S3C6400 HW Multimedia Codec (MFC) User's Guide

Samsung Electronics Co., Ltd. Mobile Solution Team, System LSI.

Though every care has been taken to ensure the accuracy of this document, Samsung Electronics Co, Ltd. cannot accept responsibility for any errors or omissions or for any loss occasioned to any person, whether legal or natural, from acting, or refraining from action, as a result of the information contained herein. Information in this document is subject to change at any time without obligation to notify any person of such changes.

Samsung Electronics Co, Ltd. may have patents or patent pending applications, trademarks copyrights or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give the recipient or reader any license to these patents, trademarks copyrights or other intellectual property rights.

The document is subject to revision without further notice.

All brand names and product names mentioned in this document are trademarks or registered trademarks of their respective owners.

Contact Address

Samsung Electronics Co., Ltd.
San #24 Nongseo-Ri, Giheung-Gu,
Yongin- City, Gyeonggi-Do, Korea
C.P.O Box #37, Suwon 449-900

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

Revision History

Date	Version	Author	Amendment
Jul. 16, 2007	V1.00	Simon Chun	
Jul. 23, 2007	V1.10	Simon Chun	
Jul. 24, 2007	V1.20	Simon Chun	Required memory size
Jul. 30, 2007	V5.30	Simon Chun	Document version #
Jul. 31, 2007	V5.31	Simon Chun	
Aug. 13, 2007	V5.40	Simon Chun	

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document is prepared for the purpose of describing the S3C6400 HW codec (MFCv1.0) device driver's API so that users can implement their multimedia application easily.

1.2 Scope

The scope of this document is to describe

- How to call the device driver's API to decode/encode.
- Usage example of Decoder

1.3 Intended Audience

Intended Audience	Tick whenever Applicable
Project Manager	Yes
Project Leader	Yes
Project Team Member	Yes
Test Engineer	Yes

1.4 Supported HW & SW

Intended Audience	Tick whenever Applicable
HW	Samsung S3C6400 HW Multimedia Codec
OS	Windows Mobile 6.0, WinCE 5.0

1.5 Definitions, Acronyms, and Abbreviations

Abbreviations	Description
MFC	Multi-Format Codec (HW codec in S3C6400 Samsung AP)
API	Application Program Interface

1.6 References

Number	Reference	Description
1	S3C6400 Datasheet	S3C6400 Datasheet

2 Installation Guide

2.1 Directory Structure

Mfc		
mfc_dd_if_layer	Device Driver OS Interface Layer	OS dependent
mfc_os_indep_layer	MFC driver operational logic	OS independent
mfc_os_dep_layer	Abstraction layer for the OS system call	OS dependent

Table 2-1 Directory Structure of MFC Driver Source Code

Mfc driver souce code was separated into three directories according to the layer structure which is shown in Fig. 2-1.

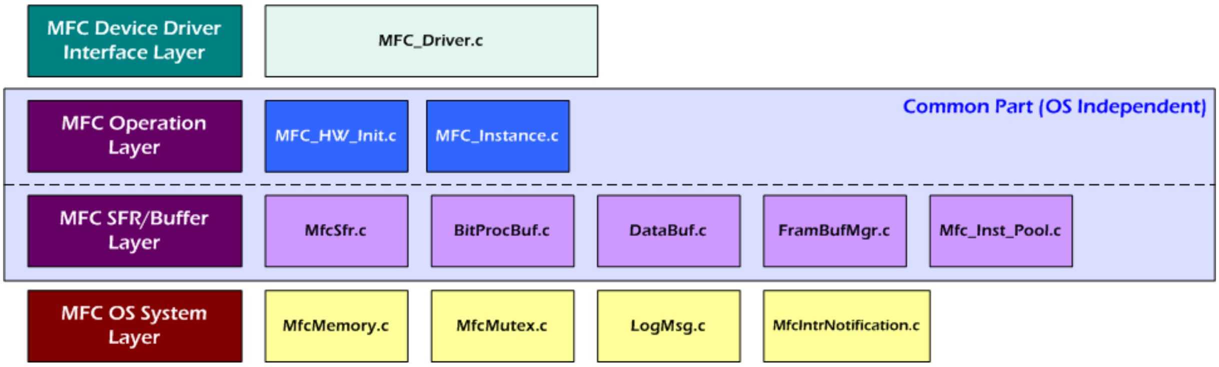

Fig. 2-1 SW Layer of MFC Driver Source Code

[MFC Device Driver Interface Layer]	It is the layer for the WinCE OS stream driver interface to be merged into one of the OEM driver. This layer is implementing the WinCE stream driver's API specification.
[MFC Operation Layer]	It is implementing the operation logics for letting the MFC HW work.
[MFC SFR/Buffer Layer]	This layer is for handling components such as MFC SFR, Bit Processor's buffer, etc.
[MFC OS System Layer]	It is an abstraction layer for the OS system call.

2.2 Installation Step (WinCE BSP)

① Copy Mfc directory to `<BSP ROOT>\Src\Drivers`.
- We have the Mfc driver source codes in `<BSP ROOT>\Src\Drivers\Mfc`.

② Open <BSP ROOT>\Src\Drivers\dirs file and add `Mfc` directory

③ Open <BSP ROOT>\Files\config.bib and add MFC memory. (Table 2-2)
- How to determine the reserved size for MFC will be explained later.

④ Open <BSP ROOT>\Files\platform.bib and add MfcDriver.dll. (Table 2-3)

⑤ Open <BSP ROOT>\Files\platform.reg and add MfcDriver.dll to registry. (Table 2-5)

⑥ Build WinCE BSP.

config.bib

```
… …
;-----------------------------------------------------
;     NAME        ADDRESS     SIZE              TYPE
;-----------------------------------------------------
      $(NKNAME)   $(NKSTART)  $(NKLEN)          RAMIMAGE
      $(RAMNAME)  $(RAMSTART) $(RAMLEN)         RAM

; Common RAM areas

      ARGS        80020800    00000800          RESERVED
      SLEEP       80028000    00002000          RESERVED

      MFC         81300000    00700000          RESERVED ; 7 MB
      DISPLAY     87000000    00F00000          RESERVED
… …
```

Table 2-2 config.bib file

platform.bib (WM 6.0)

```
MODULES
… …

      MfcDriver.dll    $(_FLATRELEASEDIR)\MfcDriver.dll    NK   SH
… …
```

Table 2-3 platform.bib file (WM6.0)

platform.bib (WinCE 6.0)

```
MODULES
… …

      MfcDriver.dll    $(_FLATRELEASEDIR)\MfcDriver.dll    NK   SHK
… …
```

Table 2-4 platform.bib file (WinCE 6.0)

platform.reg

```
… …

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\MFC]
      "Dll"="MfcDriver.dll"
      "Prefix"="MFC"
      "Index"=dword:1

… …
```

Table 2-5 platform.reg file

2.3 Memory Configuration

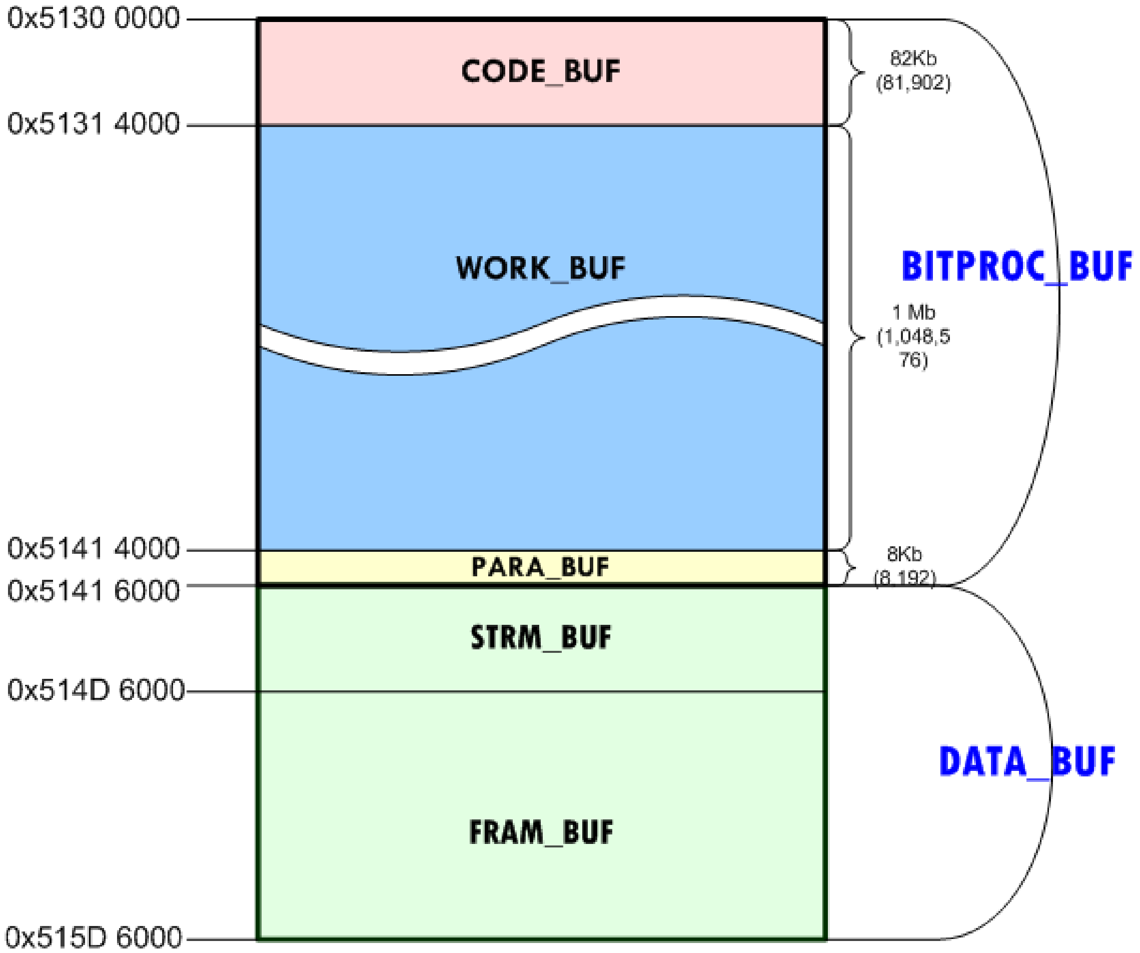

Fig. 2-2 Memory Layout in RAM for MFC

There are two memory region to be reserved for the MFC to operate. One is BITPROC_BUF and its size is fixed as 1,138,688 bytes. The other is DATA_BUF, the size for this buffer is application-dependent.

BUF name	Description		BUF size (Bytes)
BITPROC_BUF	Reserved area for the BITPROCESSOR(MFC's internal processor) Once the MFC is started, the address cannot be changed.		1,138,688
	CODE_BUF	BITPROCESSOR's F/W code	81,902
	WORK_BUF	BITPROCESSOR's working buffer	1,048,576
	PARA_BUF	Parameters on issuing command to MFC	8,192
DATA_BUF	Input/Output buffer for encoding and decoding		
	STRM_BUF	Buffer for the compressed video stream.	❶
	FRAM_BUF	Buffer for the YUV420 frame	❷

Table 2-6 MFC's Buffer Description

BITPROC_BUF is reserved for the BITPROCESSOR and the size is fixed as 1,138,688 bytes.
DATA_BUF consists of STRM_BUF and FRAM_BUF for input and output buffer. In the STRM_BUF area, we allocates the LINE_BUF

2.3.1 Configuring the MFC Buffer Address

```
Mfc\mfc_os_indep_layer\include\MfcConfig.h
```

DEFINE	Description
S3C6400_BASEADDR_MFC_SFR	• Base address of MFC SFR • [Value = 0x7e002000] is fixed. (Refer to S3C6400 Datasheet.)
S3C6400_BASEADDR_MFC_BITPROC_BUF	• Base address of MFC BITPROC_BUF • Value is in RAM region
S3C6400_BASEADDR_MFC_DATA_BUF	• Base address of MFC DATA_BUF • Value is in RAM region (Better if it is consecutive to BITPROC_BUF.)

DEFINE	Description
MFC_CODE_BUF_SIZE	• Size of CODE_BUF • [Value = 81920] is fixed.
MFC_WORK_BUF_SIZE	• Size of WORK_BUF • [Value = 1048576] is fixed.
MFC_PARA_BUF_SIZE	• Size of PARA_BUF • [Value = 8192] is fixed.
MFC_BITPROC_BUF_SIZE	• Total size of BITPROC_BUF • Value = MFC_CODE_BUF_SIZE + MFC_WORK_BUF_SIZE + MFC_PARA_BUF_SIZE

DEFINE	Description
MFC_NUM_INSTANCES_MAX	• Maximum number of MFC instances • Value = 1 ~ 8
MFC_LINE_RING_SHARE	• LINE_BUF & RING_BUF are shared? • Value = 0 or 1

DEFINE	Description
MFC_LINE_BUF_SIZE_PER_INSTANCE	• Size of LINE_BUF per instance • [Recommended Value for VGA = 200 * 1024] • [Recommended Value for QVGA = 100 * 1024]
MFC_LINE_BUF_SIZE	• MFC_LINE_BUF_SIZE_PER_INSTANCE * MFC_NUM_INSTANCES_MAX
MFC_RING_BUF_SIZE	• Size of RING_BUF • [Recommended Value = 256000 * 3]
MFC_FRAM_BUF_SIZE	• Size of FRAM_BUF • [Recommended Value = 720*480*3*4]
MFC_RING_BUF_PARTUNIT_SIZE	• Size of one PART of RING_BUF • MFC_RING_BUF_SIZE / 3
MFC_STRM_BUF_SIZE	• MFC_LINE_BUF_SIZE + MFC_RING_BUF_SIZE
MFC_DATA_BUF_SIZE	• MFC_STRM_BUF_SIZE + MFC_FRAM_BUF_SIZE

2.3.2 Required Memory Sized Calculation

In this section, how to calculate the memory will be explained step by step.

① BITPROC_BUF size is fixed. (size = 1,138,688 bytes)

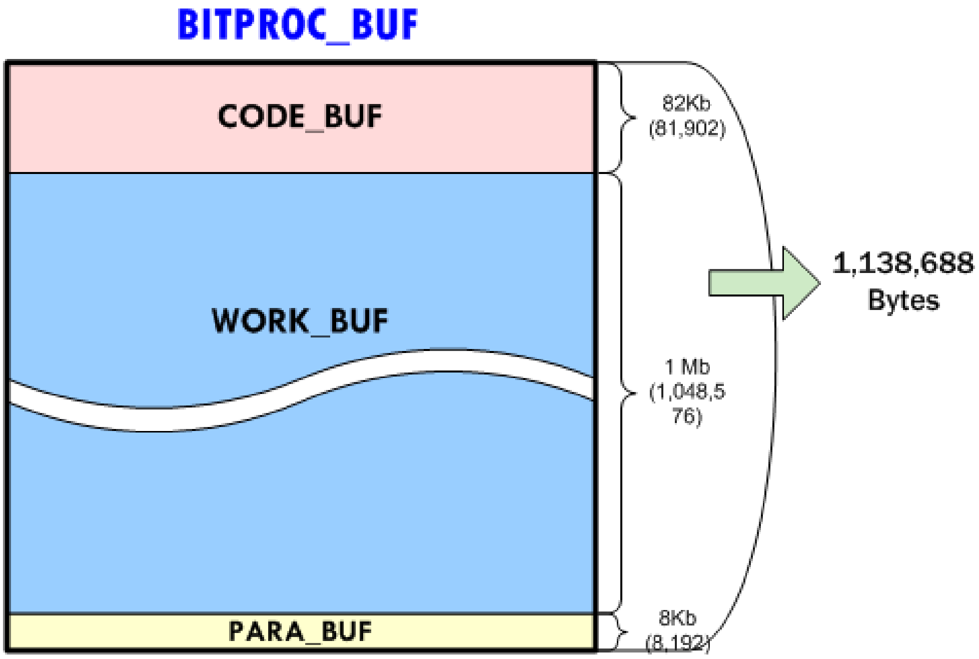

Fig. 2-3 Size of BITPROC_BUF

② STRM_BUF size calculation.

STRM_BUF has LINE_BUF and RING_BUF. They can be separated or shared.

LINE_BUF & RING_BUF are in separate mode.
As shown in Fig. 2-4, LINE_BUF and RING_BUF consume memory spaces respectively. For the 720x576-sized video, the possible maximum length of frame is 622,080 (720x576x1.5) bytes that is when it is not totally compressed. By assuming that it can be compressed to 30%, 204,800 bytes is the proper value for the maximum length of compressed frame.

Therefore, we can assume that the maximum length of compressed frame for different video size is

- SD image : 204,800 bytes or less
- CIF image : 102,400 bytes or less

LINE_BUF & RING_BUF are in shared mode.

As shown in Fig. 2-5, LINE_BUF and RING_BUF are shared so that the required size of STRM_BUF is larger one. Typically, the RING_BUF size is equal to three times of LINE_BUF_PER_INSTANCE, the size for STRM_BUF can be set 614,400 bytes for SD-sized image.

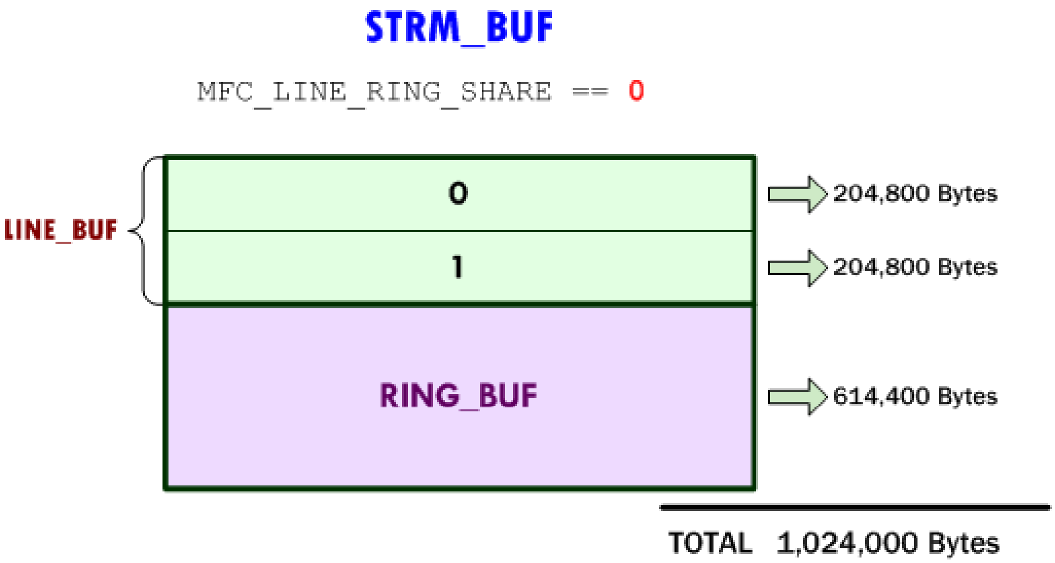

Fig. 2-4 Size of STRM_BUF with LINE_BUF and RING_BUF not-shared (Example)

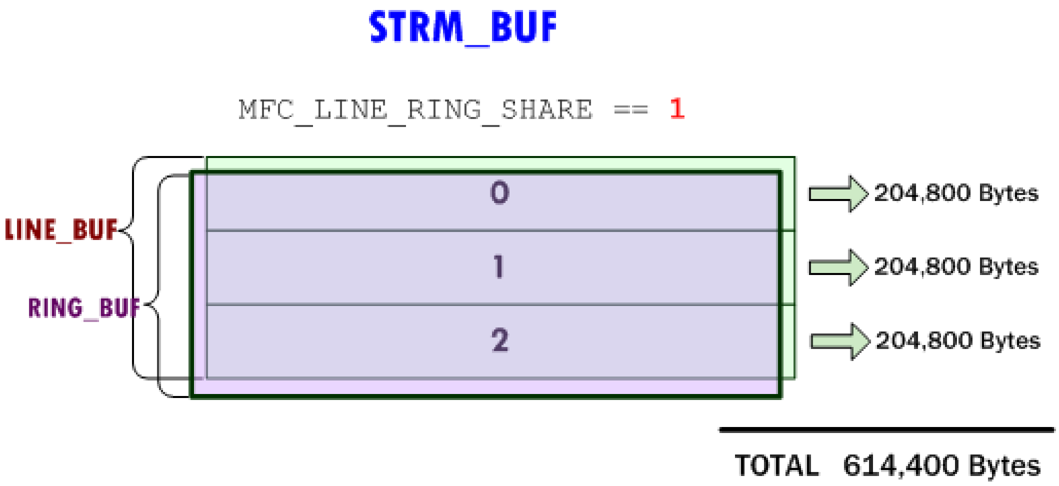

Fig. 2-5 Size of STRM_BUF with LINE_BUF and RING_BUF shared (Example)

③ FRAM_BUF size calculation.

MFC requires that the output buffer size is three or four times bigger than the YUV frame size. The number is determined by the return value (frame count in MFC DEC_PIC_RUN command.) The number is commonly 3 (three times bigger).
For the SD-sized image, the required size of FRAM_BUF is shown in Fig. 2-6.

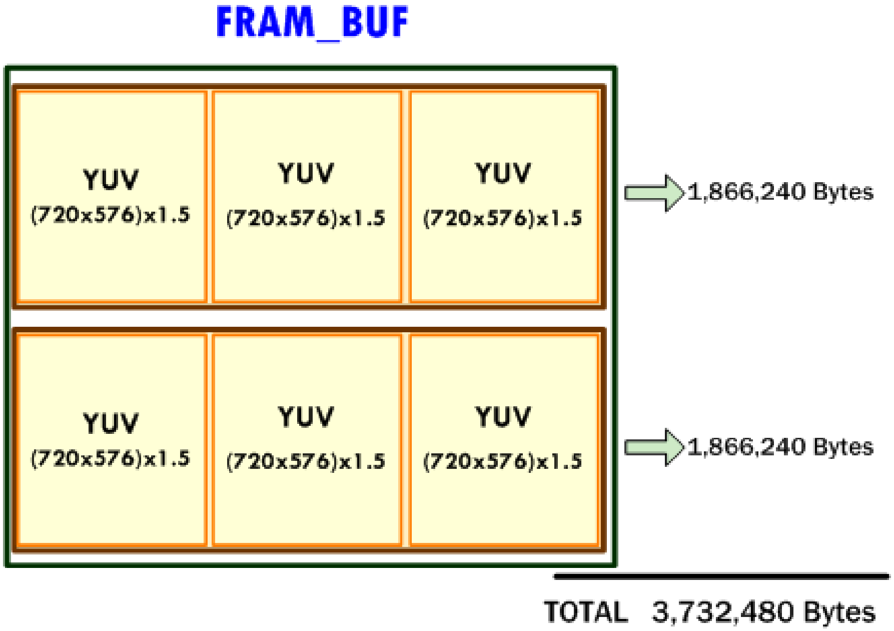

Fig. 2-6 Size of FRAM_BUF (Example)

The following tables, Table 2-7 and Table 2-8, are showing required memory size examples for common situations. If we set the MFC to support 1 SD image, it requires 3.3 MB. Since it is greater than the case of 2 CIF images, 1 SD image setting is supporting the 2CIF image setting as well.

(Unit : Bytes)

	2 SD images		1 SD image	
	LINE_RING split	LINE_RING share	LINE_RING split	LINE_RING share
BITPROC_BUF	1,138,688	1,138,688	1,138,688	1,138,688
STRM_BUF	1,024,000	614,400	512,000	307,200
FRAM_BUF	3,732,480	3,732,480	1,866,240	1,866,240
TOTAL	**5,895,168** 0x0059 F400	**5,485,568** 0x0053 B400	**3,516,928** 0x0035 AA00	**3,312,128** 0x0032 8A00

Table 2-7 Required Memory Size (1 or 2 SD images)

(Unit : Bytes)

	2 CIF images		1 CIF image	
	LINE_RING split	LINE_RING share	LINE_RING split	LINE_RING share
BITPROC_BUF	1,138,688	1,138,688	1,138,688	1,138,688
STRM_BUF	512,000	307,200	512,000	307,200
FRAM_BUF	912,384	912,384	912,384	912,384
TOTAL	**2,563,072** 0x0027 1C00	**2,358,272** 0x0023 FC00	**2,563,072** 0x0027 1C00	**2,358,272** 0x0023 FC00

Table 2-8 Required Memory Size (1 or 2 CIF images)

3 API

3.1 WinCE/WM Device Driver's APIs

API Functions	Description
CreateFile	Create the 6400 MFC instance.
DeviceIoControl	IOCTL_MFC_MPEG4_DEC_INIT IOCTL_MFC_MPEG4_ENC_INIT IOCTL_MFC_MPEG4_DEC_EXE IOCTL_MFC_MPEG4_ENC_EXE IOCTL_MFC_H264_DEC_INIT IOCTL_MFC_H264_ENC_INIT IOCTL_MFC_H264_DEC_EXE IOCTL_MFC_H264_ENC_EXE IOCTL_MFC_H263_DEC_INIT IOCTL_MFC_H263_ENC_INIT IOCTL_MFC_H263_DEC_EXE IOCTL_MFC_H263_ENC_EXE IOCTL_MFC_VC1_DEC_INIT IOCTL_MFC_VC1_DEC_EXE IOCTL_MFC_GET_LINE_BUF_ADDR IOCTL_MFC_GET_RING_BUF_ADDR IOCTL_MFC_GET_FRAM_BUF_ADDR
CloseHandle	Close the 6400 MFC instance.

3.1.1 CreateFile

CreateFile	

Syntax	HANDLE WINAPI CreateFile(
	  LPCTSTR lpFileName,
	  DWORD dwDesiredAccess,
	  DWORD dwShareMode,
	  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
	  DWORD dwCreationDisposition,
	  DWORD dwFlagsAndAttributes,
	  HANDLE hTemplateFile
	);

Description	This function creates the 6400 MFC instance. Several MFC instance can be made simultaneously. This means that CreateFile function can be called several times in a process(task).
Parameters	lpFileName [IN] : MFC's device driver name. (L"MFC1:") dwDesiredAccess [IN] : GENERIC_READ|GENERIC_WRITE dwShareMode [IN] : 0 lpSecurityAttributes [IN] : NULL dwCreationDisposition [IN] : OPEN_EXISTING dwFlagsAndAttributes [IN] : FILE_ATTRIBUTE_NORMAL hTeplateFile [IN] : NULL
Returns	HANDLE of the MFC instance. If it fails, it returns INVALID_HANDLE_VALUE.

3.1.2 DeviceIoControl

DeviceIoControl	

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

Description	Most of functions are developed in ioctl. This system call has many functions which is separated by dwIoControlCode
Parameters	hDevice [IN] : HANDLE returned by CreateFile() function dwIoControlCode [IN] : The control code for the operation. Detailed information will explain below. lpInBuffer [IN] : Structure of the MFC argument nInBufferSize [IN] : Size of MFC argument structure lpOutBuffer [OUT] : NULL nOutBufferSize [OUT] : 0 lpBytesReturned [OUT] : NULL lpOverlapped [IN] : NULL
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

3.1.3 CloseHandle

CloseHandle	

Syntax	BOOL WINAPI CloseHandle(
	  HANDLE hDevice
	);

Description	Closes an open MFC's handle.
Parameters	[IN] hDevice - HANDLE returned by CreateFile() function
Returns	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero

3.2 Control Codes for DeviceIoControl()

IOCTL_MFC_MPEG4_DEC_INIT IOCTL_MFC_H263_DEC_INIT IOCTL_MFC_H264_DEC_INIT IOCTL_MFC_VC1_DEC_INIT	
Syntax	See 3.1.2.
Description	It initializes the MFC's instance with the configure stream.
Parameters	hDevice [IN] : HANDLE returned by CreateFile() function dwIoControlCode [IN] : IOCTL_MFC_MPEG4_DEC_INIT, IOCTL_MFC_H263_DEC_INIT, IOCTL_MFC_H264_DEC_INIT, IOCTL_MFC_VC1_DEC_INIT lpInBuffer [IN] : Pointer to MFC_DEC_INIT_ARG structure. nInBufferSize [IN] : sizeof(MFC_DEC_INIT_ARG) lpOutBuffer [OUT] : NULL nOutBufferSize [OUT] : 0 lpBytesReturned [OUT] : NULL lpOverlapped [IN] : NULL
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_MFC_MPEG4_DEC_EXE IOCTL_MFC_H263_DEC_EXE IOCTL_MFC_H264_DEC_EXE IOCTL_MFC_VC1_DEC_EXE	
Syntax	See 3.1.2.

Description	It decodes the stream in the LINE_BUF or RING_BUF.
Parameters	hDevice [IN] : HANDLE returned by CreateFile() function dwIoControlCode [IN] : IOCTL_MFC_MPEG4_DEC_EXE, IOCTL_MFC_H263_DEC_EXE, IOCTL_MFC_H264_DEC_EXE, IOCTL_MFC_VC1_DEC_EXE lpInBuffer [IN] : Pointer to MFC_DEC_EXE_ARG structure. nInBufferSize [IN] : sizeof(MFC_DEC_EXE_ARG) lpOutBuffer [OUT] : NULL nOutBufferSize [OUT] : 0 lpBytesReturned [OUT] : NULL lpOverlapped [IN] : NULL
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_MFC_GET_LINE_BUF_ADDR **IOCTL_MFC_GET_RING_BUF_ADDR** **IOCTL_MFC_GET_FRAM_BUF_ADDR**	
Syntax	See 3.1.2.
Description	It obtains the address of the LINE_BUF, RING_BUF or FRAM_BUF.
Parameters	hDevice [IN] : HANDLE returned by CreateFile() function dwIoControlCode [IN] : IOCTL_MFC_GET_LINE_BUF_ADDR, IOCTL_MFC_GET_RING_BUF_ADDR, IOCTL_MFC_GET_FRAM_BUF_ADDR lpInBuffer [IN] : Pointer to MFC_GET_BUF_ADDR_ARG structure. nInBufferSize [IN] : sizeof(MFC_GET_BUF_ADDR_ARG) lpOutBuffer [OUT] : NULL nOutBufferSize [OUT] : 0 lpBytesReturned [OUT] : NULL lpOverlapped [IN] : NULL
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

3.3 Data Structure for Passing the IOCTL Arguments

3.3.1 MFC_ENC_INIT_ARG

MFC_ENC_INIT_ARG

int ret_code	[OUT] Return code
int in_width	[IN] width of YUV420 frame to be encoded
int in_height	[IN] height of YUV420 frame to be encoded
int in_bitrate	[IN] Encoding parameter: Bitrate (kbps)
int in_gopNum	[IN] Encoding parameter: GOP Number (interval of I-frame)
int in_frameRateRes	[IN] Encoding parameter: Frame rate (Res)
int in_frameRateDiv	[IN] Encoding parameter: Frame rate (Divider)

3.3.2 MFC_ENC_EXE_ARG

MFC_ENC_EXE_ARG	
int ret_code	[OUT] Return code
int out_encoded_size	[OUT] Length of Encoded video stream

3.3.3 MFC_DEC_INIT_ARG

MFC_DEC_INIT_ARG	
int ret_code	[OUT] Return code
int in_strmSize	[IN] Size of video stream filled in STRM_BUF
int out_width	[OUT] width of YUV420 frame
int out_height	[OUT] height of YUV420 frame

3.3.4 MFC_DEC_EXE_ARG

MFC_DEC_EXE_ARG	
int ret_code	[OUT] Return code
int in_strmSize	[IN] Size of video stream filled in STRM_BUF

3.3.5 MFC_GET_BUF_ADDR_ARG

MFC_GET_BUF_ADDR_ARG	
int ret_code	[OUT] Return code
int in_usr_data	[IN] User data for translating Kernel-mode address to User-mode address
int out_buf_addr	[OUT] Buffer address
int out_buf_size	[OUT] Size of buffer address

4 Annex A (H.264 Decoder Usage)

H.264 Decoding Example

```
#include <windows.h>

#include “MfcDriver.h”
#include “MfcDrvParams.h”



int h264dec_test()
{
      BOOL     r;

      int      i;
      HANDLE   hOpen;

      MFC_ARGS        mfc_args;

      unsigned char  *pStrmBuf;
      int             nStrmSize;
      int             nStrmBufSize;
      unsigned char  *pFrmeBuf;
      int             nFrmeBufSize;

      DWORD           ret;


      FILE           *fp_in = NULL;
      FILE           *fp_out;

      int             width, height;


      ////////////////////////////
      // Input Stream File Open //
      ////////////////////////////
      fp_in = fopen("\\Temp\\harryp.264", "rb");
      if (fp_in == NULL) {
            RETAILMSG(1, (L"File not found\n"));
            return -1;
      }

      /////////////////////////////
      // Output Stream File Open //
      /////////////////////////////
      fp_out = fopen("\\Temp\\Output.yuv", "wb");


      //////////////////////////////
      /////     CreateFile     /////
      //////////////////////////////
```

```
hOpen = CreateFile(L"MFC1:",
                   GENERIC_READ|GENERIC_WRITE,
                   0,
                   NULL,
                   OPEN_EXISTING,
                   FILE_ATTRIBUTE_NORMAL,
                   NULL);
if (hOpen == INVALID_HANDLE_VALUE) {
      RETAILMSG(1, (L"MFC_Open failure...\n"));
      return -1;
}


/////////////////////////////////////////////////
/////           (DeviceIoControl)           /////
/////     IOCTL_MFC_GET_INPUT_BUF_ADDR      /////
/////////////////////////////////////////////////
mfc_args.get_buf_addr.in_usr_data = GetCurrentProcessId();
r = DeviceIoControl(hOpen, IOCTL_MFC_GET_RING_BUF_ADDR,
                    &mfc_args, sizeof(MFC_GET_BUF_ADDR_ARG),
                    NULL, 0,
                    NULL,
                    NULL);
if (r == FALSE) {
      RETAILMSG(1, (L"[ERROR] IOCTL_MFC_GET_RING_BUF_ADDR error.\n"));
      CloseHandle(hOpen);
      return 0;
}
if (mfc_args.get_buf_addr.ret_code != 0) {
      RETAILMSG(1, (L"[ERROR] IOCTL_MFC_GET_RING_BUF_ADDR error.\n"));
      CloseHandle(hOpen);
      return 0;
}
pStrmBuf     = mfc_args.get_buf_addr.out_buf_addr; // Output argument
nStrmBufSize = mfc_args.get_buf_addr.out_buf_size; // Output argument


//  Fill the RING_BUF with the data from file.
//  Filling size must be "nStrmBufSize."
//  If it is less than "nStrmBufSize,"
//  then it is considered there will be no more data-filling.
nStrmSize = fread(pStrmBuf, 1, nStrmBufSize, fp_in);
if (nStrmSize < nStrmBufSize) {
      RETAILMSG(1, (L"Different size.\n"));
}


/////////////////////////////////////////////////
/////           (DeviceIoControl)           /////
/////        IOCTL_MFC_H264_DEC_INIT        /////
/////////////////////////////////////////////////
mfc_args.dec_init.in_strmSize = nStrmSize;      // Input argument
r = DeviceIoControl(hOpen, IOCTL_MFC_H264_DEC_INIT,
                    &mfc_args, sizeof(MFC_DEC_INIT_ARG),
                    NULL, 0,
                    NULL,
                    NULL);

if (!r) {
      RETAILMSG(1, (L"[DeviceIoControl] INIT return = %d.\n", r));
      CloseHandle(hOpen);
      return 0;
}
if (mfc_args.get_buf_addr.ret_code != 0) {
      RETAILMSG(1, (L"[DeviceIoControl] DEC_INIT returns = %d.\n", r));
      CloseHandle(hOpen);
      return 0;
}
width  = mfc_args.dec_init.out_width;     // Output argument
height = mfc_args.dec_init.out_height;    // Output argument


//////////////
// Decoding //
//////////////
for (i=0; i<400; i++) {


      RETAILMSG(1, (L"----------------------------------------\n"));

      /////////////////////////////////////////////////
      /////           (DeviceIoControl)           /////
      /////     IOCTL_MFC_GET_INPUT_BUF_ADDR      /////
      /////////////////////////////////////////////////
      mfc_args.get_buf_addr.in_usr_data = GetCurrentProcessId();
      r = DeviceIoControl(hOpen, IOCTL_MFC_GET_RING_BUF_ADDR,
                          &mfc_args, sizeof(MFC_GET_BUF_ADDR_ARG),
                          NULL, 0,
                          NULL,
                          NULL);
      if (r == FALSE) {
            RETAILMSG(1, (L"[ERROR] GET_RING_BUF_ADDR error.\n"));
            CloseHandle(hOpen);
            return 0;
      }
      if (mfc_args.get_buf_addr.ret_code != 0) {
            RETAILMSG(1, (L"[ERROR] GET_RING_BUF_ADDR error.\n"));
            CloseHandle(hOpen);
            return 0;
      }
      pStrmBuf     = mfc_args.get_buf_addr.out_buf_addr;
      nStrmBufSize = mfc_args.get_buf_addr.out_buf_size;


      if (nStrmBufSize > 0) {
            if (feof(fp_in)) {
                  RETAILMSG(1, (L"\n###########################"));
                  RETAILMSG(1, (L"\n##      END OF FILE      ##\n"));
                  RETAILMSG(1, (L"\n###########################"));
                  break;
            }
            nStrmSize = fread(pStrmBuf, 1, nStrmBufSize, fp_in);
            if (nStrmSize < nStrmBufSize) {
                  RETAILMSG(1, (L"\n##############################\n"));
                  RETAILMSG(1, (L"\n##                          ##"));
                  RETAILMSG(1, (L"\n##  THE LAST BLOCK OF FILE  ##"));
                  RETAILMSG(1, (L"\n##                          ##"));
                  RETAILMSG(1, (L"\n##############################\n"));
            }
      }
      else
            nStrmSize = 0;


      ////////////////////////////////////////////////
      /////           (DeviceIoControl)          /////
      /////       IOCTL_MFC_MPEG4_DEC_EXE        /////
      ////////////////////////////////////////////////
      mfc_args.dec_exe.in_strmSize = nStrmSize;
      r = DeviceIoControl(hOpen, IOCTL_MFC_H264_DEC_EXE,
                          &mfc_args, sizeof(MFC_DEC_EXE_ARG),
                          NULL, 0,
                          NULL,
                          NULL);
      if (!r) {
            RETAILMSG(1, (L"[DeviceIoControl] DEC_EXE fails.\n"));
            break;
      }

      ////////////////////////////////////////////////
      /////           (DeviceIoControl)          /////
      /////     IOCTL_MFC_GET_OUTPUT_BUF_ADDR    /////
      ////////////////////////////////////////////////
      mfc_args.get_buf_addr.in_usr_data = GetCurrentProcessId();
      r = DeviceIoControl(hOpen, IOCTL_MFC_GET_FRAM_BUF_ADDR,
                          &mfc_args, sizeof(MFC_GET_BUF_ADDR_ARG),
                          NULL, 0,
                          NULL,
                          NULL);

      if (r == FALSE) {
            RETAILMSG(1, (L"[ERROR] GET_FRAM_BUF_ADDR error.\n"));
            break;
      }
      if (mfc_args.get_buf_addr.ret_code != 0) {
            RETAILMSG(1, (L"[ERROR] GET_FRAM_BUF_ADDR error.\n"));
            break;
      }

      pFramBuf     = mfc_args.get_buf_addr.out_buf_addr;
      nFramBufSize = mfc_args.get_buf_addr.out_buf_size;


      RETAILMSG(1, (L"\nDecoding %d  frame,\n", i+1));
      if (i > 135 && i < 150)
            fwrite(pFrmeBuf, 1, (width*height*3)>>1, fp_out);
}

fclose(fp_in);
fclose(fp_out);

CloseHandle(hOpen);


return 0;
}
```