1 基于 FX2 的 USB CDC 串口回环设计

工程源码	|--60k_FX2_CDC_Loopback |--138k_FX2_CDC_Loopback |-- FX2_CDC_Loopback
相关视频课程	本教程无视频课程
本实验支持小梅哥 XilinxAC720&高云 ACG720 及板载 USB2.0 的开发板型号	

文档所涉及文件说明

本次实验配套的文件如下图，各文件功能如下所示：

- FX2_CDC_Loopback：USB CDC 串口回环工程源码
- CDC 测试数据：用于验证回环功能正确的测试数据，用户自定义测试
- fx2lp_cdc_clrfifo_nozeropkt.iic：需要烧录的 USB 固件，将 USB 配置为串行设备

章节导读

本章主要介绍 USB CDC（Communication Device Class）协议及其在 FPGA 中的应用。CDC 是一种常用的 USB 类标准，可将设备虚拟为串口，便于与上位机进行数据通信。本设计基于 AC720&ACG720 开发板 FX2 芯片，使用 USB CDC 协议，通过 SlaveFIFO 接口实现 USB 虚拟串口的数据回环功能。同时，为了方便调试，系统可解析波特率、数据位等串口参数，最终在数码管显示，方便调试与观察。

1.1 USB CDC 介绍

USB CDC（Communication Device Class）是 USB 协议里专门定义的一类通信设备标准，它为 USB 设备间提供了一套标准的通信协议，可以让设备通过 USB 枚举成一个 虚拟串口 (Virtual COM Port)。USB CDC 的出现，极大地方便

了各种外部设备的接入和交互，特别是在嵌入式系统领域，它成为实现设备通信的重要技术。

大部分操作系统（如 Windows、Linux、MacOS）都自带 CDC 驱动，插上设备就能识别为串口，无需额外驱动。在应用层上，CDC 表现为传统 UART 串口，屏蔽了 USB 复杂的底层协议，用户只需像操作串口一样收发数据。CDC 不仅能进行普通数据通信，还能传递波特率、校验位、停止位等串口配置信息，使其更贴近真实串口的使用习惯。

相比传统 RS232 串口，USB CDC 能提供更高的传输速率和更稳定的通信能力，且线缆长度受限小。

在 USB Full-Speed (12 Mbps) 下，CDC 虚拟串口通常实际速率在 800 kbps – 1 Mbps 左右。

利用 USB 的高带宽和稳定性，同时保持传统串口的编程接口和兼容性。这样开发者能获得更快的通信速度和更好的系统兼容性，而不需要额外修改上位机程序或编写复杂的 USB 驱动。

同时，USB CDC 可以和其他接口（如 SPI、I²C、GPIO）结合使用，用于配置和数据透传，是常见的嵌入式通信桥梁。

1.2 USB CDC 基本框架与驱动原理

对于一个完整的 USB CDC 设备的实现，通常包含三个部分：**1. USB 设备端 2. 主机端 3. 设备内部的应用逻辑**。驱动模型如图 1-1 所示：

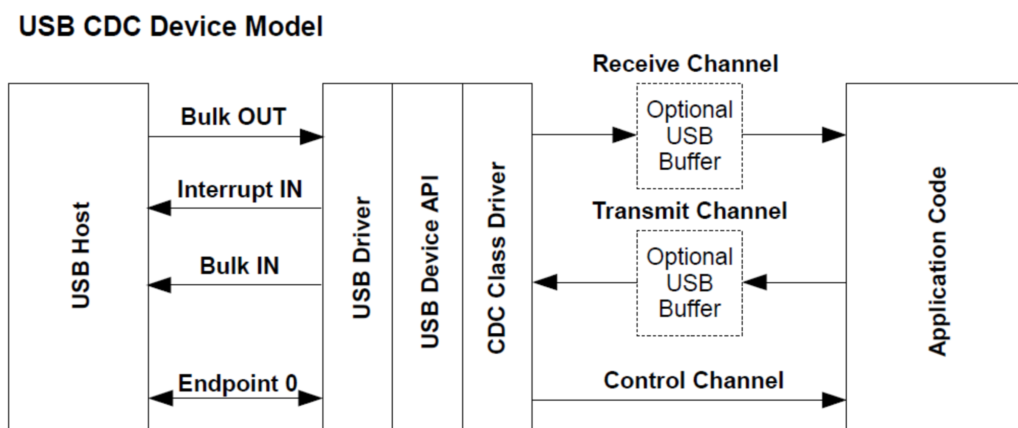

图 1-1 USB CDC 驱动模型

对于 USB 设备端，这是整个 CDC 功能的核心，用于响应主机在枚举时发出的标准 USB 请求。通过描述符 (Descriptor) 告诉主机它是一个 CDC-ACM 类

设备。具体如下：

1. 设备描述符：用来标识设备厂商、产品 ID、版本等信息。
2. 配置描述符：定义该设备的所有功能接口及其端点。
3. 接口描述符：通信接口用于处理控制和管理命令，数据接口用于实际的数据传输。
4. 端点描述符：用于定义通信的各个通道。

USB CDC 固件通过配置 Host 的设备描述符、配置描述符、接口描述符等，将 USB 接口配置为虚拟串口，再通过配置 Slave FIFO 寄存器设置端口功能和复位 FIFO 功能，如图 1-2 所示。

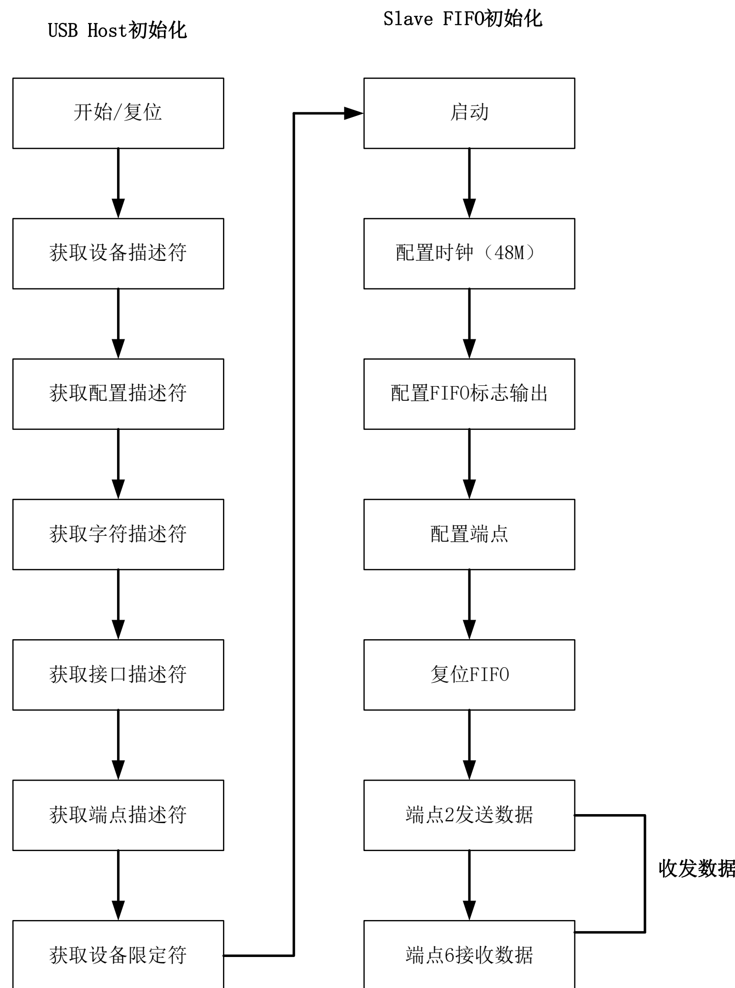

图 1-2 USB 通信流程

对于主机端，Windows 系统提供标准的 USB CDC 驱动程序。当设备使用标准的 CDC 协议枚举时，Windows 会自动加载此驱动，用户无需处理。

对应设备内部的应用逻辑，则由本次设计进行具体讲解。

1.3 USB CDC 固件介绍

如何将 USB 配置为 CDC 虚拟串口设备，关键是要把 USB 的枚举描述符和端点配置按照 CDC 类规范来实现，然后在固件中处理控制请求和收发数据。

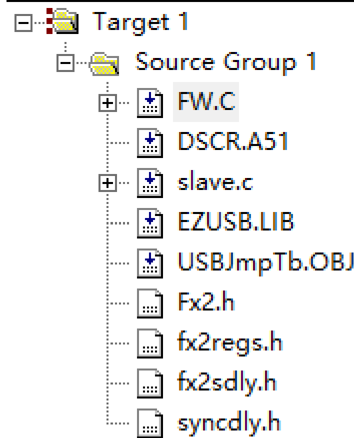

图 1-3 USB 固件文件

FW.c：为固件框架主体，包括上电枚举、重枚举、任务调度器、设备请求解析器以及上位机读参数等。

DSCR.A51：存放的就是 USB 设备的描述符信息。

slave.c：实现 USB 外设功能所需接口与 FX2 从机 FIFO 的操作以及中断服务函数。

EZUSB.LIB：USB 的库文件，包含了大量预先编写好、编译好的函数。

USBJmpTb.OBJ：目标文件，包含了中断向量跳转表。

Fx2.h：包含 FX2 的各种常量、宏、数据类型、全局变量以及库函数原型。

fx2regs.h：包含 FX2 的寄存器声明及位掩码定义。

fx2sdly.h：FX2 的同步延迟宏。

syncdly.h：FX2 的同步延迟宏。

接下来我们主要对 FW.c、DSCR.A51 和 slave.c 进行讲解。

1.3.1 DSCR.A51

DSCR.A51 包含了设备的描述符信息。设备插入后，主机读取所有这些描述符，识别出这是一个有两个接口的 CDC 设备，并加载相应的驱动程序。通过这些描述符，为主机提供设备的核心信息。当在上位机设置串口参数（如波特

率 115200）时，应用程序的调用会转化为 SET_LINE_CODING 等标准 CDC 请求，通过端点 0 发送到接口 0，固件需要解析这些请求并执行。本文件设计参考了国产沁恒的 CDC 方案，读者也可以参考如 STM32、Xilinx 的方案进行设计。

DSCR.A51 由以下部分组成：

设备描述符（DeviceDscr）：用于告知主机设备类型、设备协议以及厂家 ID 等。

通过向 DeviceDscr 放入对应值，来告诉主机设备为 CDC 类。部分代码如下：

```
db  02H            ;; 通信设备类别 02H（CDC 类）
db  00H            ;; 设备子类
db  00H            ;; 设备协议
```

设备限定描述符（DeviceQualDscr）：用于如果设备支持不同速度（如高速/全速），此描述符提供另一种速度下的基本信息。

本例中内容与设备描述符基本一致。

高速配置描述符 (HighSpeedConfigDscr)：这是设备工作在 USB 2.0 High-Speed (480 Mbps) 模式下的配置。

通过配置描述符头告知主机设备的实现等，通过接口 0（CDC）描述符定义设备类、子类、接口协议，标识此为 ACM（抽象控制模型）CDC 接口的标准代码。部分代码如下：

```
;;//Interface 0 (CDC) descriptor
    db  DSCR_INTRFC       ;; 类型：接口
    db  0                 ;; 接口编号：0
    db  0                 ;; 备用设置：0
    db  1                 ;; 端点数量：1 个 (后面的中断端点)
    db  02H               ;; 接口类：02H (CDC)**
    db  02H               ;; 接口子类：02H (Abstract Control Model)
    db  01H               ;; 接口协议：01H (AT Commands/V.250)
    db  0                 ;; 接口字符串索引：无
```

通过 CDC 功能描述符，详细定义 CDC 设备的功能。Union Functional 将接口 0 和接口 1 绑定在一起，告知主机这两个接口属于同一个功能单元（这个虚拟串口）。ACM Functional 定义了设备支持 SET_LINE_CODING, SET_CONTROL_LINE_STATE 等控制请求。部分代码如下：

```
;;//CDC Functional Descriptors (add)
    db  00H               ;; 功能描述符子类型：00H (Header Functional)
```

```
    dw  0110H           ;; CDC 规范版本：1.10

    db  02H             ;; 子类型：02H (Abstract Control Management Functional)
    db  02H             ;; 支持的功能（置位：发送 BREAK，线路编码控制）

    db  06H             ;; 子类型：06H (Union Functional)
    db  00H             ;; 主接口编号：0 (控制接口)
    db  01H             ;; 从接口编号：1 (数据接口)
```

中断 IN 端点描述符用于向主机异步地通知状态变化，例如 BREAK 信号、串行线路状态（DCD, DSR, RI 等）的变化。部分代码如下：

```
;; //Interrupt upload endpoint descriptor
    db  5               ;; 类型：端点
    db  084H            ;; **端点地址：84H (端点 4，IN 方向)**
    db  3               ;; **属性：中断传输 (ET_INT)**
    db  64              ;; 最大包大小：64 字节 (LSB)
    db  0               ;; 最大包大小：0 字节 (MSB)
    db  1               ;; 查询间隔：1 个帧
```

接口 1（CDC 数据接口）描述符明确标识这是一个专门用于数据传输的接口。部分代码如下：

```
;; //Interface 1 (data interface) descriptor */
   db  4               ;; 类型：接口
   db  0AH             ;; **接口类：0AH (CDC Data)**
```

端点描述符（Endpoint Descriptor）：定义端点的功能与大小。端点 2（out）主机->设备对应虚拟串口的接收 (RX)。端点 6 (IN)：设备 -> 主机。对应虚拟串口的发送 (TX)。包大小 512：是 USB 高速模式下批量和中断端点的最大允许值，确保了高吞吐量。部分代码如下

```
;; Endpoint Descriptor (Bulk-OUT)
    db  02H             ;; **端点地址：02H (端点 2，OUT 方向)**
    db  ET_BULK         ;; **属性：批量传输**
    db  00H             ;; 最大包大小：512 字节 (LSB)
    db  02H             ;; 最大包大小：512 字节 (MSB) 0x200 = 512
    db  00H             ;; 查询间隔：0 (批量传输忽略)


;; Endpoint Descriptor (Bulk-IN)
```

```
db  86H            ;; **端点地址：86H (端点 6，IN 方向)**
db  ET_BULK           ;; **属性：批量传输**
db  00H            ;; 最大包大小：512 字节 (LSB)
db  02H            ;; 最大包大小：512 字节 (MSB)
db  00H            ;; 查询间隔：0
```

全速配置描述符 (FullSpeedConfigDscr)：定义了设备如果运行在 USB Full-Speed (12 Mbps) 模式下的配置。其结构与高速配置类似，但通常端点数量和功能会简化，并且最大包大小更小。由于描述符需 2 字节对齐，因此在全速配置描述符中添加 1 字节使描述符 2 字节对齐。本例未使用全速模式。

```
db  00H;;补全，使后面的 StringDscr0 描述符为 2 字节对齐
```

字符串描述符(StringDscr0-3)：用于提供可读的文本类信息。如 StringDscr0 语言 ID 09H,04H 代表英语，StringDscr1 厂商字符串 wch.cn 代表南京沁恒，StringDscr2 代表产品字符串 USB Serial，StringDscr4 为序列号字符串 0123456789。

1.3.2 slave.c

slave.c 实现了将 FX2 芯片配置为一个 USB 从设备（Slave），并通过 Slave FIFO 模式与主机（PC）进行高速数据传输。

在上电后，slave.c 会执行初始化函数，将 CPU 的工作频率设为 48MHz，将 FLAG B 设置为端点 2（OUT）FIFO 的空标志（低有效），FLAG C 配置为端点 6（IN）FIFO 满标志（低有效），端点 4 设置为中断端点，并将端点都设置为最大缓冲 512 字节。部分代码如下：

```
 //设置 8051 的工作频率为 48MHz
CPUCS = 0x12; // CLKSPD[1:0]=10, for 48MHz operation, output CLKOUT

//配置 FIFO 标志输出，FLAG B 配置为 EP2 OUT FIFO 空标志
PINFLAGSAB = 0x81;          // FLAGB - EP2EF
SYNCDELAY;

//配置 FIFO 标志输出，FLAG C 配置为 EP6 IN FIFO 满标志
PINFLAGSCD = 0x1E;          // FLAGC - EP6FF
SYNCDELAY;

//将 EP2 断端点配置为 BULK-OUT 端点，使用 4 倍缓冲，512 字节 FIFO
EP2CFG = 0xA0;             //out 512 bytes, 4x, bulk
SYNCDELAY;
//将 EP6 配置为 BULK-IN 端点，
EP6CFG = 0xE0;             // in 512 bytes, 4x, bulk
SYNCDELAY;
```

```
//将端点 4 设置为中断端点
  EP4CFG = 0xF0;              // in 512 bytes, 4x, bulk
```

通过设置接口模式，将 FX2 配置为一个从机 FIFO 存储器，由外部主卡或内部来读写 FIFO。

```
//Slave 使用内部 48MHz 的时钟
  IFCONFIG = 0xF3; //Internal clock, 48 MHz, Slave FIFO interface，时钟极性翻转
  SYNCDELAY;
```

在每次设备启动时，即每次执行 TD_Init()时，都会进行一次 FIFO 复位，确保了 USB 批量传输端点的 FIFO 缓冲区开始工作时内部为空。

```
//复位 FIFO
  SYNCDELAY;
  FIFORESET = 0x80;          // activate NAK-ALL to avoid race conditions
  SYNCDELAY;                // see TRM section 15.14
  FIFORESET = 0x02;          // reset, FIFO 2
  SYNCDELAY;                //
  FIFORESET = 0x04;          // reset, FIFO 4
  SYNCDELAY;                //
  FIFORESET = 0x06;          // reset, FIFO 6
  SYNCDELAY;                //
  FIFORESET = 0x08;          // reset, FIFO 8
  SYNCDELAY;                //
  FIFORESET = 0x00;          // deactivate NAK-ALL
```

当程序运行时，通过 PC 发送的上位机配置信息的 USB 数据包，由 EP0 接收，当数据从 USB 到来后，调用 io_spi_send_byte() 等函数，将 PC 发来的数据通过 SPI 协议发送出去。FPGA 通过 SPI 接口接收从 FX2 发来的数据，在数码管显示。

PC -> USB -> FX2LP -> SPI -> FPGA ->数码管。

```
void io_spi_send_byte(unsigned char dat)
{
   char i = 0;

   // 发送数据位
   for (i = 0; i < 8; i++) {
      if (dat & 0x80) {
         SPI_MOSI_H;
      } else {
         SPI_MOSI_L;
      }
      SPI_SCLK_H;
```

```
        dat <<= 1;
        SPI_SCLK_L;
    }
    SPI_MOSI_L;
}
```

主机通过 USB 控制传输发送数据到 FX2 时，自动将这些数据通过 SPI 转发给 FPGA。当数据已经完整地到达 FX2 的端点 0 OUT 缓冲区，拉低 CS 线，调用 io_spi_send_byte 使用 SPI 发送至 FPGA，完成后拉高 CS。

```
void ISR_Ep0out( void ) interrupt 0
{
    BYTE i = 0;
    // 非 setup OUT 数据包到了（如控制传输的 Data 阶段）
    // 读取 EP0BUF 中的数据，长度由 EP0BCL 指示
    BYTE len = EP0BCL & 0x7F;
    // CS 拉低
    SPI_CS_L;

    io_spi_send_byte(len | 0x00);//长度+写位(0x00)

    for (i = 0; i < len; i++)
    {
        //io_uart_send_byte(EP0BUF[i]);//EP0BUF[i]
        io_spi_send_byte(EP0BUF[i]);
    }
    // CS 拉高
    SPI_CS_H;
    // 清除 EP0OUT 的中断标志
    EPIRQ = 0x02; // 清除 bit1，对应 EP0OUT
}
```

1.3.3 FW.c

FW.c 为主体框架，负责 USB 设备的枚举、控制请求的解析与分发、电源管理以及主任务调度。它和 slave.c 协同工作，共同构成完整的固件。

在开始定义了各种全局变量和宏，其中定义了所需的指针，指向存储在内存中的各种 USB 描述符。

```
WORD   pDeviceDscr;  // Pointer to Device Descriptor; Descriptors may be moved
WORD   pDeviceQualDscr;
WORD   pHighSpeedConfigDscr;
WORD   pFullSpeedConfigDscr;
WORD   pConfigDscr;
WORD   pOtherConfigDscr;
WORD   pStringDscr0;
WORD   pStringDscr1;
WORD   pStringDscr2;
WORD   pStringDscr3;
```

在 main(void)主函数中，调用了用户初始化 TD_Init()，这是跳转到 slave.c 的关键。这里调用的 TD_Init()正是在 slave.c 中定义的函数，它负责配置 Slave FIFO、端点、GPIO 等硬件具体参数。

```
   // Initialize user device
    TD_Init();
```

使能 EP0 的中断，当主机通过控制传输发送数据时，会触发中断，从而执行 slave.c 中的 ISR_Ep0out 函数，实现 USB 转 SPI 的功能。EA＝1：全局开启中断。

```
   EZUSB_IRQ_ENABLE();            // Enable USB interrupt (INT2)
   EZUSB_ENABLE_RSMIRQ();          // Wake-up interrupt

   INTSETUP |= (bmAV2EN | bmAV4EN);     // Enable INT 2 & 4 autovectoring

   USBIE |= bmSUDAV | bmSUTOK | bmSUSP | bmURES | bmHSGRANT;    //
Enable selected interrupts
   EPIE |= 0x02;          //开启 EP0 的 OUT 中断
   EA = 1;
```

这些 case 处理的是 CDC（虚拟串口）类的特定请求。

GET_LINE_CODING：是一个标准的类特定请求，当主机请求获取当前串口参数（波特率、数据位、停止位等）。这里的实现是通过 SPI 从外部设备（如下位机）读取这些参数，然后通过 EP0 返回给主机。

SET_LINE_CODING：主机想要设置串口参数。这里的实现是先告知 USB 核心准备好接收数据（EP0BCL=7;）。随后，主机发出的数据会触发 EP0-OUT 中断，从而在 slave.c 的 ISR_Ep0out 函数中，通过 SPI 将参数发送给下位机。

SET_CONTROL_LINE_STATE：主机设置控制线状态（如 DTR、RTS 信号）。本例未使用。

```
    case CDC_GET_LINE_CODING:               // *** Get cdc
      len = SETUPDAT[6] & 0x7F;
```

```
#if 1
        SPI_CS_L;
        //先使用 SPI 发送字节长度+读标志 0x80
        io_spi_send_byte(len | 0x80);
        //向下位机发送读指令
         //从下位机获取参数
        for (i = 0; i < len; i++)
        {
            EP0BUF[i] = io_spi_recv_byte();
        }
        SPI_CS_H;
#else
         EP0BUF[0] = 0x00;
         EP0BUF[1] = 0xC2;
         EP0BUF[2] = 0x01;
         EP0BUF[3] = 0x00;
         EP0BUF[4] = 0x00;
         EP0BUF[5] = 0x00;
         EP0BUF[6] = 0x08;
#endif

        EP0BCH = 0;
        EP0BCL = len;
        break;

  case CDC_SET_LINE_CODING:                // *** Set cdc        l
         EP0BCH = 0;           // 高字节清 0
         EP0BCL = 7;  // 向 SIE 宣布准备好接收
         break;

      case CDC_SET_LINE_CTLSTE:              // *** Set cdc
         break;
```

如何使用 USB 固件将 USB 配置为 CDC 类已经介绍完了，接下来讲解如何在应用工程内使用 USB CDC。

1.4 FX2 时序分析

本次设计使用的 CY7C68013（FX2LP）是一颗高速 USB2.0 接口控制器，

它的作用是 在 USB 总线与外部逻辑（如 FPGA、MCU、DSP 等）之间进行高速数据传输。在本次 USB CDC 回环设计中，我们使用 FX2 的同步 FIFO 模式，因此需要先了解 FX2 芯片 的同步 FIFO 读写操作机制与相关控制信号。

对于 FX2 芯片，为了实现同步 FIFO 的读操作，当发生读事件时，激活 FIFOADR[1:0]（可通过寄存器修改）（此时必须与 IFCLK 的上升沿保持同步），若 FIFO 非空，则可以置位 SLOE 激活 FD 输出，之后通过 SLRD 来递增指针。当 FIFO 读空后，应拉高 SLOE 和 SLRD。

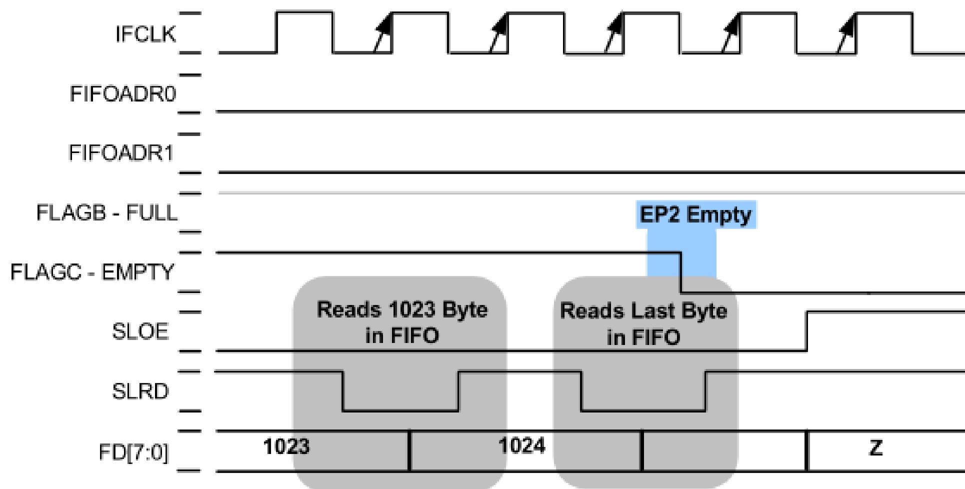

图 1-4 FX2 读操作

为了实现同步 FIFO 的写操作，当发生写事件时，激活 FIFOADR[1:0]（可通过寄存器修改）（此时必须与 IFCLK 的上升沿保持同步），若 FIFO 未满，则可以置位 SLWR 功能，开始写入数据。 当写入第一个数据后，EMPTY 就会拉高。之后通过 SLWR 来递增 FIFO 指针。

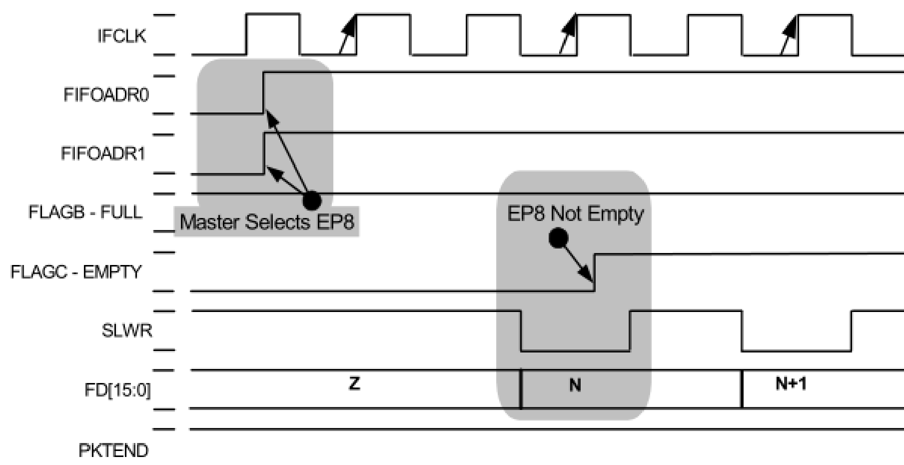

图 1-5 FX2 写入操作

FX2 默认为满 512 个字节向主机发送一包，若最后一个数据包小于 512 字节，FX2 不会主动发送。这时可以通过 PKTEND 脉冲手动提交短包。PKTEND 置位后，最后一个数据包就会被提交给 USB。当 FIFO 满时（低有效），即使 SLWR 再次置位，FD 总线数据也不再写入，只有等待主机读完整个数据包，一个缓冲器才能再次可用。

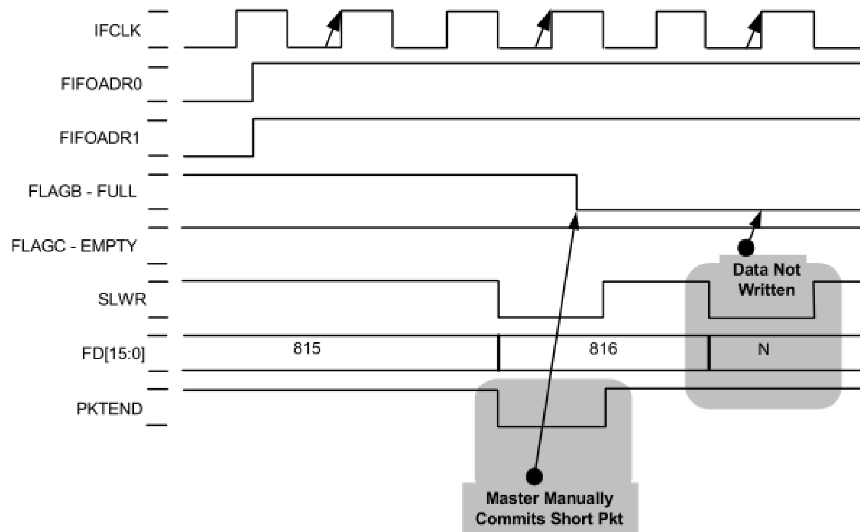

图 1-6 PKTEND 操作

FX2 的读写时序讲解完毕，接下来讲解本次实验的工程设计与验证。

1.5 系统整体设计

在 PC 端，USB CDC 驱动把 FX2 设备识别为一个 标准串口 (COMx)，对用户而言和普通 UART 串口无异。通过串口上位机发送数据时，CDC 驱动将上位机应用层的数据打包成 USB Bulk 传输包，通过 USB 总线发送给 FX2 芯片。FX2 的端点 EP2 会把数据发送给 FPGA。FPGA 内部使用 FIFO 缓存数据，同时从 FIFO 中取出数据，再送回 FX2。FX2 将 FPGA 回传的数据装入 IN 端点 EP6 缓冲区，打包成 USB Bulk 数据上传给 PC，CDC 驱动在 PC 端将 USB 数据流还原成串口数据流。最终，上位机应用在串口上读到的数据与最初写入的数据一致，实现回环通信。

同时，通过 FX2 的端点 0，解析上位机的指令，并将其显示到数码管上。

系统整体设计结构框图如下图 1-7 所示。

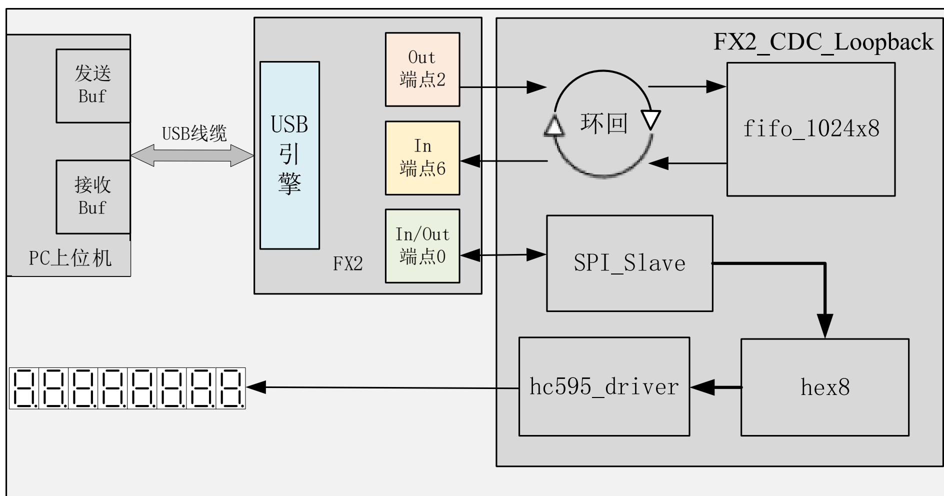

图 1-7 系统整体结构图

FX2_CDC_Loopback：顶层模块，用于处理FX2与FPGA之间的数据回环。

SPI_Slave：通用从机模块，用于通过 FX2 读取串口上位机的设置数据，如波特率，停止位等，并将其显示到数码管上。

fifo_1024x8：回送缓存 FIFO，缓存接收/发送的数据。

hc595_driver：数码管驱动模块，用于驱动 74HC595，发出数据和选通信号。

hex8：数码管动态扫描驱动模块，用于输出需要显示的数据、段选和位选值。

hc595_driver 与 **hex8 模块**详细介绍可参考开发板对应学习资料。

第 13 章 8 位 位 7 段数码管驱动设计与验证

1.5.1 FX2_CDC_Loopback

在 USB CDC 串口回环系统中，数据从 上位机串口 → USB → FX2 芯片 → FPGA 的传输路径中，FX2_CDC_Loopback 模块用于实现上位机下发数据的可靠接收与缓存，为后续 FPGA 内部处理提供数据输入。

在同步模式下，SLOE 引脚在同步模式下被置位时，会使 FD 总线以 FIFO 指针当前所指向的数据进行驱动。这些数据会被预先读取，并且只有在 SLRD 被置位时才会输出（SLOE 与 SLRD 低电平有效）。因此，上位机传递数据时需要控制 FX2 的 SlaveFIFO 的读控制信号 fx2_slrd 与输出使能信号 fx2_sloe。同时，读数据时端点 2 空标志 fx2_flagb 不能为空（低电平有效），即，当 FX2 FIFO 中有数据，且 FPGA 的 FIFO 未满时，通过置位 fx2_slrd 和 fx2_sloe 读取数据。设计代码如下：

```
assign fx2_slrd  = slrd_n;
assign fx2_sloe  = sloe_n;
//读控制信号产生逻辑
always @(*) begin
   if ((current_loop_back_state == loop_back_read) && (fx2_flagb == 1'b1) &&
(~fifo_full)) begin
  slrd_n = 1'b0;
  sloe_n = 1'b0;
      end else begin
  slrd_n = 1'b1;
  sloe_n = 1'b1;
      end
end

always @(*) begin
      if (slrd_n == 1'b0) fifo_data_in = fx2_fdata;
      else fifo_data_in = 8'd0;
end
```

具体波形如下：

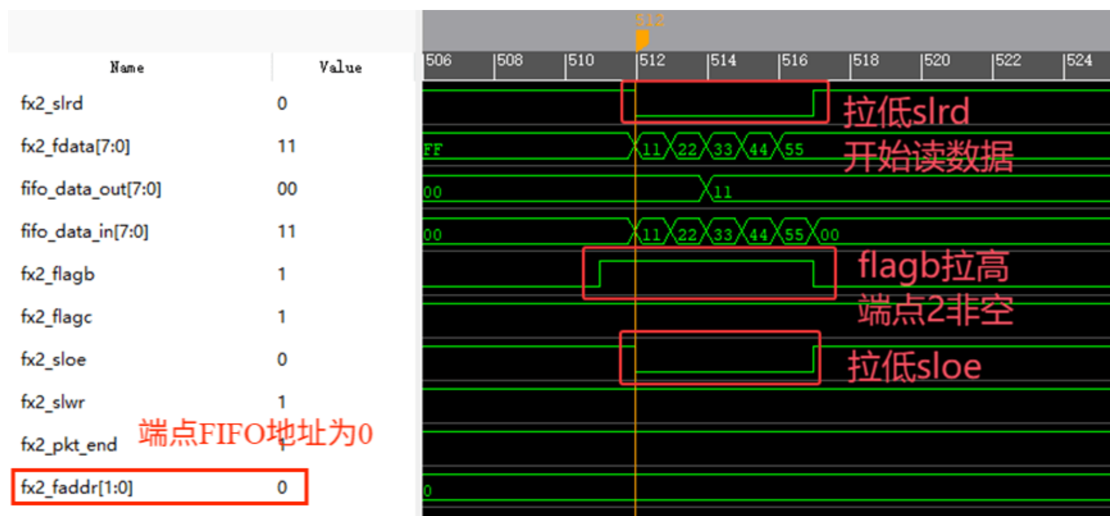

图 1-8 读数据波形

在同步模式下，FD 总线上的数据会在 IFCLK 的每次上升沿以及 SLWR 被置位时写入 FIFO，并且 FIFO 指针会递增（SLWR 为低电平有效）。因此需要控制 FX2 的 SlaveFIFO 的写控制信号 fx2_slwr。同时，读数据时端点 6 满标志 fx2_flagc 不能为满（低电平有效），即，端点6缓存 FIFO 未写满，且 FPGA FIFO 不为空的情况下，拉低 fx2_slwr。设计代码如下：

```
assign fx2_slwr  = slwr_n;
//写控制信号产生逻辑
    always @(*) begin
  if ((current_loop_back_state == loop_back_write) & (fx2_flagc == 1'b1) & (fifo_empty == 1'b0))
     slwr_n <= 1'b0;
  else slwr_n <= 1'b1;
 end

 always @(*) begin
  if (slwr_n == 1'b1) data_out = 8'dz;
  else data_out = fifo_data_out;
 end
```

FIFOADR[1:0] 引脚用于选择四个 FIFO 中的哪一个 与 FD 总线相连，因此需要控制其读写状态需要使用哪个 FIFO。设计代码如下：

```
    assign fx2_faddr = faddr_n;
    always @(*) begin
      if(((current_loop_back_state == loop_back_idle) && (delay_cnt > 8)) | (current_loop_back_state == loop_back_read))begin
        faddr_n = 2'b00;
      end else begin
        faddr_n = 2'b10;
      end
 end
```

具体波形为下：

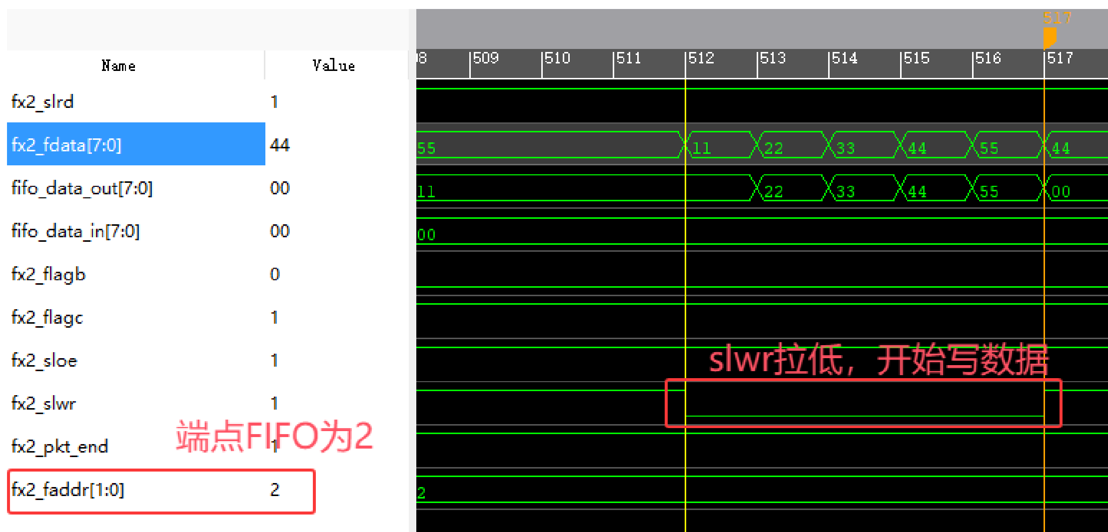

图 1-9 写数据波形

外部主控器会通过激活“PKTEND”引脚来将一个输入数据包提交至 USB，无论该数据包的长度如何。通常在主控器希望发送“短”数据包（例如，其长度小于寄存器中指定的大小）时会使用“PKTEND”引脚。

FX2 的 FIFO 默认为满 512 字节（4096Bit）发一包，如果发送的数据不是 512 字节的倍数，最后一包数据不会自动提交到 USB，因此需要用 fx2_pkt_end 信号用于向 USB 发送一个较短（小于最大数据包大小）的 IN 数据包。

```
assign fx2_pkt_end = ((current_loop_back_state == loop_back_idle) && (delay_cnt <= 4)) ? 1'b0 : 1'b1;
```

对于回环功能，我们通过状态机的方式实现，定义状态如下所示，分别为空闲状态、读状态、等待端点 2 为空状态以及写状态：

```
parameter [1:0] loop_back_idle = 2'd0;
parameter [1:0] loop_back_read = 2'd1;
parameter [1:0] loop_back_wait_flagc = 2'd2;
parameter [1:0] loop_back_write = 2'd3;
```

整体的状态转移图如所图 1-10 示：

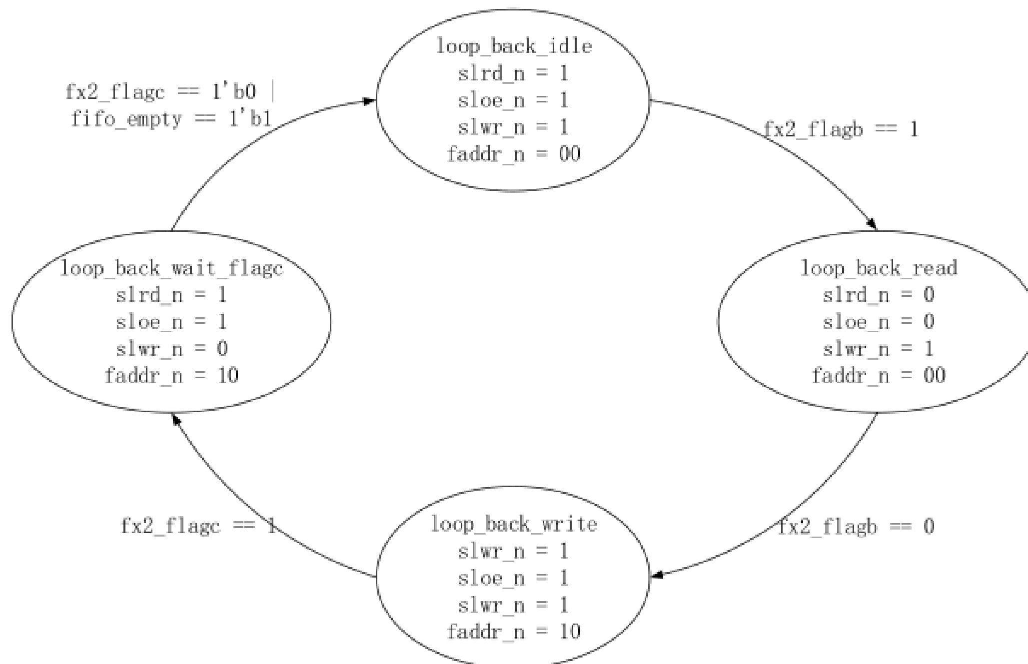

图 1-10 状态转移图

在 loop_back_idle 状态的时候，检测 fx2_flagb 的信号，也就是端点 2 为空的标志信号，当 fx2_flagb 为 1 的时候，代表此时端点 2 不为空，等待 Delay_Value 延时跳转到 loop_back_read 状态。

在 loop_back_read 状态中，FPGA 通过激活 SLRD 和 SLOE 来读取数据。当

端点 2 FIFO 空标志变为低电平时，状态机将进入 loop_back_wait_flagd 状态。同时取消激活 SLRD 和 SLOE 信号。

在 loop_back_wait_flagc 状态中，当 fx2_flagc 为高电平的时候，跳转到 loop_back_write 状态写数据。

在 loop_back_write 状态中，通过激活 SLWR 信号，FPGA 将相同的数据写入到端点 6 的 FIFO 内，代码如下所示：

```
loop_back_idle: begin
    if ((fx2_flagb == 1'b1) && (delay_cnt >= Delay_Value - 1)) begin
      next_loop_back_state = loop_back_read;
      delay_en = 1'b0;
    end else begin
      next_loop_back_state = loop_back_idle;
      delay_en = 1'b1;
    end
  end
  loop_back_read: begin
    if (((fx2_flagb == 1'b0) || (fifo_full)) && (delay_cnt >= Delay_Value - 1))
begin
      next_loop_back_state = loop_back_wait_flagc;
      delay_en = 1'b0;
    end else begin
      next_loop_back_state = loop_back_read;
      delay_en = 1'b1;
    end
  end
  loop_back_wait_flagc: begin
    if ((fx2_flagc == 1'b1) && (delay_cnt >= Delay_Value - 1)) begin
      next_loop_back_state = loop_back_write;
      delay_en = 1'b0;
    end else begin
      next_loop_back_state = loop_back_wait_flagc;
      delay_en = 1'b1;
    end
  end
  loop_back_write: begin
    if (((fx2_flagc == 1'b0) | (fifo_empty == 1'b1)) && (delay_cnt >= Delay_Value
- 1)) begin
      next_loop_back_state = loop_back_idle;
```

```
        delay_en = 1'b0;
    end else begin
        next_loop_back_state = loop_back_write;
        delay_en = 1'b1;
    end
```

SPI_Slave 模块

SPI_Slave 模块为通用 SPI 从机接口，通过 FX2 的端点 0 解析串口上位机的指令寄存器，将 FPGA 作为 SPI Slave，使用 SPI 协议读取当前串口上位机的主要寄存器的工作状态。当串口上位机连接 CDC 虚拟串口后，通过 SPI 从 USB 读取串口上位机的配置寄存器。部分设计代码如下：

对于 CDC 串口的寄存器。

- Byte 0~3: 波特率（Baud Rate），DWORD，LSB First
- Byte 4: 停止位数量（Stop Bits）: 0 = 1 位，1 = 1.5 位，2 = 2 位
- Byte 5: 校验（Parity）: 0 = None，1 = Odd，2 = Even，3 = Mark，4 = Space
- Byte 6: 数据位（Data Bits），通常为 8

```
    S_WRITE_REG: begin
         if (Recive_Data_Valid) begin
          Param_Reg[Trans_Cnt-2] <= Recive_Data;
          SM_State <= S_WRITE_WAIT;
         end else begin
          SM_State <= S_WRITE_REG;
         End
    //接收数据
    always@(posedge Clk or negedge Rst_n)
    begin
      if(!Rst_n)
        Recive_Data <= 8'h00;
      else if(Done_POS) begin
        if(BITS_ORDER == 1'b1)
          Recive_Data <= Recive;
        else
          Recive_Data                                                          <=
{Recive[0],Recive[1],Recive[2],Recive[3],Recive[4],Recive[5],Recive[6],Recive[7]
```

```
};
        end
        else
            Recive_Data <= Recive_Data;
    End
    //状态机加线性序列机采集 MOSI
    always@(negedge SCK_Sel or posedge SPI_Reset)
    begin
        if(SPI_Reset) begin
            In_Cnt <= 8'd0;
            Recive <= 8'h00;
            Trans_Done <= 1'b0;
            Trans_Cnt <= 8'h00;
        end
        else begin
            case (In_Cnt)
                8'd0: begin In_Cnt <= In_Cnt + 1'b1; Recive[7] <= SPI_MOSI; Trans_Done <= 1'b0; end
                8'd1: begin In_Cnt <= In_Cnt + 1'b1; Recive[6] <= SPI_MOSI; end
                8'd2: begin In_Cnt <= In_Cnt + 1'b1; Recive[5] <= SPI_MOSI; end
                8'd3: begin In_Cnt <= In_Cnt + 1'b1; Recive[4] <= SPI_MOSI; end
                8'd4: begin In_Cnt <= In_Cnt + 1'b1; Recive[3] <= SPI_MOSI; end
                8'd5: begin In_Cnt <= In_Cnt + 1'b1; Recive[2] <= SPI_MOSI; end
                8'd6: begin In_Cnt <= In_Cnt + 1'b1; Recive[1] <= SPI_MOSI; end
                8'd7: begin In_Cnt <= 8'd0; Recive[0] <= SPI_MOSI; Trans_Done <= 1'b1; Trans_Cnt <= Trans_Cnt + 1'b1; end
                default: In_Cnt <= 8'd0;
            endcase
        end
    end
```

读取到的数据通过开发板板载的 HC595 数码管显示，并可通过拨码开关切换波特率或数据位、校验位和停止位。

```
assign disp_data = SW ? {Param_Reg[7],Param_Reg[6],Param_Reg[5],Param_Reg[4]} : {Param_Reg[3],Param_Reg[2],Param_Reg[1],Param_Reg[0]};
```

1.6 板级验证

本次实验使用 ACG720-60K 以及其板载 CY7C68013 USB 芯片进行验证。

1.6.1 系统所需硬件

1. ACG720-60k 开发板
2. 电源线一根
3. Type-c 数据线一根
4. 下载器一个

1.6.2 硬件连接

本次设计硬件连接如下图所示：

本次实验需要使用一根 Type-c 连接开发板右上方的 USB 接口与电脑。

连接好下载器与电源适配器。

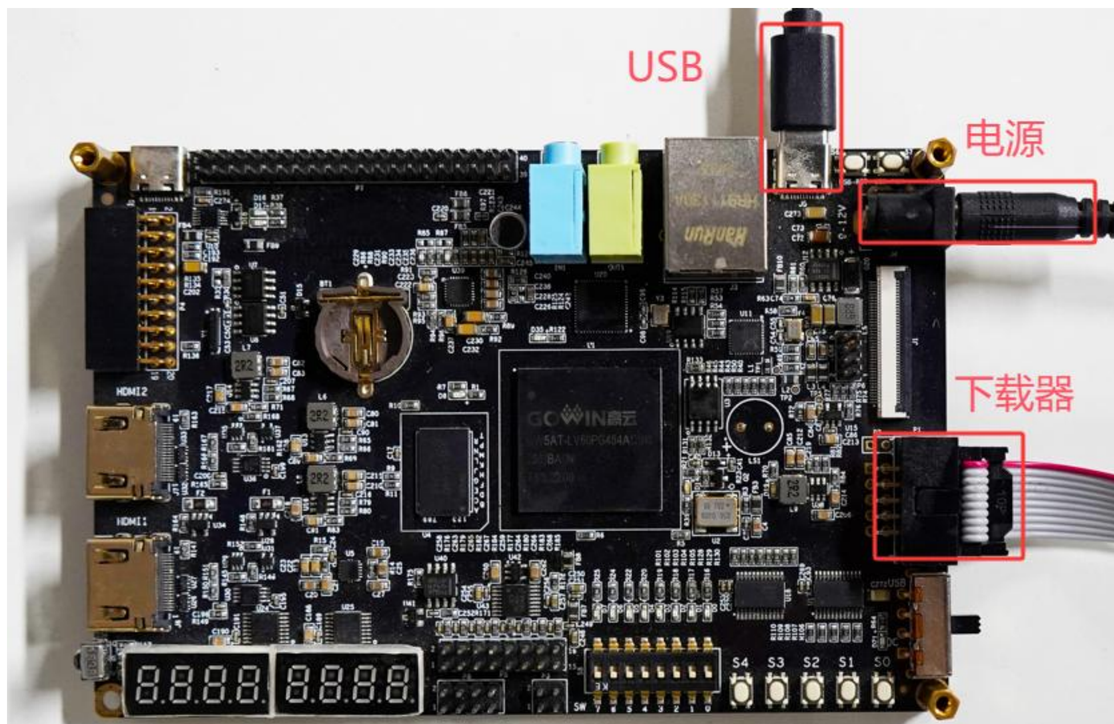

图 1-11 硬件连接

连接完毕后接下来即可进行程序下载和板级验证了。

1.6.3 管脚分配

经过上述工作，所有代码都已经设计完毕，硬件环境也已搭建完成。 接下来进行管脚分配，如下表所示：

表 1 管脚分配表

	信号名	引脚号	信号名	引脚号
基本管脚	clk	Y18	SW	G22
	reset_n	F15		
数码管	st_cp	C2	sh_cp	F4
	ds	M18		
SPI	SPI_SCLK	L15	SPI_MOSI	K16
	SPI_MISO	L16	SPI_CS	L14
FX2	fx2_slwr	L20	fx2_slrd	K19
	fx2_sloe	J15	fx2_slcs	J16
	fx2_pkt_end	J14	fx2_ifclk	L19
	fx2_flagc	J17	fx2_flagb	H18
	fx2_faddr[0]	K13	fx2_faddr[1]	H13
	fx2_fdata[0]	G18	fx2_fdata[1]	G17
	fx2_fdata[2]	G20	fx2_fdata[3]	J20
	fx2_fdata[4]	G13	fx2_fdata[5]	G16
	fx2_fdata[6]	K14	fx2_fdata[7]	G15

接下来连接 USB 接口烧写 FX2 固件。

1.6.4 烧写 FX2 固件

如果想把 USB 配置为 CDC 虚拟串口设备，需要通过固件里的 设备描述符 + 配置描述符 + 接口描述符来配置为 CDC 串口，当固件把自己枚举为 CDC-ACM 类设备时，操作系统就会加载内置驱动，把它映射成一个虚拟串口。因需要对应的固件来配置这些信息。

我们在资料中提供了一个对应的 fx2lp_cdc_clrfifo_nozeropkt.iic 固件。下面为烧写固件步骤。

打开 CyControlexe 软件，使用 USB 线连接开发板 USB 接口和电脑的 USB 接口，软件会显示出一个名为 Cypress FX2LP No EEPROM Device 的设备，选中该设备，点击 Program -> FX2 -> RAM，或选择 EEPROM 掉电不丢失，如下所图 1-12 所示。

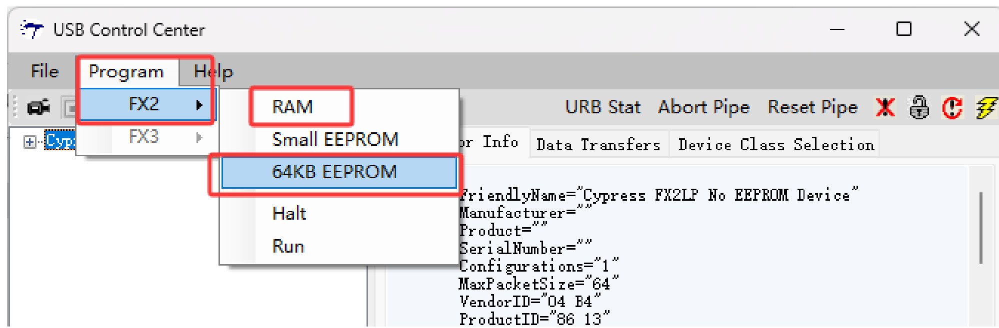

图 1-12 固件烧写界面

在弹出的文件选择框中选择 fx2lp_cdc_clrfifo_nozeropkt.iic 固件，然后点击打开，软件即开始下载固件到 FX2 芯片的 RAM/EEPROM 中，如下图 1-13 所示。

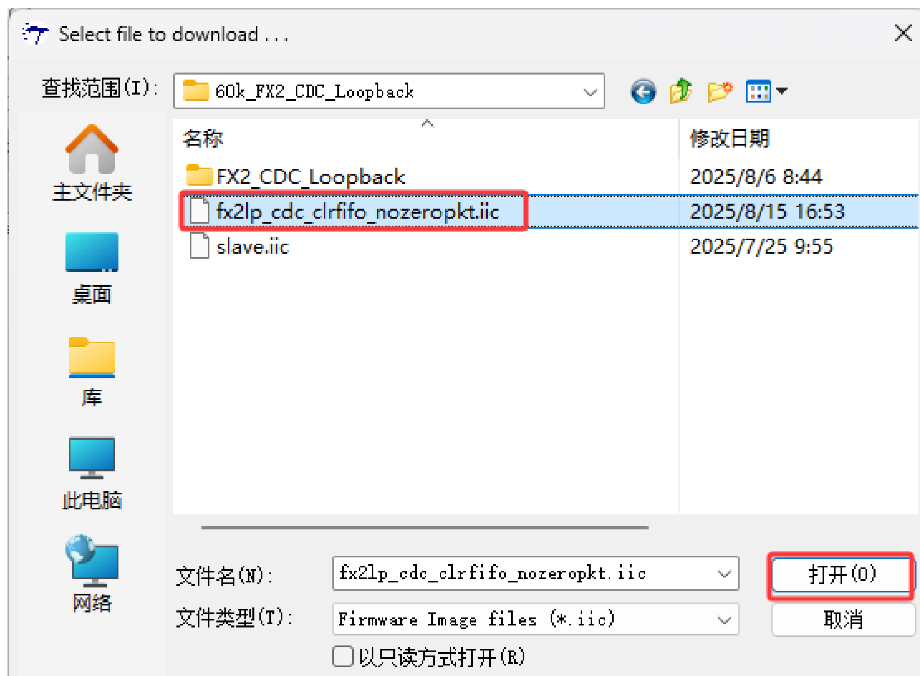

图 1-13 烧写固件

程序下载完成后，软件左下角会显示 Programming Succeeded，提示程序下载完成，按下开发板 USB 旁的 USB-RST 键，接下来 FX2 芯片会重新枚举，然后在 Control Center 软件中可以看到识别不到设备，这是因为 USB 已经映射为了虚拟串口，可以在设备管理器中看到 USB 被识别为了串行设备，如图 1-14 所示。

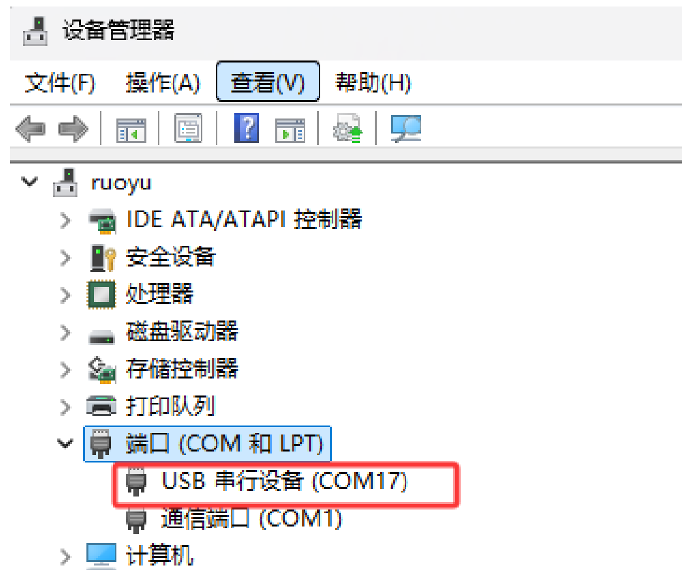

图 1-14 USB 被识别为串行设备

烧写完固件后即可以进行下载验证。

1.6.5 下载与验证

点击 GOWIN 软件的 Programmer 标志。检测到下载器后点击 Save 确认，如图 1-15 所示。

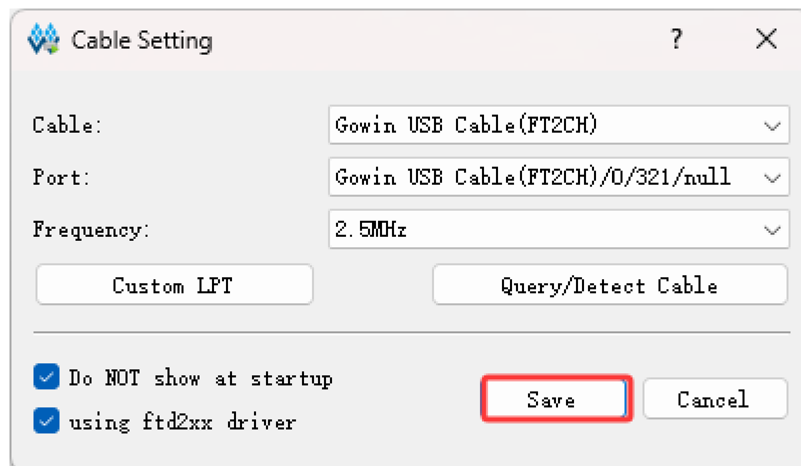

图 1-15 识别下载器

之后点击 进行程序烧录，下方的输出栏出现如下图 1-16 所示的提示即代表烧录成功。

Info	Target Cable: Gowin USB Cable(FT2CH)/0/321/null@2.5MHz
Info	Target Device: GW5AT-60B(0x0001481B)
Info	Operation "SRAM Program" for device#1...
Info	Frequency Updated: "15MHz"
Info	User Code is: 0x00001DC3
Info	Status Code is: 0x70026020
Info	Finished.
Info	Cost 4.4 second(s)

图 1-16 烧录成功

接下来打开串口助手，进行回环验证。

在串口助手中端口选择“USB 串行设备”，点击下图中 2 处打开串口助手，下图 3 处变绿即代表连接成功。（其他串口设置没有影响）

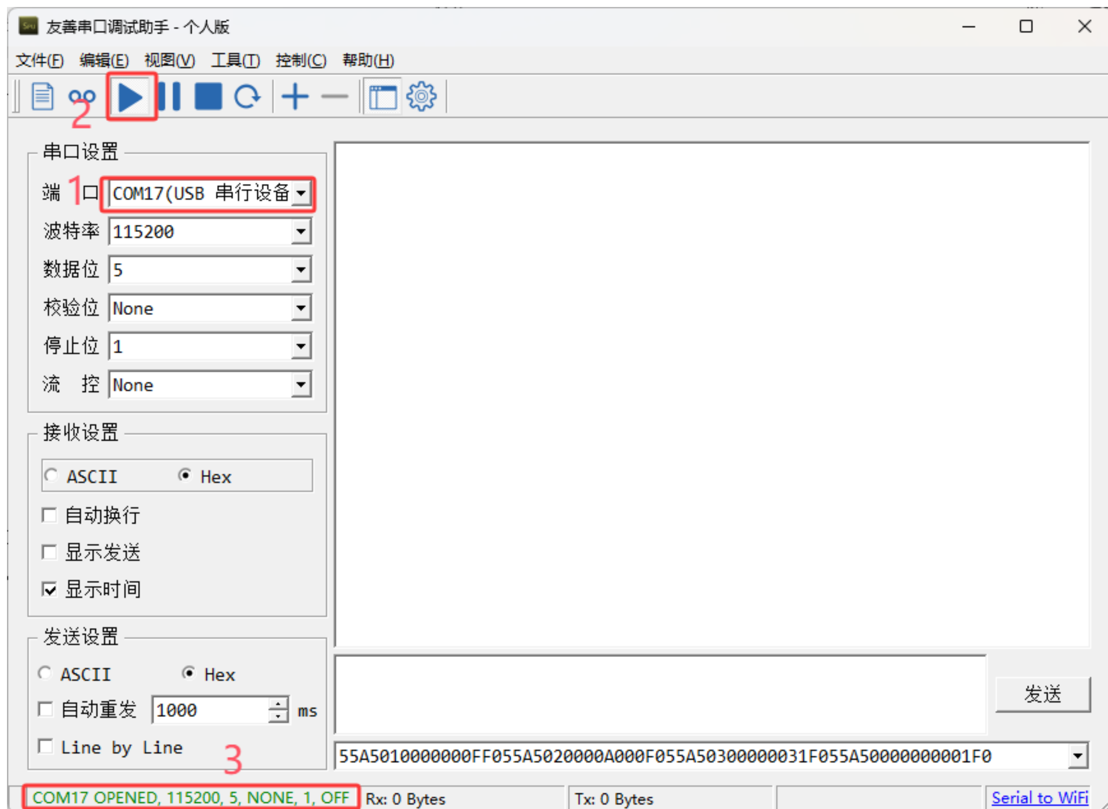

图 1-17 串口助手设置

接下来在串口助手发送栏发送想要发送的数据，或者发送我们提供的测试数据，即可看到串口助手正确回送了发送的数据。如图 1-18 与图 1-19 所示。

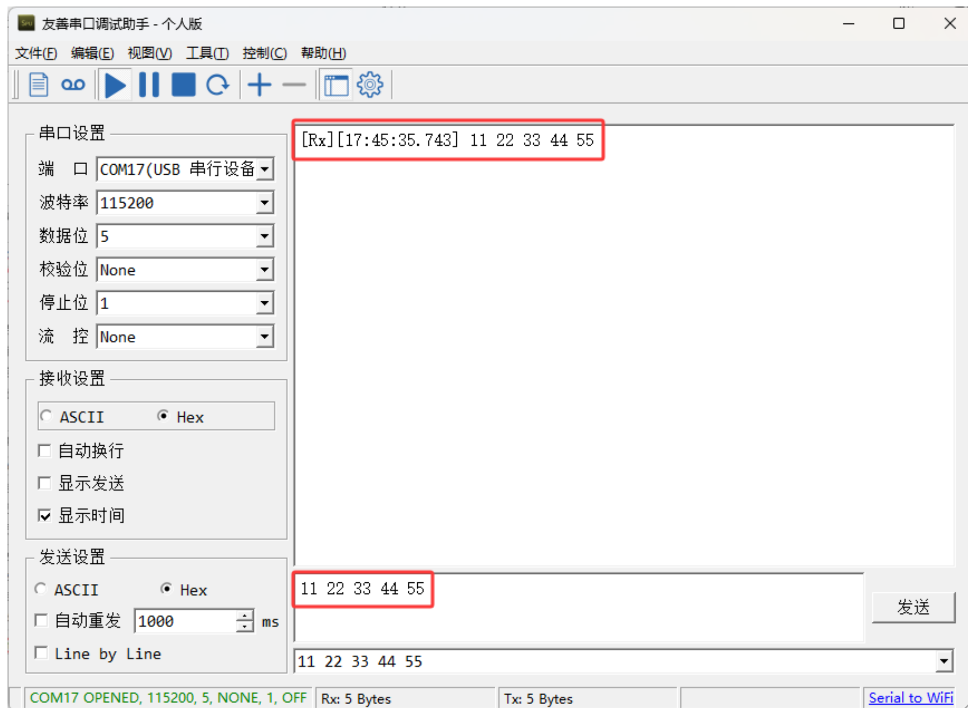

图 1-18 5 字节回环

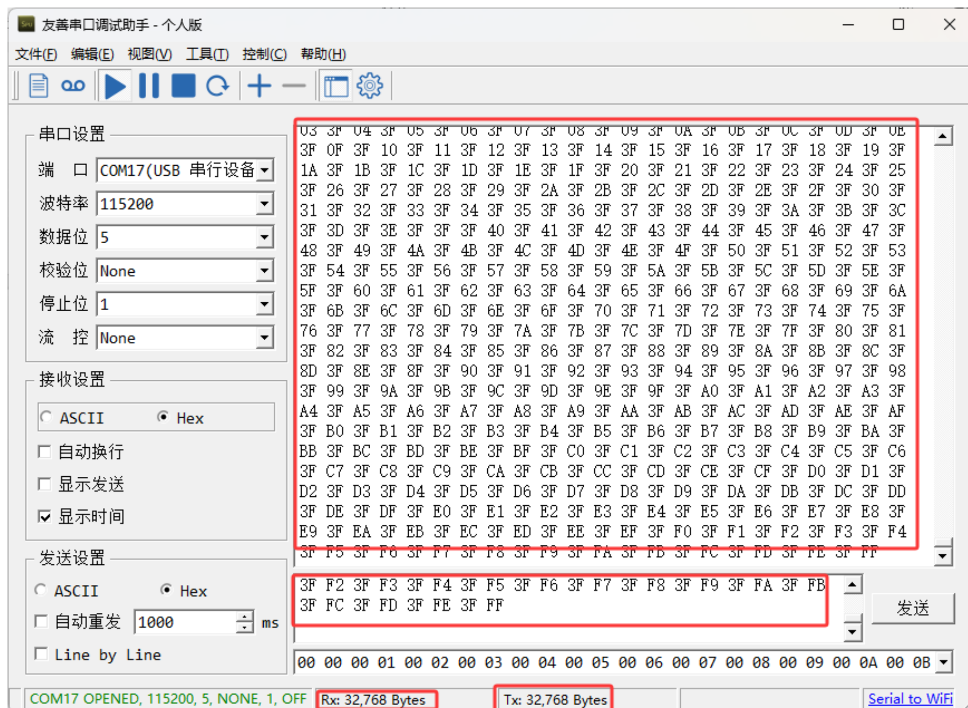

图 1-19 32768 字节

此时修改串口助手的串口设置，在开发板板载数码管上可以看到显示出对应的 16 进制波特率。将拨码开关 **SW0** 拨至上方，可以看到数码管上显示出对应的数据位、校验位和停止位的信息。（波特率 9600、数据位 5、校验位 Even、停止位 1）如下图所示：

图 1-20 数码管显示

至此，基于 FX2 的 USB CDC 串口回环的设计与实现就完成了。

注：当不使用 USB CDC 串口或需要使用正常 USB 功能时，按下 USB 接口旁的 USB-PGM 的同时按一下 USB-RST 即可。

1.7 总结与思考

本节实验基于 FX2 芯片实现了 USB 转 CDC 虚拟串口功能，并通过 FPGA 实现了数据的回环传输与串口参数的实时显示。本实验介绍了 USB CDC 协议的基本框架、枚举过程与数据传输机制，以及 FX2 SlaveFIFO 接口的读写时序控制方法和 SPI 协议在配置信息读取中的应用。建议读者能够跟随本实验内容，完整的进行整个实验。

在本实验基础上，读者可根据自己的想法，对其他寄存器进行 SPI 读写功能的设计。