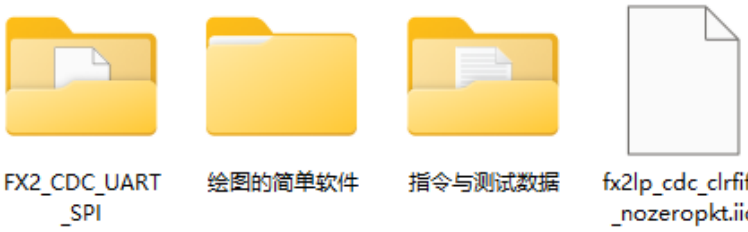
1 基于 FX2 的 USB CDC 串口与 SPI 读写设计

工程源码	|--60k_ FX2_CDC_UART_SPI |--138k_ FX2_CDC_UART_SPI |-- FX2_CDC_UART_SPI
相关视频课程	本教程无视频课程
本实验支持小梅哥 Xilinx AC720&高云 ACG720 开发板	

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_UART_SPI：USB CDC 串口与 SPI 读写工程源码
- 绘图的简单软件：用于验证 SPI 读 ADC128S 功能正确的一个简单绘图软件，可将采集到的数据绘制为波形
- 指令与测试数据：ADC128S 采集时需要用到的指令与测试数据
- 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

前面的章节讲解了 USB CDC 的原理与实现方法。本章主要介绍基于 ACG720 开发板 FX2 芯片，使用 USB CDC 协议，通过 SlaveFIFO 接口实现 USB CDC 虚拟串口与 ACG720 板载串口回环通信功能，以及通过 CDC 虚拟串口控制 ACG720 板载 ADC128S 采样数据的功能。同时，系统可解析波特率、数据位等串口参数，并利用数码管显示，方便调试与观察。

1.1 USB CDC

关于 USB CDC 的原理与实现方法，在前面的章节已介绍。具体链接如下：

【开源】使用 CY7C68013 实现 USB CDC 串口功能，含 68013 固件和 FPGA 测试程序

https://www.corecourse.cn/forum.php?mod=viewthread&tid=30141

(出处: 芯路恒电子技术论坛)

1.2 ADC128S102 型 ADC 工作原理

开发板上板载了一个模数转换器为逐次逼近型的低功耗芯片 ADC128S102，其具有 8 通道以及 12 位的分辨率。电源采用独立的模拟供电以及数字供电，其中模拟电源 VA 输入范围为 2.7V~5.25V，数字电源 VD 输入范围为 2.7V~VA。其与外部通信支持多种接口如：SPI、QSPI、MICROWIRE 以及通用的 DSP 接口。转换速度在 500 kps~1Mkps，典型情况下当 3V 供电时功耗为 2.3mW，5V 供电时为 10.7mW，如下图为该 ADC 芯片的内部结构图。

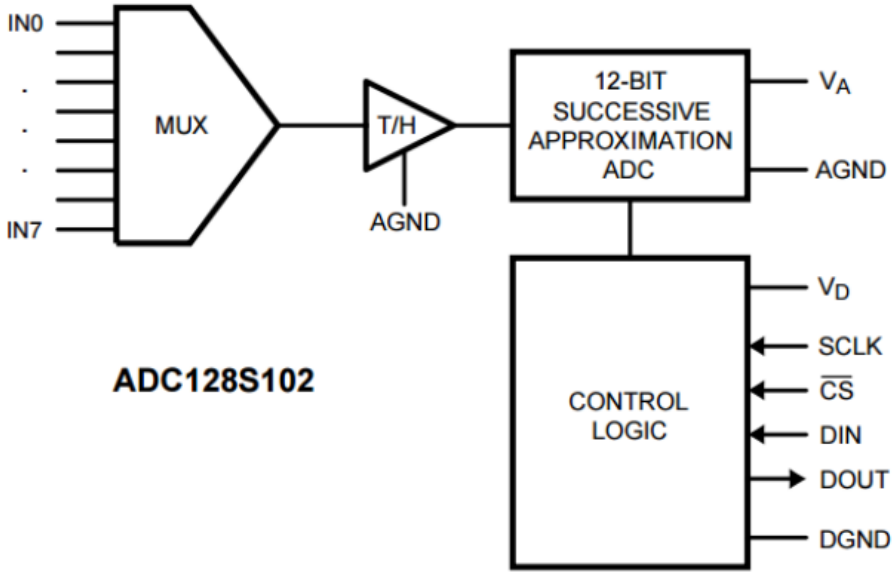


图 1-1 ADC128S102 的内部功能框图

图中左侧的 IN0~IN7 就是这个模数转换芯片（ADC）的模拟电压输入端，需要测量的模拟电压就通过这 8 个脚输入给该 ADC。右侧的 SCLK、CS、DIN、DOUT 是 ADC 的数字通信接口，该接口遵循 SPI 协议，ADC 完成了模数转换后，对输入的模拟电压转换得到一个量化的 12 位数字值，这个数字值需要通过该 SPI 接口传递给主控制器（FPGA、MCU、DSP）。本次设计通过 CDC 虚拟串口发送采集指令给 ADC128S102，通过其 SPI 接口得到采样数据，最后使用 CDC 虚拟串口输出给串口上位机。

在理解本次设计原理后，接下来讲解本次实验的工程设计与验证。

1.3 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，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。

系统整体设计结构框图如下图 1-2 所示。

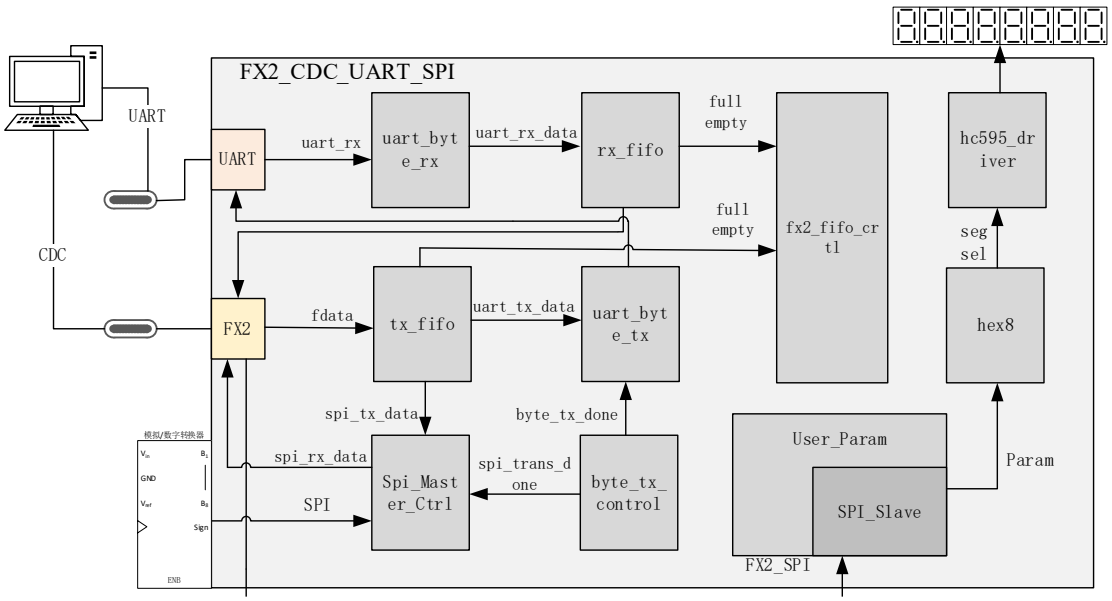


图 1-2 系统整体结构图

FX2_CDC_UART_SPI：主体工程模块，整合 FX2 SlaveFIFO 接口、CDC 虚拟串口、板载 UART 接口、板载 SPI 接口、数码管显示等功能。

User_Param：上位机寄存器读取模块，通过 SPI 读取上位机的设置数据。

SPI_Slave：通用从机模块。

fifo_1024x8：回环与 ADC 缓存 FIFO，缓存接收/发送的数据。

fx2_fifo_crtl：fx2 控制模块，输出 fx2 的各种控制信号。

byte_tx_control：串口发送使能模块，输出发送使能信号。

Spi_Master_Ctrl：ADC 控制模块，用于驱动 ADC128S。

uart_byte_tx：串口发送模块。

uart_byte_rx：串口接收模块。

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。

1.3.1 FX2_CDC_UART_SPI 模块

在 FX2_CDC_UART_SPI 中，通过拨码开关控制数据的流向来实现对应的功能。

在 **SW1=1** 时，工程为 CDC 虚拟串口与板载串口回环功能。

通过 CDC 虚拟串口发送的数据，会先经过 fifo 缓存，再通过板载串口，在连接板载串口的上位机显示。

通过板载串口发送的数据，会先经过 fifo 缓存，再通过 FX2，在连接 CDC 虚拟串口的上位机显示。具体结构如下图所示。

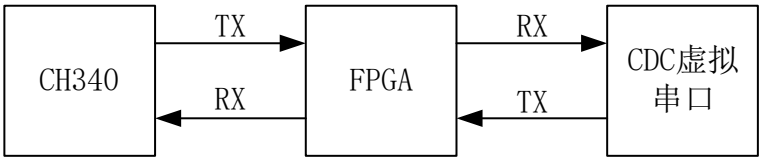


图 1-3 回环结构

在 **SW1=0** 时，工程为 ADC 采集，CDC 虚拟串口发送与接收功能。

通过 CDC 虚拟串口发送 ADC 采集指令，ADC 根据采集指令，通过 SPI 接口发送对应数据，fifo 缓存后由 FX2 发出，在连接 CDC 虚拟串口的上位机显示。具体结构如下图所示。

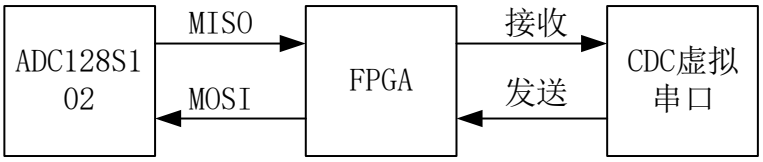


图 1-4 SPI 读写结构

FX2 发送与接收的数据以及数码管的信息显示由拨码开关控制，具体代码如下：

```
assign disp_data = SW0 ? {8'h00, Param6, Param5, Param4} : {Param3, Param2, Param1, Param0};

 assign uart_send_en = SW1 ? byte_send_en : 1'b0;
```

```
assign spi_trans_en = SW1 ? 1'b0 : byte_send_en;
assign uart_tx_data = SW1 ? byte_tx_data : 8'h00;
assign spi_tx_data = SW1 ? 8'h00 : byte_tx_data;
assign byte_tx_done = SW1 ? uart_send_done : spi_trans_done;
assign byte_rx_done = SW1 ? uart_recv_done : spi_trans_done;
assign byte_rx_data = SW1 ? uart_rx_data : spi_rx_data;
```

注：在使用高云开发板时，由于高云复位寄存器问题，需要增加一个延时复位，具体代码如下：

```
reg [23:0]rst_cnt;
 reg rst_n;
   always @(posedge clk,negedge reset_n)begin  //由于高云复位寄存器问题，增加加一个延时复位
   if(!reset_n)
      rst_cnt <= 0;
   else if(rst_cnt == 5_000_000)
      rst_cnt <= rst_cnt;
   else
      rst_cnt <= rst_cnt + 1;
   end

   always @(posedge clk ,negedge reset_n)begin
   if(!reset_n)
      rst_n <= 0;
   else if((rst_cnt >= 4_000_000) && (rst_cnt <= 4_500_000))
      rst_n <= 0;
   else
      rst_n <= 1;
   end
```

1.3.2 User_Param 与 SPI_Slave 模块

User_Param 模块用于通过 FX2 的部分接口，通过 SPI 协议，使用 SPI_Slave 模块读取当前串口上位机的主要寄存器的工作状态。

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。

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

部分读寄存器代码如下：

```
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

    //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
```

具体波形图如下：

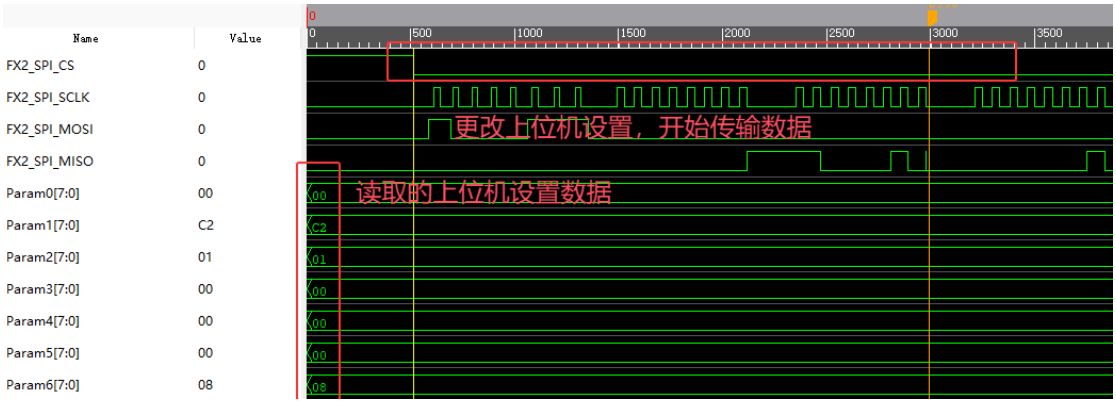


图 1-5 上位机设置数据读取

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。

1.3.3 fx2_fifo_crtl 模块

fx2_fifo_crtl 用于控制 FX2 芯片的端点 FIFO 与 FPGA 内部 FIFO 的数据交互，以及 FX2 读写信号的控制。

当 FX2 端点 2 FIFO 中有数据（fx2_flagb 为高），且 FPGA 的 tx_fifo 未满时，通过置位 fx2_slrd 和 fx2_sloe 读取数据。同时拉高 tx_fifo_push（写忙）信号。

当 FX2 端点 6 FIFO 未写满(fx2_flagc 为高)，且 FPGA 的 rx_fifo 不为空的情况下，拉低 fx2_slwr 写入数据。同时拉高 rx_fifo_pop（读忙）信号。设计代码如下：

```
// USB 接收的数据，进入 tx_fifo，后续 uart/spi 模块取出 tx_fifo 内的数据，发送出去
  always @(*) begin
    if (((~tx_fifo_full)) && (fx2_flagb == 1'b1) && (SM_State == S_READ)) begin
      fx2_slrd    = 1'b0;
      fx2_sloe    = 1'b0;
      tx_fifo_push = 1'b1;
    end else begin
      fx2_slrd    = 1'b1;
```

```
    fx2_sloe    = 1'b1;
    tx_fifo_push = 1'b0;
  end
 end

 // uart/spi 模块接收的数据，进入 rx_fifo，后续取出 rx_fifo 内的数据，写入 usb 的 in fifo 发送出去
 always @(*) begin
  if (((~rx_fifo_empty)) && (fx2_flagc == 1'b1) && (SM_State == S_WRITE))
begin
    fx2_slwr   = 1'b0;
    rx_fifo_pop = 1'b1;
  end else begin
    fx2_slwr   = 1'b1;
    rx_fifo_pop = 1'b0;
  end
  end
```

具体波形如下：

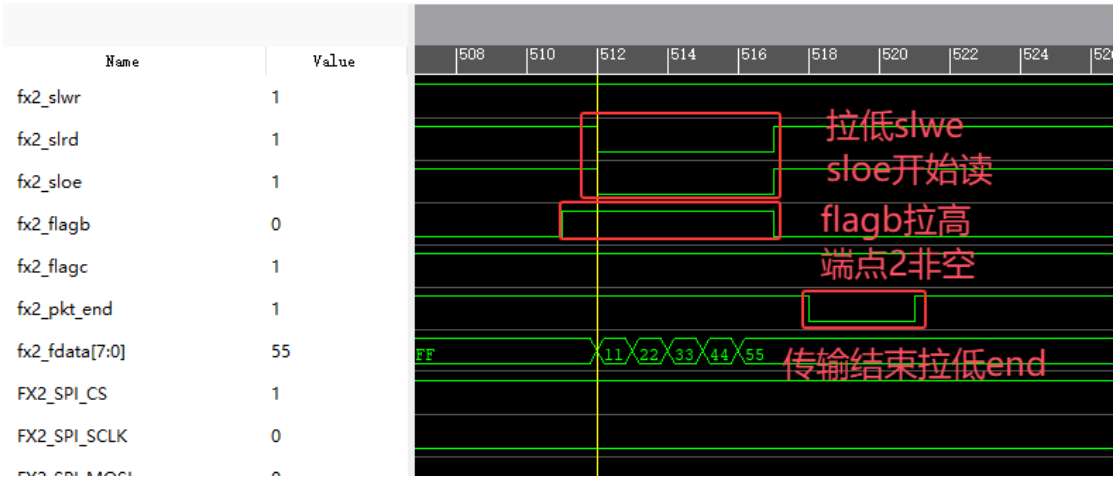


图 1-6 CDC->CH340

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：

```
 always @(*) begin
  if (((SM_State == S_IDLE) && (delay_cnt >= 4'd3)) || (SM_State == S_READ))
begin
    fx2_faddr = 2'b00;
  end else begin
```

```
    fx2_faddr = 2'b10;
  end
end
```

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 = ((SM_State == S_IDLE) && (delay_cnt < 4'd3)) ? 1'b0 : 1'b1;
```

通过一个状态机，负责内部 fifo 和 usb fifo 接口的数据交互，内部 rx fifo 不满且 usb 的 out fifo 非空，则读取数据到内部，否则不做读取。内部 tx fifo 非空且 usb 的 in fifo 非满，则写数据到 usb 的 in fifo 中。内部 tx fifo 满则强制切换为写数据状态，将 fifo 数据写入 usb 的 in fifo 中，直到把内部 fifo 写空或者将 in fifo 写满。设计代码如下：

```
      S_IDLE: begin
       if (delay_cnt < 4'd8) begin
         SM_State <= S_IDLE;
       end else if (~rx_fifo_empty) begin  // 内部 rx fifo 有数据，写入 usb 的 in fifo 中
         SM_State <= S_WRITE_WAIT;
       end else if ((~tx_fifo_full) && (fx2_flagb)) begin // 内部 tx fifo 没满，且 usb 的 out fifo 有数据，准备接收 usb 的数据
         SM_State <= S_READ;
       end else begin
         SM_State <= S_IDLE;
       end
      end
      S_READ: begin
       if (rx_fifo_full) begin  // 如果内部 rx fifo 满了则切换到写
         SM_State <= S_WRITE_WAIT;
       end else if((~fx2_flagb) || (tx_fifo_full)) begin // 内部 tx fifo 满了或者 usb 的 out fifo 空了，结束读取
         SM_State <= S_IDLE;
       end else begin
         SM_State <= S_READ;
```

```
        end
      end
      S_WRITE_WAIT: begin
        if (fx2_flagc) begin
          SM_State <= S_WRITE;
        end else if(rx_fifo_full) begin // 因为 fifo 满了所以急切地要将 rx fifo 数据通过 usb 的 in fifo 发送出去，在此期间不再接收 usb 数据
          SM_State <= S_WRITE_WAIT;
        end else begin  // 内部 rx_fifo 非满，且 usb 的 in fifo 满了，回到 idle
          SM_State <= S_IDLE;
        end
      end
      S_WRITE: begin
        if ((~fx2_flagc) || (rx_fifo_empty)) begin  // 内部 rx_fifo 写空，或者 usb 的 in fifo 写满，退出写状态
          SM_State <= S_IDLE;
        end else begin
          SM_State <= S_WRITE;
        end
      end
```

1.3.4 byte_tx_control 模块

byte_tx_control 模块负责发送使能控制，当 tx_fifo 非空时，读取其内容，写入发送模块，直到发送完成，继续发送下一个数据。设计代码如下：

```
S_TX_IDLE: begin
        if (~tx_fifo_empty) begin
          SM_TX_State <= S_TX_SEND;
        end else begin
          SM_TX_State <= S_TX_IDLE;
        end
      end
      S_TX_SEND: begin
        if (~tx_fifo_empty) begin
          tx_fifo_pop  <= 1'b1;
          byte_send_en <= 1'b1;
          SM_TX_State  <= S_TX_WAIT;
        end else begin
```

```
        SM_TX_State <= S_TX_IDLE;
      end
    end
    S_TX_WAIT: begin
      tx_fifo_pop  <= 1'b0;
      byte_send_en <= 1'b0;
      if (byte_tx_done) begin
        SM_TX_State <= S_TX_SEND;
      end else begin
        SM_TX_State <= S_TX_WAIT;
      end
    end
```

1.3.5 Spi_Master_Ctrl 模块

Spi_Master_Ctrl 是一个 SPI 主机控制器，它能够实现标准 SPI 总线协议的收发，支持 8 位数据的传输。可以通过 ADC128S102 的 SPI 接口，对其进行采集指令与采集数据的读写。

当接收到发送使能时，寄存指令数据。当采集数据接收完成后保存 rx_data。设计代码如下：

```
  always @(posedge clk or negedge rst_n) begin
   if (~rst_n) begin
     spi_busy <= `D 1'b0;
     rx_data  <= `D 8'h00;
     tx_data_r<= `D 8'h00;
   end else if (trans_en) begin
     spi_busy <= `D 1'b1;
     if(BITS_ORDER)
       tx_data_r  <= `D
{tx_data[0],tx_data[1],tx_data[2],tx_data[3],tx_data[4],tx_data[5],tx_data[6],tx_dat
a[7]};
     else
       tx_data_r<= `D tx_data;
     rx_data  <= `D rx_data;
   end else if (spi_state_cnt >= 5'd17 - CPHA) begin
     spi_busy <= `D 1'b0;
     if(BITS_ORDER)
       rx_data  <= `D
{rx_data_r[0],rx_data_r[1],rx_data_r[2],rx_data_r[3],rx_data_r[4],rx_data_r[5],rx_
data_r[6],rx_data_r[7]};
     else
       rx_data  <= `D rx_data_r;
   end else begin
```

```
    spi_busy <= `D spi_busy;
    rx_data  <= `D rx_data;
  end
end
```

模块使用 spi_state_cnt 作为状态计数器，控制整个 SPI 传输过程： 0-16：数据传输阶段（8 位数据 + 起始/停止位），17：传输完成。设计代码如下：

```
always @(posedge clk or negedge rst_n) begin
 if (~rst_n) begin
   spi_state_cnt <= `D 5'd0;
   sclk <= `D CPOL;
 end else if (spi_state_cnt >= 5'd17 - CPHA) begin
   sclk <= `D CPOL;
   spi_state_cnt <= `D 5'd0;
 end else if (spi_clk_x2) begin
   if (spi_busy) begin
     if((CPHA == 1'b0) && (spi_state_cnt == 5'd0))
        sclk <= `D sclk;
     else
        sclk <= `D ~sclk;
     spi_state_cnt <= `D spi_state_cnt + 1'd1;
   end else begin
     sclk <= `D CPOL;
     spi_state_cnt <= `D 5'd0;
   end
 end else begin
   spi_state_cnt <= `D spi_state_cnt;
 end
end
```

具体波形如下：

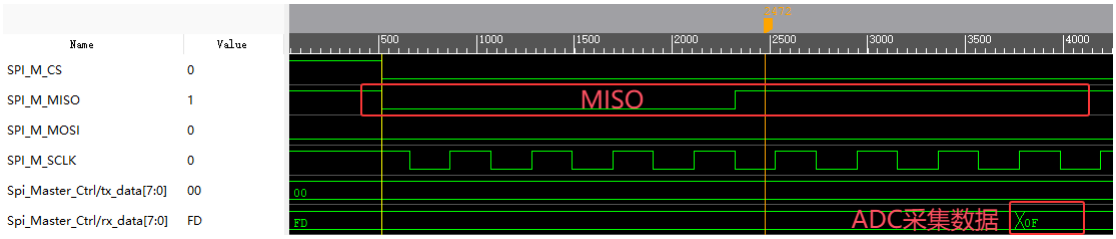


图 1-7 ADC 采集数据

其他模块的详细介绍可参考开发板对应学习资料。

【高云】【ACG720】高云 ACG720 138k&60k 教学开发板产品使用自助服务手册

https://www.corecourse.cn/forum.php?mod=viewthread&tid=29765

(出处: 芯路恒电子技术论坛)

主要模块已经介绍完了，接下来进行板级验证。

1.4 板级验证

本次实验使用 ACG720-60K 以及其板载 CY7C68013 、板载 CH340E 与板载 ADC128S102 进行验证。

1.4.1 系统所需硬件

1. ACG720-60k 开发板
2. 电源线一根
3. Type-c 数据线两根
4. 下载器一个

1.4.2 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，一根连接开发板左上方的 USB 转 TTL 串口与电脑。

连接好下载器与电源适配器。

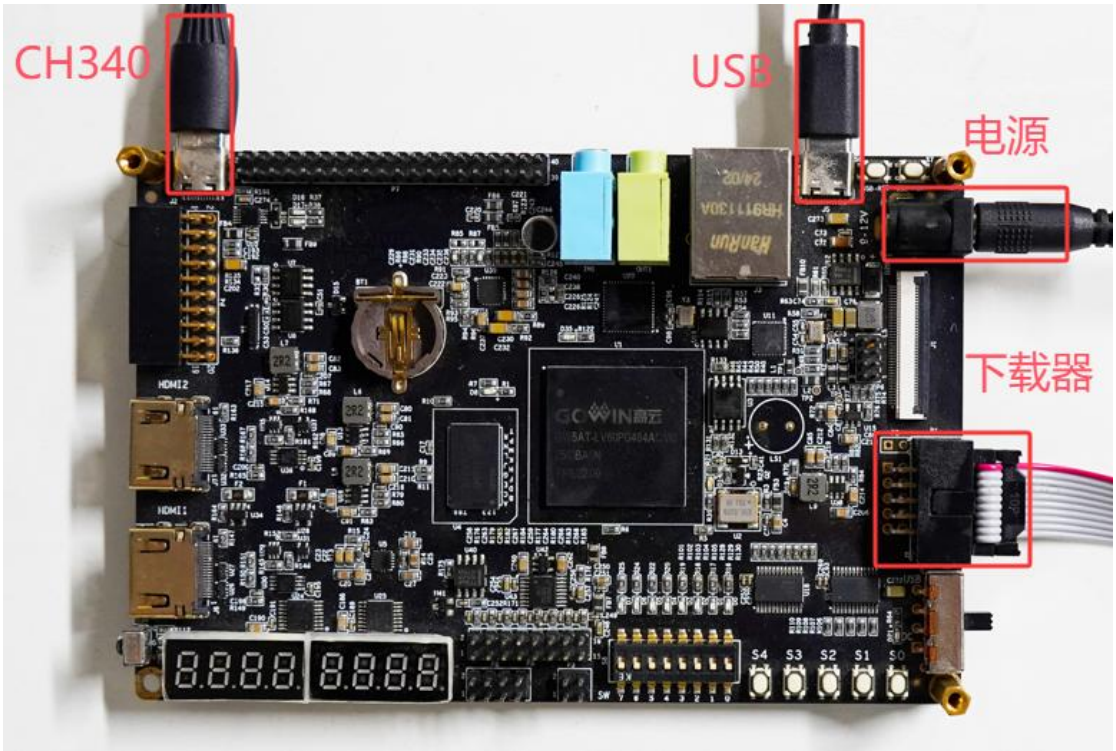


图 1-8 硬件连接

连接完毕后接下来即可进行 FX2 固件的烧写。

1.4.3 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。

我们在资料中提供了一个 fx2lp_cdc_clrfifo_nozeropkt.iic 固件。

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 1-9 所示。

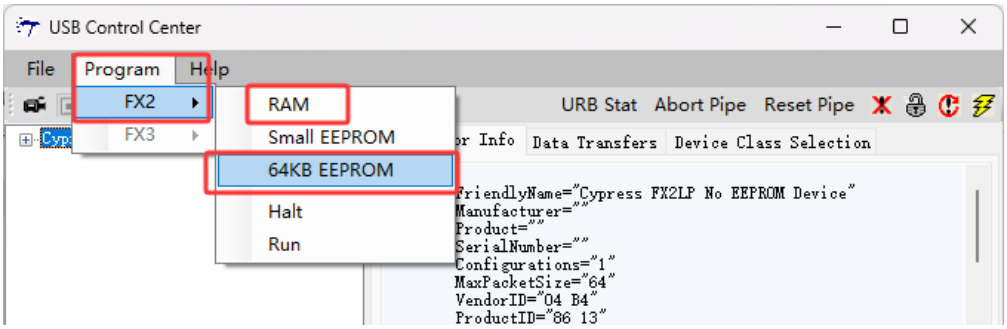


图 1-9 固件烧写界面

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 1-10 所示。

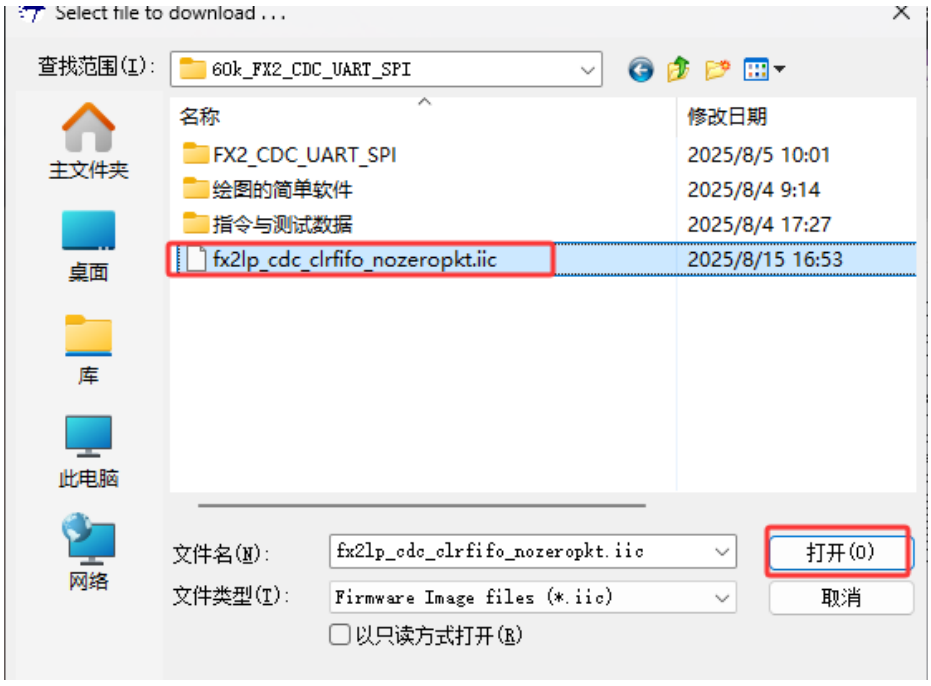


图 1-10 烧写固件

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 1-11 所示。

图 1-11 USB 被识别为串行设备

1.4.4 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	SW0	G22
	reset_n	F15	SW1	D22
数码管	st_cp	C2	sh_cp	F4
	ds	M18		
SPI	SPI_M_SCLK	N5	SPI_M_MOSI	M5
	SPI_M_MISO	P6	SPI_M_CS	M6
串口	uart_tx	M15	uart_rx	J21
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
	FX2_SPI_SCLK	L15	FX2_SPI_MOSI	K16
	FX2_SPI_MISO	L16	FX2_SPI_CS	L14

1.4.5 下载与验证

程序编译成功后即可以进行下载验证。

点击 GOWIN 软件的 Programmer 标志。检测到下载器后点击 Save 确认，如图 1-12 所示。

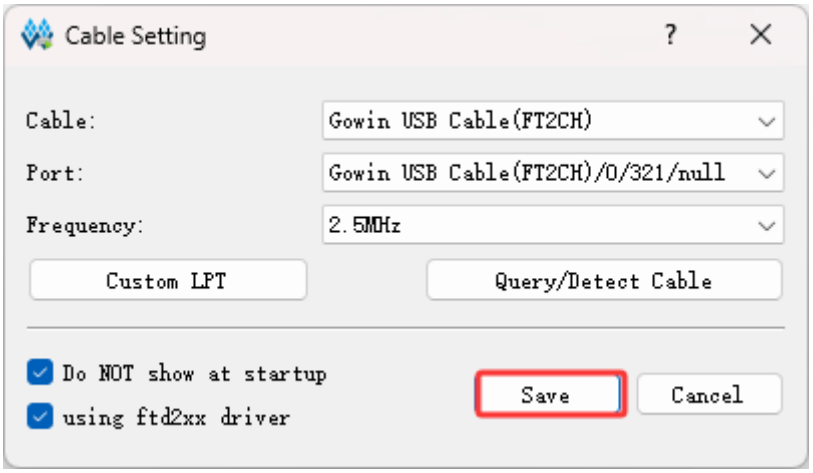


图 1-12 识别下载器

之后点击 进行程序烧录，下方的输出栏出现如下图 1-13 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 1-13 烧录成功

接下来进行回环验证。

同时打开两个串口助手，端口分别连接 CH340（波特率设置 115200）与串行设备（具体端口号以自己电脑为准）并运行。如图 1-14 所示。

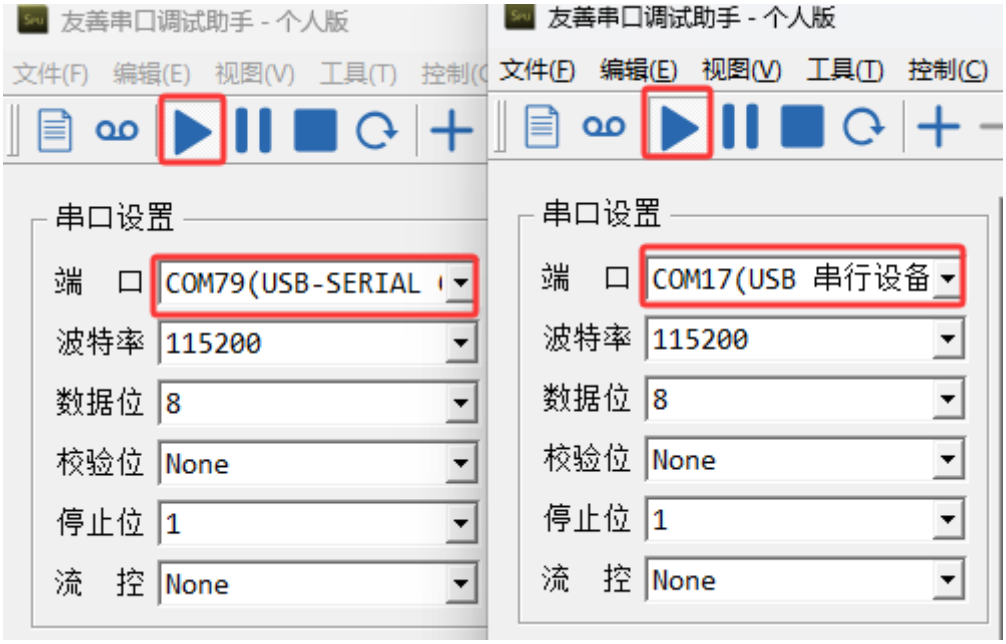


图 1-14 串口助手设置

将拨码开关 **SW1 置为 1**（拨到上方），在连接 CH340 的串口助手发送栏发送想要测试的数据，或者发送我们提供的测试数据，即可看到连接串行设备串口助手正确接收到了发送的数据。

在连接串行设备的串口助手发送栏发送想要测试的数据，或者发送我们提供的测试数据，即可看到连接 CH340 的串口助手正确接收到了发送的数据。如图 1-15 与图 1-16 所示。

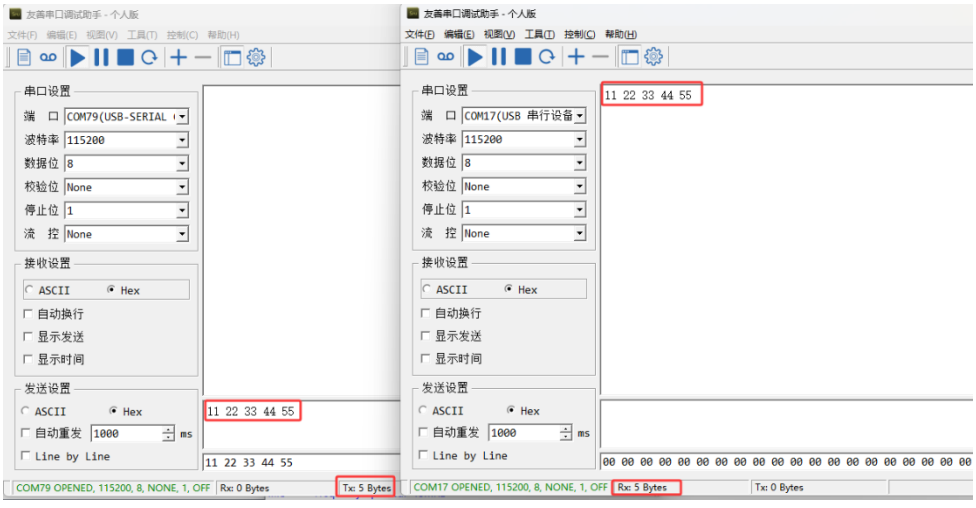


图 1-15 CH340>串行设备 5 字节回环

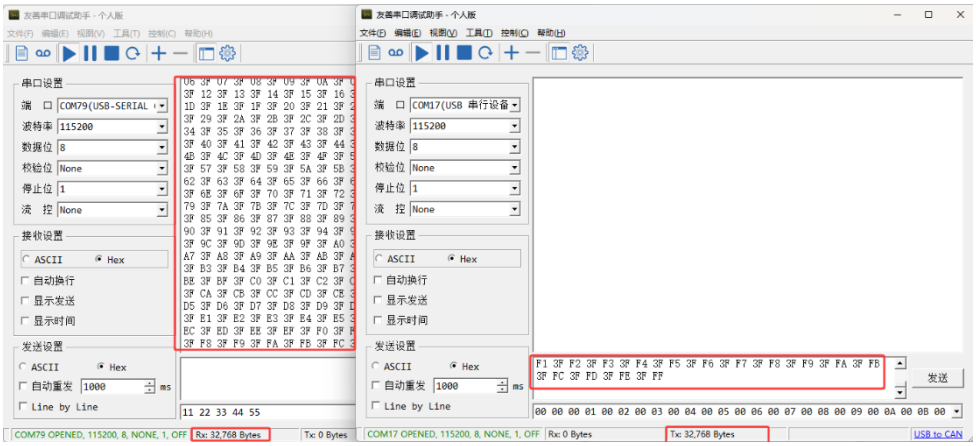


图 1-16 串行设备>CH340 32768 字节

接下来进行 ADC 采集验证。

将拨码开关 **SW1 置为 0**（拨到下方），通过连接串行设备的串口助手，发送资料里提供的采集指令，即可在串口助手中接收到 ADC 对应通道的采集数据。下图为使用 8MHz 的 SPI 抓取 ADC 通道 7 的波形（1KHz 正弦波 0~3V）与使用 12MHz 的 SPI 抓取 ADC 通道 0 的波形（1KHz 正弦波 0~3V）。

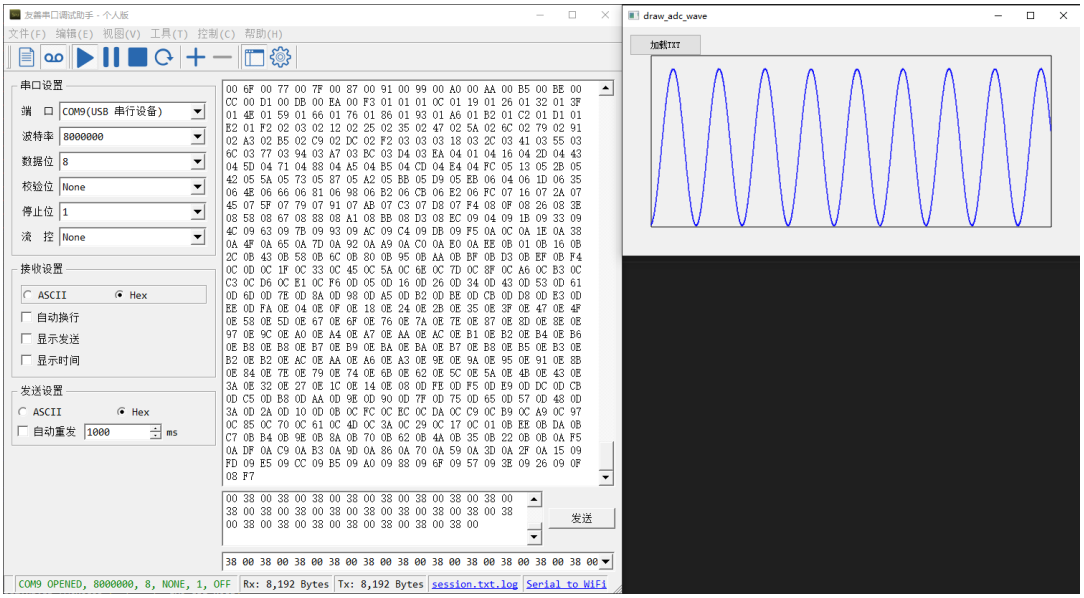


图 1-17 8MHz 抓取通道 7

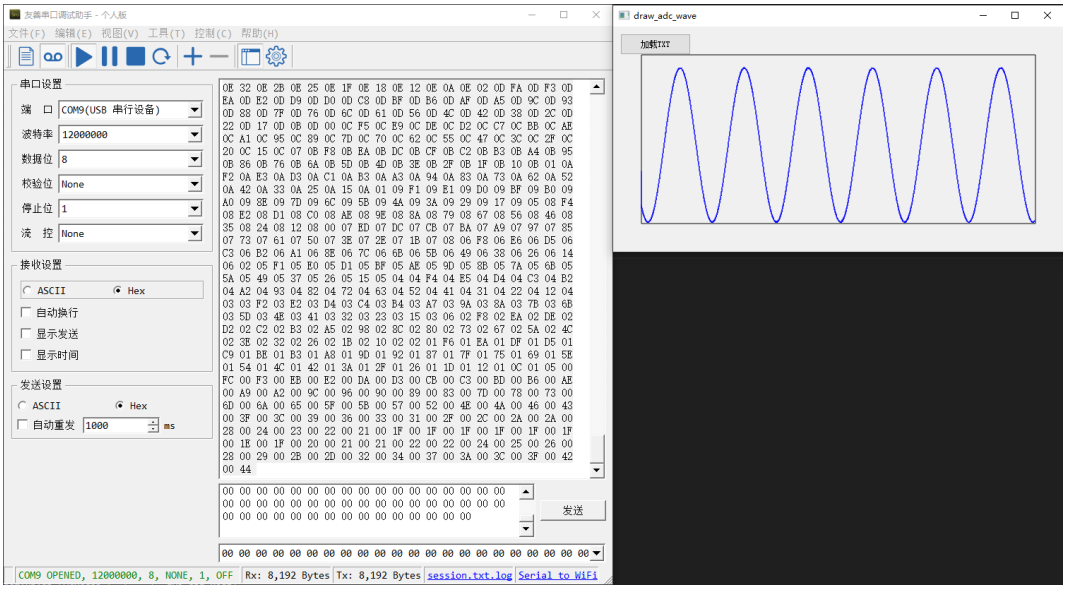


图 1-18 12MH 抓取通道 0

将串口助手采集的的数据保存为.txt 文件，可以用配套案例提供的“绘图的简单软件”绘制波形。在对应目录下找到“draw_adc_wave.exe”并双击打开，如下图所示。

图 1-19 软件路径

点击软件内的加载 TXT 并在弹出的选择页面选择保存的.txt 格式文件，即可根据采集数据绘制简单波形。如图 1-19 与图 1-21 所示。

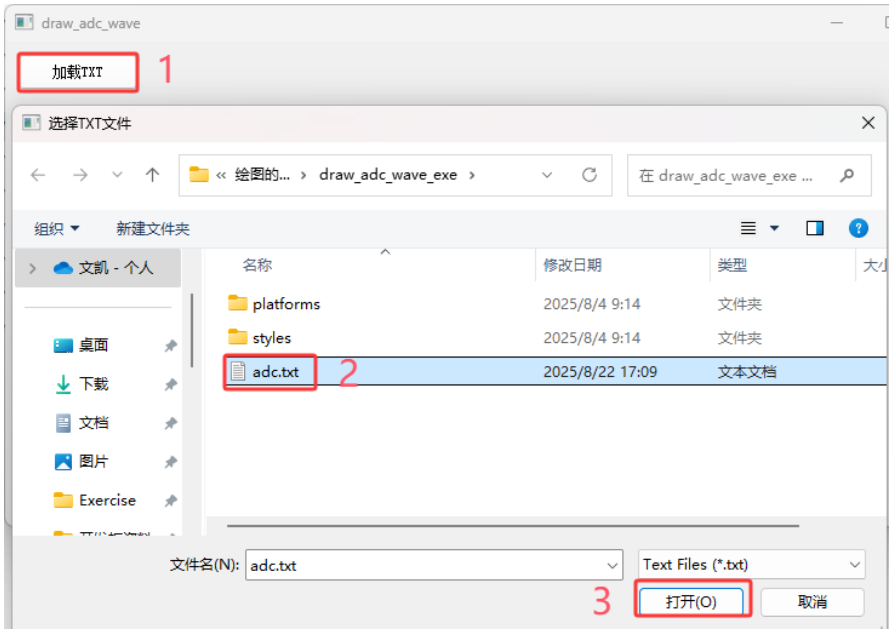


图 1-20 加载波形数据文件

图 1-21 简单波形

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 1-22 数码管显示

至此，基于 FX2 的 USB CDC 串口与 SPI 读写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。

1.5 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 实现了板载串口与 CDC 虚拟串口的回环传输、板载 ADC128S102 的 SPI 读写与 CDC 虚拟串口参数的实时显示。本实验介绍了了 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 ADC128S102 的工作原理。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，对其他模块进行 SPI 读写功能的设计。