock from the **Reference Input BNC** socket)
2 – Video Sync (the clock embedded in incoming video signal)
Return Value: true for successful
Note: **1.** For HDMI Input use Internal sync, for SDI Input use Video sync. When no input video is available use Internal sync. External Sync can only be used when a Y luminance or CVBS signal with embedded timing is connected at the Reference Input BNC socket.
2. If non-internal sync mode is set but the set sync signal is not available then internal sync is used automatically.

-- ULONG **VOVERLAYHD_GetSyncMode**(UINT uCardNo);
Function: Get Sync mode (the clock used to generate timing)
Parameters:
uCardNo: Card number, from 0 to 15
Return Value: Same as defined in the **mode** parameter in **VOVERLAYHD_SetSyncMode**.

-- bool **VOVERLAYHD_SetVideoInputSocket**(UINT uCardNo, ULONG src = 0);
Function: Select SDI or HDMI source as Input
Parameters:
uCardNo: Card number, from 0 to 15
src: = 0: SDI as input, = 1: HDMI as input
Return Value: True for success.

-- bool **VOVERLAYHD_GetVideoInputSocket**(UINT uCardNo, ULONG & src);
Function: Get current input source
Parameters:
uCardNo: Card number, from 0 to 15
src: = 0: SDI as input, = 1: HDMI as input
Return Value: True for success.

-- bool **VOVERLAYHD_SetMixMode**(UINT uCardNo, ULONG mode);
Function: Set overlay content and input video mixing mode
Parameters:
uCardNo: Card number, from 0 to 15
mode: **0** – video without overlay
1 – overlay is on video
2 – overlay is on blackness
Return Value: True for success.
Note: This setting will affect video output and retrieved video frame data.

-- bool **VOVERLAYHD_GetMixMode**(UINT uCardNo, ULONG & mode);
Function: Get overlay content and input video mixing mode
Parameters:
uCardNo: Card number, from 0 to 15
mode: **0** – video without overlay
1 – overlay is on video
2 – overlay is on blackness
Return Value: True for success.

-- bool **VOVERLAYHD_EnableVideoOutput**(UINT uCardNo, UINT enable = 1);
Function: Enable video output at output ports (SDI and HDMI output ports)
Parameters:
uCardNo: Card number, from 0 to 15
enable: 1 to enable, 0 to disable video output
Return Value: True for success.
Note: This setting will affect all video output ports

-- bool **VOVERLAYHD_SetVideoOutputEnable**(UINT uCardNo, ULONG enable);
Function: Enable video output at output ports (SDI and HDMI output ports)
Parameters:
uCardNo: Card number, from 0 to 15
enable: 1 to enable, 0 to disable video output
Return Value: True for success.
Note: This setting will affect all video output ports

-- bool **VOVERLAYHD_GetVideoOutputEnable**(UINT uCardNo, ULONG & enable);
Function: Get the Enable/Disable video output status at output ports (SDI and HDMI output ports)
Parameters:
uCardNo: Card number, from 0 to 15
enable: Same as in **VOVERLAYHD_SetVideoOutputEnable**
Return Value: true for success.

-- ULONG **VOVERLAYHD_GetLinePhase**(UINT uCardNo);
Function: Get Current Line(Horizontal) Phase
Parameters:
uCardNo: Card number, from 0 to 15
Return Value: 0~2639 valid, -1: failure.

-- bool **VOVERLAYHD_SetLinePhase**(UINT uCardNo, ULONG linePhase);
Function: Set Current Line Phase

Parameters:
uCardNo: Card number, from 0 to 15
linePhase: 0~2639 valid
Return Value: True for success.

-- ULONG **VOVERLAYHD_GetFieldPhase**(UINT uCardNo);
Function: Get Current Field(Vertical) Phase
Parameters:
uCardNo: Card number, from 0 to 15
Return Value: 0~1124 valid, -1: failure.

-- bool **VOVERLAYHD_SetFieldPhase**(UINT uCardNo, ULONG fieldPhase);
Function: Set Current Field Phase
Parameters:
uCardNo: Card number, from 0 to 15
fieldPhase: 0~1124 valid
Return Value: True for success.

-- ULONG **VOVERLAYHD_GetKeyDelay**(UINT uCardNo);
Function: Get Current Key Delay
Parameters:
uCardNo: Card number, from 0 to 15
Return Value: 0~999 valid, -1: failure.

-- bool **VOVERLAYHD_SetKeyDelay** (UINT uCardNo, ULONG keyDelay);
Function: Set Current Key Delay
Parameters:
uCardNo: Card number, from 0 to 15
keyDelay: 0~999 valid
Return Value: True for success.

-- bool **VOVERLAYHD_SetKeyOutputMode**(UINT uCardNo, ULONG keyMode);
Function: Set output mode of the KeyOut BNC socket (on the PCB)
Parameters:
uCardNo: Card number, from 0 to 15
keyMode: 1 = output normal key, 0 = output reversed key, 2 = output filling signal
Return Value: True for success.

-- ULONG **VOVERLAYHD_GetKeyOutputMode**(UINT uCardNo);
Function: Get output mode of the KeyOut BNC socket
Parameters:
uCardNo: Card number, from 0 to 15
Return Value: The **keyMode** value as set by **VOVERLAYHD_SetKeyOutputMode**.

4.3 Overlay Display Functions

-- bool **VOVERLAYHD_LoadTextSingle**(UINT uCardNo, int textBkMode, int Alpha , int TRed, int TGreen, int AlphaBk, int TBlue, int BRed, int BGreen , int BBlue, char *fontName, int fontPoint , char *textString , wchar_t *textStringWide, int textX, int textY, bool transparent , HWND DBWnd, int widthAdjust) ;

Function: Display one text string on output ports

Parameters:

uCardNo: Card number, from 0 to 15

textBkMode: Text display background mode, must be 1 for Transparent, or 2 for Opaque. This has the same meaning as the "*iBkMode*" parameter of the Windows GDI function "**SetBkMode".**

Alpha: The alpha value used to display the text on VoverlayHD Card's output ports, valid values are from 0 (overlay text is fully hidden so if there is external video pixel behind the overlay pixel then the video is fully exposed) to 255(overlay text is fully in front of the video beneath it). Values between 1 and 254 represent different exposure degree of overlaid text in front of the underneath external video (if they exist): higher values means stronger (more visible) overlay content.

TRed: The **Red** colour component for text, valid from 0 to 255

TGreen: The **Green** colour component for text, valid from 0 to 255

TBlue: The **Blue** colour component for text, valid from 0 to 255

AlphaBk: The alpha value used to display the background colour pixels (the spots where no stroke of the text is drawn) for the text on **VoverlayHD Card**'s output ports, valid values are from 0 (background is fully hidden) to 255 (background is fully in front of the video beneath it). Values between 1 and 254 represent different exposure degree of background colour in front of the external video behind(if they exist): higher values means stronger background colour pixels**.** The background Alpha as discussed here only has effects when the **textBkMode** parameter is **Opaque** and the **transparent** parameter is **false.**

BRed: The **Red** colour component for the background pixels, valid from 0 to 255

BGreen: The **Green** colour component for background pixels, valid from 0 to 255

BBlue: The **Blue** colour component for background pixels, valid from 0 to 255

fontName: The font name used to draw the text string, must be present on the PC

fontPoint: The font point used to draw the text string

textString: The ASCII (single-byte) text content to be drawn, must be a null-terminated ASCII string. If this is empty, then Unicode string in **textStringWide** must contain some content and will be used.
If both **textString** and **textStringWide** are empty error occurs.

textStringWide: The Unicode (multi-byte) text to be drawn, must be null-terminated. If both **textString** and **textStringWide** contain contents then **textStringWide** is ignored and single-byte is used.

textX: The X position on the output video area to draw the text

textY: The Y position on the output video area to draw the text

transparent: Do not use the text's background colour to fill the empty spots in-between the text strokes.

DBWnd: If supplied as a valid Window handle, error message will be printed onto this window's client area if this function call failed. Put a NULL here if no error message is to be printed.

widthAdjust: Pixels used to expand the text string's right-most area: this can be useful on

some higher resolution (e.g. 1920X1080-Pixel) input where calculated text string area often clips off some right-most pixels, default is zero.

Return Value: True if successful.

Note: **1.** The "**output video area**" is also the video memory area on board the **VoverlayHD Card** used to hold the pixels drawn by application software before they are displayed on the output video ports: this area is of fixed size for different video signals currently being used at the input ports: 1920X1080, 1280X720, 720X576, 720X480-Pixels, etc.
2. If a text string's end extends beyond the right-edge of the "output video area", the part of the text beyond the right edge will wrap around the screen to appear at the left-end of the video output area: on a TV screen they will appear at the start of the left end of the same line.

-- bool **VOVERLAYHD_GetTextWidthHeight**(UINT uCardNo, char *fontName , int fontPoint , char *textString , int *textWidth , int *textHeight) ;

Function: Retrieve the width and height of the overlay contents in the area as big as occupied by textString using fontName and fontPoint

Parameters:

uCardNo: Card number, from 0 to 15
fontName: The font name used to calculate the text string's width and height
fontPoint: The font point used to calculate the text string's width and height
textString: The text content used to calculate the area
textWidth: hold the returned area's width, must point to a 4-byte integer
textHeight: hold the returned area's height, must point to a 4-byte integer

Return Value: True if successfully retrieved the width and height value.

-- bool **VOVERLAYHD_GetTextWidthHeightWide**(UINT uCardNo, char *fontName , int fontPoint , wchar_t *textString , int *textWidth , int *textHeight) ;

Function: Exactly the same as **VOVERLAYHD_ GetTextWidthHeight** except **textString** is wide-character (Unicode) , can be used with non-empty parameter **textStringWide** in function **VOVERLAYHD_LoadTextSingle.**

-- **void VOVERLAYHD_ClearTextArea**(UINT uCardNo, char *fontName, int fontPoint , char *textString, int textX , int textY) ;

Function: Wipe out overlay contents in the area as big as occupied by textString using fontName and fontPoint on all output ports, from upper left corner (textX, textY)

Parameters:

uCardNo: Card number, from 0 to 15
fontName: The font name used to calculate the text string's width and height
fontPoint: The font point used to calculate the text string's width and height
textString: The text content used to calculate the area
textX: the clear area's upper left corner x co-ordinate: must be < video frame width
textY: the clear area's upper left corner y co-ordinate: must be < video frame height

-- **void VOVERLAYHD_ClearTextAreaWide**(UINT uCardNo, char *fontName, int fontPoint , wchar_t *textString, int textX , int textY) ;

Function: Exactly the same as **VOVERLAYHD_ClearTextArea** except **textString** is wide-character (Unicode) , can be used with non-empty parameter **textStringWide** in function **VOVERLAYHD_LoadTextSingle.**

-- void **VOVERLAYHD_GetOverlayContent**(
UINT uCardNo, ULONG *mem , int x, int y, int width, int height);

Function: Retrieve the overlay contents in the area (x, y, width, height)

Parameters:

uCardNo: Card number, from 0 to 15

mem: the buffer to hold the returned overlay content, must be >= width * height * 4 bytes

x: the area's upper left corner x co-ordinate: must be < current video frame width

y: the area's upper left corner y co-ordinate: must be < current video frame height

width: the area's width in pixels. x + width must be < current video frame width

height: the area's height in pixels. y + height must be < current video frame height

Note: **1**.**mem** must have enough space(width * height * 4 Bytes) to hold the returned content or this function will crash!

2. If x= 0, y = 0, width =0 and height =0 then return the entire overlay content according to the current video input frame. In this case the **mem** must be at least video frame width x height x 4 bytes.

-- ULONG **VOVERLAYHD_StartTimer**(UINT uCardNo , int timeBkMode ,
int Alpha , int TRed , int TGreen , int TBlue ,
int AlphaBk , int BRed , int BGreen, int BBlue ,
char *fontName, int fontPoint, int timeX, int timeY,
bool transparent ,
bool clearPrevTimeDisplay,
bool displayDate = true,
int displayMS_FN = 1,
unsigned int timerInterval = 1000,
HWND DBWnd = 0,
int widthAdjust = 0);

Function: Start to display timer as overlay

Parameters:

uCardNo: Card number, from 0 to 15

timeBkMode: same as described for **VOVERLAYHD_LoadTextSingle**

Alpha , TRed , TGreen , TBlue ,
AlphaBk , BRed , BGreen, BBlue ,
***fontName, fontPoint, timeX, timeY,**

transparent : These parameters are exactly the same as described for **VOVERLAYHD_LoadTextSingle**

clearPrevTimeDisplay: if true, the previously started time display will be wiped out when new timer defined by this call starts

displayDate: True to display date with time

displayMS_FN: 1 = display mille-seconds following the second's position,
2 = display Frame Number following the second's position
0 = display neither of them

timerInterval: The interval to display time in mille-seconds, must be >= 1

DBWnd: If supplied as a valid Window handle, error message will be printed onto this window's client area if this function call failed. Put a NULL here if no error message is to be printed.

widthAdjust: Pixels used to expand the timer string's right-most area: this can be useful on some higher resolution (e.g. 1920X1080-Pixel) input where calculated timer string area often clips off some right-most pixels, default is zero.

Return Value: If succeeds, return the timer ID(non-zero) that can be passed to Windows SDK's KillTimer function**.** If fails, return zero.

Note: 1. To create a transparent time display on live video (only time ticks without any background Colour displayed), use Black Background((**BRed**, **BGreen**, **BBlue**) = (0,0,0)), **transparent** = false, **AlphaBk** = -1, and **timeBkMode** = 2 (OPAQUE)

2. To create a time display with a half-transparent background on a live video (the background is some colour half-transparent on top of the video), use **transparent** = false, **AlphaBk** = the desired background colour (left-most/highest byte must be zero), and **timeBkMode** = 2 (OPAQUE)

3. Using SDK, each VoverlayHD Card can have at most one timer display at any time, although application software can create their own timer without using SDK's timer function.

-- void **VOVERLAYHD_StopTimer**(UINT uCardNo, bool clearPrevTimeDisplay);

Function: Stop to display timer as started by **VOVERLAYHD_StartTimer** on card uCardNo

Parameters:

uCardNo: Card number, from 0 to 15

clearPrevTimeDisplay: true to wipe out the current time display.

-- bool **VOVERLAYHD_IsTimerOn**(UINT uCardNo);

Function: If card uCardNo is displaying time

Parameters:

uCardNo: Card number, from 0 to 15

Return Value: True if this card is displaying time (has called **VOVERLAYHD_StartTimer).**

-- bool **VOVERLAYHD_LoadTargaImageFile**(UINT uCardNo, unsigned char *imageFileName,
int Alpha, int putImageX, int putImageY,
int *imageWidth = NULL, int *imageHeight = NULL);

Function: Display a Targa (.tga) graphics file's contents on all output ports

Parameters:

uCardNo: Card number, from 0 to 15

imageFileName: the Targa file's full path and name inc. disk drive letter

Alpha: Alpha value for displaying (0~255), bigger value means stronger overlay display

putImageX: the horizontal location in pixels of the starting position to display the graphics

putImageY: the vertical location in pixels of the starting position to display the graphics

imageWidth: if not NULL, will receive the Targa image's width on a successful return

imageHeight: if not NULL, will receive the Targa image's height on a successful return

Return Value: True if the loading and displaying succeed.

-- bool **VOVERLAYHD_LoadImageFile**(UINT uCardNo, char *imageFileName,
int Alpha, int putImageX, int putImageY, int putImageWidth, int putImageHeight, bool transparent , ULONG transparencyKey, DWORD rop,
bool clearOldOverlay) ;

Function: Display a BMP/JPG/GIF/PNG/TIF graphics file's contents on all output ports

Parameters:

uCardNo: Card number, from 0 to 15

imageFileName: the graphics file's full path and name inc. disk drive letter

Alpha: Alpha value for displaying (0~255), bigger value means stronger overlay display

putImageX: the horizontal location in pixels of the starting position to display the graphics

putImageY: the vertical location in pixels of the starting position to display the graphics

putImageWidth: width of the displayed image, 0 means using the graphic file's original width

putImageHeight: height of the displayed image, 0 means using the graphic file's original height

transparent: true to make the pixels with the **transparencyKey** parameter's value invisible

transparencyKey: if "**transparent**" is true, graphics pixels with this colour will not be displayed

rop: same as the dwRop(Raster Operation Code) parameter of the BitBlt function in the MS Windows SDK: defining how the graphics file's pixels are combined with the pixels previously being displayed on the same positions to achieve the final overlay result. The default value is **SRCCOPY**: copy graphics pixel over to replace the original pixel.

clearOldOverlay: This value is only meaningful when parameter "**transparent**" is TRUE:
If **clearOldOverlay** is **TRUE**, then those pixels in the graphics file where the colour value equal to the "**transparencyKey**" colour will become totally transparent, i.e., their alpha value will be set to zero.
If **clearOldOverlay** is **FALSE** (this is default), then those pixels in the graphics file where the colour value equal to the "**transparencyKey**" colour will combine (logical or) their old alpha value with the new "**Alpha**"parameter value passed by this function call.

DBWnd:If supplied as a valid Window handle, error message will be printed onto this window's client area if this function call failed. Put a NULL here if no error message is to be printed.

Return Value: True if the loading and displaying succeed.

Note: **1.** If a graphics file's content has a dimension (width by height) larger than the current overlay memory area (i.e. larger than the current input video frame), it is better to use third-party image processing software such as MS Paint, Adobe PhotoShop etc to create a shrunk image file before calling this function to display the graphics as overlay, because the built-in graphics shrinking mechanism often create a file with distorted colour.
2. The type of graphics file is determined by its file extension: .**bmp** for BITMAP, **.gif** for GIF, .**jpg** for JPEG, **.png** for PNG, and .**tif** for TIFF. Animated GIF is not supported.
3. The 4-byte long "**transparencyKey**" value represents RGB colour, with the highest byte un-used, the Red byte at the second highest position (23~16 bit), the Green byte the next highest position (15~8 bit), and the Blue byte at the lowest position (7~0 bit). Note this arrangement of RGB colour components is different from the COLORREF value used in Windows SDK, where the Red byte is at the lowest bit position.
4. The Raster Operation Code (rop) determines the combination of pixels from the graphics file and from the previously drawn overlay pixels on the same position, un-related to the input video's pixels on the same positions.

5. To achieve clear "blue screen" effect (chroma-key effect) when overlaying a graphics file with some smooth background colour on to live video, so that the background area will become completely transparent, set parameter "**clearOldOverlay**" to TRUE.

-- void **VOVERLAYHD_SetAreaAlphaColour**(UINT uCardNo, int Alpha , int x, int y,
int width, int height, int Red , int Green, int Blue,
bool transparent = false,
ULONG transparencyKey = 0,
ULONG TKErrorRange = 0);

Function: Change the alpha and/or colour values of an overlay memory area

Parameters:

uCardNo: Card number, from 0 to 15

Alpha: New alpha blending value for the overlay area, valid 0~255

x: the X-co-ordinate of the upper-left corner of the overlay area

y: the Y-co-ordinate of the upper-left corner of the overlay area

width: the width in pixels of the overlay area

height: the height in pixels of the overlay area

Red: the Red colour byte of the new colour for all the pixels within the overlay area

Green: the Green colour byte of the new colour for all the pixels within the overlay area

Blue: the Blue colour byte of the new colour for all the pixels within the overlay area

transparent: If this is FALSE(default) then change all the pixels' alpha/colour in the area.
If this is TRUE then do not change the colour values of the pixels whose RGB colour values and "**transparencyKey**" RGB colour value have minimum difference --- that is:
abs(Rp - Rt) + abs(Gp - Gt) + abs(Bp - Bt) <= **TKErrorRange**
where abs(X) means the absolute value of X,
Rp/Gp/Bp mean the RGB value of the pixel,
Rt/Gt/Bt mean the RGB value of the colour **transparencyKey**
(the lower 24 bits)

transparencyKey: as explained in the **transparent** parameter above

TKErrorRange: as explained in the **transparent** parameter above.

Note: To set the Alpha value without changing the colours of the overlay area, pass -1 to the Red, Green and Blue parameters, that is, if Red=Green=Blue=-1 then only set the alpha value of the area defined by (x,y)(width,height) without changing the pixel value of the area, otherwise set both the alpha and the (R,G,B) colour of pixels in the area.

-- void **VOVERLAYHD_GetAreaAlphaColour**(
UINT uCardNo, int x, int y, int width, int height, ULONG *buffer)

Function: Retrieve the alpha and colour values of an overlay memory area

Parameters:

uCardNo: Card number, from 0 to 15

x: the X-co-ordinate of the upper-left corner of the overlay area

y: the Y-co-ordinate of the upper-left corner of the overlay area

width: the width in pixels of the overlay area

height: the height in pixels of the overlay area

buffer: buffer to hold the retrieved data, must be at least **width** X **height** X 4 bytes

Note: Every 4-Byte retrieved in "buffer" represent the Alpha (the highest byte),

Red (the next highest byte), the Green (the third highest byte) and the Blue (the lowest byte) value of one pixel.

-- void **VOVERLAYHD_MoveArea**(UINT uCardNo, int Alpha, int sx, int sy, int width , int height, int dx, int dy, bool transparent, bool noErase);

Function: Copy the overlay contents from source location to destination location with alpha blending

Parameters:

uCardNo: Card number, from 0 to 15

Alpha: alpha value used to display the source pixels at destination location. valid 0~255

sx: the X-co-ordinate of the upper-left corner of the source overlay area

sy: the Y-co-ordinate of the upper-left corner of the source overlay area

width: the width in pixels of the source overlay area

height: the height in pixels of the source overlay area

dx: the X-co-ordinate of the upper-left corner of the destination overlay area

dy: the Y-co-ordinate of the upper-left corner of the destination overlay area

transparent: If true, and the source pixel is empty(no overlay content, i.e. black overlay colour), the destination pixel will also become empty (fully exposing the underneath input video content), regardless the **Alpha** value. If set false, then the **Alpha** value will be used to blend with the empty overlay(black colour) onto the destination location: this will show some degree of black colour on a non-black video background.

noErase: If true, the source area overlay contents remain unchanged, if false they are erased.

-- bool **VOVERLAYHD_RotateArea**(UINT uCardNo, int &x, int &y, int &width, int &height, int angle, bool noErase = false, bool fillMissingPixels = true);

Function: clockwise rotate the overlay content of rectangle area (x,y)~(x+width, y+height) with **angle** using the area's upper left corner point (**x,y**) as rotation origin.

Parameters:

uCardNo: Card number, from 0 to 15

x, y: the upper-left corner of the overlay area to be rotated within the current video frame

width: width in pixel of the area to be rotated

height: height in pixel of the area to be rotated

angle: angle in degree (within 1 ~ 359) of the rotation. 0 or 360 will do nothing.

noErase: True will not erase old image, default is to erase the old image after rotation

fillMissingPixels: True (Default) will automatically use adjacent non-empty overlay pixel content to fill the middle area's zero overlay pixel values in the rotated area, suitable for rotating areas containing solid blocks with single colours such as text, otherwise(**fillMissingPixels** == False) the rotated area's middle part of overlay contents usually contain empty overlay pixles caused by floating point calculation errors when converting rotated x/y co-ordinates to discreet integer pixel positions.

Note: **1**. After a successful rotation the **x/y/width/height** values will be updated to the newly rotated overlay area's bounding rectangle's upper left corner and width/height.

2. Any rotated pixels falling off the video frame's edges will be cut-off (not displayed).

-- bool **VOVERLAYHD_LoadBitmapPixels**(UINT uCardNo, int Alpha, bool transparent , ULONG transparencyKeyColour, char *pixels , int bytesPerPixel, bool top_down , int x, int y, int width, int height);

Function: Display a bitmap's pixels at some overlay memory area

Parameters:

uCardNo: Card number, from 0 to 15
Alpha: alpha blending value used to display the bitmap pixels at the overlay memory area.
transparent: true to make the pixels with the **transparencyKey** parameter's value invisible
transparencyKey: if "**transparent**" is true, bitmap pixels with this colour will not be displayed
pixels: buffer holding the bitmap's pixels
bytesPerPixel: either 3 or 4, number of bytes per pixel in the "**pixels**" buffer
top_down: if the scan-lines in "pixels" are arranged as 1st-line at the lower memory
x: the X-co-ordinate of the upper-left corner of the overlay area to display the bitmap
y: the Y-co-ordinate of the upper-left corner of the overlay area to display the bitmap
width: the width in pixels of the overlay area to display the bitmap
height: the height in pixels of the overlay area to display the bitmap

Return Value: True if succeeded

Note: The "scan-lines" of the bitmap pixels are counted from the upper-left corner as line 1, line 2… towards the bottom of the bitmap. If the "**top_down**" parameter is true, the pixels representing each scan line will also be arranged in buffer "**pixels**" from low memory addresses to high addresses according to scan-line1, scan-line2, … scan-lineN, where "N" is the total number of pixel lines in the bitmap. However, if the "**top_down**" parameter is false, the lowest memory addresses in "**pixels**" memory area will hold the scan-line**N,** then the next lowest addresses will hold scan-line(**N-1**), …, and the highest "**pixels**" memory addresses will hold scan-line1.

-- bool **VOVERLAYHD_LoadImageFromWindow**(UINT uCardNo , HWND wnd , int Alpha,
int putImageX, int putImageY, int putImageWidth, int putImageHeight,
DWORD rop , bool ClientAreaOnly, bool transparent, ULONG transparencyKey,
ULONG repeatTimes, ULONG pauseMS, bool clearOldOverlay,
HWND DBWnd);

Function: Display a Window's image at some overlay position

Parameters:

uCardNo: Card number, from 0 to 15
wnd: the handle of the window whose image is to be displayed
Alpha: alpha value used to display the window's image at the overlay memory area.
putImageX: X-position of the upper-left corner of the overlay area to display the window
putImageY: Y-position of the upper-left corner of the overlay area to display the window
putImageWidth: the width in pixels of the overlay area to display the window's image
putImageHeight: the height in pixels of the overlay area to display the window's image
rop: Raster Operation Code to combine the window's pixels with existing overlay pixels
ClientAreaOnly: true to only display the Client Area image of the window
transparent: true to make the pixels with the **transparencyKey** parameter's value invisible
transparencyKey: if "**transparent**" is true, window pixels with this colour will not be displayed
repeatTimes: How many times to display the window image before ending this function
pauseMS: In mille-seconds, the pause period in-between displaying the image repeatedly
clearOldOverlay: This value is only meaningful when parameter "**transparent**" is TRUE:
If **clearOldOverlay** is **TRUE**, then those pixels in the Window where the colour value equal to the "**transparencyKey**" colour will become totally transparent, i.e., their alpha value will be set to zero.
If **clearOldOverlay** is **FALSE** (this is default), then those pixels in the Window where the colour value equal to the "**transparencyKey**" colour will combine (logical or) their old alpha value with the new "**Alpha**" parameter value passed by this function call.

DBWnd: If is a Window handle, error message will be printed onto this window's client area if this function call failed. Put a NULL here if no error message is to be printed.

Return Value: true if succeeds.

Note: **1**. This function is suitable to display a window that has static contents. To display a window that has constantly changing images such as an animation or video playback window, use **VOVERLAYHD_LoadImageFromWindowOnThread** function**.**
2. To achieve clear "blue screen" (chroma-key) effect when the Window has some smooth unique background colour, so that the background area will become completely transparent, set parameter "**clearOldOverlay**" to TRUE.

-- HANDLE **VOVERLAYHD_LoadImageFromWindowOnThread**(UINT uCardNo,
HWND wnd , int Alpha, int putImageX, int putImageY,
int putImageWidth, int putImageHeight, DWORD rop,
bool ClientAreaOnly, bool transparent , ULONG transparencyKey ,
ULONG threadRunTime , ULONG threadPauseTime, int threadPriority,
bool eraseOnExit, UINT exitMsg, HWND parentWnd, bool clearOldOverlay,
ULONG TKErrorRange, HWND DBWnd);

Function: Start a separate thread to continuously display a Window's image at overlay memory area

Parameters:

uCardNo: Card number, from 0 to 15

wnd: the handle of the window whose image is to be displayed

Alpha: alpha value used to display the window's image at the overlay memory area.

putImageX: X-position of the upper-left corner of the overlay area to display the window

putImageY: Y-position of the upper-left corner of the overlay area to display the window

putImageWidth: the width in pixels of the overlay area to display the window's image

putImageHeight: the height in pixels of the overlay area to display the window's image

rop: Raster Operation Code to combine the window's pixels with existing overlay pixels

ClientAreaOnly: true to only display the Client Area image of the window

transparent: true to make the pixels with the **transparencyKey** parameter's value invisible

transparencyKey: if "**transparent**" is true, window pixels with this colour will not be displayed

threadRunTime: In seconds, how long the thread will run.
If zero is supplied then the thread will run forever until being killed by calling **VOVERLAYHD_StopImageFromWindowOnThread,** or application exits.

threadPauseTime: In mille-seconds, the thread pause time in-between updating the window's image. Longer pause time will use less CPU but might cause a rapidly changing window's image(such as video window) to appear less smooth.

threadPriority: thread running priority,
normally "THREAD_PRIORITY_NORMAL" (0), but can also be other THREAD_PRIORITY_XX values as defined in Windows SDK's winbase.h file.

eraseOnExit: True to erase the window's image when the thread exit.

exitMsg: Any WIN_USER or higher message sent to "parentWnd" when the thread ends

parentWnd: handle of the window to receive the **exitMsg** message when the thread ends

clearOldOverlay: This value is only meaningful when parameter "**transparent**" is TRUE:
If **clearOldOverlay** is **TRUE**, then those pixels in the Window whose colour and the "**transparencyKey**" colour have minimum difference --- that is:
abs(Rp - Rt) + abs(Gp - Gt) + abs(Bp - Bt) <= **TKErrorRange**
where abs(X) means the absolute value of X,
Rp/Gp/Bp mean the RGB value of the pixel,
Rt/Gt/Bt mean the RGB value of the colour **transparencyKey**

(the lower 24 bits)
will become totally transparent, i.e., their alpha value will be set to zero. If **clearOldOverlay** is **FALSE** (this is default), then those pixels in the Window where the colour and the "**transparencyKey**" colour have minimum difference as described above will combine (logical or) their old alpha value with the new "**Alpha**"parameter value passed by this function call.

TKErrorRange: used to calculate colour difference as in the **clearOldOverlay** parameter

DBWnd: If is a Window handle, error message will be printed onto this window's client area if this function call failed. Put a NULL here if no error message is to be printed.

Return Value: If succeeds, the thread handle (4-Byte long) that can be passed to function **VOVERLAYHD_StopImageFromWindowOnThread** to stop the thread. Zero means failure.

Note: **1.** This function is useful to continuously display a content-changing window on input video, such as a video or animation playback window. To simply display a static window content, the function **VOVERLAYHD_LoadImageFromWindow** can be used.
2. To achieve clear "blue screen" (chroma-ley) effect when the Window has some smooth unique background colour, so that the background area will become completely transparent, set parameter "**clearOldOverlay**" to TRUE.

-- bool **VOVERLAYHD_StopImageFromWindowOnThread**(
UINT uCardNo, HANDLE hThread , bool eraseOnExit, ULONG waitTime = 0);

Function: Stop a thread started by calling **VOVERLAYHD_LoadImageFromWindowOnThread**

Parameters:

uCardNo: Card number, from 0 to 15

hThread: the thread handle returned by **VOVERLAYHD_LoadImageFromWindowOnThread**

eraseOnExit: true to erase the window's image when ending the thread

waitTime: In mille-seconds, time to wait before exiting this function (to make sure the thread ends)

Return value: true if succeeds.

-- bool **VOVERLAYHD_ClearCurrentScreen**(UINT uCardNo = 0, , ULONG colour = 0);

Function: Paint all overlay contents on all output ports using a colour.

Parameters:

uCardNo: Card number, from 0 to 15

colour: The ARGB colour painted to all pixles of the video, default is 0 which makes all pixels cleared of overlay contents, thus exposing all incoming video if they exist.

Return Value: True if successful

Note: **1.** After wiping out overlay contents with colour==0, incoming video signal will fully appear on the video output ports. If there is no incoming video the video output ports will show total blackness.
2. If parameter **colour** is not zero then the entire output frame will show this colour in front of the input video signal, e.g., if **colour** is 0xFF00FF00, then the entire output video will be green without seeing any video signal underneath since Alpha is 255; if **colour** is 0x77FFFF00 then the entire output frame is covered with half-transparent yellow overlay.

-- void **VOVERLAYHD_ClearArea**(
UINT uCardNo, int x, int y, int width, int height, ULONG colour);

Function: Fill overlay contents in the area (x, y, width, high) with "colour" on all output ports

Parameters:

uCardNo: Card number, from 0 to 15

x: the clear area's upper left corner x co-ordinate: must be < current input signal width

y: the clear area's upper left corner y co-ordinate: must be < current input signal height

width: the clear area's width in pixels. x + width must be < current input signal width

height: the clear area's height in pixels. y + height must be < current input signal height

Note: **1.** After wiping out overlay contents with colour==0, incoming video signal will fully appear in the area bounded by (x, y, width, height) . If there is no incoming video the area will be black.

2. If parameter **colour** is not zero then the (x, y, width, height) area will show this colour in front of the input video signal, e.g., if **colour** is 0xFF0000FF, then the area will be blue without seeing any video signal underneath since Alpha is 255; if **colour** is 0x77FF0000 then the area is covered with half-transparent red overlay.

4.4 Video Frame Retrieval Functions

These functions can be used to obtain individual video frames from the **VoverlayHD** card for still image creation and continuous video preview or analysis operation.

-- bool **VOVERLAYHD_GetAFrameData**(UINT uCardNo, ULONG* value, HWND DBWnd);

Function: Retrieve one input video frame's YUV 4:2:2 pixel data as still image

Parameters:

uCardNo: Card number, from 0 to 15

value: memory to store the video pixel data,
must be >= this card's current Video Frame's width X height X 2 Bytes

DBWnd: If is a Window handle, error message will be printed onto this window's client area if this function call failed. Put a NULL here if no error message is to be printed.

Return value: true if succeeds.

Note: 1. Retrieved frame data is of packed YUV 4:2:2 format, can be converted to RGB format by calling function **VOVERLAYHD_YUVto24RGB** as described below

2. This function is suitable to retrieve individual frame data as still image infrequently. Although this function retrieves the same full frame video data as function **VOVERLAYHD_GetPreviewFrame** after **VOVERLAYHD_SetFullFramePreview** is called with parameter fmtType == 1, long time repeated calling **VOVERLAYHD_GetAFrameData** is not recommended since that can disturb driver and SDK causing unexpected result.

-- bool **VOVERLAYHD_GetPreviewFrame**(UINT uCardNo, ULONG* value,
UINT& width, UINT& height, BOOL &format,
HWND DBWnd);

Function: Retrieve one input video frame's YUV 4:2:2 pixel data for continuous video preview

Parameters:

uCardNo: Card number, from 0 to 15

value: memory to store the video pixel data,
must must be >= this card's current Video Frame's width X height X 2 Bytes

width: return the retrieved video frame width,
this is either the current frame's full width or its ¼ width, depending on function **VOVERLAYHD_SetFullFramePreview** described below

height: return the retrieved video frame height,
this is either the current frame's full height or its ¼ height, depending on function **VOVERLAYHD_SetFullFramePreview** described below

format: Currently not being used

DBWnd: If is a Window handle, error message will be printed onto this window's client area if this function call failed. Put a NULL here if no error message is to be printed.

Return value: true if succeeds.

Note: 1. Retrieved data is in packed YUV 4:2:2 format that can be converted to RGB format by function **VOVERLAYHD_YUVto24RGB** or supplied to 3rd party analysis programs.

2. This function is suitable to be called repeatedly to retrieve continuous frames for live video preview or analysis purpose, e.g. implementing dynamic blue-screen/chroma-key effects can use the retrieved pixel values to decide if to overlay background graphics/PC-video onto the pixels' positions.

3. When full width/height is returned, the retrieved video frame represents the full input resolution video frame although the function takes some time to return. When ¼ of width/height is returned, the retrieved video frame represent a quarter of the full input resolution video frame, i.e. it is a shrunk image, but the function returns quickly.

4. This function can only succeed once function **VOVERLAYHD_SetPreviewEnable**(uCardNo, TRUE) is called.

5. Examples using this function can be found in the CVideoPreview.CPP source file in the **VoverlayHD.exe** program that allows live video preview. Note after retrieving frame data the CVideoPreview.CPP uses Direct3D functions to display frames under Windows Vista/7/8, but uses DirectDraw functions to display frames under Windows XP.

-- bool **VOVERLAYHD_SetFullFramePreview** (UINT uCardNo, UINT fmtType);

Function: Set the retrieved width/height length for function **VOVERLAYHD_GetPreviewFrame**

Parameters:

uCardNo: Card number, from 0 to 15

fmtType: 0 means the returned width/height will be ¼ of current video frame's width/height when function **VOVERLAYHD_GetPreviewFrame** is called; 1 means the returned width/height will be current video frame's full width/height when function **VOVERLAYHD_GetPreviewFrame** is called, that is, the full video frame data is returned by **VOVERLAYHD_GetPreviewFrame**

Return value: true if succeeds.

-- bool **VOVERLAYHD_SetPreviewEnable**(UINT uCardNo, BOOL Enable = TRUE);

Function: Enable/Disable successful calling to function **VOVERLAYHD_GetPreviewFrame**

Parameters:

uCardNo: Card number, from 0 to 15

Enable: TRUE means subsequent calling to function **VOVERLAYHD_GetPreviewFrame** will succeed

Return value: true if succeeds.

-- bool **VOVERLAYHD_GetPreviewEnable**(UINT uCardNo, BOOL & Enable);

Function: Retrieve the status for successful calling function **VOVERLAYHD_GetPreviewFrame**

Parameters:

uCardNo: Card number, from 0 to 15

Enable: Same as in **VOVERLAYHD_SetPreviewEnable**

Return value: true if succeeds.

4.5 Utility Functions

-- char * **VOVERLAYHD_GetSDKVer**(void);

Function: Return the version of the **VOVERLAY** SDK as a string constant such as "1.0.0.0".

-- int **VOVERLAYHD_GetCardNum** (void);

Function: Return the total number of installed **VoverlayHD** Cards in the PC.

Note: 1. This function depends on the properly installed device drivers of the VoverlayHD Cards, not the physical cards themselves. So if one **VoverlayHD** Card's driver is not installed properly then that card will not be counted as being present in the PC, even though that **VoverlayHD Card** sits in a PCIe slot.

2. The SDK supports maximum 16 **VoverlayHD Card**s in one PC.

Return Value: the number of the VoverlayHD Card in this PC, 0 means no card.

-- ULONG **VOVERLAYHD_GetCardModel**(UINT uCardNo);

Function: Return the VoverlayHD Card model

Return: 1: SDI-HD card, 2: SDI-3G card, -1: failure.

-- bool **VOVERLAYHD_GetCardVer**(UINT uCardNo,
ULONG & nCardHardVer,
ULONG & nCardSoftVer,
ULONG & nCardTypeNo);

Function: Retrieve the card hardware related information.

Parameters:

uCardNo: Card number, from 0 to 15

nCardHardVer: Return Hardware version

nCardSoftVer: Return Firmware version

nCardTypeNo: No significance currently.

Return Value: True for success.

-- bool **VOVERLAYHD_YUVto24RGB**(
ULONG *YUVBuf,
ULONG *RGBBuf,
int Width,
int Height,
int Format = 1,
BYTE Alpha = 0xFF);

Function: Convert packed YUV 4:2:2 image inside **YUVBuf** to RGB inside **RGBBuf**

Parameters:

uCardNo: Card number, from 0 to 15

YUVBuf: must be >= Width X Height X 2 Bytes long holding the YUV 4:2:2 data

RGBBuf: must be >= Width X Height X 4 Bytes long receiving the converted RGB data, Each pixel has 4-bytes with the highest byte set to **Alpha**, the rest will get the correponding pixle's RGB values.

Width: Width in pixel of the image to be converted

Height: Height in pixel of the image to be converted

Format: 1 for UYVY (VoverlayHD always uses this), or 0 for YUY2(not used by SDK)

Alpha: Alpha value assigned to all pixels in **RGBBuf**

Return Value: True for success.

Note: This function can be called any time regarless VoverlayHD card situation

-- bool **VOVERLAYHD_SaveRGBBitmap**(
char *szFilename, char *pBuf, int fileFormat, ULONG width, ULONG height, ULONG RGBSize, int deinterlace = 0, ULONG cutoffBottomRows = 0);

Function: Save RGB bitmap data to file.

Parameters:

szFilename: file name to be saved to, existing file will be deleted without warning

pBuf: image buffer pBuf[width * height * 4] is the ARGB pixels in 32-bit per pixel format

fileFormat: 0=BMP, 1=JPG, 2=GIF(WinXP)/TIF(Win7/8), 3=PNG

width: image width in pixels

height: image height in pixels

RGBSize: size in bytes of **pBuf**, must be >= **width** * **height** * 4

deinterlace: 0 = No deinterlace(default);
1 = use even field only (copy 0, 2, 4, ... lines to 1, 3, ... lines);
2 = use odd field only (copy 1,3,5... lines to 0, 2, 4... lines);

cutoffBottomRows: number of rows at the bottom of the frame to be cut off (will be filled with blackness), default is 0(none zero will not affect the height of the created image file). This is useful to cut off noises at the image bottom.

Return Value: True for success.
Note: This function can be called any time regardless VoverlayHD card situation

-- bool **VOVERLAYHD_SetCardErrPrint**(UINT uCardNo, bool printErr);
Function: Set the card's printing error flag.
Parameters:
uCardNo: Card number, from 0 to 15
printErr: True to print error message to **DBWnd** window passed by functions like VOVERLAYHD _LoadTextSingle, VOVERLAYHD_LoadImageFile, VOVERLAYHD_StartTimer etc. when they fail. Default is False.
Return Value: True for success.

-- bool **VOVERLAYHD_GetCardErrPrint**(UINT uCardNo, bool &printErr);
Function: Get the card's printing error flag.
Parameters:
uCardNo: Card number, from 0 to 15
printErr: Same as in **VOVERLAYHD_SetCardErrPrint**
Return Value: True for success.

-- bool **VOVERLAYHD_SetEEPROMByte**(UINT uCardNo, ULONG eeprom_addr, BYTE byte);
Function: Write the card's eeprom data.
Parameters:
uCardNo: Card number, from 0 to 15
eeprom_addr: eeprom address, must be within 0 ~ 0x2F.
byte: data to write at **eeprom_addr.**
Return Value: True for success.
Note: **1**. eeprom has some limited write time (approx. < 1 million times) so do not use this eeprom write function for long time repeated data updates: it is only intended for occasional hardware ID storage/retrieval, e.g. using a unique value for each card's identification or match with proprietary software etc.
2. eeprom write/read is a a slow process, so do not do consecutive write/read too fast.

-- bool **VOVERLAYHD_GetEEPROMByte**(UINT uCardNo, ULONG eeprom_addr, BYTE &byte);
Function: Read the card's eeprom data.
Parameters:
uCardNo: Card number, from 0 to 15
eeprom_addr: eeprom address, must be within 0 ~ 0x2F.
byte: data read from **eeprom_addr** when function is successful.
Return Value: True for success.

5. SDK Function Calling Sequences

(1). **VOVERLAYHD_GetSDKVer**();
VOVERLAYHD_GetCardNum():
VOVERLAYHD_IsCardInited();
VOVERLAYHD_YUVto24RGB();
VOVERLAYHD_SaveRGBBitmap();
these functions can be called anytime anywhere.

(2). **VOVERLAYHD_Initial**();
VOVERLAYHD_InitialEvent();
must be called before all other functions on card number "uCardNo" can be used

(3). **VOVERLAYHD_Close**();
must be called before exiting software for every card that has been initialised

(4). **All other functions:** must be called between
VOVERLAYHD_Initial / VOVERLAYHD_InitialEvent and **VOVERLAYHD_Close.**

6. SDK Operation Requirement

6.1 To use **VoverlayHD** SDK functions to write application software, **VoverlayHD.DLL-related files** must be copied to the development PC's Library and Include search folder: these include all files under the "**lib**"and the "**inc**" folders on the **VoverlayHD** Setup CD.

To run the linked C++ program **VoverlayHD.exe** under the "**exe**" folder of the Setup CD, **VoverlayHD** device driver must be installed from the folder "**driver**" on the Setup CD.

6.2 To prepare a target PC to run your application software written with **VoverlayHD** SDK, copy all the files from "**lib**" folder on the Setup CD to the target PC's execution searchable folder, preferably the same folder as the application software, install the **VoverlayHD Card**'s device driver from the "**driver**" folder. Note the different library files for different versions of Windows, while files under **lib\common** are needed by all versions of Windows.

6.3 The function prototype declaration include file **VOVERLAYHD.H** needs to be used by C++ projects; while VisualBasic projects need to declare individual **VoverlayHD** functions one by one; C# projects need to create a class library including all the SDK functions, as shown in the included C# project.

6.4 The **SDK** works on Microsoft **WindowsXP, Vista, Win7/8** Operating Systems, 32-Bit and 64-Bit.

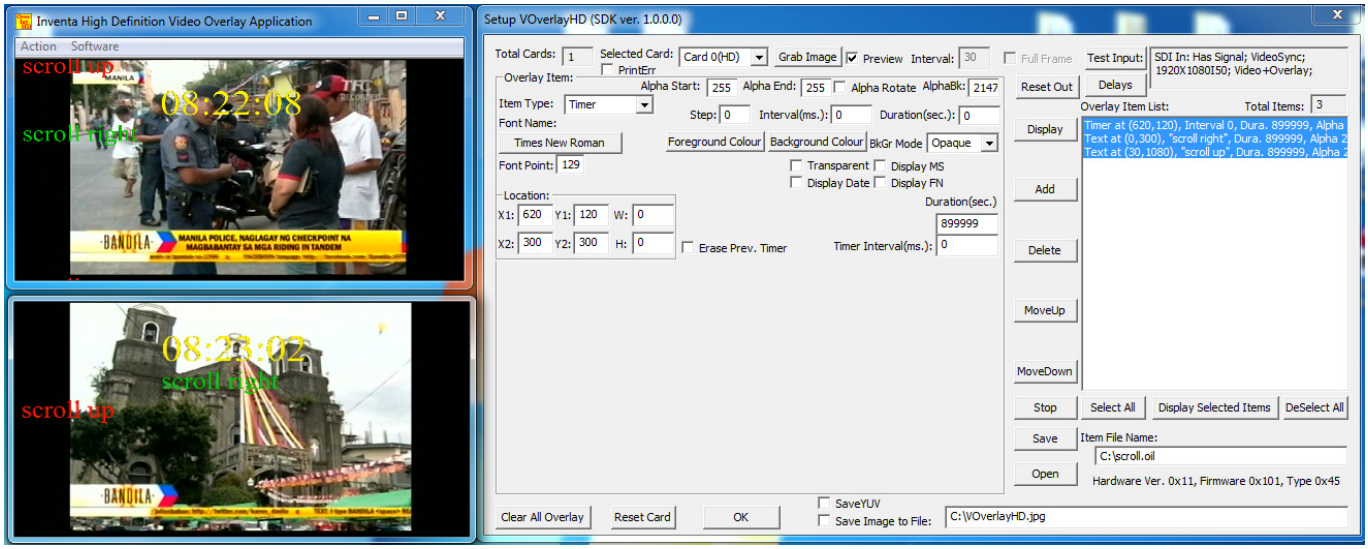
7. Sample Source Code

Fully functioning MS Visual **C++** Application software "**VoverlayHD.EXE**" with full source codes and project files are included under the "**src**" folder of the Setup CD. Also included are **C#** and **VisualBasic** source codes, their VisualStudio projects and compiled executables. All VisualStudio projects are created with VisualStudio 2008 Pro. Note on 64-bit Windows these VisualStudio projects must set "**Platform targets**" to **x86** in their **build** property.

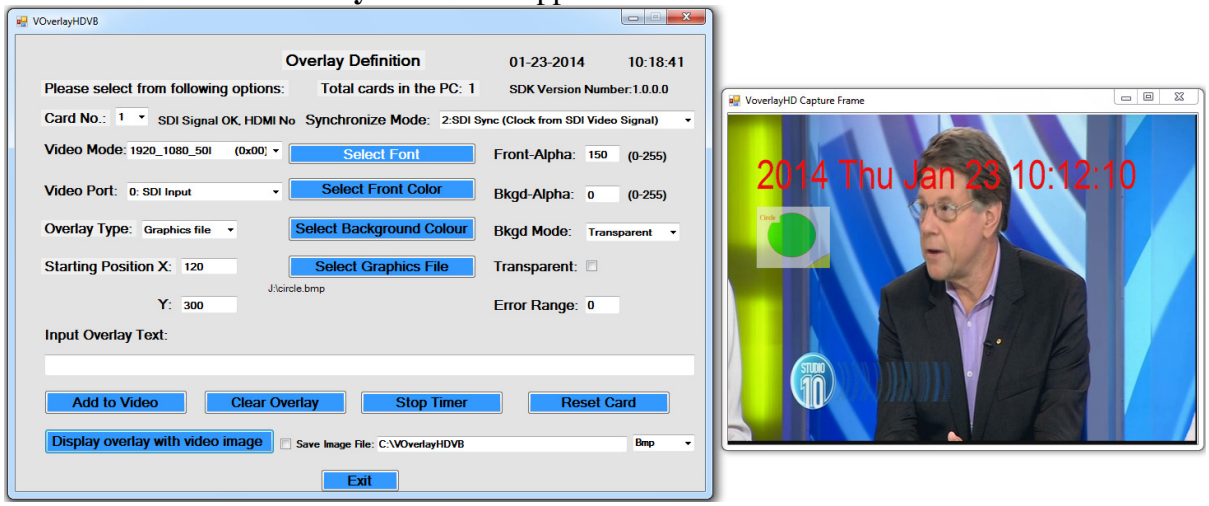
The VisualStudio compiler pre-processor **ABOVE_WINXP** must be defined appropriately for C++ project **VoverlayHD** according to the Windows versions used: 0 for WinXP, 1 for all other Windows.

To re-compile the C++ sample code **VoverlayHD.exe**, MS Windws' SDK (version 7.1) must be installed and its root path pointed to by environment variable **WINSDK**, and MS Windows' DirectX SDK(2010-Feb Version) must be installed and its root directory pointed to by environment variable **DXSDK_DIR**. For different versions of Windows (32-Bit vs 64-Bit, Windows XP vs Windows Vista/7/8 etc.), use different .Dll and .Lib libraries files resided under the **lib** folder on the Setup CD, while the libraries under folder **lib\common** are needed for all versions of Windows.

Screenshot for the **VoverlayHD.exe** application software:



Screenshot for the **VoverlayHDVB.exe** application software:



Screenshots for the **CSharp.exe** application software: