**拼接处理器在线升级固件说明**

**步骤1：**

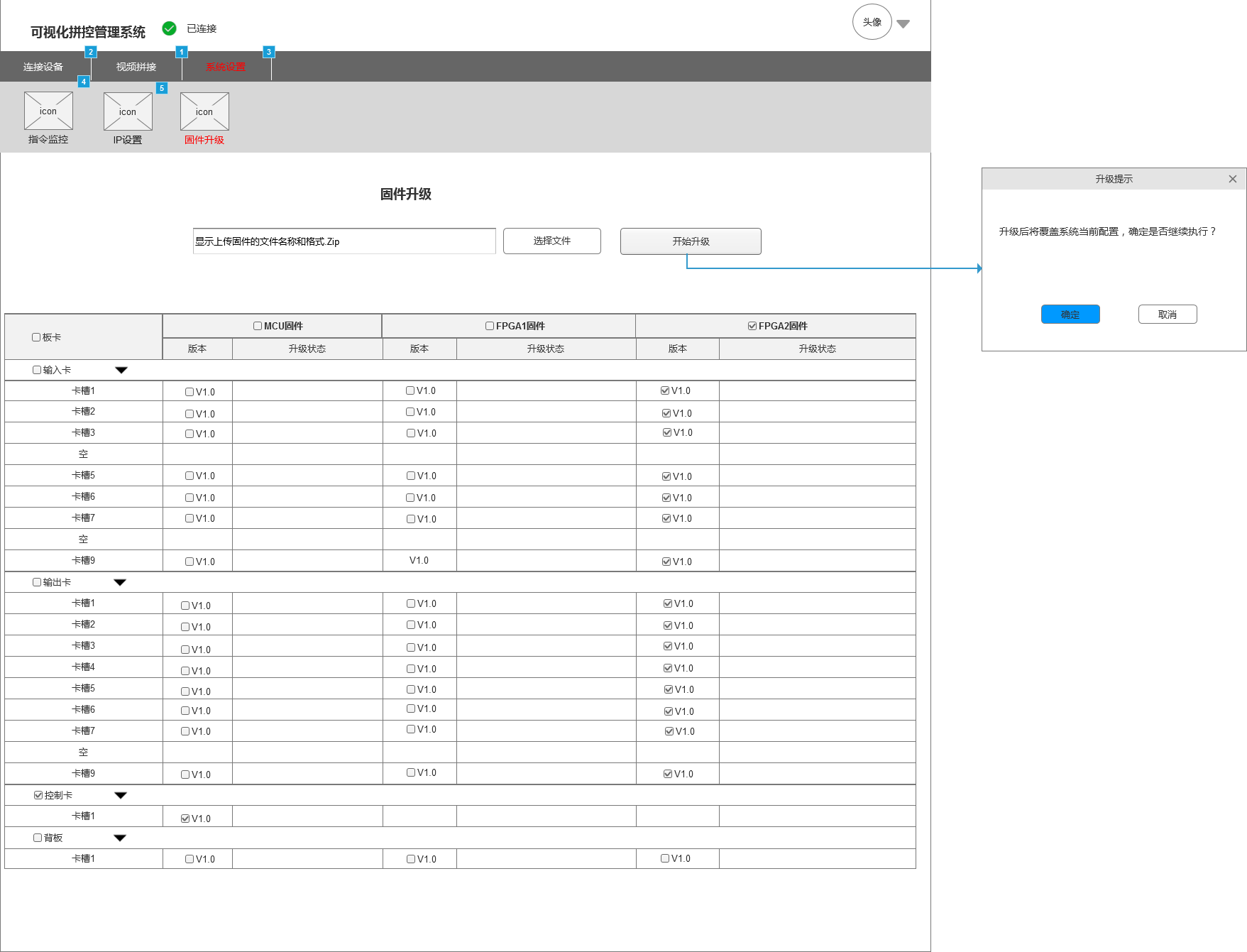
把所有板卡的升级固件（.bin文件，包含MCU和FPGA）打包压缩成一个文件（.json）。

**步骤2：**

把json文件上传到拼接软件PC端，PC软件根据协议识别固件类型（板卡类型、MCU或FPGA1或FPGA2固件和固件版本号）

**步骤3：**

在固件升级界面中选择需要更新固件的板卡（自动识别板卡类型和当前固件版本号）和升级固件类型（MCU、FPGA1或FPGA2）



**步骤4：**

升级过程：

点击“升级”后，PC软件端发送在线升级固件指令给控制卡如下图所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **十六进制指令(电脑到矩阵)** | **指令功能** | **返回**  **(十六进制，矩阵到电脑)** | **举例与说明** |
| 42 4c 13 89 00 04 00 35 x1 x2 x3 fe | 在线升级固件指令 | 0x35 | 说明：  x1：板卡类型（0：输入，1：输出）  x2：板卡通道  0：控制卡  1-4：卡槽1通道，取首通道值1  5-8：卡槽2通道，取首通道值5  ... ...  x3：固件类型  0：MCU固件  1：FPGA1固件  2：FPGA2固件  控制板重启并进入固件升级模式 |

PC软件等待控制卡返回应答信息“C”后，采用1K -Xmodem协议把固件数据发送给控制卡，然后控制卡再转发给其它板卡。

**Xmodem协议说明：**

XMODEM协议是一种串口通信中 广泛用到的异步文件传输协议。分为标准Xmodem和1k-Xmodem两种，前者以128字节块的形式传输数据，后者字节块为1k即1024字节，并且每个块都使用一个CRC检验过程来进行错误检测。在校验过程中如果接收方关于一个块的CRC检验与它在发送方的CRC检验相同时，接收方就向发送方发送一个确认字节 (ACK)。

**Xmodem协议相关控制字符**

1. SOH              0x01          //Xmodem数据头
2. STX              0x02           //1K-Xmodem数据头
3. EOT              0x04           //发送结束
4. ACK             0x06           //认可响应
5. NAK             0x15           //不认可响应
6. CAN             0x18           //撤销传送
7. CTRLZ         0x1A          //填充数据包

**标准1k -Xmodem协议（每个数据包含有1024字节数据）帧格**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte1 | Byte2 | Byte3 | Byte4--1027 | Byte1028--1029 |
| Start of Hearder | Packet Number | ~(Packet Number) | Packet Data | CRC16 |

1k Xmodem 协议的传输数据单位为信息包，包含一个标题开始字符<SOH>或者<STX>,一个单字节包序号，一个单字节包包序号的补码，1024个字节数据和一个双字节的CRC16检验。

**数据包说明**

对于标准1k-Xmodem协议来说，如果传送的文件不是1024的整数倍，那么最后一个数据包的有效内容肯定小于帧长，不足的部分需要用CTRL-Z(0x1A)来填充。

**如何启动传输**

Xmodem协议的传输由接收方启动，接收方向发送方发送"C" (ASCII字符)或者NAK（这里的NAK是用来启动传输的。下面我们用到的NAK是用来对数据产生重传机制）。其中接收方发送NAK信号表示接收方打算用累加和校验；发送字符"C"则表示接收方打算使用CRC校验。

**传输过程**

当接收方发送的第一个"C"或者NAK到达发送方，发送方认为可以发送第一个数据包了，传输启动。发送方接着接着应该将数据以每次1024字节的数据加上包头，包号，包号补码，末尾加上CRC16校验，打包成帧格式传送。发送方发了第一个包后就等待接收方的确认字节<ACK>，收到接收方传来的<ACK>确认，就认为数据包被接收方正确接收，并且接收方要求发送方继续发送下一个包；如果发送方收到接收方传来的<NAK>（这里的表示重发），则表示接收方请求重发刚才的数据包；如果发送方收到接收方传来的<CAN>字节，则表示接收方请求无条件停止传输。

**结束传输**

如果发送方正常传输完全部数据，需要结束传输，正常结束需要发送方发送<EOT>通知接收方。接收方回以<ACK>进行确认。如果接收方发送<CAN>给发送方也可以强制停止传输，发送方受到<CAN>后不需要发送<EOT>确认，此时传输已经结束。

**CRC 校验信息包**  
<STX ><blk #><255-blk #><1024data bytes><CRC hi><CRC lo>  
其中：

<STX > = 02 hex

<blk #> = 信息包序号， 从 01 开始以发送一包将加 1，加到 FF hex

<255-blk #> = 信息包序号的补码。

<CRC hi> = 信息包序号的补码。

<CRC lo> = CRC16 低字节。

2、 CRC 描述

计算 16－Bit CRC 校验的除数多项式为 X^16 + X^12 + X^5 + 1，信息包中的 1024 数据字节将参加 CRC 校验的计算。在发送端 CRC16 的高字节在先，低字节在后。  
3、 CRC 校验方式数据传输流程  
接收方要求发送方以 CRC 校验方式发送时以‘C’ 来请求， 发送方将对此做出应答。 如表 1－2  
所示传输 3 包数据的示意过程。

表 1－2 CRC 校验数据传输过程

|  |  |  |
| --- | --- | --- |
| Sender | Flow | Receiver |
|  | <--- | ‘C’ |
|  |  | Time out after 3 Second |
|  | <--- | ‘C’ |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | STX | 0x01 | 0xFE | Data[0-1024] | CRC16 | | ---> | Packet OK |
|  | <--- | ACK |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | STX | 0x02 | 0xFD | Data[0-1024] | CRC16 | | ---> | Line hit during transmission |
|  | <--- | NAK |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | STX | 0x02 | 0xFD | Data[0-1024] | CRC16 | | ---> | Packet OK |
|  | <--- | ACK |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | STX | 0x03 | 0xFC | Data[0-1024] | CRC16 | | ---> | Packet OK |
|  | <--- | ACK |
| EOT | ---> | Packet OK |
| Finished | <--- | ACK |

**CRC16检验函数：**

/\* CRC16 implementation acording to CCITT standards \*/

static const unsigned short crc16tab[256]= {

0x0000,0x1021,0x2042,0x3063,0x4084,0x50a5,0x60c6,0x70e7,

0x8108,0x9129,0xa14a,0xb16b,0xc18c,0xd1ad,0xe1ce,0xf1ef,

0x1231,0x0210,0x3273,0x2252,0x52b5,0x4294,0x72f7,0x62d6,

0x9339,0x8318,0xb37b,0xa35a,0xd3bd,0xc39c,0xf3ff,0xe3de,

0x2462,0x3443,0x0420,0x1401,0x64e6,0x74c7,0x44a4,0x5485,

0xa56a,0xb54b,0x8528,0x9509,0xe5ee,0xf5cf,0xc5ac,0xd58d,

0x3653,0x2672,0x1611,0x0630,0x76d7,0x66f6,0x5695,0x46b4,

0xb75b,0xa77a,0x9719,0x8738,0xf7df,0xe7fe,0xd79d,0xc7bc,

0x48c4,0x58e5,0x6886,0x78a7,0x0840,0x1861,0x2802,0x3823,

0xc9cc,0xd9ed,0xe98e,0xf9af,0x8948,0x9969,0xa90a,0xb92b,

0x5af5,0x4ad4,0x7ab7,0x6a96,0x1a71,0x0a50,0x3a33,0x2a12,

0xdbfd,0xcbdc,0xfbbf,0xeb9e,0x9b79,0x8b58,0xbb3b,0xab1a,

0x6ca6,0x7c87,0x4ce4,0x5cc5,0x2c22,0x3c03,0x0c60,0x1c41,

0xedae,0xfd8f,0xcdec,0xddcd,0xad2a,0xbd0b,0x8d68,0x9d49,

0x7e97,0x6eb6,0x5ed5,0x4ef4,0x3e13,0x2e32,0x1e51,0x0e70,

0xff9f,0xefbe,0xdfdd,0xcffc,0xbf1b,0xaf3a,0x9f59,0x8f78,

0x9188,0x81a9,0xb1ca,0xa1eb,0xd10c,0xc12d,0xf14e,0xe16f,

0x1080,0x00a1,0x30c2,0x20e3,0x5004,0x4025,0x7046,0x6067,

0x83b9,0x9398,0xa3fb,0xb3da,0xc33d,0xd31c,0xe37f,0xf35e,

0x02b1,0x1290,0x22f3,0x32d2,0x4235,0x5214,0x6277,0x7256,

0xb5ea,0xa5cb,0x95a8,0x8589,0xf56e,0xe54f,0xd52c,0xc50d,

0x34e2,0x24c3,0x14a0,0x0481,0x7466,0x6447,0x5424,0x4405,

0xa7db,0xb7fa,0x8799,0x97b8,0xe75f,0xf77e,0xc71d,0xd73c,

0x26d3,0x36f2,0x0691,0x16b0,0x6657,0x7676,0x4615,0x5634,

0xd94c,0xc96d,0xf90e,0xe92f,0x99c8,0x89e9,0xb98a,0xa9ab,

0x5844,0x4865,0x7806,0x6827,0x18c0,0x08e1,0x3882,0x28a3,

0xcb7d,0xdb5c,0xeb3f,0xfb1e,0x8bf9,0x9bd8,0xabbb,0xbb9a,

0x4a75,0x5a54,0x6a37,0x7a16,0x0af1,0x1ad0,0x2ab3,0x3a92,

0xfd2e,0xed0f,0xdd6c,0xcd4d,0xbdaa,0xad8b,0x9de8,0x8dc9,

0x7c26,0x6c07,0x5c64,0x4c45,0x3ca2,0x2c83,0x1ce0,0x0cc1,

0xef1f,0xff3e,0xcf5d,0xdf7c,0xaf9b,0xbfba,0x8fd9,0x9ff8,

0x6e17,0x7e36,0x4e55,0x5e74,0x2e93,0x3eb2,0x0ed1,0x1ef0

};

unsigned short crc16\_ccitt(const unsigned char \*buf, int len)

{

register int counter;

register unsigned short crc = 0;

for( counter = 0; counter < len; counter++)

crc = (crc<<8) ^ crc16tab[((crc>>8) ^ \*(char \*)buf++)&0x00FF];

return crc;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* 函数名称 : check

\*\* 功能描述 : 校验，CRC为真则为CRC校验，否则为校验和

\*\* 入口参数 : <crc>[in] 选择是CRC校验还是SUM校验

\*\* <buf>[in] 校验的原始数据

\*\* <sz>[in] 校验的数据长度

\*\* 出口参数 : 无

\*\* 返 回 值 : 校验无误返回TURE,反之返回FALSE

\*\* 其他说明 : 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static int check(int crc, const unsigned char \*buf, int sz)

{

if (crc)

{

unsigned short crc = crc16\_ccitt(buf, sz);

unsigned short tcrc = (buf[sz]<<8)+buf[sz+1];

if (crc == tcrc)

return TRUE;

}

else

{

int i;

unsigned char cks = 0;

for (i = sz; i != 0; i--)

{

cks += \*(buf++);

}

if (cks == \*buf)

return TRUE;

}

return FALSE;

}