S3C64xx series Video Driver User Manual
(Windows Embedded CE 6.0 & Windows Mobile 6.x)

S3C64xx series

Feb. 11, 2009

(Preliminary) REV 0.5

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typical" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product

S3C64xx series Video Driver User Manual For For Windows Embedded CE 6.0 & Windows Mobile 6.1

Samsung Electronics Co., Ltd.
San #24 Nongseo-Dong, Giheung-Gu
Yongin-City Gyeonggi-Do, Korea
446-711

Home Page: http://www.samsungsemi.com/

E-Mail: mobilesol.cs@samsung.com

Printed in the Republic of Korea

Revision History

Revision No	Description of Change	Refer to	Author(s)	Date
0.1	Preliminary draft	-	GabJoo Lim	2008-04-04
0.2	Revised for Rotator API	-	GabJoo Lim	2008-04-24
0.3	Revised for TV Out API	-	GabJoo Lim	2008-05-03
0.4	Revised for 64xx series common contents		Jiwon Kim	2008-07-04
0.5	Revised for modified directory layout		Jiwon Kim	2009-02-11

NOTE: REVISED PARTS ARE WRITTEN IN BLUE.

Contents

Figures

1 Introduction

Standard APIs(GDI, GAPI, Direct Draw and Direct Show) supported by Windows CE or Windows Mobile OS has some abstraction to eliminate HW dependency. And this abstraction restricts some feature of HW IP. So we introduce the Video Driver supporting the Custom API can control multimedia HW IP directly.

Video Driver supports APIs control 5 multimedia HW IP(Display Controller, Post Processor, Rotator, TV Scaler and TV Encoder). User Application can access and control HW using the APIs and pass the Physical Address as argument of API. But this means application can use Physical Address as an argument and does not means application can access to Physical memory directly. If application want to access to Physical Memory, The Physical Memory Region should be mapped to Virtual Memory region by some system calls like VirtualAlloc() and VirtualCopy().

Display Driver(GDI, Direct Draw) in the BSP is also using Video Driver to control HW IP in the same way of User Application. Application must reserve some HW resource before calling API, because Display Driver or some other application can call APIs of Video Driver at any time.

2 Directory Layout

Directory layout of source code related with Video Driver is shown in Figure 2-1. The all examples including directory name, file name, function name, and variables show in case of S3C6400. In case of S3C6410, “6400” will be replaced to “6410”. All prefix on directory name is removed, Figure 2-2 shows that.

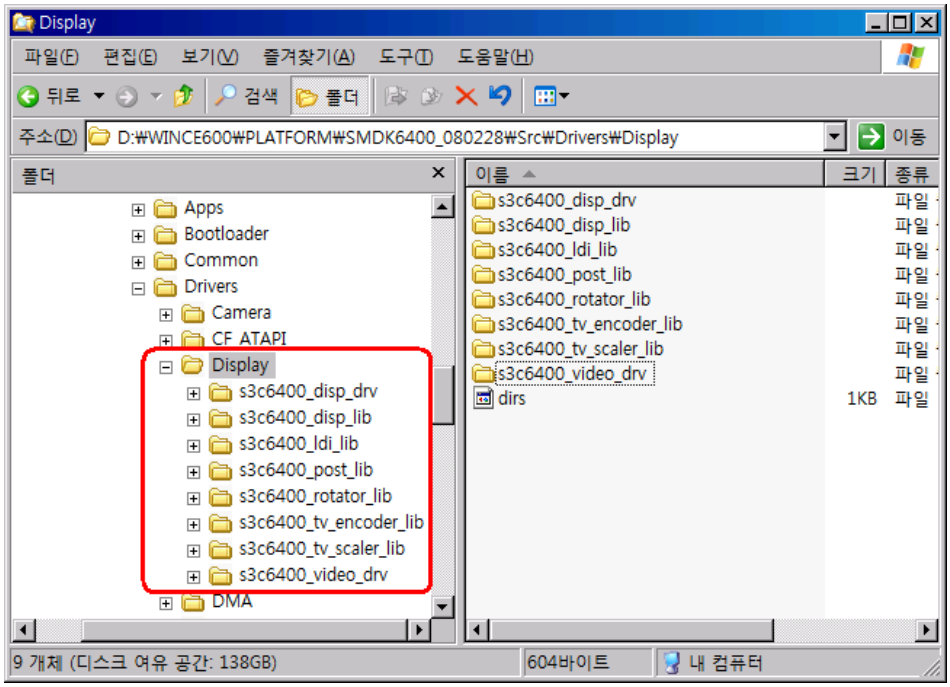

Figure 2-1 Video Driver Directory Layout(Old)

Directory Name	New Directory Name	Description
S3c6400_disp_drv	DISPLAY_DRV	Display Driver (GDI, Direct Draw support)
S3c6400_disp_lib	DISPCON_LIB	Display Controller Function Library
S3c6400_ldi_lib	LDI_LIB	LCD module Function Library
S3c6400_post_lib	POST_PROCESSOR_LIB	Post Processor Function Library
S3c6400_rotator_lib	IMAGE_ROTATOR_LIB	Image Rotator Function Library
S3c6400_tv_encoder_lib	TV_ENCODER_LIB	TV Encoder Function Library
S3c6400_tv_scaler_lib	TV_SCALER_LIB	TV Scaler Function Library
S3c6400_video_drv	VIDEO_DRV	Video Driver

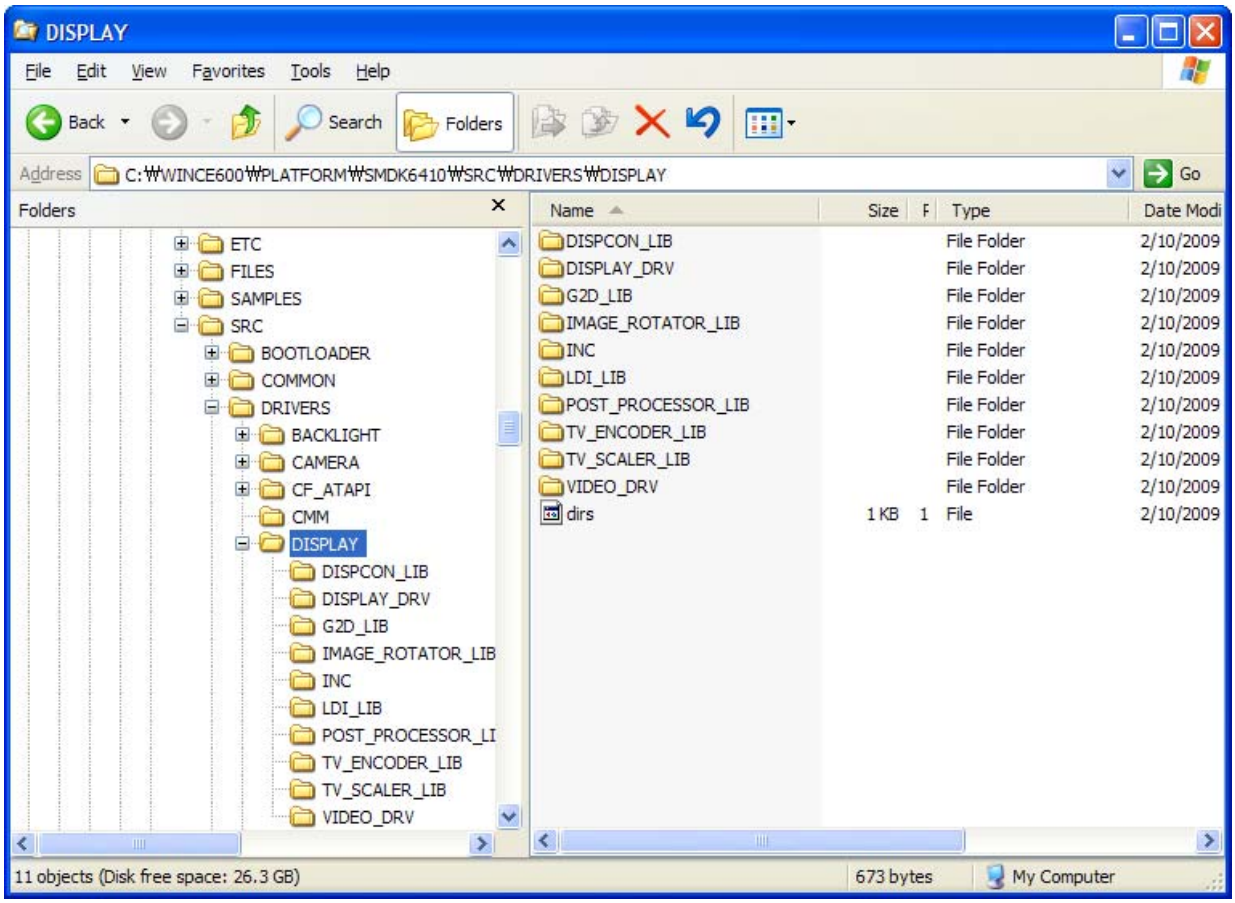

Figure 2-2 Video Driver Directory Layout(New)

3 Driver Loading & Driver Open

To using Video Driver API, The Video Driver should be loaded and each application must get the handle of Video Driver by opening driver. This sequence is same as other stream drivers.

Video Driver is built-in driver and OS load the driver automatically in the OS booting process.

Application can open the driver and get the handle with CreateFile() System Call. Device name of Video Driver is **"VDE0:"**

```
m_hVideoDrv = CreateFile( L"VDE0:", GENERIC_READ|GENERIC_WRITE,
                          FILE_SHARE_READ|FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, 0 );
```

4 Video Driver API

Application can call API of Video Driver with DeviceIoControl() System Call.

```
BOOL DeviceIoControl(
    HANDLE hDevice,
    DWORD dwIoControlCode,
    LPVOID lpInBuffer,
    DWORD nInBufferSize,
    LPVOID lpOutBuffer,
    DWORD nOutBufferSize,
    LPDWORD lpBytesReturned,
    LPOVERLAPPED lpOverlapped
);
```

Video Driver API is classified as Resource Request/Release API, Function API and Power Management API.

Resource Request/Release API is used to reserve some HW IP before application use function API. Function API is used to control some feature of HW IP. And Power Management API is used to control Clock or Block Power of HW IP and Video Engine.

All IOCTL codes, Data structures and Enumeration Values are defined in "<<BSP ROOT>>\Src\Inc\SVEDriverAPI.h" file.

4.1 Resource Request/Release API

Resource Request/Release API need only dwIoControlCode as argument. The reserve states of resource are stored in Video Driver and handle is not passed to application. Video Driver approves access right of application based on open context of application pass to driver.

IOCTL Code	Description
IOCTL_SVE_RSC_REQUEST_FIMD_INTERFACE	Request the resource of Display Controller Output Interface.
IOCTL_SVE_RSC_RELEASE_FIMD_INTERFACE	Release the resource of Display Controller Output Interface.
IOCTL_SVE_RSC_REQUEST_FIMD_WIN0	Request the resource of Display Controller Window0.
IOCTL_SVE_RSC_RELEASE_FIMD_WIN0	Release the resource of Display Controller Window0.
IOCTL_SVE_RSC_REQUEST_FIMD_WIN1	Request the resource of Display Controller Window1.
IOCTL_SVE_RSC_RELEASE_FIMD_WIN1	Release the resource of Display Controller Window1.
IOCTL_SVE_RSC_REQUEST_FIMD_WIN2	Request the resource of Display Controller Window2.
IOCTL_SVE_RSC_RELEASE_FIMD_WIN2	Release the resource of Display Controller Window2.
IOCTL_SVE_RSC_REQUEST_FIMD_WIN3	Request the resource of Display Controller Window3.
IOCTL_SVE_RSC_RELEASE_FIMD_WIN3	Release the resource of Display Controller Window3.
IOCTL_SVE_RSC_REQUEST_FIMD_WIN4	Request the resource of Display Controller Window4.
IOCTL_SVE_RSC_RELEASE_FIMD_WIN4	Release the resource of Display Controller Window4.
IOCTL_SVE_RSC_REQUEST_POST	Request the resource of Post processor.
IOCTL_SVE_RSC_RELEASE_POST	Release the resource of Post processor.
IOCTL_SVE_RSC_REQUEST_ROTATOR	Request the resource of Rotator.
IOCTL_SVE_RSC_RELEASE_ROTATOR	Release the resource of Rotator.
IOCTL_SVE_RSC_REQUEST_TVSCALER_TVENCODER[1]	Request the resource of TV Scaler & TV

	Encoder.
IOCTL_SVE_RSC_RELEASE_TVSCALER_TVENCODER(1)	Release the resource of TV Scaler & TV Encoder.

(1) The resource management API of TV Scaler and TV Encoder is same because TV Encoder can't work without TV Scaler. TV Encoder always needs output of TV Scaler as the input path.

4.2 Function API

Function API is used to control the feature of each HW IP.

Function API supports control each HW IP (Display Controller, Post Processor, Rotator, TV Scaler and TV Encoder). Also supports Local Path API control FIFO mode between Post Processor and Display Controller.

Application must reserve the resource of HW IP by Resource Request API before calling Function API. To use Local Path API, application should reserve resource of Post Processor and Display Controller.

4.2.1 Display Controller API (FIMD)

IOCTL Code	Description
IOCTL_SVE_FIMD_SET_INTERFACE_PARAM	Configuring Parameters of Output Interface (LCD module, TV out)
IOCTL_SVE_FIMD_SET_OUTPUT_RGBIF	Change Display Output to RGB I/F (LCD)
IOCTL_SVE_FIMD_SET_OUTPUT_TV	Change Display Output to TV Encoder (TV Out)
IOCTL_SVE_FIMD_SET_OUTPUT_ENABLE	Display Output Enable
IOCTL_SVE_FIMD_SET_OUTPUT_DISABLE	Display Output Disable
IOCTL_SVE_FIMD_SET_WINDOW_MODE	Configuring Window Parameter (Mode, Format, Size, Position)
IOCTL_SVE_FIMD_SET_WINDOW_POSITION	Change Window Position
IOCTL_SVE_FIMD_SET_WINDOW_FRAMEBUFFER	Configuring Window Frame Buffer Address
IOCTL_SVE_FIMD_SET_WINDOW_COLORMAP(1)	Configuring Window Color Map
IOCTL_SVE_FIMD_SET_WINDOW_ENABLE	Window Enable
IOCTL_SVE_FIMD_SET_WINDOW_DISABLE	Window Disable
IOCTL_SVE_FIMD_SET_WINDOW_BLEND_DISABLE(1)	Disable Window Blending (Always show upper window first)
IOCTL_SVE_FIMD_SET_WINDOW_BLEND_COLORKEY(2)	Configuring Color Key
IOCTL_SVE_FIMD_SET_WINDOW_BLEND_ALPHA(2)	Configuring Alpha Blending
IOCTL_SVE_FIMD_WAIT_FRAME_INTERRUPT(2)	Waiting for Frame Interrupt
IOCTL_SVE_FIMD_GET_OUTPUT_STATUS	Request the status of Display Output
IOCTL_SVE_FIMD_GET_WINDOW_STATUS(1)	Request the status of each Window

(1) Function is not implemented at current version

(2) API can be used without request HW resource at current version

4.2.1.1 IOCTL_SVE_FIMD_SET_INTERFACE_PARAM

API Description	
Configuring Parameters and initializing Output Interface (LCD module, TV out)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_INTERFACE_PARAM
lpInBuffer	SVEARG_FIMD_OUTPUT_IF *
nInBuffer	sizeof(SVEARG_FIMD_OUTPUT_IF)
lpOutBuffer	None
nOutBufferSize	None
Note	
This API is used by Display Driver when initializing Display Controller. The parameters of RGB I/F is filled by library of external LCD Module. In most cases, TVout screen size is same with LCD size (but dwTVOutScreenWidths can not be greater than 720 pixel)	
Data Structure of **SVEARG_FIMD_OUTPUT_IF**	

```
typedef struct
{
  tDevInfo tRGBDevInfo;              // RGB I/F Device Information
  DWORD dwTVOutScreenWidth;          // TV Out Device Screen Width
  DWORD dwTVOutScreenHeight;         // TV Out Device Screen Height
} SVEARG_FIMD_OUTPUT_IF;
```

4.2.1.2 IOCTL_SVE_FIMD_SET_OUTPUT_RGBIF

API Description	
Change Display Output to RGB I/F (LCD)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_OUTPUT_RGBIF
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API is used by Display Driver.	

4.2.1.3 IOCTL_SVE_FIMD_SET_OUTPUT_TV

API Description	
Change Display Output to TV Encoder (TV Out)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_OUTPUT_TV
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API is used by Display Driver.	

4.2.1.4 IOCTL_SVE_FIMD_SET_OUTPUT_ENABLE

API Description	
Display Controller Output Enable	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_OUTPUT_ENABLE
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API is used by Display Driver.	

4.2.1.5 IOCTL_SVE_FIMD_SET_OUTPUT_DISABLE

API Description	
Display Controller Output Disable	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_OUTPUT_DISABLE
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API is used by Display Driver.	

4.2.1.6 IOCTL_SVE_FIMD_SET_WINDOW_MODE

API Description	
Configuring Window Parameter (Mode, Format, Size, Position)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_WINDOW_MODE
lpInBuffer	SVEARG_FIMD_WIN_MODE *
nInBuffer	sizeof(SVEARG_FIMD_WIN_MODE)
lpOutBuffer	None
nOutBufferSize	None
Note	
Only Window0 can support local path with Post Processor at current version	
Data Structure of SVEARG_FIMD_WIN_MODE	

```
typedef struct
{
   DWORD dwWinMode;  // FIMD Window Mode
   DWORD dwBPP;      // BitPerPixel
   DWORD dwWidth;    // Window Horizontal Pixel
   DWORD dwHeight;   // Window Vertical Pixel
   DWORD dwOffsetX;  // Window Horizontal Offset
   DWORD dwOffsetY;  // Window Vertical Offset
} SVEARG_FIMD_WIN_MODE;
```

Member	**Description**	**Possible value or unit**
dwWinMode	Window Number and Operation Mode (DMA or Local Path)	*DISP_WINDOW_MODE*
dwBPP	Frame Buffer Image Format	*DISP_BPP_MODE*
dwWidth	Window Horizontal Size	Pixel
dwHeight	Window Vertical Size	Pixel
dwOffsetX	Window Horizontal Position	Pixel
dwOffsetY	Window Vertical Position	Pixel

4.2.1.7 IOCTL_SVE_FIMD_SET_WINDOW_POSITION

API Description		
Change Window Position		
Arguments	**Type**	
dwIoControlCode	IOCTL_SVE_FIMD_SET_WINDOW_POSITION	
lpInBuffer	SVEARG_FIMD_WIN_POS *	
nInBuffer	sizeof(SVEARG_FIMD_WIN_POS)	
lpOutBuffer	None	
nOutBufferSize	None	
Note		
This API operates asynchronously when display output is enabled, because position of window can be changed only in blank area.		
Data Structure of SVEARG_FIMD_WIN_POS		

typedef struct
{
DWORD dwWinNum; // FIMD Window Number
DWORD dwOffsetX; // Window Horizontal Offset
DWORD dwOffsetY; // Window Vertical Offset
} SVEARG_FIMD_WIN_POS;		

Member	**Description**	**Possible value or unit**
dwWinNum	Window Number	*DISP_WINDOW*
dwOffsetX	Window Horizontal Position	Pixel
dwOffsetY	Window Vertical Position	Pixel

4.2.1.8 IOCTL_SVE_FIMD_SET_WINDOW_FRAMEBUFFER

API Description		
Change Window Position		
Arguments	**Type**	
dwIoControlCode	IOCTL_SVE_FIMD_SET_WINDOW_FRAMEBUFFER	
lpInBuffer	SVEARG_FIMD_WIN_FRAMEBUFFER *	
nInBuffer	sizeof(SVEARG_FIMD_WIN_FRAMEBUFFER)	
lpOutBuffer	None	
nOutBufferSize	None	
Note		
This API operates asynchronously when display output is enabled, because frame buffer address can be changed only in blank area.		
Data Structure of SVEARG_FIMD_WIN_FRAMEBUFFER		

```
typedef struct
{
   DWORD dwWinNum;          // FIMD Window Number
   DWORD dwFrameBuffer;     // Frame Buffer Address
   BOOL bWaitForVSync;      // Blocked Operation
} SVEARG_FIMD_WIN_FRAMEBUFFER;
```

Member	**Description**	**Possible value or unit**
dwWinNum	Window Number	*DISP_WINDOW*
dwFrameBuffer	Frame buffer address	Physical Address
bWaitForVSync	If bWaitForVSync is set as TRUE, Function will be blocked until new address applied to HW	*TRUE* or *FALSE*

4.2.1.9 IOCTL_SVE_FIMD_SET_WINDOW_ENABLE

API Description	
Window Enable	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_WINDOW_ENABLE
lpInBuffer	DWORD *
nInBuffer	sizeof(DWORD)
lpOutBuffer	None
nOutBufferSize	None
Note	
lpInBuffer is the pointer of variable contains window number to enable. Refer to enumeration *DISP_WINDOW*	

4.2.1.10 IOCTL_SVE_FIMD_SET_WINDOW_DISABLE

API Description	
Window Disable	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_WINDOW_DISABLE
lpInBuffer	DWORD *
nInBuffer	sizeof(DWORD)
lpOutBuffer	None
nOutBufferSize	None
Note	
lpInBuffer is the pointer of variable contains window number to disable. Refer to enumeration *DISP_WINDOW*	

4.2.1.11 IOCTL_SVE_FIMD_SET_WINDOW_BLEND_COLORKEY

API Description	
Configuring Color Key	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_SET_WINDOW_BLEND_COLORKEY
lpInBuffer	SVEARG_FIMD_WIN_COLORKEY *
nInBuffer	sizeof(SVEARG_FIMD_WIN_COLORKEY)
lpOutBuffer	None
nOutBufferSize	None

Data Structure of SVEARG_FIMD_WIN_COLORKEY

```
typedef struct
{
  DWORD dwWinNum;          // FIMD Window Number
  DWORD dwDirection;       // Keying Direction
  DWORD dwColorKey;        // Color Key Value
  DWORD dwCompareKey;      // Compare Key Value
  BOOL bOnOff;             // Color Key Enable/Disable
} SVEARG_FIMD_WIN_COLORKEY;
```

Member	**Description**	**Possible value or unit**
dwWinNum	Window Number	*DISP_WINDOW*
dwDirection	Color Key Direction Select which window will be shown above	*DISP_COLOR_KEY_DIRECTION*
dwColorKey	Color key value for the transparent pixel effect	24 bit RGB value
dwCompareKey	Color key mask value	24 bit RGB value
bOnOff	Color key enable or disable	*TRUE* or *FALSE*

4.2.1.12 IOCTL_SVE_FIMD_SET_WINDOW_BLEND_ALPHA

API Description		
Configuring Alpha Blending		
Arguments	**Type**	
dwIoControlCode	IOCTL_SVE_FIMD_SET_WINDOW_BLEND_ALPHA	
lpInBuffer	SVEARG_FIMD_WIN_ALPHA *	
nInBuffer	sizeof(SVEARG_FIMD_WIN_ALPHA)	
lpOutBuffer	None	
nOutBufferSize	None	
Data Structure of SVEARG_FIMD_WIN_ALPHA		

typedef struct
{
DWORD dwWinNum; // FIMD Window Number
DWORD dwMethod; // Blending Method (per Pixel or per Plane)
DWORD dwAlpha0; // Alpha Value 0
DWORD dwAlpha1; // Alpha Value 1
} SVEARG_FIMD_WIN_ALPHA;		

Member	**Description**	**Possible value or unit**
dwWinNum	Window Number	*DISP_WINDOW*
dwMethod	Blending Method Per Pixel or Per Plane	*DISP_ALPHA_BLEND_METHOD*
dwAlpha0	Alpha value 0	4 bit alpha value (0x0~0xF) *0x0* will show lower window *0xF* will show upper window
dwAlpha1	Alpha value 1	4 bit alpha value (0x0~0xF) *0x0* will show lower window *0xF* will show upper window

4.2.1.13 IOCTL_SVE_FIMD_WAIT_FRAME_INTERRUPT

API Description	
Waiting for Frame Interrupt.	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_FIMD_WAIT_FRAME_INTERRUPT
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API can be used when application want to wait for current display frame. Actually frame interrupt is occurred at front porch of display output. This API use WaitForSingleObject() internally to waiting for HW interrupt. So you can release CPU scheduled time until HW operation finished.	

4.2.1.14 IOCTL_SVE_FIMD_GET_OUTPUT_STATUS

API Description		
Request the status of Display Output		
Arguments	**Type**	
dwIoControlCode	IOCTL_SVE_FIMD_GET_OUTPUT_STATUS	
lpInBuffer	None	
nInBuffer	None	
lpOutBuffer	SVEARG_FIMD_OUTPUT_STAT *	
nOutBufferSize	sizeof(SVEARG_FIMD_OUTPUT_STAT)	
Note		
When display output is disabled, the return values except bENVID can be incorrect		
Data Structure of SVEARG_FIMD_WIN_FRAMEBUFFER		

```
typedef struct
{
   DWORD dwLineCnt;            // Line Counter
   DWORD dwVerticalStatus;     // Vertical Status
   DWORD dwHorizontalStatus;   // Horizontal Status
   BOOL bENVID;                // ENVID Field of VIDCON0
} SVEARG_FIMD_OUTPUT_STAT;
```

Member	**Description**	**Possible value or unit**
dwLineCnt	Current line of display output signal	Line number
dwVerticalStatus	Vertical status of display output signal	*DISP_VERTICAL_STATUS*
dwHorizontalStatus	Horizontal status of display output signal	*DISP_HORIZONTAL_STATUS*
bENVID	Display output is enabled or not	*TRUE* or *FALSE*

4.2.2 Post Processor API (Post Processor)

IOCTL Code	Description
IOCTL_SVE_POST_SET_PROCESSING_PARAM	Configuring Parameters of Post processor (Operating mode, source &, destination image size, offset)
IOCTL_SVE_POST_SET_SOURCE_BUFFER	Configuring Source Image Address
IOCTL_SVE_POST_SET_NEXT_SOURCE_BUFFER(1)	Change Next Source Image Address (for Free run mode only)
IOCTL_SVE_POST_SET_DESTINATION_BUFFER	Configuring Destination Image Address
IOCTL_SVE_POST_SET_NEXT_DESTINATION_BUFFER(1)	Change Next Destination Image Address (for Free run mode only)
IOCTL_SVE_POST_SET_PROCESSING_START	Post Processing Start
IOCTL_SVE_POST_SET_PROCESSING_STOP	Post Processing Stop
IOCTL_SVE_POST_WAIT_PROCESSING_DONE(2)	Waiting for Post Processing to be finished. (for Per Frame mode only)
IOCTL_SVE_POST_GET_PROCESSING_STATUS	Request the status of Post Processor

(1) API can be used when Post processor is operating in free run mode.

(2) API can be used when Post processor is operating in per frame mode.

4.2.2.1 IOCTL_SVE_POST_SET_PROCESSING_PARAM

API Description	
Configuring Parameters of Post processor (Operating mode and image format, size, offset)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_POST_SET_PROCESSING_PARAM
lpInBuffer	SVEARG_POST_PARAMETER *
nInBuffer	sizeof(SVEARG_POST_PARAMETER)
lpOutBuffer	None
nOutBufferSize	None

Note

Post Processing parameter can be configured only when HW is not running.

To use post processor with display controller as local path, dwDstType must be configured as *POST_DST_FIFO_YUV444* or *POST_DST_FIFO_RGB888* and dwScanMode must be configured as *POST_PROGRESSIVE.*

dwScanMode is applied to HW when Post processor is configured as FIFO mode (Local Path)

If application changes the parameters of post processor, Application should reconfigure Image Buffer Address also.

Data Structure of SVEARG_POST_PARAMETER

```
typedef struct
{
  DWORD dwOpMode;          // Operation Mode (Frame or Free Run)
  DWORD dwScanMode;        // Scan Mode (Progressive or Interlaced)
  DWORD dwSrcType;         // Src Image Type
  DWORD dwSrcBaseWidth;
  DWORD dwSrcBaseHeight;
  DWORD dwSrcWidth;
  DWORD dwSrcHeight;
  DWORD dwSrcOffsetX;
  DWORD dwSrcOffsetY;
  DWORD dwDstType;         // Dst Image Type
  DWORD dwDstBaseWidth;
  DWORD dwDstBaseHeight;
  DWORD dwDstWidth;
  DWORD dwDstHeight;
  DWORD dwDstOffsetX;
```

DWORD dwDstOffsetY; } SVEARG_POST_PARAMETER;		
Member	**Description**	**Possible value or unit**
dwOpMode	Operating mode Per Frame Mode or Free run mode	*POST_OP_MODE*
dwScanMode	Output scan method Progressive or Interlaced mode	*POST_SCAN_MODE*
dwSrcType	Source Image Format	*POST_SRC_TYPE*
dwSrcBaseWidth	Source Image Width	Pixel
dwSrcBaseHeight	Source Image Height	Pixel
dwSrcWidth	Source Image Clipping Width	Pixel
dwSrcHeight	Source Image Clipping Height	Pixel
dwSrcOffsetX	Horizontal offset of Source Clipping area	Pixel
dwSrcOffsetY	Vertical offset of Source Clipping area	Pixel
dwDstType	Destination Image Format	*POST_DST_TYPE*
dwDstBaseWidth	Destination Image Width	Pixel
dwDstBaseHeight	Destination Image Height	Pixel
dwDstWidth	Destination Image Clipping Width	Pixel
dwDstHeight	Destination Image Clipping Height	Pixel
dwDstOffsetX	Horizontal offset of Destination Clipping area	Pixel
dwDstOffsetY	Vertical offset of Destination Clipping area	Pixel

4.2.2.2 IOCTL_SVE_POST_SET_SOURCE_BUFFER

API Description	
Configuring Source Image Address	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_POST_SET_SOURCE_BUFFER
lpInBuffer	SVEARG_POST_BUFFER *
nInBuffer	sizeof(SVEARG_POST_BUFFER)
lpOutBuffer	None
nOutBufferSize	None
Note	
Source Image Address can be configured only when HW is not running.	
Data Structure of SVEARG_POST_BUFFER	

```
typedef struct
{
  DWORD dwBufferRGBY;
  DWORD dwBufferCb;
  DWORD dwBufferCr;
  BOOL bWaitForVSync;          // Blocked Operation
} SVEARG_POST_BUFFER;
```

Member	**Description**	**Possible value or unit**
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image	Physical Address
bWaitForVSync	Don't care in this API	TRUE or FALSE

4.2.2.3 IOCTL_SVE_POST_SET_NEXT_SOURCE_BUFFER

API Description	
Change Next Source Image Address	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_POST_SET_NEXT_SOURCE_BUFFER
lpInBuffer	SVEARG_POST_BUFFER *
nInBuffer	sizeof(SVEARG_POST_BUFFER)
lpOutBuffer	None
nOutBufferSize	None
Note	
Setting Next address can be used only when Post processor is running in free run mode. If user configure parameter as per frame mode, next address will be ignored This API operates asynchronously when post processor is running, because next address value is updated to address in HW when current operation is finished.	
Data Structure of SVEARG_POST_BUFFER	

```
typedef struct
{
   DWORD dwBufferRGBY;
   DWORD dwBufferCb;
   DWORD dwBufferCr;
   BOOL bWaitForVSync;          // Blocked Operation
} SVEARG_POST_BUFFER;
```

Member	**Description**	**Possible value or unit**
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image	Physical Address
bWaitForVSync	If bWaitForVSync is set as TRUE, Function will be blocked until new address applied to HW	*TRUE* or *FALSE*

4.2.2.4 IOCTL_SVE_POST_SET_DESTINATION_BUFFER

API Description		
Configuring Destination Image Address		
Arguments	**Type**	
dwIoControlCode	IOCTL_SVE_POST_SET_DESTINATION_BUFFER	
lpInBuffer	SVEARG_POST_BUFFER *	
nInBuffer	sizeof(SVEARG_POST_BUFFER)	
lpOutBuffer	None	
nOutBufferSize	None	
Note		
Destination Image Address can be configured only when HW is not running. Destination Image address is ignored when post processor is running as Local path.		
Data Structure of SVEARG_POST_BUFFER		

typedef struct
{
DWORD dwBufferRGBY;
DWORD dwBufferCb;
DWORD dwBufferCr;
BOOL bWaitForVSync; // Blocked Operation
} SVEARG_POST_BUFFER;		

Member	**Description**	**Possible value or unit**
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image	Physical Address
bWaitForVSync	Don't care in this API	TRUE or FALSE

4.2.2.5 IOCTL_SVE_POST_SET_NEXT_DESTINATION_BUFFER

API Description	
Change Next Destination Image Address	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_POST_SET_NEXT_DESTINATION_BUFFER
lpInBuffer	SVEARG_POST_BUFFER *
nInBuffer	sizeof(SVEARG_POST_BUFFER)
lpOutBuffer	None
nOutBufferSize	None

Note

Setting Next address can be used only when Post processor is running in free run mode. If user configure parameter as per frame mode, next address will be ignored

This API operates asynchronously when post processor is running, because next address value is updated to address in HW when current operation is finished.

Destination Image address is ignored when post processor is running as Local path.

Data Structure of SVEARG_POST_BUFFER

```
typedef struct
{
  DWORD dwBufferRGBY;
  DWORD dwBufferCb;
  DWORD dwBufferCr;
  BOOL bWaitForVSync;          // Blocked Operation
} SVEARG_POST_BUFFER;
```

Member	Description	Possible value or unit
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image	Physical Address
bWaitForVSync	If bWaitForVSync is set as TRUE, Function will be blocked until new address applied to HW	*TRUE* or *FALSE*

4.2.2.6 IOCTL_SVE_POST_SET_PROCESSING_START

API Description	
Post Processing Start	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_POST_SET_PROCESSING_START
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
Post processing can start after last operation is finished. User can wait for finishing last operation with *IOCTL_SVE_POST_WAIT_PROCESSING_DONE* API	

4.2.2.7 IOCTL_SVE_POST_SET_PROCESSING_STOP

API Description	
Post Processing Stop	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_POST_SET_PROCESSING_STOP
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API can't stop Post processing immediately because Post Processor can not be terminated before DMA operation is completed. When Post Processor is configured as Per Frame mode, HW will stop operation automatically after finishing one frame.	

4.2.2.8 IOCTL_SVE_POST_WAIT_PROCESSING_DONE

API Description	
Waiting for Post Processing to be finished. (for Per Frame Mode Only)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_POST_WAIT_PROCESSING_DONE
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API can be used when application want to wait for finishing last post processing. This API should be used before reconfiguring Post Processing Parameter or Address. Do not use this API when Post processor is configured as Free run mode. In Free run mode, Post processor run continually. This API use WaitForSingleObject() internally to waiting for HW interrupt. So you can release CPU scheduled time until HW operation finished.	

4.2.2.9 IOCTL_SVE_POST_GET_PROCESSING_STATUS

API Description	
Request the status of Post Processing	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_POST_GET_PROCESSING_STATUS
lpInBuffer	None
nInBuffer	None
lpOutBuffer	DWORD *
nOutBufferSize	sizeof(DWORD)
Note	
lpInBuffer is the pointer of variable contains status value. Refer to enumeration *POST_STATE*. (0: *POST_IDLE*, 1: *POST_BUSY*)	

4.2.3 Local Path API (FIMD + Post Processor)

IOCTL Code	Description
IOCTL_SVE_LOCALPATH_SET_WIN0_START	Start the local path between Post Processor and FIMD Window0 (Post Processor Start & Window0 enable)
IOCTL_SVE_LOCALPATH_SET_WIN0_STOP	Stop the local path between Post Processor and FIMD Window0 (Post Processor Stop & Window0 disable)
IOCTL_SVE_LOCALPATH_SET_WIN1_START(1)	Start the local path between Camera Interface Preview Path and FIMD Window1 (Preview Path MSDMA Start & Window1 enable)
IOCTL_SVE_LOCALPATH_SET_WIN1_STOP(1)	Stop the local path between Camera Interface Preview Path and FIMD Window1 (Preview Path MSDMA Stop & Window1 disable)
IOCTL_SVE_LOCALPATH_SET_WIN2_START(1)	Start the local path between Camera Interface Codec Path and FIMD Window2 (Codec Path MSDMA Start & Window2 enable)
IOCTL_SVE_LOCALPATH_SET_WIN2_STOP(1)	Stop the local path between Camera Interface Codec Path and FIMD Window2 (Codec Path MSDMA Stop & Window2 disable)

(1) Function is not implemented at current version

4.2.3.1 IOCTL_SVE_LOCALPATH_SET_WIN0_START

API Description	
Starting Local Path between Post Processor and FIMD Window0 (Post Processor Start & Window0 enable)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_LOCALPATH_SET_WIN0_START
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
Before using this API, Post Processor and Display Window0 should be configured properly for Local Path. Post Processing parameter should be configured dwDstType as *POST_DST_FIFO_YUV444* or *POST_DST_FIFO_RGB888* for Local Path Display Window0 mode should be configured dwWinMode as *DISP_WIN0_POST_RGB* or *DISP_WIN0_POST_YUV* for Local Path Do not stop local path with *IOCTL_SVE_FIMD_SET_WINDOW_DISABLE* or *IOCTL_SVE_POST_SET_PROCESSING_STOP*, Unless Display Controller will be unpredictable state.	

4.2.3.2 IOCTL_SVE_LOCALPATH_SET_WIN0_STOP

API Description	
Stopping Local Path between Post Processor and FIMD Window0 (Post Processor Stop & Window0 disable)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_LOCALPATH_SET_WIN0_STOP
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API should be used when Local Path is enabled. Do not stop local path with *IOCTL_SVE_FIMD_SET_WINDOW_DISABLE* or *IOCTL_SVE_POST_SET_PROCESSING_STOP*, Unless Display Controller will be unpredictable state.	

4.2.4 Rotator API (Image Rotator)

IOCTL Code	Description
IOCTL_SVE_ROTATOR_SET_OPERATION_PARAM	Configuring Image Format, Rotation Angle and Source Image Size Parameter for Rotator Operation
IOCTL_SVE_ROTATOR_SET_SOURCE_BUFFER	Configuring Buffer Address of Source Image
IOCTL_SVE_ROTATOR_SET_DESTINATION_BUFFER	Configuring Buffer Address of Destination Image
IOCTL_SVE_ROTATOR_SET_OPERATION_START	Starting Rotator Operation
IOCTL_SVE_ROTATOR_SET_OPERATION_STOP(1)	Stopping Rotator Operation.
IOCTL_SVE_ROTATOR_WAIT_OPERATION_DONE	Waiting for Rotator Operation to be finished
IOCTL_SVE_ROTATOR_GET_STATUS(2)	Request the status of Rotator Operation

(1) Actually *IOCTL_SVE_ROTATOR_SET_OPERATION_STOP* API is same with *IOCTL_SVE_ROTATOR_WAIT_OPERATION_DONE* API. Because Rotator can not be terminated before DMA operation is completed.

(2) Function is not implemented at current version

4.2.4.1 IOCTL_SVE_ROTATOR_SET_OPERATION_PARAM

API Description	
Configuring Image Format, Rotation Angle and Source Image Size Parameter for Rotator Operation.	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_ROTATOR_SET_OPERATION_PARAM
lpInBuffer	SVEARG_ROTATOR_PARAMETER *
nInBuffer	sizeof(SVEARG_ROTATOR_PARAMETER)
lpOutBuffer	None
nOutBufferSize	None
Note	
The Configuration of Image Rotator can not be changed while HW is running.	
Data Structure of SVEARG_ROTATOR_PARAMETER	

```
typedef struct
{
  DWORD dwImgFormat;        // Source Image Format
  DWORD dwOpType;           // Rotator Operation Type
  DWORD dwSrcWidth;
  DWORD dwSrcHeight;
} SVEARG_ROTATOR_PARAMETER;
```

Member	**Description**	**Possible value or unit**
dwImgFormat	Source Image Format Supporting YUV420, YUV422(Interleaved), RGB565 and RGB888	*ROTATOR_IMAGE_FORMAT*
dwOpType	Rotating Angle or Flipping Direction	*ROTATOR_OPERATION_TYPE*
dwSrcWidth	Source Image Width	Pixel
dwSrcHeight	Source Image Height	Pixel

4.2.4.2 IOCTL_SVE_ROTATOR_SET_SOURCE_BUFFER

API Description	
Configuring Buffer Address of Source Image	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_ROTATOR_SET_SOURCE_BUFFER
lpInBuffer	SVEARG_ROTATOR_BUFFER *
nInBuffer	sizeof(SVEARG_ROTATOR_BUFFER)
lpOutBuffer	None
nOutBufferSize	None
Note	
The Configuration of Image Rotator can not be changed while HW is running.	
Data Structure of SVEARG_ROTATOR_BUFFER	

```
typedef struct
{
  DWORD dwBufferRGBY;
  DWORD dwBufferCb;
  DWORD dwBufferCr;
} SVEARG_ROTATOR_BUFFER;
```

Member	**Description**	**Possible value or unit**
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image(YUV420)	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image(YUV420)	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image(YUV420)	Physical Address

4.2.4.3 IOCTL_SVE_ROTATOR_SET_DESTINATION_BUFFER

API Description	
Configuring Buffer Address of Destination Image	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_ROTATOR_SET_DESTINATION_BUFFER
lpInBuffer	SVEARG_ ROTATOR_BUFFER *
nInBuffer	sizeof(SVEARG_ ROTATOR_BUFFER)
lpOutBuffer	None
nOutBufferSize	None
Note	
The Configuration of Image Rotator can not be changed while HW is running.	
Data Structure of SVEARG_ROTATOR_BUFFER	

```
typedef struct
{
   DWORD dwBufferRGBY;
   DWORD dwBufferCb;
   DWORD dwBufferCr;
} SVEARG_ROTATOR_BUFFER;
```

Member	**Description**	**Possible value or unit**
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image(YUV420)	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image(YUV420)	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image(YUV420)	Physical Address

4.2.4.4 IOCTL_SVE_ROTATOR_SET_OPERATION_START

API Description	
Starting Rotator Operation	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_ROTATOR_SET_OPERATION_START
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
Rotator Operation can start after last operating is finished. User can wait for finishing last operation with *IOCTL_SVE_ROTATOR_WAIT_OPERATION_DONE* API	

4.2.4.5 IOCTL_SVE_ROTATOR_SET_OPERATION_STOP

API Description	
Stopping Rotator Operation	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_ROTATOR_SET_OPERATION_STOP
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
Actually This API can not stop rotator operation while HW is running. Instead, waiting for finishing HW Operation. This API is exactly same with *IOCTL_SVE_ROTATOR_WAIT_OPERATION_DONE API*	

4.2.4.6 IOCTL_SVE_ROTATOR_WAIT_OPERATION_DONE

API Description	
Waiting for Rotator Operation to be finished.	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_ROTATOR_WAIT_OPERATION_DONE
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API can be used when application want to wait for finishing last rotator operation. This API should be used before reconfiguring Rotator Parameter or Address. This API use WaitForSingleObject() internally to waiting for HW interrupt. So you can release CPU scheduled time until HW operation finished.	

4.2.5 TV Scaler API (TV Scaler)

The function of TV Scaler is almost same with Post Processor. But only TV Scaler can be the input path of TV Encoder.

IOCTL Code	Description
IOCTL_SVE_TVSC_SET_PROCESSING_PARAM	Configuring Parameters of TV Scaler (Operating mode, source &, destination image size, offset)
IOCTL_SVE_TVSC_SET_SOURCE_BUFFER	Configuring Source Image Address
IOCTL_SVE_TVSC_SET_NEXT_SOURCE_BUFFER(1)	Change Next Source Image Address (for Free run mode only)
IOCTL_SVE_TVSC_SET_DESTINATION_BUFFER	Configuring Destination Image Address
IOCTL_SVE_TVSC_SET_NEXT_DESTINATION_BUFFER(1)	Change Next Destination Image Address (for Free run mode only)
IOCTL_SVE_TVSC_SET_PROCESSING_START	TV Scaler Start
IOCTL_SVE_TVSC_SET_PROCESSING_STOP	TV Scaler Stop
IOCTL_SVE_TVSC_WAIT_PROCESSING_DONE(2)	Waiting for TV Scaler to be finished. (for Per Frame mode only)
IOCTL_SVE_TVSC_GET_PROCESSING_STATUS	Request the status of TV Scaler

(1) API can be used when TV Scaler is operating in free run mode.

(2) API can be used when TV Scaler is operating in per frame mode.

4.2.5.1 IOCTL_SVE_TVSC_SET_PROCESSING_PARAM

API Description	
Configuring Parameters of TV Scaler (Operating mode and image format, size, offset)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVSC_SET_PROCESSING_PARAM
lpInBuffer	SVEARG_ TVSC_PARAMETER *
nInBuffer	sizeof(SVEARG_ TVSC_PARAMETER)
lpOutBuffer	None
nOutBufferSize	None

Note

TV Scaler parameter can be configured only when HW is not running.

To use TV Scaler as input path of TV Encoder, dwDstType must be configured as *TVSC_DST_FIFO_YUV444* and dwScanMode must be configured as *TVSC_INTERLACE*.

To use FIMD as input path of TV Scaler, dwSrcType must be configured as *TVSC_SRC_FIFO.*

dwScanMode is applied to HW when TV Scaler is configured as FIFO mode(TV Out)

If application changes the parameters of TV Scaler, Application should reconfigure Image Buffer Address also.

Data Structure of SVEARG_TVSC_PARAMETER

```
typedef struct
{
  DWORD dwOpMode;          // Operation Mode (Frame or Free Run)
  DWORD dwScanMode;        // Scan Mode (Progressive or Interlaced)
  DWORD dwSrcType;         // Src Image Type
  DWORD dwSrcBaseWidth;
  DWORD dwSrcBaseHeight;
  DWORD dwSrcWidth;
  DWORD dwSrcHeight;
  DWORD dwSrcOffsetX;
  DWORD dwSrcOffsetY;
  DWORD dwDstType;         // Dst Image Type
  DWORD dwDstBaseWidth;
  DWORD dwDstBaseHeight;
  DWORD dwDstWidth;
  DWORD dwDstHeight;
```

```
    DWORD dwDstOffsetX;
    DWORD dwDstOffsetY;
} SVEARG_TVSC_PARAMETER;
```

Member	Description	Possible value or unit
dwOpMode	Operating mode Per Frame Mode or Free run mode	*TVSC_OP_MODE*
dwScanMode	Output scan method Progressive or Interlaced mode	*TVSC_SCAN_MODE*
dwSrcType	Source Image Format	*TVSC_SRC_TYPE*
dwSrcBaseWidth	Source Image Width	Pixel
dwSrcBaseHeight	Source Image Height	Pixel
dwSrcWidth	Source Image Clipping Width	Pixel
dwSrcHeight	Source Image Clipping Height	Pixel
dwSrcOffsetX	Horizontal offset of Source Clipping area	Pixel
dwSrcOffsetY	Vertical offset of Source Clipping area	Pixel
dwDstType	Destination Image Format	*TVSC_DST_TYPE*
dwDstBaseWidth	Destination Image Width	Pixel
dwDstBaseHeight	Destination Image Height	Pixel
dwDstWidth	Destination Image Clipping Width	Pixel
dwDstHeight	Destination Image Clipping Height	Pixel
dwDstOffsetX	Horizontal offset of Destination Clipping area	Pixel
dwDstOffsetY	Vertical offset of Destination Clipping area	Pixel

4.2.5.2 IOCTL_SVE_TVSC_SET_SOURCE_BUFFER

API Description		
Configuring Source Image Address		
Arguments	**Type**	
dwIoControlCode	IOCTL_SVE_TVSC_SET_SOURCE_BUFFER	
lpInBuffer	SVEARG_TVSC_BUFFER *	
nInBuffer	sizeof(SVEARG_TVSC_BUFFER)	
lpOutBuffer	None	
nOutBufferSize	None	
Note		
Source Image Address can be configured only when HW is not running.		
Data Structure of SVEARG_TVSC_BUFFER		

```
typedef struct
{
  DWORD dwBufferRGBY;
  DWORD dwBufferCb;
  DWORD dwBufferCr;
  BOOL bWaitForVSync;          // Blocked Operation
} SVEARG_TVSC_BUFFER;
```

Member	**Description**	**Possible value or unit**
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image	Physical Address
bWaitForVSync	Don't care in this API	TRUE or FALSE

4.2.5.3 IOCTL_SVE_TVSC_SET_NEXT_SOURCE_BUFFER

API Description	
Change Next Source Image Address	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVSC_SET_NEXT_SOURCE_BUFFER
lpInBuffer	SVEARG_TVSC_BUFFER *
nInBuffer	sizeof(SVEARG_TVSC_BUFFER)
lpOutBuffer	None
nOutBufferSize	None
Note	
Setting Next address can be used only when TV Scaler is running in free run mode. If user configure parameter as per frame mode, next address will be ignored This API operates asynchronously when TV Scaler is running, because next address value is updated to address in HW when current operation is finished.	
Data Structure of SVEARG_TVSC_BUFFER	

typedef struct
{
DWORD dwBufferRGBY;
DWORD dwBufferCb;
DWORD dwBufferCr;
BOOL bWaitForVSync; // Blocked Operation
} SVEARG_TVSC_BUFFER;	

Member	Description	Possible value or unit
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image	Physical Address
bWaitForVSync	If bWaitForVSync is set as TRUE, Function will be blocked until new address applied to HW	*TRUE* or *FALSE*

4.2.5.4 IOCTL_SVE_TVSC_SET_DESTINATION_BUFFER

API Description	
Configuring Destination Image Address	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVSC_SET_DESTINATION_BUFFER
lpInBuffer	SVEARG_TVSC_BUFFER *
nInBuffer	sizeof(SVEARG_TVSC_BUFFER)
lpOutBuffer	None
nOutBufferSize	None
Note	
Destination Image Address can be configured only when HW is not running. Destination Image address is ignored when TV Scaler is running as input of TV Encoder.	
Data Structure of SVEARG_TVSC_BUFFER	

```
typedef struct
{
  DWORD dwBufferRGBY;
  DWORD dwBufferCb;
  DWORD dwBufferCr;
  BOOL bWaitForVSync;          // Blocked Operation
} SVEARG_TVSC_BUFFER;
```

Member	Description	Possible value or unit
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image	Physical Address
bWaitForVSync	Don't care in this API	TRUE or FALSE

4.2.5.5 IOCTL_SVE_TVSC_SET_NEXT_DESTINATION_BUFFER

API Description	
Change Next Destination Image Address	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVSC_SET_NEXT_DESTINATION_BUFFER
lpInBuffer	SVEARG_TVSC_BUFFER *
nInBuffer	sizeof(SVEARG_TVSC_BUFFER)
lpOutBuffer	None
nOutBufferSize	None
Note	
Setting Next address can be used only when TV Scaler is running in free run mode. If user configure parameter as per frame mode, next address will be ignored This API operates asynchronously when TV Scaler is running, because next address value is updated to address in HW when current operation is finished. Destination Image address is ignored when TV Scaler is running as input of TV Encoder.	
Data Structure of SVEARG_TVSC_BUFFER	

typedef struct
{
DWORD dwBufferRGBY;
DWORD dwBufferCb;
DWORD dwBufferCr;
BOOL bWaitForVSync; // Blocked Operation
} SVEARG_TVSC_BUFFER;	

Member	Description	Possible value or unit
dwBufferRGBY	Address of Source Image or Address of Y Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCb	Address of Cb(U) Component of Non-interleaved YUV Source Image	Physical Address
dwBufferCr	Address of Cr(V) Component of Non-interleaved YUV Source Image	Physical Address
bWaitForVSync	If bWaitForVSync is set as TRUE, Function will be blocked until new address applied to HW	*TRUE* or *FALSE*

4.2.5.6 IOCTL_SVE_TVSC_SET_PROCESSING_START

API Description	
TV Scaler Start	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVSC_SET_PROCESSING_START
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
TV Scaler can start after last operation is finished. Application can wait for last operation to be finished with *IOCTL_SVE_TVSC_WAIT_PROCESSING_DONE* API. Use *IOCTL_SVE_ TVENC_SET_ENCODER_ON* API to start TV Scaler when TV scaler is input of TV Encoder. Unless TV Scaler will be unpredictable state. *IOCTL_SVE_TVENC_SET_ENCODER_ON* API enables TV Scaler and TV Encoder in proper sequence.	

4.2.5.7 IOCTL_SVE_TVSC_SET_PROCESSING_STOP

API Description	
TV Scaler Processing Stop	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVSC_SET_PROCESSING_STOP
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API can't stop TV Scaler immediately because TV Scaler can not be terminated before DMA operation is completed. When TV Scaler is configured as Per Frame mode, HW will stop operation automatically after finishing one frame. Use *IOCTL_SVE_TVENC_SET_ENCODER_OFF* API to stop TV Scaler when TV scaler is input of TV Encoder. Unless TV Scaler will be unpredictable state. *IOCTL_SVE_TVENC_SET_ENCODER_OFF* API disables TV Scaler and TV Encoder in proper sequence.	

4.2.5.8 IOCTL_SVE_TVSC_WAIT_PROCESSING_DONE

API Description	
Waiting for TV Scaler operation to be finished. (for Per Frame Mode Only)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVSC_WAIT_PROCESSING_DONE
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
This API can be used when application want to wait for finishing last TV Scaler processing. This API should be used before reconfiguring TV Scaler Parameter or Address. Do not use this API when TV Scaler is configured as Free run mode. In Free run mode, TV Scaler run continually. This API use WaitForSingleObject() internally to waiting for HW interrupt. So you can release CPU scheduled time until HW operation finished.	

4.2.5.9 IOCTL_SVE_TVSC_GET_PROCESSING_STATUS

API Description	
Request the status of TV Scaler Processing	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVSC_GET_PROCESSING_STATUS
lpInBuffer	None
nInBuffer	None
lpOutBuffer	DWORD *
nOutBufferSize	sizeof(DWORD)
Note	
lpInBuffer is the pointer of variable contains status value. Refer to enumeration *TVSC_STATE*. (0: *TVSC_IDLE*, 1: *TVSC_BUSY*)	

4.2.6 TV Encoder API (TV Encoder)

TV Encoder always needs output of TV Scaler as the input path. So TV Encoder can't work without TV Scaler.

IOCTL Code	Description
IOCTL_SVE_TVENC_SET_INTERFACE_PARAM	Configuring Parameters of TV Encoder (Output Type, Output System Standard, source image size from TV Scaler)
IOCTL_SVE_TVENC_SET_ENCODER_ON	Enable TV Out (Start TV Scaler and Enable TV Encoder)
IOCTL_SVE_TVENC_SET_ENCODER_OFF	Disable TV Out (Stop TV Scaler and Disable TV Encoder)
IOCTL_SVE_TVENC_GET_INTERFACE_STATUS(1)	Request the status of TV Encoder

(1) Function is not implemented at current version

4.2.6.1 IOCTL_SVE_TVENC_SET_INTERFACE_PARAM

API Description	
Configuring Parameters of TV Encoder (Output Type, Output System Standard, source image size from TV Scaler)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVENC_SET_INTERFACE_PARAM
lpInBuffer	SVEARG_TVENC_PARAMETER *
nInBuffer	sizeof(SVEARG_TVENC_PARAMETER)
lpOutBuffer	None
nOutBufferSize	None
Note	
TV Encoder parameter can be configured only when HW is not running.	
Data Structure of SVEARG_TVENC_PARAMETER	

```
typedef struct
{
  DWORD dwOutputType;        // Output Interface Type
  DWORD dwOutputStandard;    // Output System
  DWORD dwMVisionPattern;    // Macrovision Pattern
  DWORD dwSrcWidth;
  DWORD dwSrcHeight;
} SVEARG_TVENC_PARAMETER;
```

Member	**Description**	**Possible value or unit**
dwOutputType	TV Out Interface Type Composite or S-Video	*TVENC_OUTPUT_TYPE*
dwOutputStandard	Standard system of Output Signal NTSC M/J and PAL system	*TVENC_OUTPUT_STANDARD*
dwMVisionPattern	Can not work at this version. Use *TVENC_MACROVISION_OFF* only	*TVENC_MACROVISION_PATTERN*
dwSrcWidth	Source Image Width from TV Scaler	Pixel
dwSrcHeight	Source Image Height from TV Scaler	Pixel

4.2.6.2 IOCTL_SVE_TVENC_SET_ENCODER_ON

API Description	
Enable TV Out (Start TV Scaler and Enable TV Encoder)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVENC_SET_ENCODER_ON
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
Before enable TV Out, TV Scaler should be configured properly for TV Encoder. Do nou use *IOCTL_SVE_TVSC_SET_PROCESSING_START* API when TV scaler is input of TV Encoder. Unless TV Scaler will be unpredictable state. *IOCTL_SVE_TVENC_SET_ENCODER_ON* API enables TV Scaler and TV Encoder in proper sequence.	

4.2.6.3 IOCTL_SVE_TVENC_SET_ENCODER_OFF

API Description	
Disable TV Out (Stop TV Scaler and Disable TV Encoder)	
Arguments	**Type**
dwIoControlCode	IOCTL_SVE_TVENC_SET_ENCODER_OFF
lpInBuffer	None
nInBuffer	None
lpOutBuffer	None
nOutBufferSize	None
Note	
Do nou use *IOCTL_SVE_TVSC_SET_PROCESSING_STOP* API when TV scaler is input of TV Encoder. Unless TV Scaler will be unpredictable state. *IOCTL_SVE_TVENC_SET_ENCODER_OFF* API disables TV Scaler and TV Encoder in proper sequence.	

4.3 Power Management API

Power Management APIs control clock and block power of each HW IP.

Power Management of Video Driver is controlled by Display Driver. So, Do not use Power Management API in user application. It can cause unpredictable state.

4.4 API Usage Sample

4.4.1 Display Video Screen to Window0

At default, Display Driver of OS uses Window1 as Primary Surface.

Normally, when playing video movie, the overlay is used as video screen, that is window 0. And see the case, in that time, the YUV420 video frame is out to Window0 and Window0 is overlaid on Window 1

1st Step : Driver Open

```
// Open Video Driver
HANDLE hVideoDrv = INVALID_HANDLE_VALUE;
hVideoDrv = CreateFile( L"VDE0:", GENERIC_READ|GENERIC_WRITE,
                        ILE_SHARE_READ|FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, 0);
if (hVideoDrv == INVALID_HANDLE_VALUE)
{
  RETAILMSG(1,(L"[VDE:ERR] VDE0 Open Device Failed\n"));
  return;
}
```

2nd Step : Request Resource

```
// Request FIMD Window0 Resource
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_REQUEST_FIMD_WIN0, NULL, 0, NULL, 0,
                        &dwBytes, NULL);

// Request Post Processor Resourece
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_REQUEST_POST, NULL, 0, NULL, 0,
                        &dwBytes, NULL);
```

3rd Step : Configure Display Controller Window0 mode

```
SVEARG_FIMD_WIN_MODE tParamMode;

tParamMode.dwWinMode = DISP_WIN0_DMA;
tParamMode.dwBPP = DISP_16BPP_565;
tParamMode.dwWidth = DST_WIDTH;
tParamMode.dwHeight = DST_HEIGHT;
tParamMode.dwOffsetX = 0;
tParamMode.dwOffsetY = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_MODE, &tParamMode,
                        sizeof(SVEARG_FIMD_WIN_MODE), NULL, 0, &dwBytes, NULL);
```

Now, Window 0 is set with RGB565 color format and DST_WIDTH x DST_HEIGHT size, using DMA

4th Step : Configure Overlay Blending

```
SVEARG_FIMD_WIN_COLORKEY tParamCKey;

// Color Key Disable
tParamCKey.dwWinNum = DISP_WIN1;
tParamCKey.bOnOff = FALSE;
tParamCKey.dwDirection = DISP_FG_MATCH_BG_DISPLAY;
tParamCKey.dwColorKey = 0;
tParamCKey.dwCompareKey = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_BLEND_COLORKEY,
                        &tParamCKey, sizeof(SVEARG_FIMD_WIN_COLORKEY), NULL, 0,
                        &dwBytes, NULL);

SVEARG_FIMD_WIN_ALPHA tParamAlpha;

// Alpha Set to 0x0 (Show Window0)
tParamAlpha.dwWinNum = DISP_WIN1;
tParamAlpha.dwMethod = DISP_ALPHA_PER_PLANE;
tParamAlpha.dwAlpha0 = 0x0;
tParamAlpha.dwAlpha1 = 0x0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_BLEND_ALPHA,
                        &tParamAlpha, sizeof(SVEARG_FIMD_WIN_ALPHA), NULL, 0,
                        &dwBytes, NULL);
```

5th Step : Configure Window0 Frame Buffer and Enable Window0

```
SVEARG_FIMD_WIN_FRAMEBUFFER tParamFB;

// Set Window0 Framebuffer
tParamFB.dwWinNum = DISP_WIN0;
tParamFB.dwFrameBuffer = 0x57400000;
tParamFB.bWaitForVSync = FALSE;
DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_FRAMEBUFFER, &tParamFB,
                        sizeof(SVEARG_FIMD_WIN_FRAMEBUFFER), NULL, 0, &dwBytes,
                        NULL);

DWORD dwWinNum;

// Window0 Enable
dwWinNum = DISP_WIN0;
DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_ENABLE, &dwWinNum,
                        sizeof(DWORD), NULL, 0, &dwBytes, NULL);
```

The framebuffer of Window0 is set as 0x57400000 physical address, now you can write video image RGB565 format to 0x57400000 Physical address, then that will be displayed in Window0 and it's shown over Window1

.

6th Step : Configure Post Processor Parameter

```
SVEARG_POST_PARAMETER tParamPost;

tParamPost.dwOpMode = POST_PER_FRAME_MODE;
tParamPost.dwScanMode = POST_PROGRESSIVE;
tParamPost.dwSrcType = POST_SRC_YUV420;
tParamPost.dwSrcBaseWidth = VIDEO_WIDTH;
tParamPost.dwSrcBaseHeight = VIDEO_HEIGHT;
tParamPost.dwSrcWidth = VIDEO_WIDTH;
tParamPost.dwSrcHeight = VIDEO_HEIGHT;
tParamPost.dwSrcOffsetX = 0;
tParamPost.dwSrcOffsetY = 0;
tParamPost.dwDstType = POST_DST_RGB16;
tParamPost.dwDstBaseWidth = DST_WIDTH;
tParamPost.dwDstBaseHeight = DST_HEIGHT;
tParamPost.dwDstWidth = DST_WIDTH;
tParamPost.dwDstHeight = DST_HEIGHT;
tParamPost.dwDstOffsetX = 0;
tParamPost.dwDstOffsetY = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_PROCESSING_PARAM, &tParamPost,
                        sizeof(SVEARG_POST_PARAMETER), NULL, 0, &dwBytes, NULL);
```

Now, Set the Post Processor to use Per Frame Mode, YUV420 to RGB565 Color Conversion, Scaling from VIDEO_WIDTH x VIDEO_HEIGHT size to DST_WIDTH x DST_HEIGHT size.

7th Step : Configure Source & Destination Image Address

```
SVEARG_POST_BUFFER tParamBuffer;

// Source Address
tParamBuffer.dwBufferRGBY = VIDEO_YUV420_BUFFER;
tParamBuffer.dwBufferCb = tParamBuffer.dwBufferRGBY + VIDEO_WIDTH *
                        VIDEO_HEIGHT;
tParamBuffer.dwBufferCr = tParamBuffer.dwBufferCb + (VIDEO_WIDTH*VIDEO_HEIGHT)/4;
tParamBuffer.bWaitForVSync = FALSE;

DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_SOURCE_BUFFER, &tParamBuffer,
                        sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes, NULL);

// Destination Address
tParamBuffer.dwBufferRGBY = 0x57400000;
tParamBuffer.dwBufferCb = 0;
tParamBuffer.dwBufferCr = 0;
tParamBuffer.bWaitForVSync = FALSE;

DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_DESTINATION_BUFFER, &tParamBuffer,
                        sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes, NULL);
```

Set the Source Image Address to VIDEO_YUV420_BUFFER address, and Destination Image Address to the Frame Buffer Address, 0x57400000 that is the frambuffer of Display Window 0

8th Step : Post Processing Start

```
// Post Processing Start
DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_PROCESSING_START, NULL, 0, NULL, 0,
                        &dwBytes, NULL);
```

Now the Post Processor will process 1 video frame, and the result will be stored in the framebuffer of window 0

9th Step : Process Next frame

```
// Wait for Post Processing Finished
DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_WAIT_PROCESSING_DONE, NULL, 0, NULL,
                        0, &dwBytes, NULL);

// Source Address
tParamBuffer.dwBufferRGBY = NEXT_VIDEO_YUV420_BUFFER;
tParamBuffer.dwBufferCb = tParamBuffer.dwBufferRGBY + VIDEO_WIDTH *
                        VIDEO_HEIGHT;
tParamBuffer.dwBufferCr = tParamBuffer.dwBufferCb + (VIDEO_WIDTH*VIDEO_HEIGHT)/4;
tParamBuffer.bWaitForVSync = FALSE;

DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_SOURCE_BUFFER, &tParamBuffer,
                        sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes, NULL);

// Post Processing Start
DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_PROCESSING_START, NULL, 0, NULL, 0,
                        &dwBytes, NULL);
```

To wait for completion about processing of previous frame, we use IOCTL_SVE_POST_WAIT_PROCESSING_DONE API

After set the new address for new video, restart the Post Processor, this step will repeat to the end frame

10th Step : Driver Close

```
// Window0 Disable
dwWinNum = DISP_WIN0;
DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_DISABLE, &dwWinNum,
                        sizeof(DWORD), NULL, 0, &dwBytes, NULL);

// Release FIMD Win0 H/W Resource to Video Engine Driver
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_RELEASE_FIMD_WIN0, NULL, 0, NULL, 0,
                        &dwBytes, NULL);

// Release Post Processor H/W Resource to Video Engine Driver
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_RELEASE_POST, NULL, 0, NULL, 0, &dwBytes,
                        NULL);

CloseHandle(hVideoDrv);
```

Disable Window 0, return the Resource, close the Driver handler

4.4.2 Display Video Screen with Local Path

This show the example how to use LocalPath when using YUV420 Source video frame, output to the framebuffer of Window 0

1st Step : Driver Open

```
// Open Video Driver
HANDLE hVideoDrv = INVALID_HANDLE_VALUE;
hVideoDrv = CreateFile( L"VDE0:", GENERIC_READ|GENERIC_WRITE,
                        ILE_SHARE_READ|FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, 0);
if (hVideoDrv == INVALID_HANDLE_VALUE)
{
  RETAILMSG(1,(L"[VDE:ERR] VDE0 Open Device Failed\n"));
  return;
}
```

2nd Step : Request Resource

```
// Request FIMD Window0 Resource
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_REQUEST_FIMD_WIN0, NULL, 0, NULL, 0,
                        &dwBytes, NULL);

// Request Post Processor Resourece
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_REQUEST_POST, NULL, 0, NULL, 0,
                        &dwBytes, NULL);
```

3rd Step : Configure Display Controller Window0 mode as Local Path

```
SVEARG_FIMD_WIN_MODE tParamMode;

tParamMode.dwWinMode = DISP_WIN0_POST_RGB;
tParamMode.dwBPP = DISP_24BPP_888;
tParamMode.dwWidth = DST_WIDTH;
tParamMode.dwHeight = DST_HEIGHT;
tParamMode.dwOffsetX = 0;
tParamMode.dwOffsetY = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_MODE, &tParamMode,
                        sizeof(SVEARG_FIMD_WIN_MODE), NULL, 0, &dwBytes, NULL);
```

Set the destination size of Window 0 as DST_WIDTH x DST_HEIGHT to use Post Processor as LocalPath mode. the dwBPP value must be DISP_24BPP_888 when you use DISP_WIN0_POST_RGB mode that mean using LocalPath FIFO.

4th Step : Configure Overlay Blending

```
SVEARG_FIMD_WIN_COLORKEY tParamCKey;

// Color Key Disable
tParamCKey.dwWinNum = DISP_WIN1;
tParamCKey.bOnOff = FALSE;
tParamCKey.dwDirection = DISP_FG_MATCH_BG_DISPLAY;
tParamCKey.dwColorKey = 0;
tParamCKey.dwCompareKey = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_BLEND_COLORKEY,
                        &tParamCKey, sizeof(SVEARG_FIMD_WIN_COLORKEY), NULL, 0,
                        &dwBytes, NULL);

SVEARG_FIMD_WIN_ALPHA tParamAlpha;

// Alpha Set to 0x0 (Show Window0)
tParamAlpha.dwWinNum = DISP_WIN1;
tParamAlpha.dwMethod = DISP_ALPHA_PER_PLANE;
tParamAlpha.dwAlpha0 = 0x0;
tParamAlpha.dwAlpha1 = 0x0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_BLEND_ALPHA,
                        &tParamAlpha, sizeof(SVEARG_FIMD_WIN_ALPHA), NULL, 0,
                        &dwBytes, NULL);
```

5th Step : Configure Window0 Frame Buffer and Enable Window0

In the case that using Local Path, the Frame buffer of Windows does not need to be set. Because the Display Controller does not use DMA to get data. And Post will put the result to Display Controller's Stride FIFO directly.

6th Step : Configure Post Processor Parameter

```
SVEARG_POST_PARAMETER tParamPost;

tParamPost.dwOpMode = POST_FREE_RUN_MODE;
tParamPost.dwScanMode = POST_PROGRESSIVE;
tParamPost.dwSrcType = POST_SRC_YUV420;
tParamPost.dwSrcBaseWidth = VIDEO_WIDTH;
tParamPost.dwSrcBaseHeight = VIDEO_HEIGHT;
tParamPost.dwSrcWidth = VIDEO_WIDTH;
tParamPost.dwSrcHeight = VIDEO_HEIGHT;
tParamPost.dwSrcOffsetX = 0;
tParamPost.dwSrcOffsetY = 0;
tParamPost.dwDstType = POST_DST_FIFO_RGB888;
tParamPost.dwDstBaseWidth = DST_WIDTH;
tParamPost.dwDstBaseHeight = DST_HEIGHT;
tParamPost.dwDstWidth = DST_WIDTH;
tParamPost.dwDstHeight = DST_HEIGHT;
tParamPost.dwDstOffsetX = 0;
tParamPost.dwDstOffsetY = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_PROCESSING_PARAM, &tParamPost,
                        sizeof(SVEARG_POST_PARAMETER), NULL, 0, &dwBytes, NULL);
```

To set Local Path of Post Processor, Set Free Run mode, and the destination type as POST_DST_FIFO_RGB888.

7th Step : Configure Source & Destination Image Address

```
SVEARG_POST_BUFFER tParamBuffer;

tParamBuffer.dwBufferRGBY = VIDEO_YUV420_BUFFER;
tParamBuffer.dwBufferCb = tParamBuffer.dwBufferRGBY + VIDEO_WIDTH *
                        VIDEO_HEIGHT;
tParamBuffer.dwBufferCr = tParamBuffer.dwBufferCb + (VIDEO_WIDTH*VIDEO_HEIGHT)/4;
tParamBuffer.bWaitForVSync = FALSE;

// Source Address
DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_SOURCE_BUFFER, &tParamBuffer,
                        sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes, NULL);

// Next Source Address
DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_NEXT_SOURCE_BUFFER,
                        &tParamBuffer, sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes,
                        NULL);
```

Due to Free run mode, the frame processing will continue to end frame automatically. The interrupt will occur between frames, and the Post Processor will update the current source address as the next address. So when you use Local Path, you must set the next address before start Local Path. Using Local Path, you don't need to set the destination address.

8th Step : Local Path Start

```
// Local Path Start
DeviceIoControl(hVideoDrv, IOCTL_SVE_LOCALPATH_SET_WIN0_START, NULL, 0, NULL, 0,
                &dwBytes, NULL);
```

Now start Local Path. This will enable Window 0.

9th Step : Process Next frame

```
// Next Source Address
tParamBuffer.dwBufferRGBY = NEXT_VIDEO_YUV420_BUFFER;
tParamBuffer.dwBufferCb = tParamBuffer.dwBufferRGBY + VIDEO_WIDTH *
                VIDEO_HEIGHT;
tParamBuffer.dwBufferCr = tParamBuffer.dwBufferCb + (VIDEO_WIDTH*VIDEO_HEIGHT)/4;
tParamBuffer.bWaitForVSync = TRUE;

DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_NEXT_SOURCE_BUFFER,
                &tParamBuffer, sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes,
                NULL);
```

When you use Local Path, Post Processor always run, So you should set the next frame address till running. This is get by using IOCTL_SVE_POST_SET_NEXT_SOURCE_BUFFER_API. This IOCTL will wait for frame done, and when the current frame processing is done, will update the current address to next address.

10th Step : Driver Close

```
// Local Path Stop
DeviceIoControl(hVideoDrv, IOCTL_SVE_LOCALPATH_SET_WIN0_STOP, NULL, 0, NULL, 0,
                &dwBytes, NULL);

// Release FIMD Win0 H/W Resource to Video Engine Driver
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_RELEASE_FIMD_WIN0, NULL, 0, NULL, 0,
                &dwBytes, NULL);

// Release Post Processor H/W Resource to Video Engine Driver
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_RELEASE_POST, NULL, 0, NULL, 0, &dwBytes,
                NULL);

CloseHandle(hVideoDrv);
```

Stop the LocalPath process, return Resource, close the driver's handle.

4.4.3 Display Video Screen with Local Path and display to TV Out

Display YUV420 video frame on window0 of LCD using Local Path. And Display YUV420 video frame to TV using TV Scaler and TV Encoder.

1st Step : Driver Open

```
// Open Video Driver
HANDLE hVideoDrv = INVALID_HANDLE_VALUE;
hVideoDrv = CreateFile( L"VDE0:", GENERIC_READ|GENERIC_WRITE,
                        ILE_SHARE_READ|FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, 0);
if (hVideoDrv == INVALID_HANDLE_VALUE)
{
  RETAILMSG(1,(L"[VDE:ERR] VDE0 Open Device Failed\n"));
  return;
}
```

2nd Step : Request Resource

```
// Request FIMD Window0 Resource
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_REQUEST_FIMD_WIN0, NULL, 0, NULL, 0,
                        &dwBytes, NULL);

// Request Post Processor Resourece
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_REQUEST_POST, NULL, 0, NULL, 0,
                        &dwBytes, NULL);

// Request TV Scaler and TV Encoder Resourece
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_REQUEST_TVSCALER_TVENCODER, NULL, 0,
                        NULL, 0, &dwBytes, NULL);
```

3rd Step : Configure Display Controller Window0 mode as Local Path

```
SVEARG_FIMD_WIN_MODE tParamMode;

tParamMode.dwWinMode = DISP_WIN0_POST_RGB;
tParamMode.dwBPP = DISP_24BPP_888;
tParamMode.dwWidth = DST_WIDTH;
tParamMode.dwHeight = DST_HEIGHT;
tParamMode.dwOffsetX = 0;
tParamMode.dwOffsetY = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_MODE, &tParamMode,
                        sizeof(SVEARG_FIMD_WIN_MODE), NULL, 0, &dwBytes, NULL);
```

When FIMD window0 is used as local path(DISP_WIN0_POST_RGB), dwBPP should be configured to DISP_24BPP_888.

4th Step : Configure Overlay Blending

```
SVEARG_FIMD_WIN_COLORKEY tParamCKey;

// Color Key Disable
tParamCKey.dwWinNum = DISP_WIN1;
tParamCKey.bOnOff = FALSE;
tParamCKey.dwDirection = DISP_FG_MATCH_BG_DISPLAY;
tParamCKey.dwColorKey = 0;
tParamCKey.dwCompareKey = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_BLEND_COLORKEY,
                        &tParamCKey, sizeof(SVEARG_FIMD_WIN_COLORKEY), NULL, 0,
                        &dwBytes, NULL);

SVEARG_FIMD_WIN_ALPHA tParamAlpha;

// Alpha Set to 0x0 (Show Window0)
tParamAlpha.dwWinNum = DISP_WIN1;
tParamAlpha.dwMethod = DISP_ALPHA_PER_PLANE;
tParamAlpha.dwAlpha0 = 0x0;
tParamAlpha.dwAlpha1 = 0x0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_FIMD_SET_WINDOW_BLEND_ALPHA,
                        &tParamAlpha, sizeof(SVEARG_FIMD_WIN_ALPHA), NULL, 0,
                        &dwBytes, NULL);
```

5th Step : Configure Window0 Frame Buffer and Enable Window0

In case of Local Path, There is no need to configure frame buffer address of FIMD window because DMA of FIMD Window is not used.

(continued in next pages)

6th Step : Configure Post Processor Parameter

```
SVEARG_POST_PARAMETER tParamPost;

tParamPost.dwOpMode = POST_FREE_RUN_MODE;
tParamPost.dwScanMode = POST_PROGRESSIVE;
tParamPost.dwSrcType = POST_SRC_YUV420;
tParamPost.dwSrcBaseWidth = VIDEO_WIDTH;
tParamPost.dwSrcBaseHeight = VIDEO_HEIGHT;
tParamPost.dwSrcWidth = VIDEO_WIDTH;
tParamPost.dwSrcHeight = VIDEO_HEIGHT;
tParamPost.dwSrcOffsetX = 0;
tParamPost.dwSrcOffsetY = 0;
tParamPost.dwDstType = POST_DST_FIFO_RGB888;
tParamPost.dwDstBaseWidth = DST_WIDTH;
tParamPost.dwDstBaseHeight = DST_HEIGHT;
tParamPost.dwDstWidth = DST_WIDTH;
tParamPost.dwDstHeight = DST_HEIGHT;
tParamPost.dwDstOffsetX = 0;
tParamPost.dwDstOffsetY = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_PROCESSING_PARAM, &tParamPost,
                        sizeof(SVEARG_POST_PARAMETER), NULL, 0, &dwBytes, NULL);
```

To configure Local Path, Post Processor is configured to Free Run mode and destination is configured to POST_DST_FIFO_RGB888.

7th Step : Configure Source & Destination Image Address

```
SVEARG_POST_BUFFER tParamBuffer;

tParamBuffer.dwBufferRGBY = VIDEO_YUV420_BUFFER;
tParamBuffer.dwBufferCb = tParamBuffer.dwBufferRGBY + VIDEO_WIDTH *
                        VIDEO_HEIGHT;
tParamBuffer.dwBufferCr = tParamBuffer.dwBufferCb + (VIDEO_WIDTH*VIDEO_HEIGHT)/4;
tParamBuffer.bWaitForVSync = FALSE;

// Source Address
DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_SOURCE_BUFFER, &tParamBuffer,
                        sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes, NULL);

// Next Source Address
DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_NEXT_SOURCE_BUFFER,
                        &tParamBuffer, sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes,
                        NULL);
```

In Free run mode, Post Processor restart next frame after finishing a frame. When a frame is finished, next address is update to current address register automatically. So, Next address should be configured before starting Local Path.

8th Step : Local Path Start

```
// Local Path Start
DeviceIoControl(hVideoDrv, IOCTL_SVE_LOCALPATH_SET_WIN0_START, NULL, 0, NULL, 0,
                &dwBytes, NULL);
```

When local path is enable, Post Processor will process the video frame and pass it to display window through local path.

(continued in next page)

9th Step : Configure TV Scaler Parameter

```
SVEARG_TVSC_PARAMETER tParamTVSC;

tParamTVSC.dwOpMode = TVSC_FREE_RUN_MODE;
tParamTVSC.dwScanMode = TVSC_INTERLACE;
tParamTVSC.dwSrcType = POST_SRC_YUV420;
tParamTVSC.dwSrcBaseWidth = VIDEO_WIDTH;
tParamTVSC.dwSrcBaseHeight = VIDEO_HEIGHT;
tParamTVSC.dwSrcWidth = VIDEO_WIDTH;
tParamTVSC.dwSrcHeight = VIDEO_HEIGHT;
tParamTVSC.dwSrcOffsetX = 0;
tParamTVSC.dwSrcOffsetY = 0;
tParamTVSC.dwDstType = TVSC_DST_FIFO_YUV444;
tParamTVSC.dwDstBaseWidth = 720*2;
tParamTVSC.dwDstBaseHeight = 480;
tParamTVSC.dwDstWidth = 720*2;
tParamTVSC.dwDstHeight = 480;
tParamTVSC.dwDstOffsetX = 0;
tParamTVSC.dwDstOffsetY = 0;

DeviceIoControl(hVideoDrv, IOCTL_SVE_TVSC_SET_PROCESSING_PARAM, &tParamTVSC,
                        sizeof(SVEARG_TVSC_PARAMETER), NULL, 0, &dwBytes, NULL);
```

To using TV Encoder, TV Scaler is configured to Free Run mode and interlaced mode, and destination is configured to TVSC_DST_FIFO_YUV444.

dwDstBaseWidth and dwDstWidth should be configured double of image width, because TV Scaler is running in Interlaced mode.

10th Step : Configure Source & Destination Image Address

```
SVEARG_TVSC_BUFFER tParamBuffer;

tParamBuffer.dwBufferRGBY = VIDEO_YUV420_BUFFER;
tParamBuffer.dwBufferCb = tParamBuffer.dwBufferRGBY + VIDEO_WIDTH *
                        VIDEO_HEIGHT;
tParamBuffer.dwBufferCr = tParamBuffer.dwBufferCb + (VIDEO_WIDTH*VIDEO_HEIGHT)/4;
tParamBuffer.bWaitForVSync = FALSE;

// Source Address
DeviceIoControl(hVideoDrv, IOCTL_SVE_TVSC_SET_SOURCE_BUFFER, &tParamBuffer,
                        sizeof(SVEARG_TVSC_BUFFER), NULL, 0, &dwBytes, NULL);

// Next Source Address
DeviceIoControl(hVideoDrv, IOCTL_SVE_TVSC_SET_NEXT_SOURCE_BUFFER,
                        &tParamBuffer, sizeof(SVEARG_TVSC_BUFFER), NULL, 0, &dwBytes,
                        NULL);
```

In Free run mode, TV Scaler restart next frame after finishing a frame. When a frame is finished, next address is update to current address register automatically. So, Next address should be configured before starting TV Scaler.

11th Step : Configure TV Encoder

```
SVEARG_TVENC_PARAMETER tParamTVEnc;

tParamTVEnc.dwOutputType = TVENC_COMPOSITE;
tParamTVEnc.dwOutputStandard = TVENC_NTSC_M;
tParamTVEnc.dwMVisionPattern = TVENC_MACROVISION_OFF;
tParamTVEnc.dwSrcWidth = 720;
tParamTVEnc.dwSrcHeight = 480;

// TV Encoder Initialize
DeviceIoControl(hVideoDrv, IOCTL_SVE_TVENC_SET_INTERFACE_PARAM, & tParamTVEnc,
                        sizeof(SVEARG_TVENC_PARAMETER), NULL, 0, &dwBytes, NULL);
```

Input size of TV Encoder is same with output size of TV Scaler.

12th Step : Enable TV Out

```
// TV Out Enable
DeviceIoControl(hVideoDrv IOCTL_SVE_TVENC_SET_ENCODER_ON, NULL, NULL, 0, NULL,
                        0, &dwBytes, NULL);
```

To enable TV out, Do not use IOCTL_SVE_TVSC_SET_PROCESSING_START to starting TV Scaler. IOCTL_SVE_TVENC_SET_ENCODER_ON starts TV Scaler and enables TV Encoder in properly sequence.

(continued in next page)

13th Step : Next frame Processing for Local Path

```
// Next Source Address
tParamBuffer.dwBufferRGBY = NEXT_VIDEO_YUV420_BUFFER;
tParamBuffer.dwBufferCb = tParamBuffer.dwBufferRGBY + VIDEO_WIDTH *
                          VIDEO_HEIGHT;
tParamBuffer.dwBufferCr = tParamBuffer.dwBufferCb + (VIDEO_WIDTH*VIDEO_HEIGHT)/4;
tParamBuffer.bWaitForVSync = TRUE;

DeviceIoControl(hVideoDrv, IOCTL_SVE_POST_SET_NEXT_SOURCE_BUFFER,
                          &tParamBuffer, sizeof(SVEARG_POST_BUFFER), NULL, 0, &dwBytes,
                          NULL);
```

14th Step : Next frame Processing for TV Out

```
// Next Source Address
tParamBuffer.dwBufferRGBY = NEXT_VIDEO_YUV420_BUFFER;
tParamBuffer.dwBufferCb = tParamBuffer.dwBufferRGBY + VIDEO_WIDTH *
                          VIDEO_HEIGHT;
tParamBuffer.dwBufferCr = tParamBuffer.dwBufferCb + (VIDEO_WIDTH*VIDEO_HEIGHT)/4;
tParamBuffer.bWaitForVSync = TRUE;

DeviceIoControl(hVideoDrv, IOCTL_SVE_TVSC_SET_NEXT_SOURCE_BUFFER,
                          &tParamBuffer, sizeof(SVEARG_TVSC_BUFFER), NULL, 0, &dwBytes,
                          NULL);
```

(go to 13th Step until video playback is finished)

15th Step : Driver Close

```
// Local Path Stop
DeviceIoControl(hVideoDrv, IOCTL_SVE_LOCALPATH_SET_WIN0_STOP, NULL, 0, NULL, 0,
                &dwBytes, NULL);

// TV Out Disable
DeviceIoControl(hVideoDrv IOCTL_SVE_TVENC_SET_ENCODER_OFF, NULL, NULL, 0, NULL,
                0, &dwBytes, NULL);

// Release FIMD Win0 H/W Resource to Video Engine Driver
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_RELEASE_FIMD_WIN0, NULL, 0, NULL, 0,
                &dwBytes, NULL);

// Release Post Processor H/W Resource to Video Engine Driver
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_RELEASE_POST, NULL, 0, NULL, 0, &dwBytes,
                NULL);

// Release TV Scaler and TV Encoder H/W Resource to Video Engine Driver
DeviceIoControl(hVideoDrv, IOCTL_SVE_RSC_RELEASE_TVSCALER_TVENCODER, NULL, 0,
                NULL, 0, &dwBytes, NULL);

CloseHandle(hVideoDrv);
```

To diable TV out, Do not use IOCTL_SVE_TVSC_SET_PROCESSING_STOP to stop TV Scaler. IOCTL_SVE_TVENC_SET_ENCODER_OFF stops TV Scaler and disables TV Encoder in properly sequence.

After stop local path and TV out, release HW resources to Video Driver and close driver handle.

4.5 Appendix

4.5.1 Video Driver Argument Data Structure

```
typedef struct
{
  tDevInfo tRGBDevInfo;              // RGB I/F Device Information
  DWORD dwTVOutScreenWidth;          // TV Out Device Screen Width
  DWORD dwTVOutScreenHeight;         // TV Out Device Screen Height
} SVEARG_FIMD_OUTPUT_IF;

typedef struct
{
  DWORD dwLineCnt;                   // Line Counter
  DWORD dwVerticalStatus;            // Vertical Status
  DWORD dwHorizontalStatus;          // Horizontal Status
  BOOL bENVID;                       // ENVID Field of VIDCON0
} SVEARG_FIMD_OUTPUT_STAT;

typedef struct
{
  DWORD dwWinMode;          // FIMD Window Mode
  DWORD dwBPP;              // BitPerPixel
  DWORD dwWidth;            // Window Horizontal Pixel
  DWORD dwHeight;           // Window Vertical Pixel
  DWORD dwOffsetX;          // Window Horizontal Offset
  DWORD dwOffsetY;          // Window Vertical Offset
} SVEARG_FIMD_WIN_MODE;

typedef struct
{
  DWORD dwWinNum;           // FIMD Window Number
  DWORD dwOffsetX;          // Window Horizontal Offset
  DWORD dwOffsetY;          // Window Vertical Offset
} SVEARG_FIMD_WIN_POS;

typedef struct
{
  DWORD dwWinNum;           // FIMD Window Number
  DWORD dwFrameBuffer;      // Frame Buffer Address
  BOOL bWaitForVSync;       // Blocked Operation
} SVEARG_FIMD_WIN_FRAMEBUFFER;

typedef struct
{
  DWORD dwWinNum;           // FIMD Window Number
  DWORD dwDirection;        // Keying Direction
  DWORD dwColorKey;         // Color Key Value
  DWORD dwCompareKey;       // Compare Key Value
  BOOL bOnOff;              // Color Key Enable/Disable
} SVEARG_FIMD_WIN_COLORKEY;
```

```
typedef struct
{
  DWORD dwWinNum;          // FIMD Window Number
  DWORD dwMethod;          // Blending Method (per Pixel or per Plane)
  DWORD dwAlpha0;          // Alpha Value 0
  DWORD dwAlpha1;          // Alpha Value 1
} SVEARG_FIMD_WIN_ALPHA;

typedef struct
{
  DWORD dwOpMode;          // Operation Mode (Frame or Free Run)
  DWORD dwScanMode;        // Scan Mode (Progressive or Interace)
  DWORD dwSrcType;         // Src Image Type
  DWORD dwSrcBaseWidth;
  DWORD dwSrcBaseHeight;
  DWORD dwSrcWidth;
  DWORD dwSrcHeight;
  DWORD dwSrcOffsetX;
  DWORD dwSrcOffsetY;
  DWORD dwDstType;         // Dst Image Type
  DWORD dwDstBaseWidth;
  DWORD dwDstBaseHeight;
  DWORD dwDstWidth;
  DWORD dwDstHeight;
  DWORD dwDstOffsetX;
  DWORD dwDstOffsetY;
} SVEARG_POST_PARAMETER;

typedef struct
{
  DWORD dwBufferRGBY;
  DWORD dwBufferCb;
  DWORD dwBufferCr;
  BOOL bWaitForVSync;      // Blocked Operation
} SVEARG_POST_BUFFER;

typedef struct
{
  DWORD dwImgFormat;       // Source Image Format
  DWORD dwOpType;          // Rotator Operation Type
  DWORD dwSrcWidth;
  DWORD dwSrcHeight;
} SVEARG_ROTATOR_PARAMETER;

typedef struct
{
  DWORD dwBufferRGBY;
  DWORD dwBufferCb;
  DWORD dwBufferCr;
} SVEARG_ROTATOR_BUFFER;
```

```
typedef struct
{
  DWORD dwOpMode;          // Operation Mode (Frame or Free Run)
  DWORD dwScanMode;        // Scan Mode (Progressive or Interace)
  DWORD dwSrcType;         // Src Image Type
  DWORD dwSrcBaseWidth;
  DWORD dwSrcBaseHeight;
  DWORD dwSrcWidth;
  DWORD dwSrcHeight;
  DWORD dwSrcOffsetX;
  DWORD dwSrcOffsetY;
  DWORD dwDstType;         // Dst Image Type
  DWORD dwDstBaseWidth;
  DWORD dwDstBaseHeight;
  DWORD dwDstWidth;
  DWORD dwDstHeight;
  DWORD dwDstOffsetX;
  DWORD dwDstOffsetY;
} SVEARG_TVSC_PARAMETER;

typedef struct
{
  DWORD dwBufferRGBY;
  DWORD dwBufferCb;
  DWORD dwBufferCr;
  BOOL bWaitForVSync;      // Blocked Operation
} SVEARG_TVSC_BUFFER;

typedef struct
{
  DWORD dwOutputType;              // Output Interface Type
  DWORD dwOutputStandard;          // Output System
  DWORD dwMVisionPattern;          // Macrovision Pattern
  DWORD dwSrcWidth;
  DWORD dwSrcHeight;
} SVEARG_TVENC_PARAMETER;
```

4.5.2 Video Driver Pre-defined Parameter Enumeration

```
typedef enum
{
  DISP_WIN0_DMA,
  DISP_WIN0_POST_RGB,
  DISP_WIN0_POST_YUV,
  DISP_WIN1_DMA,
  DISP_WIN1_TVSCALER_RGB,
  DISP_WIN1_TVSCALER_YUV,
  DISP_WIN1_CIPREVIEW_RGB,
  DISP_WIN1_CIPREVIEW_YUV,
  DISP_WIN2_DMA,
  DISP_WIN2_TVSCALER_RGB,
  DISP_WIN2_TVSCALER_YUV,
  DISP_WIN2_CICODEC_RGB,
  DISP_WIN2_CICODEC_YUV,
  DISP_WIN3_DMA,
  DISP_WIN4_DMA
} DISP_WINDOW_MODE;

typedef enum
{
  DISP_WIN0,
  DISP_WIN1,
  DISP_WIN2,
  DISP_WIN3,
  DISP_WIN4
} DISP_WINDOW;

typedef enum
{
  DISP_1BPP = 0,
  DISP_2BPP,
  DISP_4BPP,
  DISP_8BPP_PAL,
  DISP_8BPP_NOPAL,
  DISP_16BPP_565     ,
  DISP_16BPP_A555,
  DISP_16BPP_I555,
  DISP_18BPP_666,
  DISP_18BPP_A665,
  DISP_19BPP_A666,
  DISP_24BPP_888,
  DISP_24BPP_A887,
  DISP_25BPP_A888
} DISP_BPP_MODE;

typedef enum
{
  DISP_FG_MATCH_BG_DISPLAY,
  DISP_BG_MATCH_FG_DISPLAY
} DISP_COLOR_KEY_DIRECTION;
```

```
typedef enum
{
  DISP_ALPHA_PER_PLANE,
  DISP_ALPHA_PER_PIXEL
} DISP_ALPHA_BLEND_METHOD;

typedef enum
{
  DISP_V_VSYNC = 0,
  DISP_V_BACKPORCH,
  DISP_V_ACTIVE,
  DISP_V_FRONTPORCH
} DISP_VERTICAL_STATUS;

typedef enum
{
  DISP_H_HSYNC = 0,
  DISP_H_BACKPORCH,
  DISP_H_ACTIVE,
  DISP_H_FRONTPORCH
} DISP_HORIZONTAL_STATUS;

typedef enum
{
  POST_PER_FRAME_MODE,
  POST_FREE_RUN_MODE
} POST_OP_MODE;

typedef enum
{
  POST_PROGRESSIVE,
  POST_INTERLACE
} POST_SCAN_MODE;

typedef enum
{
  POST_SRC_RGB16 = 0,
  POST_SRC_RGB24,
  POST_SRC_YUV420,
  POST_SRC_YUV422_YCBYCR,
  POST_SRC_YUV422_CBYCRY,
  POST_SRC_YUV422_YCRYCB,
  POST_SRC_YUV422_CRYCBY
} POST_SRC_TYPE;

typedef enum
{
  POST_DST_RGB16 = 0,
  POST_DST_RGB24,
  POST_DST_YUV420,
  POST_DST_YUV422_YCBYCR,
  POST_DST_YUV422_CBYCRY,
  POST_DST_YUV422_YCRYCB,
  POST_DST_YUV422_CRYCBY,
  POST_DST_FIFO_YUV444,
```

```
    POST_DST_FIFO_RGB888
} POST_DST_TYPE;

typedef enum
{
    POST_IDLE = 0,
    POST_BUSY
} POST_STATE;

typedef enum
{
    ROT_FORMAT_YUV420 = 0,
    ROT_FORMAT_YUV422,
    ROT_FORMAT_RGB565,
    ROT_FORMAT_RGB888
} ROTATOR_IMAGE_FORMAT;

typedef enum
{
    ROT_OP_ROTATE_90 = 0,
    ROT_OP_ROTATE_180,
    ROT_OP_ROTATE_270,
    ROT_OP_FLIP_VERTICAL,
    ROT_OP_FLIP_HORIZONTAL
} ROTATOR_OPERATION_TYPE;

typedef enum
{
    TVSC_PER_FRAME_MODE,
    TVSC_FREE_RUN_MODE
} TVSC_OP_MODE;

typedef enum
{
    TVSC_PROGRESSIVE,
    TVSC_INTERLACE
} TVSC_SCAN_MODE;

typedef enum
{
    TVSC_SRC_RGB16 = 0,
    TVSC_SRC_RGB24,
    TVSC_SRC_YUV420,
    TVSC_SRC_YUV422_YCBYCR,
    TVSC_SRC_YUV422_CBYCRY,
    TVSC_SRC_YUV422_YCRYCB,
    TVSC_SRC_YUV422_CRYCBY,
    TVSC_SRC_FIFO
} TVSC_SRC_TYPE;

typedef enum
{
    TVSC_DST_RGB16 = 0,
    TVSC_DST_RGB24,
    TVSC_DST_YUV420,
    TVSC_DST_YUV422_YCBYCR,
```

```
    TVSC_DST_YUV422_CBYCRY,
    TVSC_DST_YUV422_YCRYCB,
    TVSC_DST_YUV422_CRYCBY,
    TVSC_DST_FIFO_YUV444,
    TVSC_DST_FIFO_RGB888
} TVSC_DST_TYPE;

typedef enum
{
    TVSC_IDLE = 0,
    TVSC_BUSY
} TVSC_STATE;

typedef enum
{
    TVENC_COMPOSITE = 0,
    TVENC_S_VIDEO
} TVENC_OUTPUT_TYPE;

typedef enum
{
    TVENC_NTSC_M = 0,
    TVENC_NTSC_J,
    TVENC_PAL_BDGHI,
    TVENC_PAL_M,
    TVENC_PAL_NC
} TVENC_OUTPUT_STANDARD;

typedef enum
{
    TVENC_MACROVISION_AGC4L = 0,
    TVENC_MACROVISION_AGC2L,
    TVENC_MACROVISION_N01,
    TVENC_MACROVISION_N02,
    TVENC_MACROVISION_P01,
    TVENC_MACROVISION_P02,
    TVENC_MACROVISION_OFF
} TVENC_MACROVISION_PATTERN;
```

(End of Document)