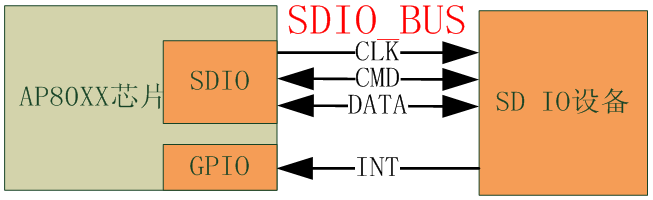
**SDIO Module Description Document**

**1. Introduction**  
SDIO is a communication protocol interface that supports the SDIO 2.0 protocol standard. It is mainly used for communication with devices such as SD Memory Cards, Wi-Fi, Bluetooth, and MMC.  
The AP80 series chip serves as an SDIO Host, and the hardware circuit connection is as follows:



Translation:

AP80XX芯片 = AP80XX chip

SD IO设备 = SD IO device

Figure 1 — SDIO application block diagram

Note: The INT signal is not required for every device. For example, an SD Memory Card does not need this signal, whereas a Wi-Fi chip does.

**2. Main Features**

1 Supports a maximum clock output of 24 MHz.

2 Supports 1-bit data transmission; 4-bit transmission is not supported.

3 Supports all CMD and Response commands.

4 Supports interrupt output signals for CMD completion and Data transfer completion.

5 Built-in 4 KB dedicated DMA for data transmission.

6 Supports 3 GPIO multiplexing configurations.

**3. Configuration Procedure**

**3.1.1 Initialization**  
Configure as follows:

1. Set the GPIO multiplexing configuration using GpioSdIoConfig(PortNum); see gpio.h for details.

2. Initialize SDIO using SdioControllerInit();.

3. If necessary, set the clock divider using SdioSetClk(DIV\_NUM);.

**3.1.2 Sending CMD and Receiving Response**

1. To send a CMD, call the following function:

bool SdioSendCommand(uint8\_t Cmd, uint32\_t Param, uint16\_t TimeOut)

**Parameter:** Cmd indicates the CMD number to be sent. For example, if CMD7, then Cmd = 7; if CMD52, then Cmd = 52.

Parameter: Param is related to CMD. For example, CMD7:



The Param part consists of RCA and stuff16, so Param = (RCA << 16) + Stuff16;

For CMD52, CMD = 0x34 = 52:



The Param part is as follows:



The code configuration is shown below:

202 **uint32\_t** SDIO\_Argument = 0;

203 SDIO\_Argument = write ? 0x80000000 : 0x00000000;

204 SDIO\_Argument |= fn << 28;

205 SDIO\_Argument |= (write && out) ? 0x08000000 : 0x00000000;

206 SDIO\_Argument |= addr << 9;

207 SDIO\_Argument |= data;

208 **if**(SdioSendCommand(SD\_IO\_RW\_DIRECT,SDIO\_Argument,100) == NO\_ERR)

209 {

210     SdioGetCmdResp(ResBuf,4);

211 }

212 **else**

213 {

214 //error

215 }

When using SdioSendCommand to send a command that does not have a response, you should ignore the return value of this function.

**2. Receiving Response**

After SdioSendCommand is executed, the actual response has already been received and stored in the internal registers of the SDIO. The software can use the function SdioGetCmdResp to read the response content.

The format of the response content: it does not include the CMD number or the CRC part.

For example, if the actual return content from SDIO\_BUS is:



The actual value returned by SdioGetCmdResp(ResBuf, 4) is:



**3.1.3 Sending Data**

Configuration method:

218 SdioSendCommand(X,X,X);

219 SdioStartSendData(pBuffer, TotalNumberOfBytes);

220 **while**(!SdioIsDatTransDone());

221 SdioEndDatTrans();

222 **while**(!SdioIsBusy());

Note:TotalNumberOfBytes must be ≤ 512 Bytes. pBuffer range: 0x20000000 ~ 0x20000000 + 96\*1024.

**3.1.4 Receiving Data**

Configuration method:

224 SdioStartReciveData(pBuffer, TotalNumberOfBytes);

225 SdioSendCommand(X,X,X);

226 **while**(!SdioIsDatTransDone());

227 SdioEndDatTrans();

Note: TotalNumberOfBytes must be ≤ 512 Bytes. pBuffer range: 0x20000000 ~ 0x20000000 + 96\*1024.