

The World Leader in High Performance Signal Processing Solutions



# **An USB Mass Storage device on bf54x/52x**



# Introduction

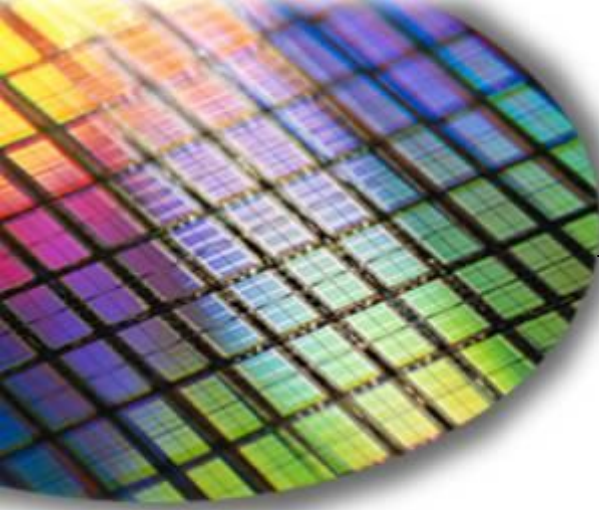
## ◆ 文档的主要内容

- 主要介绍关于**BF52x/54x**上**USB Host**驱动程序的开发知识
- 涉及到**FSS**的使用
- **USB Host** 驱动代码的组织结构，及实现逻辑

## ◆ 文档的组织结构

- 从一个**DEMO**程序开始
  - ◆ 演示程序的是一个读写U盘例子,它实现了对U盘的全部操作
  - ◆ **DEMO**程序是基于**ADI**的**System Services** 和 **Device Driver**来实现，其中**USB**驱动是我们的重点
- 细化**DEMO**程序中**USB**相关的部分,对驱动实现的过程进行分析
  - ◆ **USB Mass Storage Class Driver**
  - ◆ **OTG Controller Driver**
  - ◆ **USB PID**

◆ 如需本文**DEMO**的源代码，请联系[processor.china@analog.com](mailto:processor.china@analog.com)

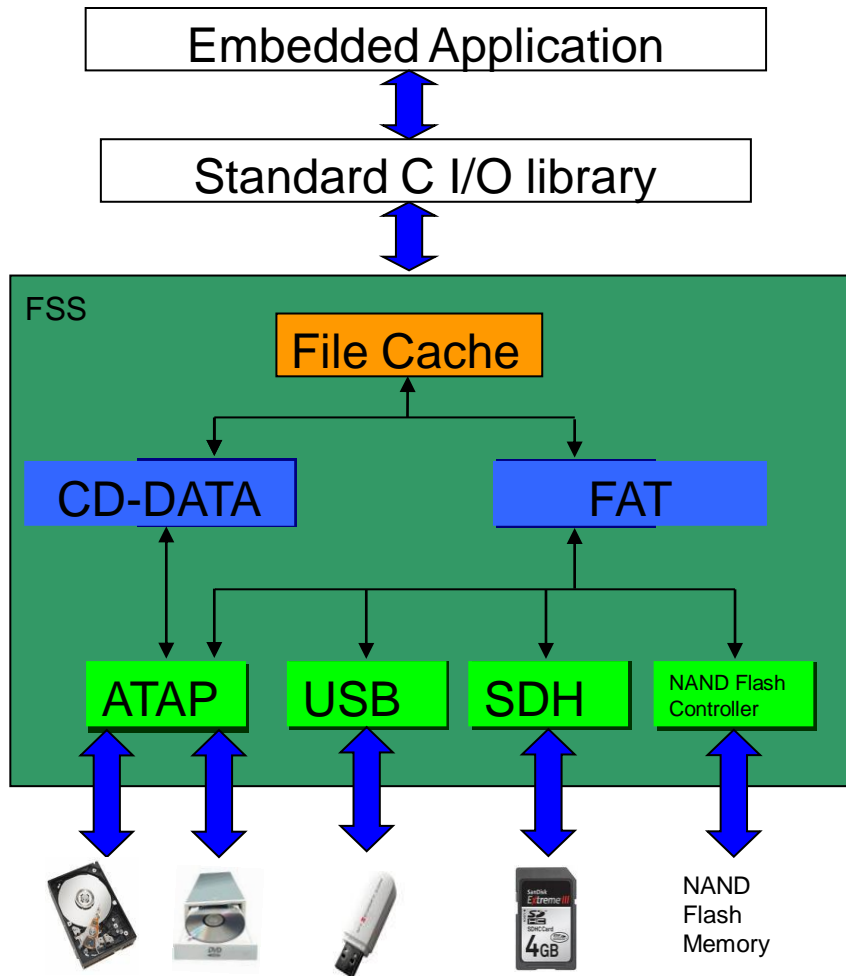


The World Leader in High Performance Signal Processing Solutions



# USB Demo

# File System Service - Architecture



## ◆ Portable

- Written in C
- Based on ADI System Services & Device Drivers
- ADSP-BF52x/BF53x/54x
- **Standard C I/O Library**

## ◆ Extensible

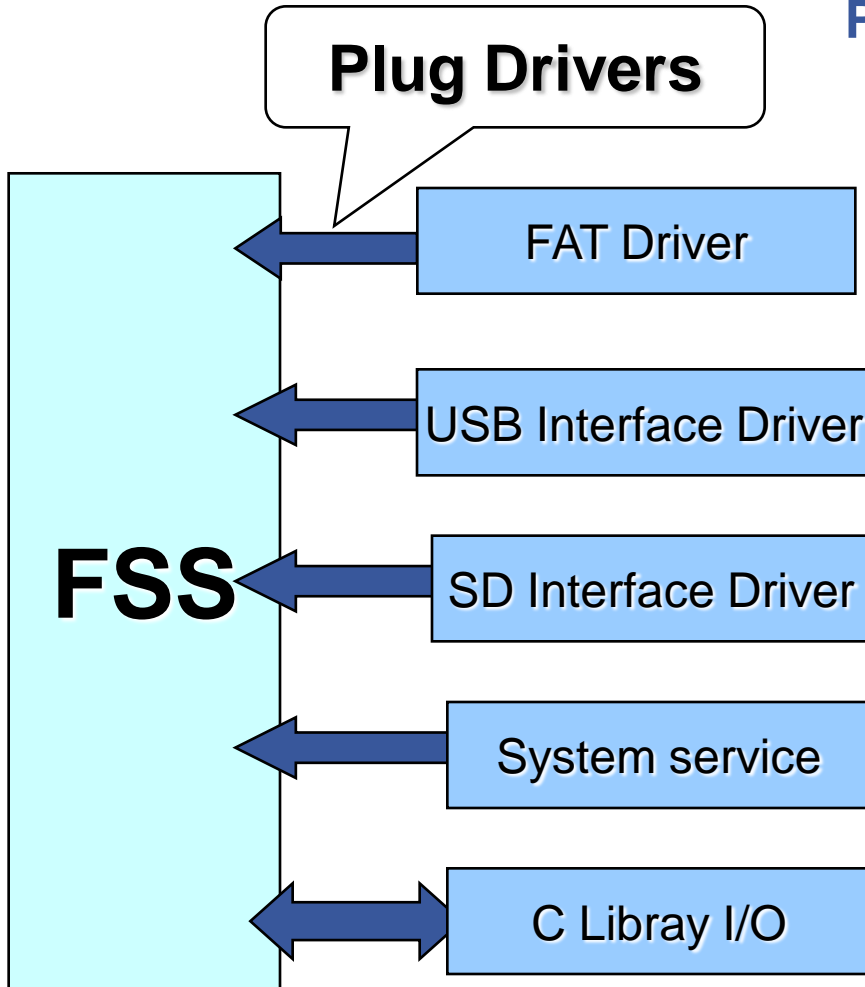
- Plug and Play architecture - easy insertion of new/alternative File Systems and Physical Interfaces

## ◆ Supported Memory Interfaces and Formats

- **File System Drivers**
  - ◆ FAT 12/16/32 with long filename support
  - ◆ In Future: ISO 9660 (CD-Audio & CD-Data), UDF (DVD)
- **Plug-In Drivers**
  - ◆ ATA/ATAPI
  - ◆ SD Card
  - ◆ USB Mass Storage
  - ◆ In Future – NAND with wear-levelling; automotive optical device support

# File System Service - Make use of it

## Plug and Play architecture

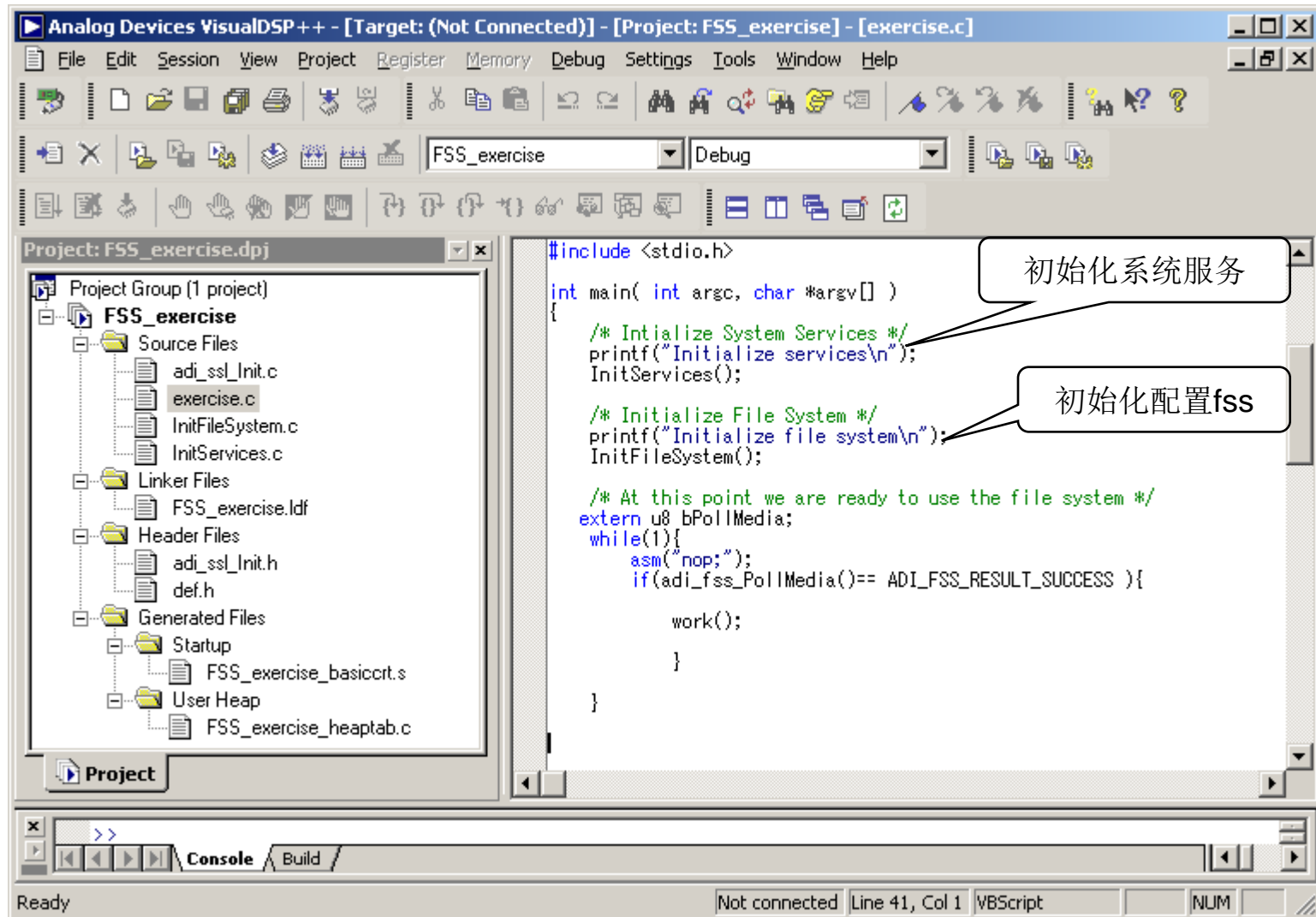


1. Register drivers with FSS
  - a. FSD (File system driver)
  - b. PID (Physical interface driver)

2. Register System Services & In
  - a. DMA component
  - b. Device Manager component
  - c. etc..

3. Standard C I/O Library
  - a. file operation
  - b. directory operation

# File System Service - Exercise



# Demo – System service Initialization

- ◆ 直接拷贝 **adi\_ssl\_init.c** 和 **adi\_ssl\_init.h** 这两个文件到你的应用程序中，并且参照示例代码中的使用方法
- ◆ 惟一需要修改的地方是**adi\_ssl\_init.h**中

```
/******  
Definitions/Sizings  
The user should modify the values in parenthesis as needed by their  
application.  
*****/  
#define ADI_SSL_INT_NUM_SECONDARY_HANDLERS (3) // number of secondary interrupt handlers  
#define ADI_SSL_DCB_NUM_SERVERS (0) // number of DCB servers  
#define ADI_SSL_DMA_NUM_CHANNELS (2) // number of DMA channels  
#define ADI_SSL_FLAG_NUM_CALLBACKS (0) // number of flag callbacks  
#define ADI_SSL_SEM_NUM_SEMAPHORES (9) // number of semaphores  
#define ADI_SSL_DEV_NUM_DEVICES (6) // number of device drivers
```



# Demo – File System Initialization

- ◆ **Register drivers with FSS**
  - **FAT file system**
    - ◆ Add header file and macro
    - ◆ Add command to initialization table
  - **USB physical interface driver**
    - ◆ Add header file and macro
    - ◆ Add command to initialization table

```
#define _ADI_FAT_DEFAULT_DEF_  
#include <drivers/fsd/fat/adi_fat.h>  
#define _ADI_USB_DEFAULT_DEF_  
#include <drivers/pid/usb/adi_usb.h>
```

```
void InitFileSystem(void)  
{  
    ADI_FSS_CMD_VALUE_PAIR adi_fss_Config[] = {  
        { ADI_FSS_CMD_ADD_DRIVER,          (void*)&ADI_FAT_Def },  
        { ADI_FSS_CMD_ADD_DRIVER,          (void*)&ADI_USB_Def }  
    };  
}
```





# Demo – File System Initialization

## ◆ Register System Services components:

- Add “adi\_ssl\_init.h” header file
- Add commands to initialization table:
  - ◆ DMA Manager
  - ◆ Device Manager

```
#define _ADI_FAT_DEFAULT_DEF_
#include <drivers/fsd/fat/adi_fat.h>
#define _ADI_USB_DEFAULT_DEF_
#include <drivers/pid/usb/adi_usb.h>
```

```
#include “adi_ssl_init.h”
```

```
void InitFileSystem(void)
{
    ADI_FSS_CMD_VALUE_PAIR adi_fss_Config[] = {
        { ADI_FSS_CMD_ADD_DRIVER,          (void*)&ADI_FAT_Def },
        { ADI_FSS_CMD_ADD_DRIVER,          (void*)&ADI_SDH_Def },
        { ADI_FSS_CMD_SET_DMA_MGR_HANDLE,  (void*)adi_dma_ManagerHandle },
        { ADI_FSS_CMD_SET_DEV_MGR_HANDLE,  (void*)adi_dev_ManagerHandle },
    }
```



# Demo – File System Initialization

## ◆ Initialize File System Service

```
#define _ADI_FAT_DEFAULT_DEF_
#include <drivers/fsd/fat/adi_fat.h>
#define _ADI_USB_DEFAULT_DEF_
#include <drivers/pid/usb/adi_usb.h>

#include "adi_ssl_init.h"
void InitFileSystem(void)
{
    ADI_FSS_CMD_VALUE_PAIR adi_fss_Config[] = {
        { ADI_FSS_CMD_ADD_DRIVER,          (void*)&ADI_FAT_Def },
        { ADI_FSS_CMD_ADD_DRIVER,          (void*)&ADI_SDH_Def },
        { ADI_FSS_CMD_SET_DMA_MGR_HANDLE,  (void*)adi_dma_ManagerHandle },
        { ADI_FSS_CMD_SET_DEV_MGR_HANDLE,  (void*)adi_dev_ManagerHandle },
        { ADI_FSS_CMD_END,                  (void*)0 }
    };

    adi_fss_Init ( adi_fss_Config );
}
```



# Demo – File System Initialization

- ◆ **Register FSS with IO library**
  - **Entry points defined in FSS header**
  - **Address passed to library function**

```
#include <services/fss/adi_fss.h>
#define _ADI_FAT_DEFAULT_DEF_
#include <drivers/fsd/fat/adi_fat.h>
#define _ADI_USB_DEFAULT_DEF_
#include <drivers/pid/usb/adi_usb.h>

#include "adi_ssl_init.h"
void InitFileSystem(void)
{
    ADI_FSS_CMD_VALUE_PAIR adi_fss_Config[] = {
        { ADI_FSS_CMD_ADD_DRIVER,          (void*)&ADI_FAT_Def },
        { ADI_FSS_CMD_ADD_DRIVER,          (void*)&ADI_SDH_Def },
        { ADI_FSS_CMD_SET_DMA_MGR_HANDLE,   (void*)adi_dma_ManagerHandle },
        { ADI_FSS_CMD_SET_DEV_MGR_HANDLE,   (void*)adi_dev_ManagerHandle },
        { ADI_FSS_CMD_END,                  (void*)0 }
    };

    adi_fss_Init ( adi_fss_Config );

    add_devtab_entry( &adi_fss_entry );
}
```

# Demo – Ready to go

- ◆ **The application is now ready to start using the file system**
  - **Add Standard C I/O header to application source**

```
#include <stdio.h>

void work(void)
{
    size_t n;
    FILE *fp;

}
```



# Demo – Write To File

- ◆ Open file for write access
- ◆ Write to file
- ◆ Close file

```
#include <stdio.h>

void work(void)
{
    size_t n;
    FILE *fp;

    /* write to a file */
    fp = fopen("junk1.dat", "w");
    fprintf(fp, "Hello World!\n");
    fclose(fp);
}
```



# Demo – Read from File

- ◆ Open file for read access
- ◆ Read file
- ◆ Close file

```
#include <stdio.h>

void work(void)
{
    size_t n;
    FILE *fp;

    /* write to a file */
    fp = fopen("junk1.dat", "w");
    fprintf(fp, "Hello World!\n");
    fclose(fp);

    /* read back from the file */
    fp = fopen("junk1.dat", "r");
    n = fread(buf, 1, 80, fp);
    printf("%d bytes read: %s\n", n, buf);
    fclose(fp);
}
```



# Demo – List Files

- ◆ List all files in USB Disk
- ◆ Print file name

```
#include <stdio.h>

void work(void)
{
    size_t n;
    FILE *fp;

    /* write to a file */
    fp = fopen("junk1.dat", "w");
    fprintf(fp, "Hello World!\n");
    fclose(fp);

    /* read back from the file */
    fp = fopen("junk1.dat", "r");
    n = fread(buf, 1, 80, fp);
    printf("%d bytes read: %s\n", n, buf);
    fclose(fp);

    /* List all files */
    if(ListDirectory("bao", FILETYPE_DIRECTORY))
        printf("Fail to list files\n");
}
```



# Demo – main

```
int main( int argc, char *argv[] )
{
    /* Intialize System Services */
    printf("Initialize services\n");
    InitServices();

    /* Initialize File System */
    printf("Initialize file system\n");
    InitFileSystem();

    /* At this point we are ready to use the file system */

    while(1){
        asm("nop;");
        if(adi_fss_PollMedia() == ADI_FSS_RESULT_SUCCESS ){

            work();

        }

    }
}
```

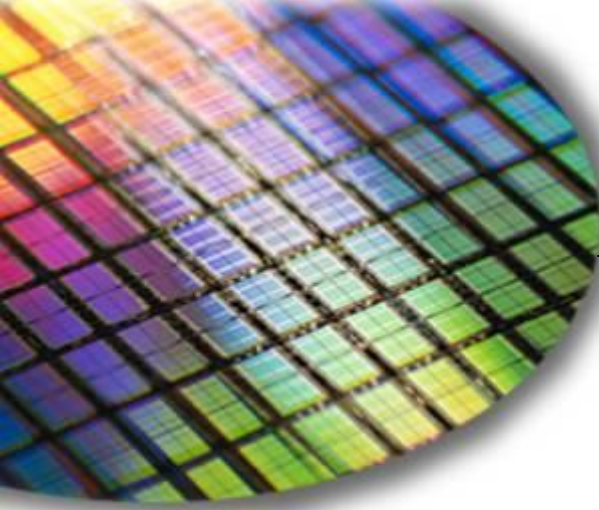




# NOTE

## Hardware

- ◆ Format your USB disk as “FAT” or ”FAT32”
- ◆ Make sure you are using Mini-A USB plugger
- ◆ Use the default Ez-Kit jumper setting
- ◆ 速度(cclk:525M,sclk:105M)
  - 2.5MB/sec
- ◆ 文件系统
  - 避免使用中文,涉及到编码的问题



The World Leader in High Performance Signal Processing Solutions



# USB Driver Architecture

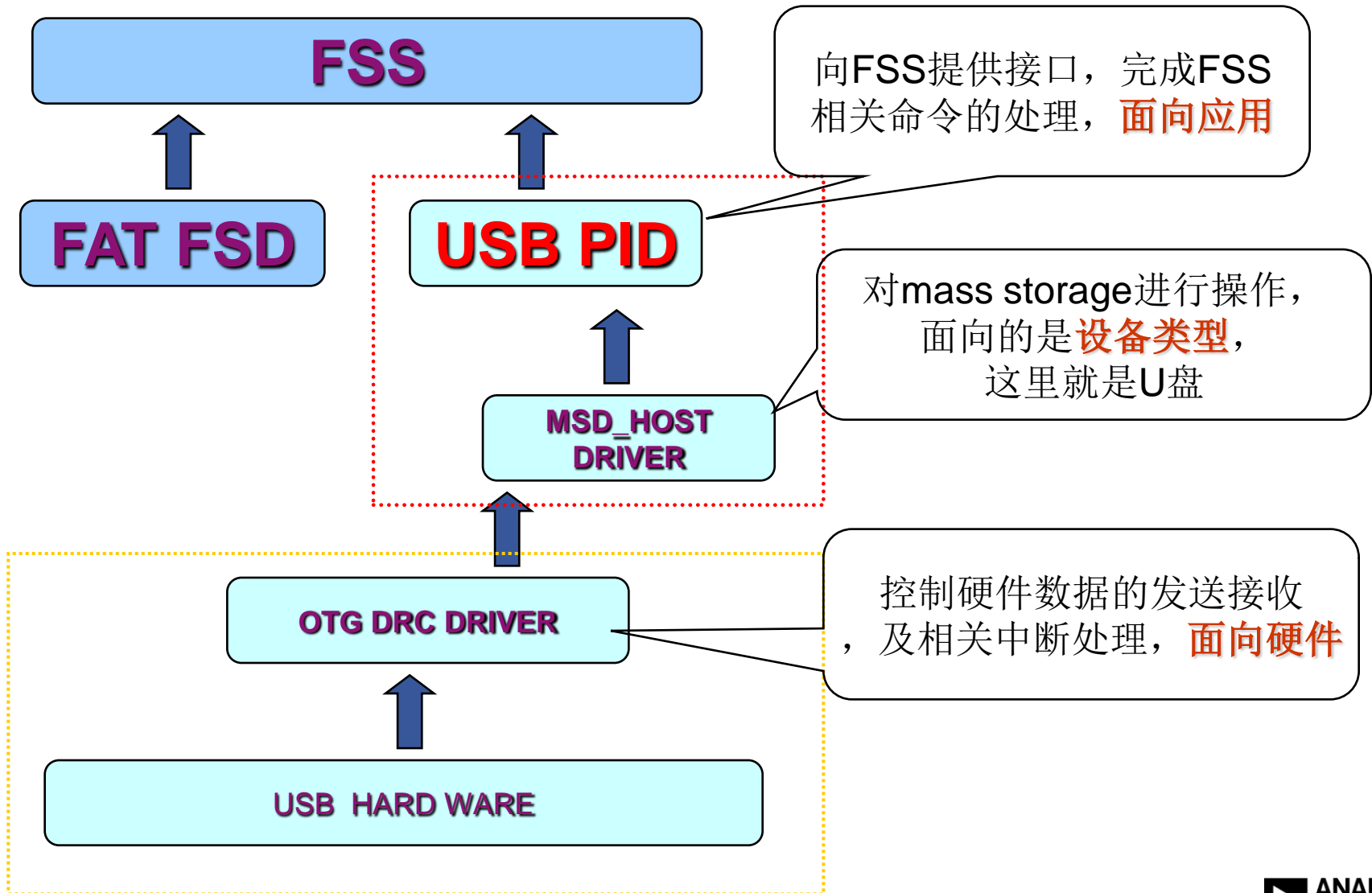
# USB Driver Architecture

注册文件系统时的参数:

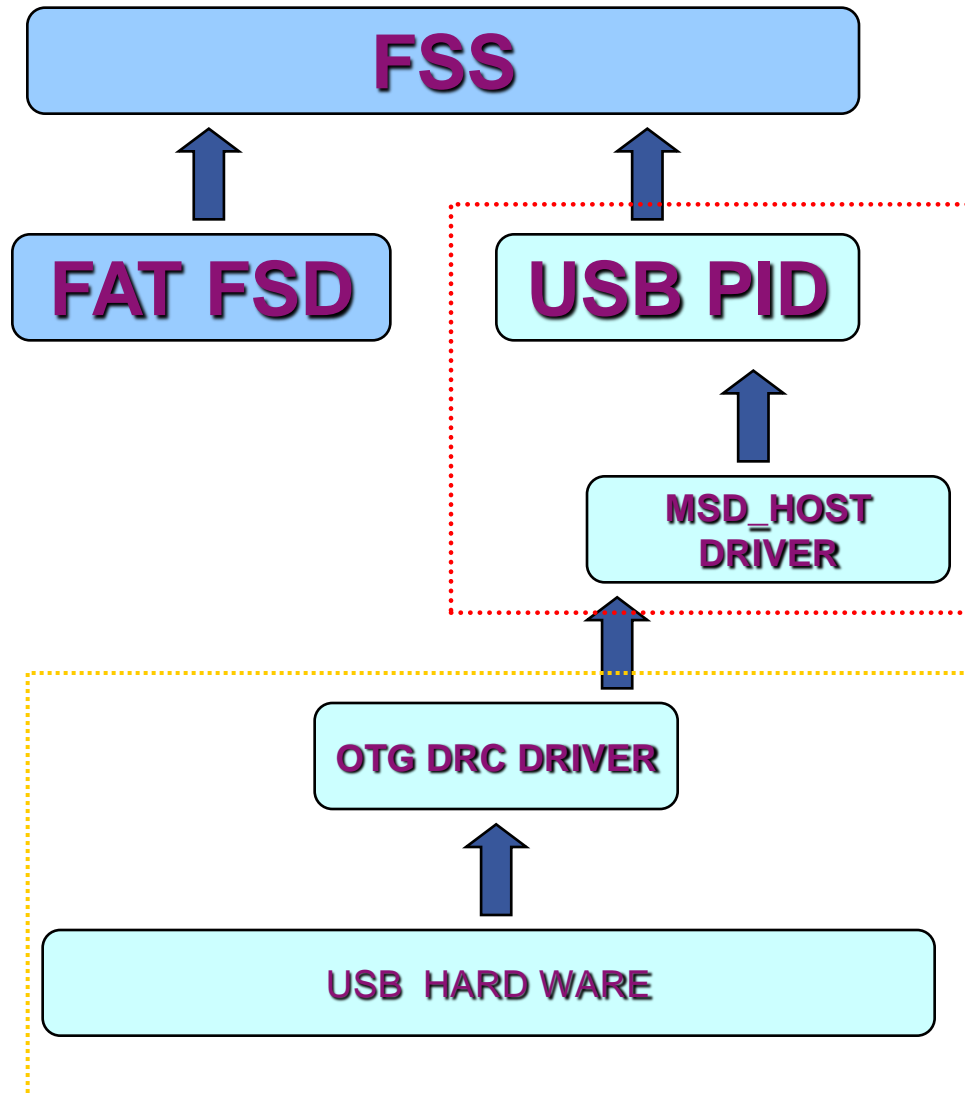
```
/* *****  
 * Configuration table to initialize the FSS  
 * ***** */  
ADI_FSS_CMD_VALUE_PAIR adi_fss_Config[] =  
{  
    { ADI_FSS_CMD_SET_NUMBER_CACHE_BLOCKS, (void*) },  
    { ADI_FSS_CMD_SET_NUMBER_CACHE_SUB_BLOCKS, (void*)1 },  
    { ADI_FSS_CMD_SET_GENERAL_HEAP_ID, (void*)GeneralHeapID },  
  
    { ADI_FSS_CMD_ADD_DRIVER, (void*)&ADI_USB_Def },  
  
    /* Add File Systems - one command per type of File System Present */  
    { ADI_FSS_CMD_ADD_DRIVER, (void*)&ADI_FAT_Def },  
    /* DMA and Device Manager Handles */  
    { ADI_FSS_CMD_SET_DMA_MGR_HANDLE, (void*)adi_dma_ManagerHandle },  
    { ADI_FSS_CMD_SET_DEV_MGR_HANDLE, (void*)adi_dev_ManagerHandle },  
    /* Command Table Terminator */  
    { ADI_FSS_CMD_END, (void*)NULL },  
};
```

USB(PID)  
Physical Interface Driver

# USB Driver Architecture-logical block



# USB Driver Architecture-logical block



结论:

USB开发的过程就是从下到上实现整个USB所能涉及到的多个设备驱动程序，从

1. 下层的硬件驱动程序DRC Driver
2. 完成特定设备功能的驱动，这里是mass storage
3. 和应用接口的驱动，这里是USB PID，（不是必需的）

需要根据应用需求来定制

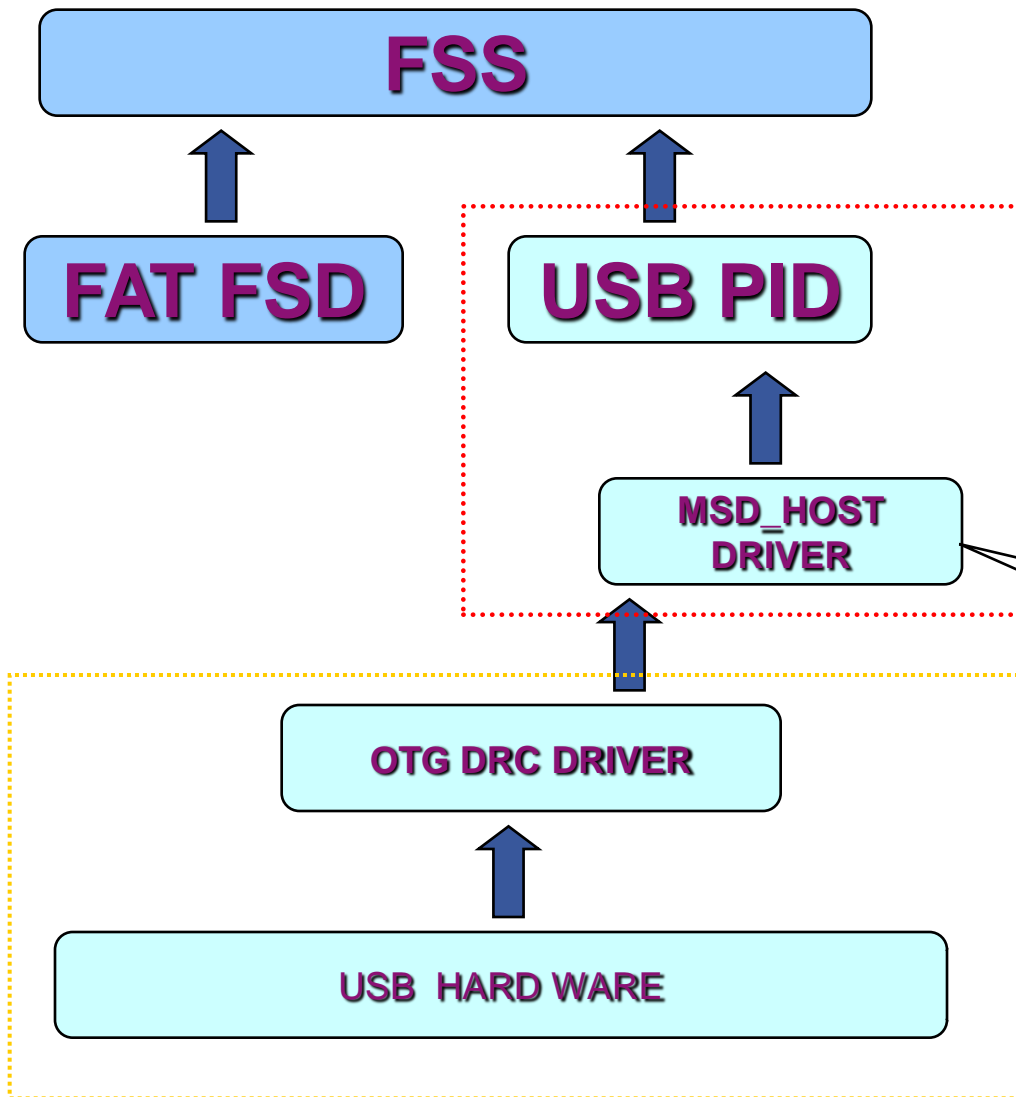
这一层ADI已经实现好了，可以直接调用



# USB Driver Architecture-development flow

- ◆ **ADI** 已经提供了完整的**USB**核和硬件控制器(**OTG Controller**)驱动
- ◆ **ADI** 已经对相关的资源有统一接口, **DMA**控制, 中断控制
- ◆ **ADI** 已经提供了进行**USB**应用开发的代码,包括**HOST**和**SLAVE**
  
- ◆ 我们的工作:
  - ◆ 分析**ADI** 提供的类设备驱动的实现, 熟悉下层驱动调用过程
  - ◆ 根据需求定义自己的类设备
  - ◆ 编写应用程序
  - ◆ 调试程序功能

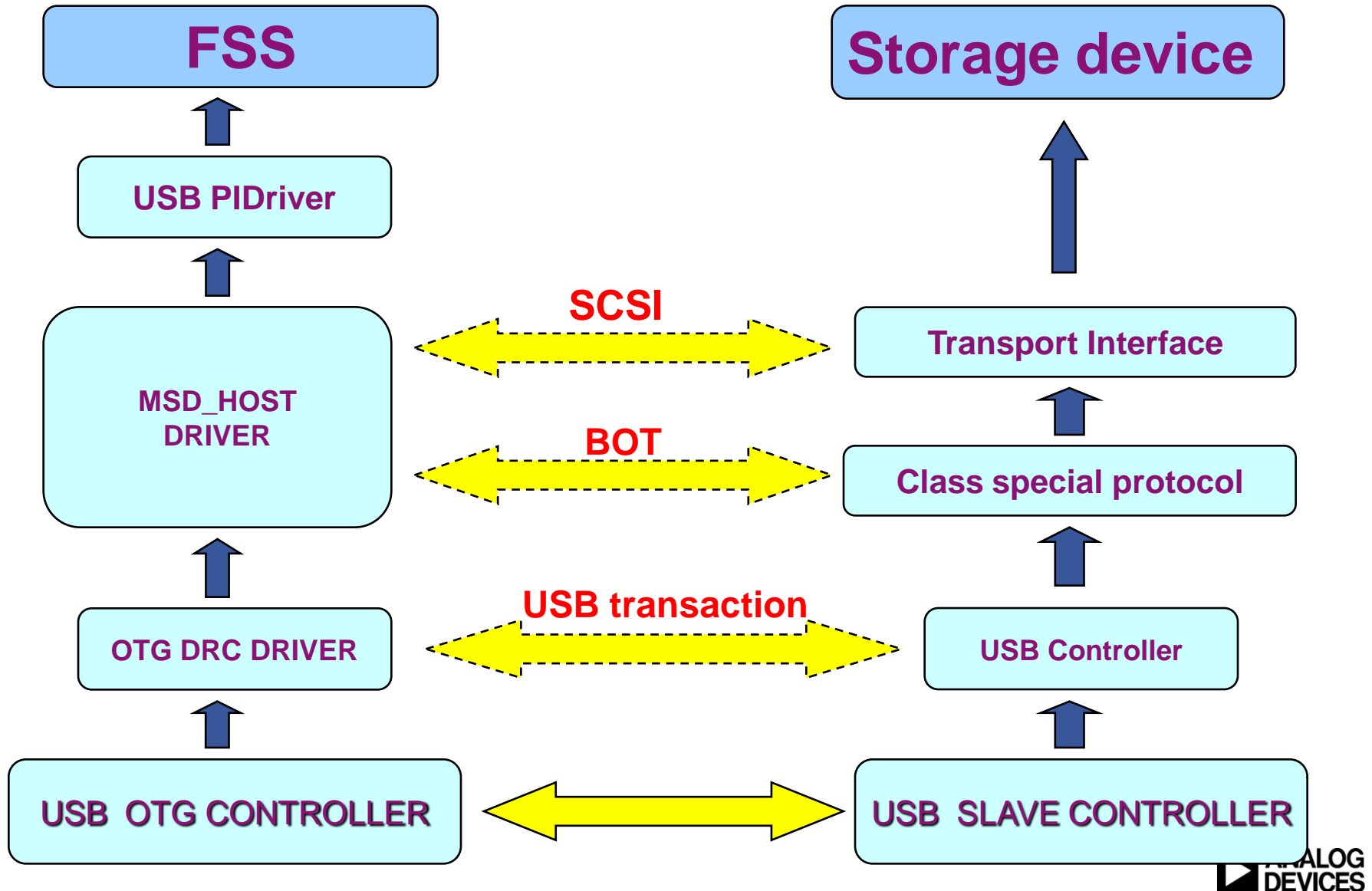
# USB Driver Architecture-MSD的实现



以下的部分我们来看一下MSD\_HOST DRIVER的实现过程

对mass storage进行操作，  
面向的是**设备类型**，  
这里就是U盘

# USB Driver Architecture





# USB Driver Architecture

## 驱动程序的组成

驱动程序 = 驱动数据结构体 + 驱动操作结构体

这个结构体包含了该驱动程序使用到相关资源如  
dmahandler, callbackhandler,  
等数据

这个结构体包含了该对外的  
接口函数  
如文件打开, 读写控制操作

```
ADI_DEV_PDD_ENTRY_POINT
ADI_USB_Host_MassStorageClass_Entrypoint = {
    adi_pdd_Open,
    adi_pdd_Close,
    adi_pdd_Read,
    adi_pdd_Write,
    adi_pdd_Control
};
```

# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypont

```
/* Mass Storage Device data structure */  
typedef struct MassStorageDeviceData
```

```
{  
    ADI_DEV_DEVICE_HANDLE DeviceHandle;  
    ADI_DMA_MANAGER_HANDLE DMAHandle;  
    ADI_DCB_CALLBACK_FN DMCallback;  
  
    ADI_DCB_HANDLE DCBHandle;  
    ADI_DEV_DIRECTION Direction;  
    void *CriticalData;  
  
    ADI_DEV_DEVICE_HANDLE PeripheralDevHandle;  
    s32 DeviceID;  
    u32 DeviceNumber;  
  
    ADI_DEV_MANAGER_HANDLE ManagerHandle;  
    ADI_DEV_PDD_HANDLE *pPDDHandle;  
  
    volatile u32 TransferComplete;  
    volatile u32 DeviceConfigured;
```

```
    ADI_DEV_1D_BUFFER *pBuffer;  
    ADI_DEV_BUFFER *pCurrentBuffer;  
  
    u32 dwCBWTransferLength;  
    u32 dwDataNotProcessedLength;  
  
    volatile u32 dwDeviceStalled;  
    volatile u32 dwDataPhase;  
  
    COMMAND_STATUS_STATE CommandState;  
    CLASS_SPECIFIC_COMMAND CSCCommand;  
  
    u32 SCSICommandSent;  
  
    u32 RemainingDataToComeIn;  
    u32 RemainingDataToSend;  
  
    u8 HostOutEP;  
    u8 HostInEP;  
  
    u32 TransferLength;  
  
    ADI_USB_HDRC_DMA_CONFIG dmaConfig;  
  
    SCSI_FORMAT_CAPACITIES_DEF FormatCapacities;  
    SCSI_CAPACITY_10_DEF CapacityTen;
```

```
} ADI_USB_DEV_DEF;
```

```
u32 (*adi_pdd_Open) (  
    ADI_DEV_MANAGER_HANDLE ManagerHandle,  
    u32 DevNumber,  
    ADI_DEV_DEVICE_HANDLE DeviceHandle,  
    ADI_DEV_PDD_HANDLE *pPDDHandle,  
    ADI_DEV_DIRECTION Direction,  
    void *pCriticalRegionA  
    ADI_DMA_MANAGER_HANDLE DMAHandle,  
    ADI_DCB_HANDLE DCBHandle,  
    ADI_DCB_CALLBACK_FN DMCallback  
);
```

根据驱动的逻辑应用需求来定义的

# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoint

```
ADI_USB_MSH_SECTION_DATA
ADI_DEV_PDD_ENTRY_POINT ADI_USB_Host_
    adi_pdd_Open,
    adi_pdd_Close,
    adi_pdd_Read,
    adi_pdd_Write,
    adi_pdd_Control
};
```

**adi\_pdd\_Open:** 初始化设备，及结构体中的相关变量

**adi\_pdd\_Read:** 实现的数据读操作；

**adi\_pdd\_Write:** 实现的数据写操作

**adi\_pdd\_Control(**  
ADI\_DEV\_PDD\_HANDLE PDDHandle, /\* PDD handle \*/  
u32 Command, /\* command ID \*/  
void \*pArg /\* pointer to argument \*/  
)

对驱动进行通过命令进行控制

**adi\_pdd\_Close:** 关闭设备释放资源

# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypont

```
static u32 adi_pdd_Open(  
    ADI_DEV_MANAGER_HANDLE    ManagerHandle,  
    u32                        DeviceNumber,  
    ADI_DEV_DEVICE_HANDLE     DeviceHandle,  
    ADI_DEV_PDD_HANDLE         *pPDDHandle,  
    ADI_DEV_DIRECTION          Direction,  
    void                       *pCriticalRegionArg,  
    ADI_DMA_MANAGER_HANDLE     DMAHandle,  
    ADI_DCB_HANDLE             DCBHandle,  
    ADI_DCB_CALLBACK_FN        DMCallback  
)  
{  
    /* Create an instance of the device driver */  
    ADI_USB_DEV_DEF *pDevice = &adi_msd_usb_host_dev_def;  
  
    pDevice->ManagerHandle = ManagerHandle;  
    pDevice->Direction      = Direction;  
    pDevice->DCBHandle      = DCBHandle;  
    pDevice->DMCallback     = DMCallback;  
    pDevice->DeviceHandle   = DeviceHandle;  
    pDevice->DeviceNumber   = DeviceNumber;  
    pDevice->DMAHandle       = DMAHandle;  
    pDevice->CriticalData   = pCriticalRegionArg;  
  
    *pPDDHandle = (ADI_DEV_PDD_HANDLE *)pDevice;  
    pDevice->pPDDHandle = (ADI_DEV_PDD_HANDLE *)pDevice;  
  
    /* Reset to initial Values */  
    ResetState(pDevice);  
  
    return(Result);  
}
```

初始化，是一个驱动执行的第一个函数

```
static void ResetState(ADI_USB_DEV_DEF *pDevice)  
{  
    pDevice->pCurrentBuffer = NULL;  
  
    pDevice->TransferComplete = FALSE;  
    pDevice->DeviceConfigured = FALSE;  
  
    pDevice->dwDeviceStalled = FALSE;  
  
    pDevice->RemainingDataToSend = 0;  
    pDevice->RemainingDataToComeIn = 0;  
  
    pDevice->CSCommand = CS_NONE;  
  
    pDevice->CommandState = STATE_CBW_SENT;  
}
```

# USB Driver Architecture

/\* Common Commands implemented by all peripheral Drivers \*/

```
enum
{
    ADI_USB_CMD_GET_DEVICE_ID = ADI_USB_ENUMERATION_START,
    ADI_USB_CMD_ENABLE_USB,
    ADI_USB_CMD_DISABLE_USB,
    ADI_USB_CMD_SET_STALL_EP,
    ADI_USB_CMD_CLEAR_STALL_EP,
    ADI_USB_CMD_SET_DEV_ADDRESS,
    ADI_USB_CMD_GET_BUFFER_PREFIX,
    ADI_USB_CMD_SET_BUFFER_PREFIX,
    ADI_USB_CMD_UPDATE_ACTIVE_EP_LIST,
    ADI_USB_CMD_ENABLE_CNTRL_STATUS_HANDSHAKE,
    ADI_USB_CMD_OTG_REQ_IN_TOKEN,
    ADI_USB_CMD_OTG_SEND_ZERO_LEN_PKT,
    ADI_USB_CMD_SET_DEV_MODE,
    ADI_USB_CMD_GET_DEV_MODE,
    ADI_USB_CMD_BUFFERS_IN_CACHE,
    ADI_USB_CMD_CLASS_ENUMERATE_ENDPOINTS,
    ADI_USB_CMD_CLASS_SET_CONTROLLER_HANDLE,
    ADI_USB_CMD_CLASS_GET_CONTROLLER_HANDLE,
    ADI_USB_CMD_CLASS_CONFIGURE,
    ADI_USB_CMD_GET_DEV_SPEED,
    ADI_USB_CMD_SET_DEV_SPEED,
    ADI_USB_CMD_SET_IVG,
    ADI_USB_CMD_GET_IVG,
    ADI_USB_CMD_ENTER_TEST_MODE,
    ADI_USB_CMD_IS_DEVICE_PRESENT,
    ADI_USB_CMD_OTG_SET_POLL_TIMEOUT,
    ADI_USB_CMD_PEEK_EP_FIFO,
    ADI_USB_CMD_STOP_EP_TOKENS,
    ADI_USB_CMD_SETUP_RESPONSE,
    ADI_USB_CMD_HIBERNATE,
    ADI_USB_CMD_RESTORE
};
```

这里定义了一个USB驱动所需要实现的相关命令



# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoint

```
static u32 adi_pdd_Control(  
    ADI_DEV_PDD_HANDLE PDDHandle,    /* PDD handle */  
    u32 Command,                      /* command ID */  
    void *pArg                        /* pointer to argument */  
)  
{  
    switch(Command)  
    {  
        case ADI_USB_CMD_CLASS_SET_CONTROLLER_HANDLE:  
            pDevice->PeripheralDevHandle = (ADI_DEV_DEVICE_HANDLE)pArg;  
            // the following line is not extensible  
            adi_msd_host_PeripheralDevHandle = (ADI_DEV_DEVICE_HANDLE)pArg;  
            break;  
  
        case ADI_USB_CMD_CLASS_CONFIGURE:  
            Result = DevMassStorageHostConfigure(pDevice);  
            break;  
  
        case ADI_USB_CMD_ENABLE_USB:  
            Result = adi_dev_Control(pDevice->PeripheralDevHandle,  
                                    Command,  
                                    (void*)pArg);  
            break;  
  
        default:  
            Result = ADI_DEV_RESULT_NOT_SUPPORTED;  
            break;  
    }  
  
    return(Result);  
}
```



# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoin

```
case ADI_USB_CMD_CLASS_SET_CONTROLLER_HANDLE:  
    pDevice->PeripheralDevHandle =(ADI_DEV_DEVICE_HANDLE)pArg;  
    // the following line is not extensible  
    adi_msd_host_PeripheralDevHandle = (ADI_DEV_DEVICE_HANDLE)pArg;  
break;
```

首先被执行命令是：**ADI\_USB\_CMD\_CLASS\_SET\_CONTROLLER\_HANDLE**，在这里的**pArg**所指的就是硬件控制器的驱动handler，使其内部的一个**DEVICE HANDLE**指向硬件控制器驱动。

### **ADI\_USB\_CMD\_CLASS\_SET\_CONTROLLER\_HANDLE:**

作用就是将类驱动和硬件控制器驱动联系在一起，使得类设备能够操作硬件





# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoin

```
case ADI_USB_CMD_CLASS_CONFIGURE:  
    Result = DevMassStorageHostConfigure(pDevice);  
break;
```

使用之前正确配置驱动，  
关键就是回调函数

### **ADI\_USB\_CMD\_CLASS\_CONFIGURE:**

这个命令对USB下层驱动作配置，其中包括设置USB工作模式，创建Device,Configure,Interface,Endpoint描述符，所调用的函数全部由下层服务来提供，特别注意这个命令的实现过程。





# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypnt

```
static u32 DevMassStorageHostConfigure(ADI_USB_DEV_DEF *pDevice)
{
    u32 Result = ADI_DEV_RESULT_SUCCESS;

    /* setup the entrypnt and peripheral handle for the usb core */
    adi_usb_SetPhysicalDriverHandle(pDevice->PeripheralDevHandle);

    /* Setup the OTG Endpoint Zero Callback */
    adi_usb_otg_SetEpZeroCallback(1,
        (ADI_DCB_CALLBACK_FN)HostEndpointZeroCompleteCallback);

    /* Set MODE to OTG Host */
    adi_usb_SetDeviceMode(MODE_OTG_HOST);

    /* Get the HDRC DMA Mode */
    pDevice->dmaConfig.DmaChannel = ADI_USB_HDRC_RX_CHANNEL;
    adi_dev_Control(pDevice->PeripheralDevHandle, ADI_USB_CMD_GET_DMA_MODE,

    /* Reset to initial Values */
    ResetState(pDevice);

    return Result;
}
```

回调函数，当下层驱动完成上层驱动下发的命令或读写操作时回调，通过它向上层传递状态消息，回调函数一般是在下层的中断服务程序中调用



# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoin

/\* USB class driver or application events \*/

ADI\_USB\_EVENT\_DATA\_RX, /\* TODO: DELETE later \*/

ADI\_USB\_EVENT\_DATA\_TX, /\* TODO: DELETE later \*/

**ADI\_USB\_EVENT\_RX\_COMPLETE, /\* Data Received \*/**

**ADI\_USB\_EVENT\_TX\_COMPLETE, /\* Data Transmitted \*/**

ADI\_USB\_EVENT\_PKT\_RCVD\_NO\_BUFFER, /\* No buffer for the received packet \*/

**ADI\_USB\_OTG\_EVENT\_ENUMERATION\_COMPLETE, /\* Device Enumeration completed \*/**

ADI\_USB\_OTG\_EVENT\_SET\_CONFIG\_COMPLETE, /\* Set Configuration successfully completed

\*/

ADI\_USB\_OTG\_EVENT\_SET\_INTERFACE\_COMPLETE, /\* Set Interface successfully completed \*/

**ADI\_USB\_EVENT\_DISCONNECT, /\* Disconnect Event \*/**

ADI\_USB\_EVENT\_RX\_STALL, /\* RX STALL Event \*/

ADI\_USB\_EVENT\_TX\_STALL, /\* TX STALL Event \*/

**ADI\_USB\_EVENT\_CONNECT, /\* Connect Event \*/**

ADI\_USB\_EVENT\_RX\_NAK\_TIMEOUT, /\* RX NAK Timeout reached \*/

ADI\_USB\_EVENT\_TX\_NAK\_TIMEOUT, /\* TX NAK Timeout reached \*/

ADI\_USB\_EVENT\_RX\_ERROR, /\* RX ERROR \*/

ADI\_USB\_EVENT\_TX\_ERROR, /\* TX ERROR \*/

ADI\_USB\_EVENT\_SET\_INTERFACE /\* Host request to set an interface \*/



# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoint

```
case ADI_USB_CMD_ENABLE_USB:  
    Result = adi_dev_Control(pDevice->PeripheralDevHandle,  
                             Command,  
                             (void*)pArg);  
break;
```

### ADI\_USB\_CMD\_ENABLE\_USB:

使能**USB**硬件，这条命令会传递给下层硬件驱动，使硬件初始化相关寄存器，配置中断等工作。

**ADI\_USB\_CMD\_ENABLE\_USB:** 硬件开始工作，对于主机来说等待设备，以进入枚举过程

# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoin

```

void HostEndpointZeroCompleteCallback
( void *Handle, u32 Event, void *pArg)
{
    switch(Event)
    {
        /* D->H transfer complete */
        3 case ADI_USB_EVENT_RX_COMPLETE:
            . . . . .

        /* H->D transfer complete */
        2 case ADI_USB_EVENT_TX_COMPLETE:
            . . . . .

        1 case ADI_USB_OTG_EVENT_ENUMERATION_COMPLETE:
            {

                case ADI_USB_EVENT_DISCONNECT:
                {

                    break;

                default:
                    break;

            }
        }
    }
}

```

# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoint

```
case ADI_USB_OTG_EVENT_ENUMERATION_COMPLETE:
{
```

```
/* Set our configuration */
```

```
adi_usb_otg_SetConfiguration(1);
```

这是标准的  
USB请求

```
PDEVICE_OBJECT    pDevO = (PDEVICE_OBJECT)pArg;
PCONFIG_OBJECT    pCfgO = pDevO->pConfigObj;
PENDPOINT_OBJECT  pEpO  = pCfgO->pActiveEpObj;
PENDPOINT_OBJECT  pActiveEpO = pEpO;
```

```
pActiveEpO->EPInfo.EpCallback = EndpointCompleteCallback_Host;
```

```
/* Get MAX LUN from the device */
```

```
GetMaxLun_Cmd(pDevice);
```

```
}
```

这是BOT，类相关命令，获取  
LUN,相当于主机向设备数据

# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoint

```
/* H->D transfer complete */
case ADI_USB_EVENT_TX_COMPLETE:
    if((pDevice->DeviceConfigured == FALSE) &&
        (pDevice->CSCommand == CS_GET_MAX_LUN))
    {
        ReadMaxLun_Cmd(pDevice);
    }
    else
    {
        if((pDevice->dwDeviceStalled == TRUE) &&
            (pDevice->DeviceConfigured == TRUE))
        {
            /* Issue a request for IN_TOKEN to complete clear_feature */
            adi_dev_Control(pDevice->PeripheralDevHandle, ADI_USB_CMD_OTC,
                pDevice->dwDeviceStalled = FALSE;
        }
    }
    break;

/* D->H transfer complete */
case ADI_USB_EVENT_RX_COMPLETE:
    if((pDevice->DeviceConfigured == FALSE) &&
        (pDevice->CSCommand == CS_GET_MAX_LUN))
    {
        /* Complete the MAX LUN transaction */
        pDevice->CSCommand = CS_NONE;
        pDevice->DeviceConfigured = TRUE;
        pDevice->DMCallback(pDevice->DeviceHandle, ADI_USB_EVENT_CONNECT, 0);
    }
    break;
```

这里当事件时说明主机的请求已经发出，开启一次读操作

当我们得到数据，并且是我们期待的命令则向上报告事件设备已连接



# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoint

设备已经就绪了，可以向上边提供数据服务了！

**pDevice->DeviceConfigured = TRUE;**

这里设备状态已经成为配置成功

```
case ADI_USB_MSD_CMD_IS_DEVICE_ENUMERATED:  
    *((u32 *)pArg) = pDevice->DeviceConfigured;  
break;
```

上层驱动可以通过这个命令来查询设备状态，而采取相应的操作，作为**MASS Storage**设备接下来就是实现**scsi**接口进行文件操作



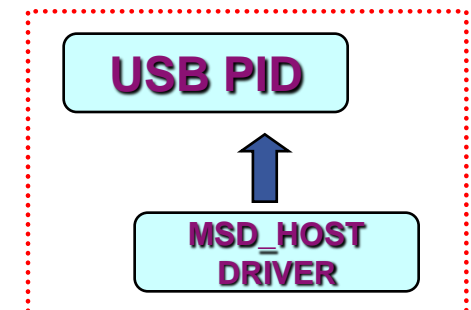
# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoint

```
#define ADI_USB_MSD_CMD_IS_DEVICE_ENUMERATED
#define ADI_USB_MSD_CMD SCSI_READ10
#define ADI_USB_MSD_CMD SCSI_WRITE10
#define ADI_USB_MSD_CMD INQUIRY
#define ADI_USB_MSD_CMD TEST_UNIT_READY
#define ADI_USB_MSD_CMD READ_FORMAT_CAPACITIES
#define ADI_USB_MSD_CMD READ_CAPACITY
#define ADI_USB_MSD_CMD MODE_SENSE6
#define ADI_USB_MSD_DISABLE_LDEB2_URGENT
#define ADI_USB_MSD_CMD_REQ_SENSE
#define ADI_USB_MSD_CMD_GET_FORMAT_CAPACITIES_DATA
#define ADI_USB_MSD_CMD_GET_CAPACITY_TEN_DATA

#define ADI_USB_MSD_EVENT_INQUIRY_COMPLETE
#define ADI_USB_MSD_EVENT_READ_FORMAT_CAPACITY_COMPLETE
#define ADI_USB_MSD_EVENT_READ_CAPACITY_COMPLETE
#define ADI_USB_MSD_EVENT_TEST_UNIT_READY_COMPLETE
#define ADI_USB_MSD_EVENT_MODE_SENSE6_COMPLETE
#define ADI_USB_MSD_EVENT_REQ_SENSE_COMPLETE
#define ADI_USB_MSD_EVENT SCSI_CMD_ERROR
#define ADI_USB_MSD_EVENT SCSI_CMD_COMPLETE
#define ADI_USB_MSD_EVENT_CSW_PHASE_ERROR
#define ADI_USB_MSD_EVENT_CSW_GARbled_ERROR
```

这些命令  
是在我们的  
MSD\_HOST  
ST驱动中  
实现的，  
它向上PID  
提供了  
SCSI服务

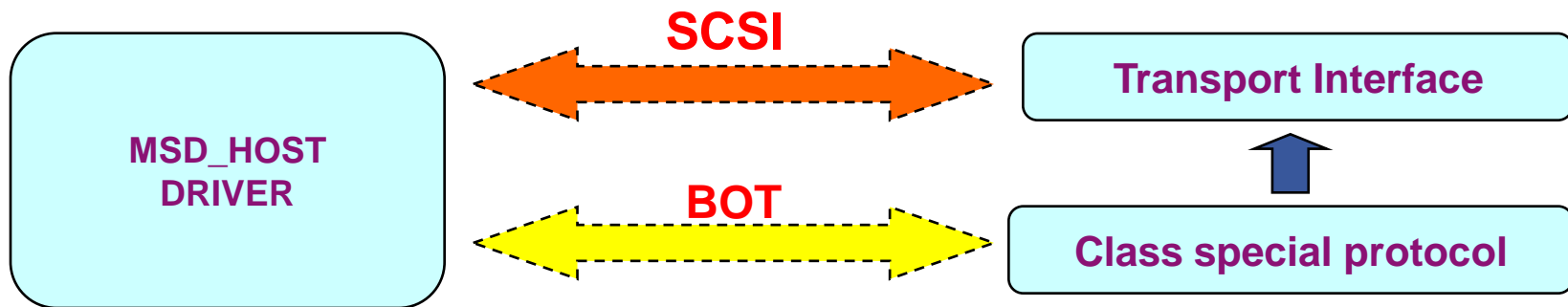




# USB Driver Architecture

## ---SCSI

```
u32 SCSI_WRITE10_CMD(ADI_USB_DEV_DEF *pDevice, u32 LBASector, u16 BlockSize);
u32 SCSI_READ10_CMD(ADI_USB_DEV_DEF *pDevice, u32 LBASector, u16 BlockSize);
u32 SCSI_TEST_UNIT_READY_CMD(ADI_USB_DEV_DEF *pDevice); //测试SCSI设备的状态
u32 SCSI_INQUIRY_CMD(ADI_USB_DEV_DEF *pDevice); //获取 设备的配置信息
u32 SCSI_READ_FORMAT_CAPACITIES_CMD(ADI_USB_DEV_DEF *pDevice);
u32 SCSI_READ_CAPACITY_CMD(ADI_USB_DEV_DEF *pDevice);
u32 SCSI_MODE_SENSE6_CMD(ADI_USB_DEV_DEF *pDevice);
```



# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypont

```
static u32 adi_pdd_Read(  
    ADI_DEV_PDD_HANDLE PDDHandle,    /* PDD handle */  
    ADI_DEV_BUFFER_TYPE BufferType,  /* buffer type */  
    ADI_DEV_BUFFER *pBuffer          /* pointer to buffer */  
)  
{  
    u32 Result = ADI_DEV_RESULT_SUCCESS;  
  
    ADI_USB_DEV_DEF *pDevice = (ADI_USB_DEV_DEF *)PDDHandle;  
  
    pDevice->pCurrentBuffer = pBuffer;  
    pDevice->TransferComplete = FALSE;  
  
    ((ADI_DEV_1D_BUFFER*)pBuffer)->ElementCount =  
        (((ADI_DEV_1D_BUFFER*)pBuffer)->ElementCount * ((ADI_DEV_1D_BUFFER*)pBuffer)->ElementWidth);  
  
    ((ADI_DEV_1D_BUFFER*)pBuffer)->ElementWidth = 1;  
  
    ((ADI_DEV_1D_BUFFER*)pBuffer)->Reserved[BUFFER_RSVD_EP_ADDRESS] = pDevice->HostInEP;  
    ((ADI_DEV_1D_BUFFER*)pBuffer)->pNext = NULL;  
  
    /* Wait until we are in DATA phase */  
    while(pDevice->dwDataPhase != TRUE)  
    {  
    };  
  
    Result = adi_dev_Read(    pDevice->PeripheralDevHandle,  
                             BufferType,  
                             pBuffer);  
  
    return(Result);  
}
```

在参数中指定使用  
哪一个Endpoint

会在回调函数中  
修改它的状态

# USB Driver Architecture

## ---ADI\_USB\_Host\_MassStorageClass\_Entrypoint

```
adi_pdd_Control(  
    case ADI_USB_CMD_ENABLE_USB:  
        Result = adi_dev_Control(pDevice->PeripheralDevHandle,  
                                Command,  
                                (void*)pArg);  
break;
```



```
void HostEndpointZeroCompleteCallback  
( void *Handle, u32 Event, void *pArg)  
{  
    ADI_USB_EVENT_CONNECT  
  
    ADI_USB_OTG_EVENT_ENUMERATION_COMPLETE:  
    /* Set our configuration */  
    adi_usb_otg_SetConfiguration(1);
```



```
adi_pdd_Read(          adi_pdd_Write(
```

等待事件完成



打开驱动

配置驱动

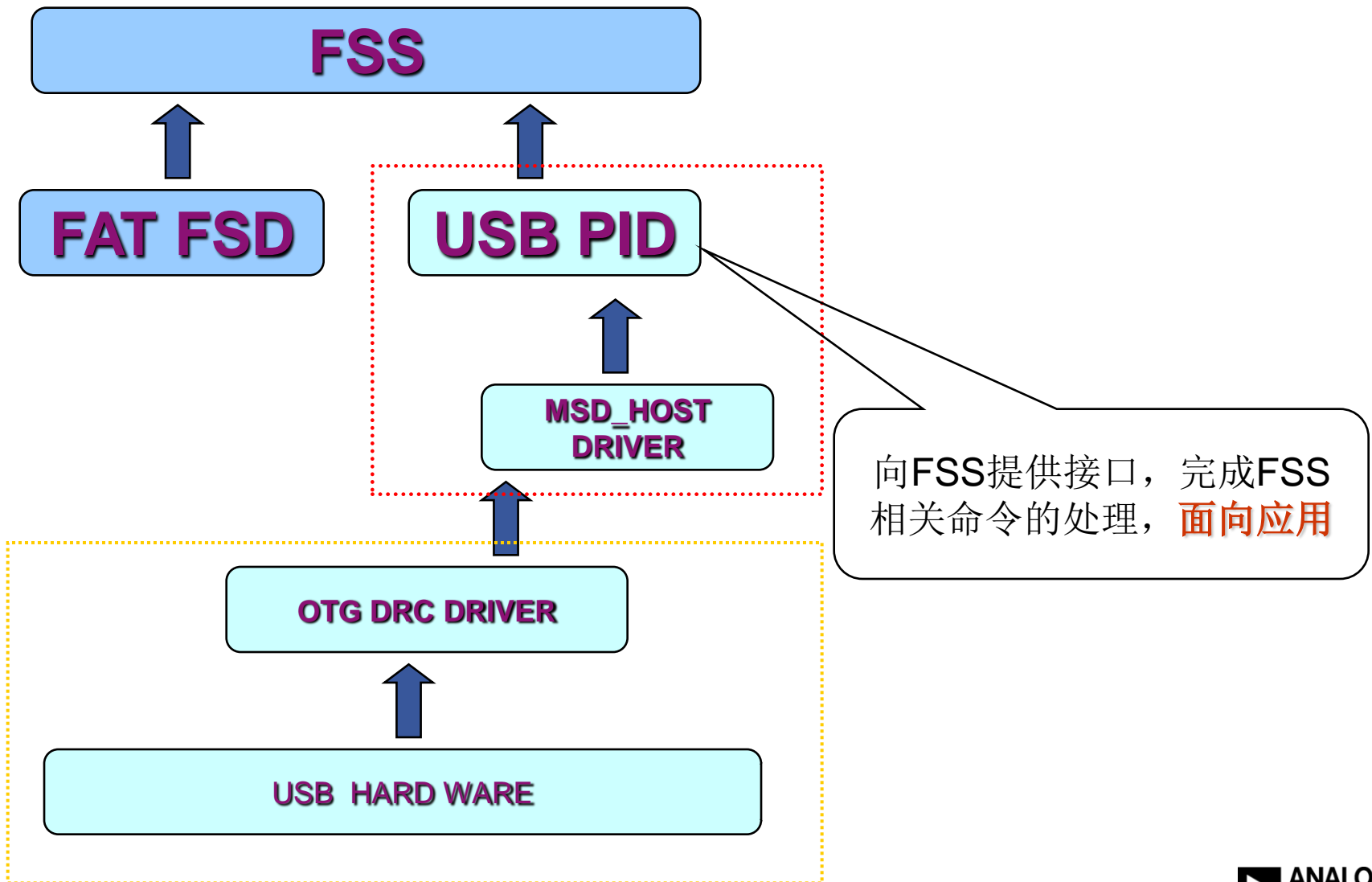
通过标准的命令序列配置设备

等待回调函数响应命令

数据读写

循环等待

# USB Driver Architecture-PID的实现





# USB Driver Architecture---usb pid

```
typedef struct ADI_USB_DEF {
    ADI_DEV_MANAGER_HANDLE ManagerHandle;           /* device manager handle */
    ADI_DMA_MANAGER_HANDLE DMAHandle;               /* handle to the DMA manager */
    ADI_DEV_DEVICE_HANDLE DeviceHandle;             /* device manager handle */
    u32 DeviceNumber;                               /* Device Number */
    ADI_DEV_DIRECTION Direction;                   /* Direction of Device Driver */
    ADI_DCB_HANDLE DCBHandle;                       /* callback manager handle */
    ADI_DCB_CALLBACK_FN DMCallback;                 /* the callback function supplied by the Device Manager */
    void *pEnterCriticalArg;                       /* critical region argument */
    int CacheHeapID;                               /* Heap Index for Device transfer buffers */
    ADI_FSS_LBA_REQUEST CurrentLBARequest;          /* The current LBA request being processed */
    ADI_SEM_HANDLE DataSemaphoreHandle;             /* Semaphore for Internal data transfers */
    ADI_SEM_HANDLE LockSemaphoreHandle;            /* Semaphore for Lock Semaphore operation */
    ADI_DEV_DEVICE_HANDLE ControllerHandle;         /* Device handle of USB controller driver */
    ADI_DEV_DEVICE_HANDLE ClassDriverHandle;        /* Device handle of USB Mass Storage Class Driver */
    u32 UseDefaultController;                      /* Flag to indicate that the default controller is used */
    u32 UseDefaultClassDriver;                     /* Flag to indicate that the default controller is used */
    u32 MediaPresent;                              /* Flag to indicate whether media is available */
    u32 ClassCommandComplete;                     /* Flag to indicate whether class driver command is completed */
    u32 SCSICommandError;                         /* Flag to indicate SCSI command error */

    #if defined(_BF527_SDRAM_ISSUE_WORKAROUND)
        ADI_DMA_STREAM_HANDLE MemDMAStreamHandle; /* Memory DMA Stream Handle for copies from internal buffer */
    #endif
} ADI_USB_DEF;
```

# USB Driver Architecture---usb pid

```
/* ****  
 * Device Driver Model entry point structure  
 * ****  
 */  
ADI_DEV_PDD_ENTRY_POINT ADI_USB_Entrypoint = {  
    adi_pdd_Open,  
    adi_pdd_Close,  
    adi_pdd_Read,  
    adi_pdd_Write,  
    adi_pdd_Control  
};
```



```
static ADI_FSS_DEVICE_DEF ADI_USB_Def = {  
    0, /* Not Applicable */  
    &ADI_USB_Entrypoint, /* Entry Points for Driver */  
    NULL, /* Command Table to configure USB Driver */  
    NULL, /* Critical region data */  
    ADI_DEV_DIRECTION_BIDIRECTIONAL, /* Direction (RW media) */  
    NULL /* Device Handle */  
};
```



```
{ ADI_FSS_CMD_ADD_DRIVER, (void*)&ADI_USB_Def },
```

# USB Driver Architecture---usb pid

```
/* ****  
 * Device Driver Model entry point structure  
 * ****  
 */  
ADI_DEV_PDD_ENTRY_POINT ADI_USB_Entrypoint = {  
    adi_pdd_Open,  
    adi_pdd_Close,  
    adi_pdd_Read,  
    adi_pdd_Write,  
    adi_pdd_Control  
};
```

这一层次的读写全部会发到下层来实现，这就是我们刚才实现的MSD\_HOST DRIVER

adi\_pdd\_Control中有关键的两条命令，来自fss的操作

```
case (ADI_PID_CMD_MEDIA_ACTIVATE):  
    Result=Activate(pDevice, (u32)Value);  
    break;  
  
case ADI_PID_CMD_POLL_MEDIA_CHANGE:  
    Result = DevicePollMedia(pDevice);  
    break;
```

这个命令将打开控制器驱动，并初始化硬件

fss检测设备的状态，是否能工作



# USB Driver Architecture---Active function

```
case (ADI_PID_CMD_MEDIA_ACTIVATE):  
    Result=Activate(pDevice, (u32)Value);  
    break;
```

————→ 我们打开所有会用到的设备驱动，并将其关联

首先我们会打开otg控制器的驱动程序，  
这个驱动程序面向硬件进行操作

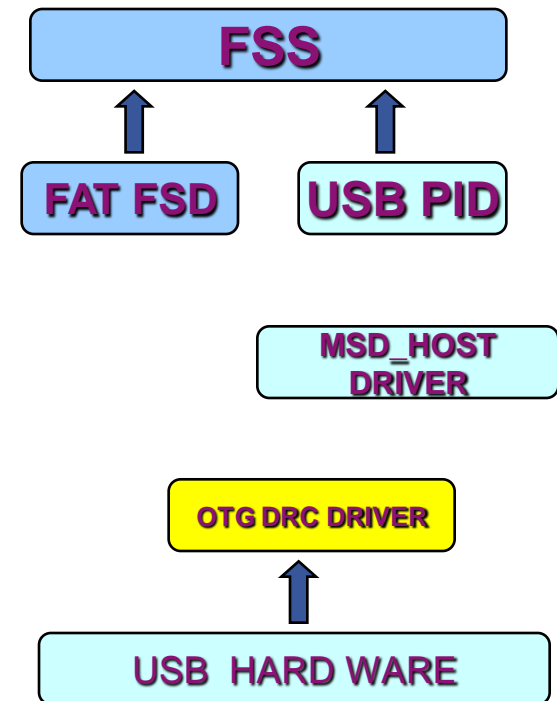
```
/******
```

Function:     Activate

Description:   Activates the host and enumerates the attached  
                  device

```
*****/  
static u32 Activate(ADI_USB_DEF *pDevice, u32 EnableFlag)
```

```
{  
  
/* Initialize the core */  
if (pDevice->UseDefaultController)  
{  
    adi_usb_CoreInit((void*)&Result);  
  
/* Open the BF54x USB Driver */  
Result = adi_dev_Open(  
    pDevice->ManagerHandle,                 /* DevMgr handle */  
    &ADI_USBDRC_Entrypoint,                 /* pdd entry point */  
    0,                                         /* device instance */  
    (void*)1,                                 /* client handle callback identifier */  
    &pDevice->ControllerHandle,               /* device handle */  
    ADI_DEV_DIRECTION_BIDIRECTIONAL,         /* data direction for this device */  
    pDevice->DMAHandle,                       /* handle to DmaMgr for this device */  
    pDevice->DCBHandle,                       /* handle to deferred callback service */  
    ControllerCallback                         /* client's callback function */  
);  
}  
}
```





# USB Driver Architecture---usb pid

```
case (ADI_PID_CMD_MEDIA_ACTIVATE):  
    Result=Activate(pDevice, (u32)Value);  
    break;
```

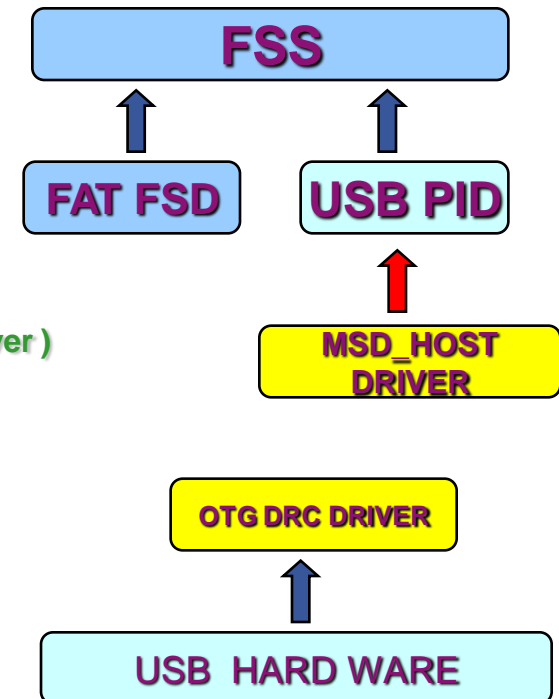
```
/******
```

Function: Activate

Description: Activates the host and enumerates the attached device

```
*****/  
static u32 Activate(ADI_USB_DEF *pDevice, u32 EnableFlag)  
{  
    .....  
    if ( Result==ADI_DEV_RESULT_SUCCESS && pDevice->UseDefaultClassDriver )  
    {  
        if ( Result==ADI_DEV_RESULT_SUCCESS && pDevice->UseDefaultClassDriver )  
        {  
            /* open the class driver */  
            Result = adi_dev_Open(  
                pDevice->ManagerHandle, /* DevMgr handle */  
                &ADI_USB_Host_MassStorageClass_Entrypoint, /* pdd entry point */  
                0, /* device instance */  
                pDevice, /* client handle callback identifier */  
                &pDevice->ClassDriverHandle, /* device handle */  
                ADI_DEV_DIRECTION_BIDIRECTIONAL, /* data direction for this device */  
                pDevice->DMAHandle, /* handle to DmaMgr for this device */  
                pDevice->DCBHandle, /* handle to deferred callback service */  
                ClassDriverCallback /* client's callback function */  
            );  
        }  
    }  
}
```

打开设备类驱动程序，这里是  
mass storage class  
这个驱动程序面向对mass storage设备  
进行操作



# USB Driver Architecture---usb pid

```
case (ADI_PID_CMD_MEDIA_ACTIVATE):  
    Result=Activate(pDevice, (u32)Value);  
    break;
```

```
if (pDevice->UseDefaultClassDriver && Result == ADI_DEV_RESULT_SUCCESS)
```

```
{
```

```
    /* configure the controller mode */
```

```
    Result = adi_dev_Control(  
        pDevice->ClassDriverHandle,  
        ADI_USB_CMD_CLASS_SET_CONTROLLER_HANDLE,  
        (void*)pDevice->ControllerHandle  
    );
```

1

```
    /* configure the class driver */
```

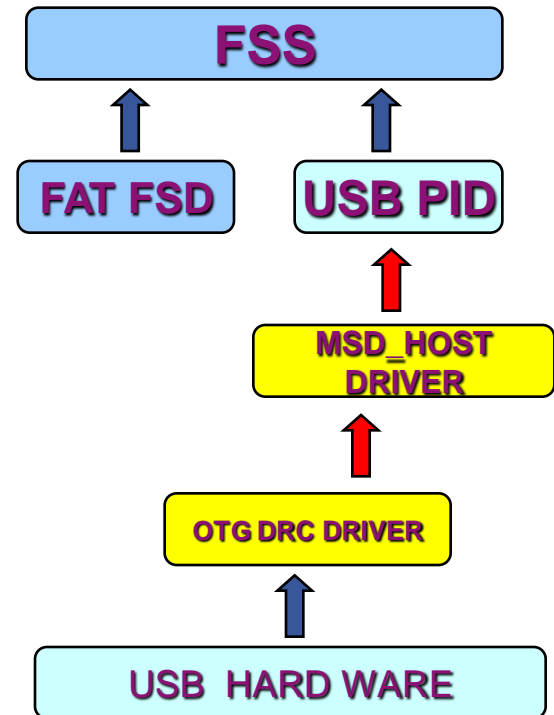
```
    if (Result == ADI_DEV_RESULT_SUCCESS)
```

```
    {  
        Result = adi_dev_Control(  
            pDevice->ClassDriverHandle,  
            ADI_USB_CMD_CLASS_CONFIGURE,  
            (void*)0  
        );  
    }
```

2

发配置命令，到我们刚才实现的函数

通过control函数来配置打开的两个设备，并初始化硬件



# USB Driver Architecture---usb pid

```
case (ADI_PID_CMD_MEDIA_ACTIVATE):
    Result=Activate(pDevice, (u32)Value);
    break;

/* configure the data flow method */
if (Result == ADI_DEV_RESULT_SUCCESS)
{
    Result = adi_dev_Control(
        pDevice->ClassDriverHandle,
        ADI_DEV_CMD_SET_DATAFLOW_METHOD,
        (void*)ADI_DEV_MODE_CHAINED
    );
}
/* enable data flow */
if (Result == ADI_DEV_RESULT_SUCCESS)
{
    Result = adi_dev_Control(
        pDevice->ClassDriverHandle,
        ADI_DEV_CMD_SET_DATAFLOW,
        (void*)TRUE
    );
}
if (Result == ADI_DEV_RESULT_SUCCESS)
{
    Result = adi_dev_Control(
        pDevice->ControllerHandle,
        ADI_USB_CMD_ENABLE_USB,
        0
    );
}
}
```

从这句开始会检测硬件插入  
以及进入**USB**的枚举进程

# USB Driver Architecture---poll media

```
case ADI_PID_CMD_POLL_MEDIA_CHANGE:
    Result = DevicePollMedia(pDevice);
    break;
```

检测设备是否可用

```
/******
```

```
Function:    DevicePollMedia
```

```
Description: Detect media change, this is used in particular to detect
              when a USB memory stick is removed or inserted.
```

```
*****/
```

```
u32 DevicePollMedia(ADI_USB_DEF *pDevice)
{
```

```
    adi_dev_Control
```

```
    (pDevice->ClassDriverHandle, ADI_USB_MSD_CMD_IS_DEVICE_ENUMERATED,
    &DeviceEnumerated);
```

→ 向下层驱动查询是否完成枚举

```
/* Perform additional communications with device to establish
 * the connection and inform the FSS of media insertion.
 */
```

```
ConfigureUsbDevice(pDevice);
```

```
/* Use Result to pass the device number */
```

```
Result = 0;
```

```
/* call back to FSS to inform it of media insertion. The FSS will unmount any previous mount, and will
 * instruct this PID to detect volumes by reading the MBR. On successful return to FSS the Result value
 * will be set accordingly. If successful the MediaPresent flag is set to true.
 */
```

```
if (FSSDirectCallbackFunction)
{
    (FSSDirectCallbackFunction)( @pDevice->DeviceHandle, ADI_FSS_EVENT_MEDIA_INSERTED, @Result );
}
```

```
else
{
    (pDevice->DMCallback) ( pDevice->DeviceHandle, ADI_FSS_EVENT_MEDIA_INSERTED, @Result );
}
```

```
-----
```

# Demo – main

```
int main( int argc, char *argv[] )
{
    /* Intialize System Services */
    printf("Initialize services\n");
    InitServices();

    /* Initialize File System */
    printf("Initialize file system\n");
    InitFileSystem();

    /* At this point we are ready to use the file system */

    while(1){
        asm("nop;");
        if(adi_fss_PollMedia()==
ADI_FSS_RESULT_SUCCESS ){

            work();

        }

    }
```



实际上调用的就是  
**DevicePollMedia**  
函数



# USB Driver Architecture

## 主要源码目录文件

### ◆ PID源码文件

- <\$VDSP>\Blackfin\lib\src\drivers\pid\usb\adi\_usb.c
- <\$VDSP>\Blackfin\include\drivers/pid/usb/adi\_usb.h

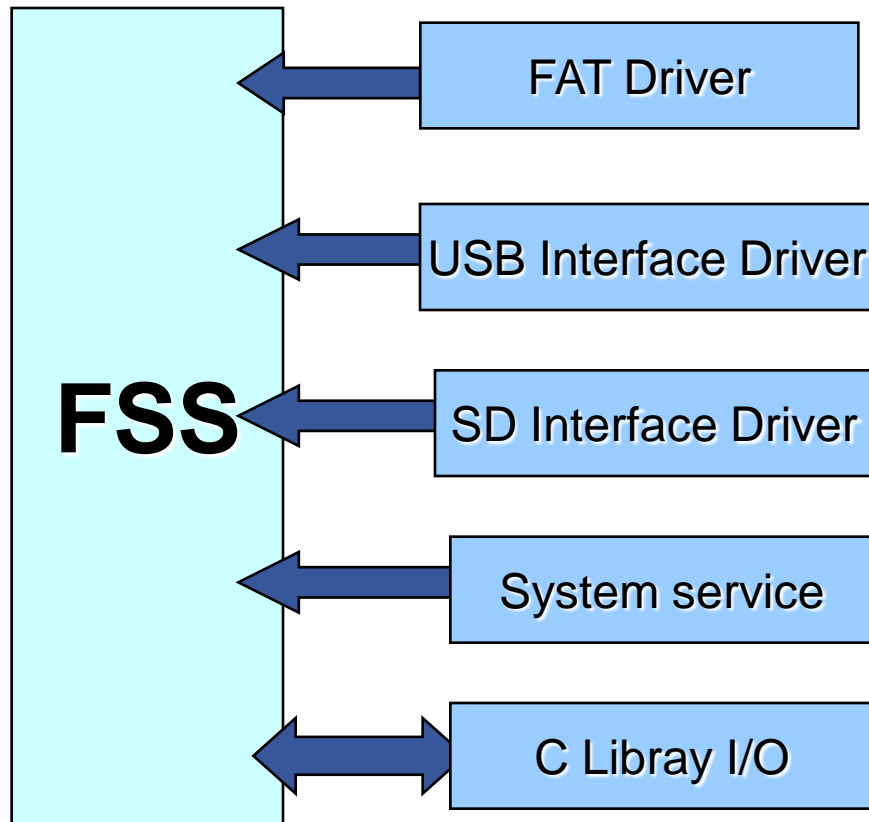
### MSD源码文件

- <\$VDSP>\Blackfin\lib\src\drivers\usb\class\otg\mass\_storage\adi\_usb\_msd\_class\_host.c
- <\$VDSP>\Blackfin\include\drivers/usb/class/otg/mass\_storage/adi\_usb\_msd\_class\_host.h

- 在下边目录中可以找到所有和usb相关的文件

<\$VDSP>\Blackfin\lib\src\drivers\usb

# Make use of it



1. Register drivers with FSS
  - a. FSD (File system driver)
  - b. PID (Physical interface driver)

2. Register System Services & In
  - a. DMA component
  - b. Device Manager component
  - c. etc..

3. Standard C I/O Library
  - a. file operation
  - b. directory operation

# 兼容性问题

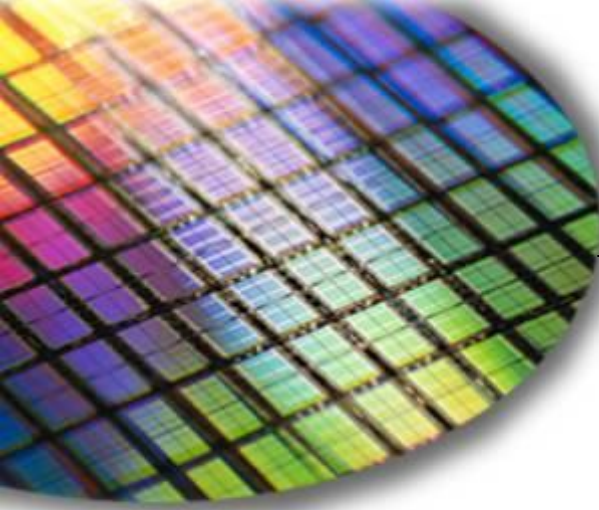
目前我们驱动程序在识别上存在兼容性问题，原因是多个方面的，可能和地域有关，各个地域生产厂家的具体实现不同，有的厂家并没有按照标准来做。

解决方法：

- 1 主要是在**MSD\_DRIVER**这个驱动里做修改
- 2 利用**Bus Hound** 这个工具来对不能识别的**U盘**进行分析
- 3 利用一些标准的测试工具来发现问题







The World Leader in High Performance Signal Processing Solutions



Thank You!

