浙江大學

《EDA技术应用》实验报告

项目名称：UART 模块设计与通信实现

姓　　名：

学　　号：

学　　院：生物医学工程与仪器科学学院

专　　业：生物医学工程

指导老师：黄添添、裘利坚

2026 年 6 月 8 日

浙江大学 实验报告

课程名称：EDA 技术应用　实验类型：EDA 硬件实验　指导老师：黄添添、裘利坚

实验名称：实验五：UART 模块设计与通信实现

姓　　名：　　学　　号：　　专　　业：生物医学工程

实验地点：玉泉教六 204　　日　　期：2026 年 6 月 8 日

目　录

一、实验目的

(1) 了解 Quartus II 开发环境，熟悉 Verilog 语言的基本语法；
(2) 了解 UART 模块功能；
(3) 通过 UART 模块实现开发板与电脑进行回环通信。

二、实验原理与模块架构

本实验主要设计了一个串行数据的回环系统：由 PC 端通过串口发送数据给 FPGA，FPGA 接收解析后将其存入内部缓冲队列，随后再从队列中读出数据并原封不动地通过串口发送回 PC 端。整个工程采用自顶向下的模块化设计。

2.1 总体架构设计

整个回环系统采用自顶向下（Top-Down）的模块化设计方法，在顶层文件 `uart.v` 中对三大核心功能模块进行物理网表级互连，分别为：串口接收接口模块（`rx_interface.v`）、中间控制模块（`inter_control_module.v`）以及串口发送接口模块（`tx_interface.v`）。

在物理层面上，PC 端通过串口调试助手将并行数据拆解为标准的物理电平流，经由 USB 转串口芯片输入至 FPGA 的接收引脚 `RX_Pin_in`。FPGA 接收接口模块将其还原为并行字节并送入缓冲区排队；中间控制模块扮演数据搬运工角色，在监测到接收队列非空且发送队列未满时，将并行字节无缝分发至发送端缓冲区；最终发送接口模块将并行字节再次拆解为串行比特流，通过 `TX_Pin_out` 引脚逆向发回 PC 端，完成闭环验证。

利用弹性 FIFO（先进先出）数据队列作为收发两端能够有效平滑 PC 端突发高速写入与 FPGA 硬件串行输出之间的速率吞吐差，阻断低速发送链路对高速接收链路的反向挤压。

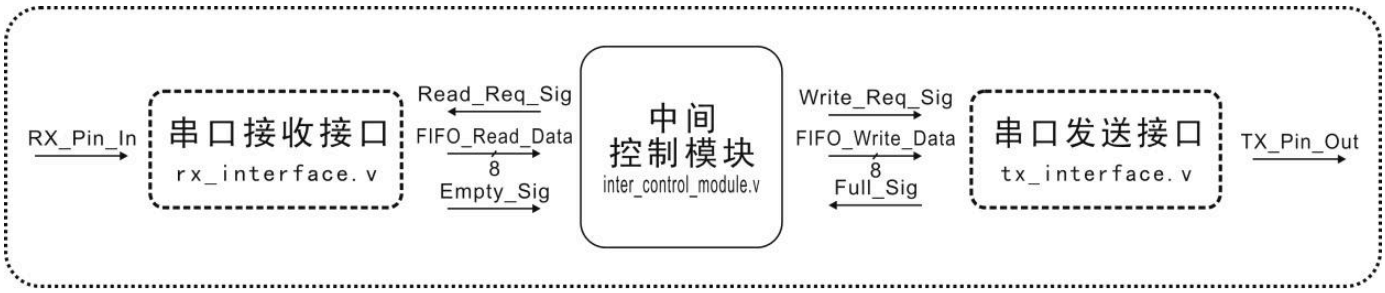

图 2.1 – UART 总模块结构

2.2 串口接收接口结构

串口接收接口模块（`rx_interface.v`）是系统捕获外部串行信号的关口，其内部由边缘检测、波特率生成、移位解析、顶层同步和队列缓冲五个子模块协同构建：

(1) 边缘检测模块（`detect_module`）：常态下串口物理线维持高电平（1），数据帧开端由一个周期的低电平（0）作为起始位。该模块采用双级同步寄存器对 `RX_Pin_in` 进行硬件打拍，引入单周期时钟级延时。通过比对前后时钟周期的电平状态（`H2L_Sig = F2 & !F1`），能够在消除跨时钟域亚稳态的同时，精准捕获起始位的下降沿，并向总管发出脉冲信号。
(2) 接收波特率时钟控制模块（`rx_bps_module`）：建立在 FPGA 核心的 50MHz 主频之上。针对 9600 bps 的预设波特率，该模块维护一个模值为 5208（$50,000, \frac{000}{9600}$）的分频计数器。为了规避信号在线路传输中受电容效应形成的上升沿/下降沿电平毛刺干扰，本设计核心在于将数据采样点精

确定位在波特率计数的正中央位置（第 2604 个时钟周期），此时物理线上的电压信号处于全周期内最稳定的波峰或波谷，能最大程度提高抗干扰信噪比。

(3) 底层移位接收核（rx_control_module）：这是一个基于换位时钟脉冲驱动的有限状态机（FSM）。当总管使能开启后，它在每个中央采样脉冲到达时递增状态计数器（0-10），依次跳过起始位，顺序抓取串行线上的最低有效位（LSB）至最高有效位（MSB），将其存入内部移位寄存器的对应抽屉，并在最终捕获到高电平停止位后拉高 RX_Done_Sig。

(4) 接收控制状态机（rx_top_control_module）：该模块作为核心总管，采用严格的握手机制协调底层。常态下关闭接收使能以保持静默，一旦接收到边缘检测传入的下降沿触发脉冲后，才正式启动底层接收核。在接收完全部比特后，它会主动校验内置 FIFO 的满状态。若队列未满，则向下游的缓冲队列施加单时钟周期的写使能脉冲 Write_Req_Sig，将组装好的并行字节强行灌入仓库，随后立即撤销使能重回监听状态。

(5) 硬件 FIFO 缓冲区（rx_fifo）：提供 16 字节深度的自控队列。负责接纳瞬时打包完成的并行字节，并向中间控制模块抛出多负载状态信号（如 Empty_Sig、Full_Sig）。

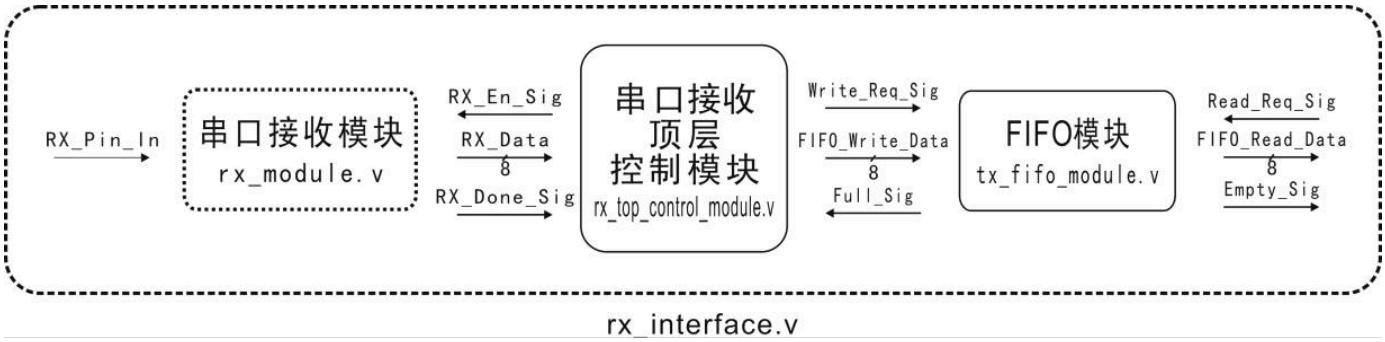

图 2.2 – 串口接收接口模块结构 rx_interface.v

2.3 串口发送接口结构

串口发送接口模块（tx_interface.v）负责逆向将 FPGA 内存中的并行字节打碎为串行物理电平输出，其工作流向与接收端方向相反、逻辑对称：

(1) 硬件 FIFO 缓冲区（tx_fifo）：存放由中间控制模块的待发送回环字节，其内部读指针的滚动直接受控于发送端顶层。

(2) 发送控制状态机（tx_top_control_module）：该模块构成发送端的调度中枢。它常态化扫描发送 FIFO 的状态指标。一旦空信号 Empty_Sig 为低，意味着缓冲区内有排队积压的待发包裹。此时状态机启动，拉高 Read_Req_Sig。针对硬件存储器（双口 RAM 架构）从触发读使能到总线数据稳定呈现存在 1 个时钟周期流水线延迟的物理特性，本设计在状态机中特别注入了一个周期的静止等待状态。在确保总线数据完全建立并收敛后，再提取有效字节交付底层物理移位核，并拉高发送使能开关 TX_En_Sig。

(3) 底层串行发送控制核（tx_control_module）：标准的并串转换（Parallel to Serial）移位寄存器实体。在使能有效的背景下，状态机在每个发送波特率切换脉冲处演进。它首先强行将发送物理引脚 TX_Pin_out 拉低 1 个波特率周期（作为起始位 0），随后依照状态机计数，由低位到高位（D0-D7）依次将锁存的 8 位并行字节刷新到引脚上，最后拉高引脚电平（作为停止位 1）完成一帧协议的硬件驱动，并向总管反馈 TX_Done_Sig。

(4) 发送波特率时钟控制模块（tx_bps_module）：分频基准同样为 5208。由于发送端属于信号驱动，为了保障发送出来的方波电平饱满、脉宽严格符合 1 bit 的物理长度，其时钟脉冲 BPS_CLK 产生

在波特率计数器的边界交界点（第 5207 个时钟周期）。此时产生脉冲，能够瞬间切位，直接刷新下一个比特位的物理电平状态。

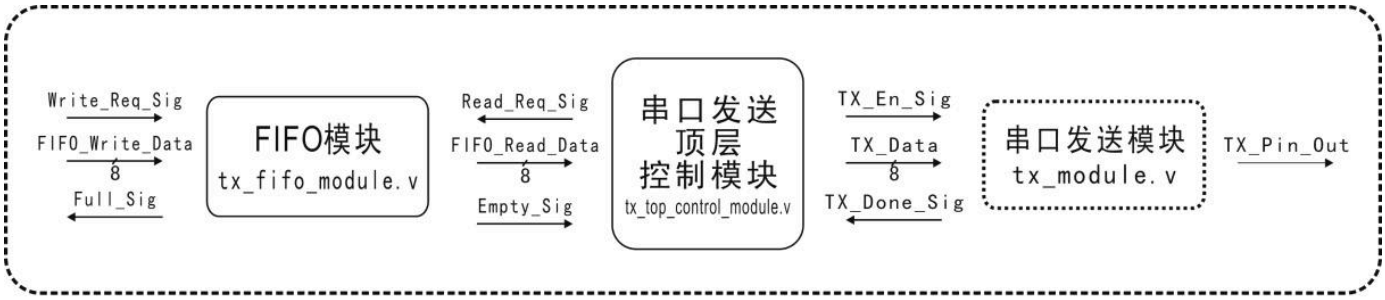

图 2.3 – 串口发送接口模块结构 tx_interface.v

三、主要仪器设备

Quartus ii 13.1，DE2-115 开发板，串口调试工具。

四、操作方法与实验步骤

实验项目中的顶层文件为 uart.v，可以看到 uart.v 中实现了三个结构模块（rx、control、tx 模块）之间的连接。以下列出其中一些我认为比较重要的模块的讲解。

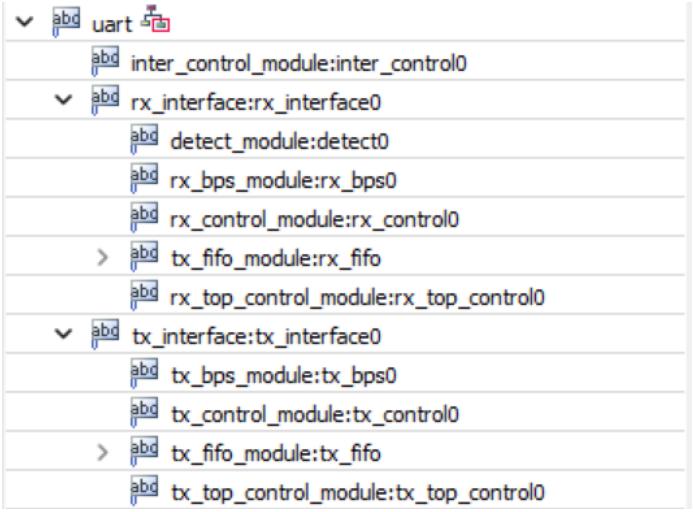

图 4.1 – 项目结构

4.1 下降沿边缘检测模块（detect_module.v）

该模块利用双级寄存器打拍子的方式，精准捕获串行输入线上的起始位低电平跳变。

```
// ====================================================================
// 模块名称: detect_module
// 功    能: 使用双级寄存器实现串口起始位下降沿（从高到低）跳变检测
// ====================================================================
module detect_module (
    input  CLK,          // 外部50MHz系统同步主时钟
    input  RSTn,         // 外部低电平复位信号（接拨码开关SW7）
    input  RX_Pin_In,    // 外部物理串口串行输入线
    output H2L_Sig       // 成功捕获到下降沿起始位的单周期脉冲警告信号
);

    reg H2L_F1;          // 第一级硬件打拍寄存器，锁存当前时钟周期的引脚状态
```

```
    reg H2L_F2;          // 第二级硬件打拍寄存器，锁存上一个时钟周期的引脚状态

    always @(posedge CLK or negedge RSTn) begin
        if (!RSTn) begin
            H2L_F1 <= 1'b1; // 空闲电平默认保持高电平1
            H2L_F2 <= 1'b1;
        end else begin
            H2L_F1 <= RX_Pin_In; // 将当前周期的引脚电平移入第一级
            H2L_F2 <= H2L_F1;    // 将前一电平推入第二级，形成时间差
        end
    end

    // 经典电路：若前一个时刻为高电平(1)且当前时刻为低电平(0)，则逻辑输出为1
    assign H2L_Sig = H2L_F2 & !H2L_F1;

endmodule
```

4.2 接收波特率时钟控制模块（rx_bps_module.v）

通过对 50MHz 系统时钟进行分频运算，锁定 9600 串口传输速率，并在波特率周期的正中央产生数据采样脉冲。

```
// ============================================================================
// 模块名称：rx_bps_module
// 功    能：9600波特率计数分频，并在数据比特正中间位置输出最佳采样时钟
// ============================================================================
module rx_bps_module (
    input  CLK,          // 50MHz系统主时钟
    input  RSTn,         // 低电平全局复位信号
    input  Count_Sig,    // 底层接收核传入的计数允许使能信号
    output BPS_CLK       // 处于波特率周期正中间的最佳数据采样脉冲信号
);

    reg [12:0] Count_BPS; // 50000000 / 9600 = 5208.3次计数，故使用13位宽寄存器

    always @(posedge CLK or negedge RSTn) begin
        if (!RSTn)
            Count_BPS <= 13'd0;
        else if (Count_BPS == 13'd5207) // 计满一整个波特率宽度（0~5207共5208个时钟周期）
            Count_BPS <= 13'd0;         // 计数器归零，开始下一位数据的周期计数
        else if (Count_Sig)             // 只有当底层接收核处于工作状态时才累加计数
            Count_BPS <= Count_BPS + 1'b1;
        else
            Count_BPS <= 13'd0;         // 空闲期计数器强制清零保持
    end

    // 核心设计：在周期的正中心位置（5208 / 2 = 2604）发出采样脉冲，此时电平最稳定
    assign BPS_CLK = (Count_BPS == 13'd2604) ? 1'b1 : 1'b0;

endmodule
```

4.3 接收顶层控制状态机（rx_top_control_module.v）

该模块作为接收端总管，采用有限状态机（FSM）管理接收时序。

```
// ============================================================================
// 模块名称：rx_top_control_module
// 功    能：接收端顶层有限状态机控制，生成严格同步的写FIFO控制时序
// ============================================================================
module rx_top_control_module (
    input            CLK,
    input            RSTn,
```

```
    input             H2L_Sig,          // 边缘检测模块传入的起跑枪信号
    input             RX_Done_Sig,      // 底层核心组装完一字节后的汇报信号
    input      [7:0]  RX_Data,          // 底层接收核串转并出来的8位并行结果
    input             Full_Sig,         // 接收内置FIFO缓冲发出的满警报
    output            RX_En_Sig,        // 允许底层串转并硬件启动的开关信号
    output            Write_Req_Sig,    // 往接收FIFO执行写入的命令脉冲
    output     [7:0]  FIFO_Write_Data   // 准备压入FIFO的并行数据总线
);

    reg [2:0] i;                        // 扩展为3位宽，支持8个状态演进
    reg       isWrite;                  // 内部写使能寄存器
    reg       isRX;                     // 内部接收使能寄存器

    always @(posedge CLK or negedge RSTn) begin
        if (!RSTn) begin
            i       <= 3'd0;
            isWrite <= 1'b0;
            isRX    <= 1'b0;
        end else begin
            case (i)
                3'd0: begin
                    if (H2L_Sig) begin    // 严密监控：只有听到起跑枪响（发现真正的起始位）
                        isRX <= 1'b1;     // 才准许开启底层硬件接收使能
                        i    <= 3'd1;     // 跳入下一阶段
                    end else begin
                        isRX <= 1'b0;     // 空闲期坚决不启动，防止误抓线上高电平造成FF乱码
                    end
                end
                3'd1: begin
                    if (RX_Done_Sig) begin // 挂起并静静等待底层10个比特位串并转换完全部结束
                        isRX <= 1'b0;       // 接收完毕后立即关闭接收使能
                        i    <= 3'd2;
                    end
                end
                3'd2: begin
                    if (!Full_Sig) begin    // 检查接收FIFO内部是否还有富余空间
                        i <= 3'd3;
                    end else begin
                        i <= 3'd0;          // 溢出满则丢弃当前帧，重回空闲状态0
                    end
                end
                3'd3: begin
                    isWrite <= 1'b1;        // 触发一个时钟周期的高电平写FIFO命令
                    i       <= 3'd4;
                end
                3'd4: begin
                    isWrite <= 1'b0;        // 迅速撤销写请求，结束本次数据的入库搬运
                    i       <= 3'd0;        // 重新回到状态0等待下一帧起始位的降临
                end
                default: i <= 3'd0;
            endcase
        end
    end

    assign RX_En_Sig       = isRX;
    assign Write_Req_Sig   = isWrite;
    assign FIFO_Write_Data = RX_Data;

endmodule
```

4.4 通用硬件 FIFO 数据缓冲队列模块（tx_fifo_module.v）

分别在接收接口和发送接口中各例化一次，利用双口 RAM 空间与读写指针的双向滚动，实现数据的排队与吞吐弹性缓冲。

```
// ============================================================================
// 模块名称: tx_fifo_module
// 功    能: 通用FIFO数据队列缓冲组件（深度16，位宽8），负责平滑收发流速差
// ============================================================================
`define FIFO_DEPTH 16
`define DATA_WIDTH 8

module tx_fifo_module (
    input                       CLK,              // 全局同步时钟线
    input                       RSTn,             // 全局低电平复位线
    input  [`DATA_WIDTH-1:0] FIFO_Write_Data, // 外部传入的待写入并行字节
    input                       Write_Req_Sig,    // 外部发起的写请求使能指令
    input                       Read_Req_Sig,     // 外部发起的读请求使能指令
    output reg [`DATA_WIDTH-1:0] FIFO_Read_Data, // FIFO内读出送往外部的总线数据
    output                      Full_Sig,         // 队列存满警报（禁止写入）
    output                      Empty_Sig         // 队列放空警报（禁止读取）
);

    parameter FIFO_WIDH = 4; // 16个存储位置对应4位指针（0~15环形寻址）

    reg [FIFO_WIDH-1:0] rp_reg;  // 内部读格指针计数器，指向当前即将被读取的数据位置
    reg [FIFO_WIDH-1:0] wp_reg;  // 内部写格指针计数器，指向当前即将被填入的数据位置
    reg [FIFO_WIDH:0]   cnt_reg; // 存储总量计数器（0~16存储，用5位宽防止溢出边界）

    // 内存阵列形式定义的物理双口RAM寄存器组（16个深度的8位寄存器）
    reg [`DATA_WIDTH-1:0] men [`FIFO_DEPTH-1:0];

    // 1. 环形读指针维护逻辑
    always @(posedge CLK or negedge RSTn) begin
        if (RSTn == 0)
            rp_reg <= 4'd0;
        else if (!Empty_Sig && Read_Req_Sig == 1'b1)
            rp_reg <= rp_reg + 1'b1; // 如果没空且收到读请求，指针向前滚动一格
        else
            rp_reg <= rp_reg;
    end

    // 2. 环形写指针维护逻辑
    always @(posedge CLK or negedge RSTn) begin
        if (RSTn == 0)
            wp_reg <= 4'd0;
        else if (!Full_Sig && Write_Req_Sig == 1'b1)
            wp_reg <= wp_reg + 1'b1; // 如果没满且收到写请求，指针向前滚动一格
        else
            wp_reg <= wp_reg;
    end

    // 3. FIFO内部动态载荷存量实时维护状态机
    always @(posedge CLK or negedge RSTn) begin
        if (RSTn == 0)
            cnt_reg <= 5'd0;
        // 读写操作同时有效发生，存量相对保持平衡抵消
        else if ((Read_Req_Sig && !Empty_Sig) && (Write_Req_Sig && !Full_Sig))
            cnt_reg <= cnt_reg;
        // 仅发生了有效的数据读出动作，容量计数递减 1
        else if (Read_Req_Sig == 1'b1 && !Empty_Sig)
            cnt_reg <= cnt_reg - 1'b1;
        // 仅发生了有效的数据存入动作，容量计数递增 1
        else if (Write_Req_Sig == 1'b1 && !Full_Sig)
            cnt_reg <= cnt_reg + 1'b1;
```

```
        else
            cnt_reg <= cnt_reg;
    end

    // 4. 同步时钟作用下的物理双口RAM寄存器读写映射
    always @(posedge CLK) begin
        if (!Full_Sig && Write_Req_Sig == 1'b1)
            men[wp_reg] <= FIFO_Write_Data; // 数据正式落盘存入当前的写格
        if (!Empty_Sig && Read_Req_Sig == 1'b1)
            FIFO_Read_Data <= men[rp_reg];  // 提取出当前读格内的数据抛向外部总线
    end

    // 逻辑组合判断输出空满指示标志
    assign Empty_Sig = (cnt_reg == 5'd0)  ? 1'b1 : 1'b0;
    assign Full_Sig  = (cnt_reg == 5'd16) ? 1'b1 : 1'b0;

endmodule
```

五、实验结果

使用串口调试助手并且连接开发板后得到的发送接收数据的情况如下，可以看到能够能够正确发送数据，并且能够正确地将这一帧数据接收回来。

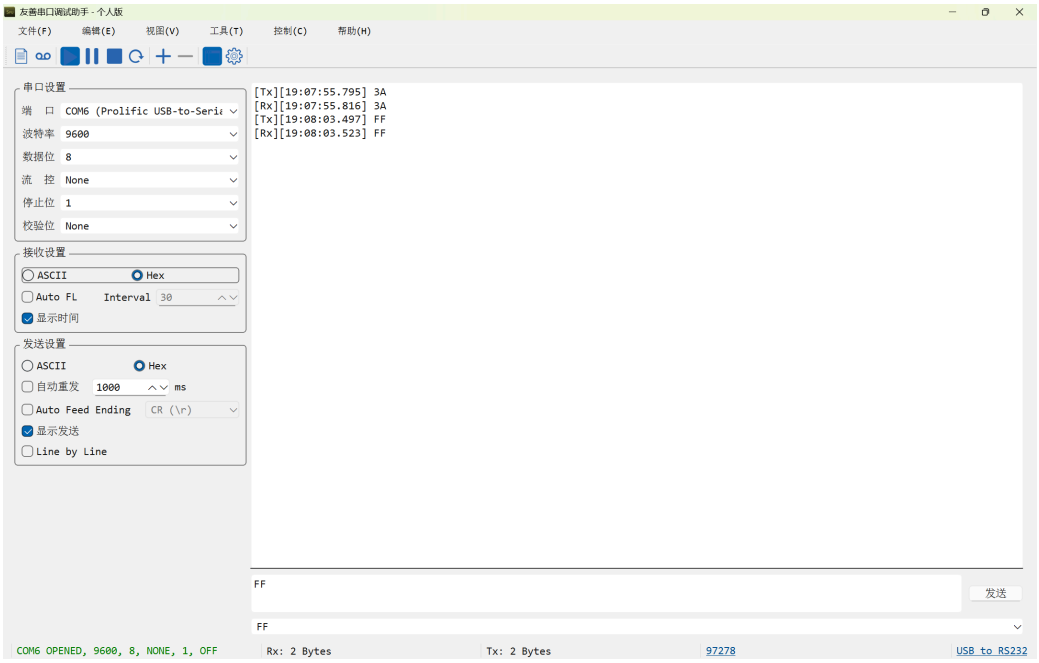

图 5.1 – 项目结构

六、实验心得与总结

在整个模块设计与上板调试的过程中，我遇到了许多具有代表性的底层硬件问题，通过不断地“现象分析—定位 bug—修改时序”，我对 EDA 设计有了极其深刻的体悟，总结如下：

(1) 软硬件编程思维的范式转换与流水线延迟。在前期调试中，系统曾出现过“发送新数据，却接收上一次历史数据”的诡异错位现象。经深入排查底层逻辑后发现，这是由于用纯软件逻辑的惯性思维去操作硬件 RAM 导致的。在硬件实体中，双口 RAM（FIFO 缓冲）从接收到读使能信号 Read_Req_Sig，到将总线电平稳定呈现出来，客观上存在 1 个时钟周期的物理延迟。原始状态机在发出读请求后立即抓取数据，导致抓取到的永远是总线上的旧电平。通过在状态机中果断引入一拍“等待状态”，数据错位就能得到解决。

(2) 异步系统中的同步化与边界触发机制的绝对必要性。在系统联调初期，曾出现过串口空闲时开发板向 PC 端“狂发 FF 乱码”的系统假死现象。通过对 UART 协议和底层时序图的倒推，我定位到了问题根源：由于收发双方没有全局同步时钟，若接收端状态机缺少严格的起始位检测，就会将总线空闲期的高电平（持续的 1）盲目采集并转换为 FF。在顶层控制模块中引入边缘检测产生的 H2L_Sig（下降沿单周期脉冲）作为严格的触发条件后，系统恢复了极高的稳定性。这表明，在缺乏握手时钟的异步通信中，利用双级寄存器打拍消除亚稳态，并施加严苛的边沿触发约束，是保障系统不陷入伪工作状态的核心防线。

(3) 工程容错率设计与中点采样抗干扰机制。在分频波特率时钟的设计上，本次实验摒弃了在计数器边缘采样的简陋做法，而是将数据抓取脉冲 BPS_CLK 严格锁定在计数周期的一半（2604 处）。这种中点采样策略有效地规避了物理连线上由分布电容引发的上升沿/下降沿毛刺。

(4) “自顶向下”的架构分割与基于现象的硬件 Debug 方法论。本次实验面对的是一个多模块协同的复杂工程，如果采用扁平化代码结构将无从下手。通过划分“物理移位核—缓冲层 FIFO—顶层控制状态机”，各个模块职责明确、松耦合，极大降低了综合难度。同时，我也建立了一套标准的 FPGA 排错流程：从 Quartus 的综合 Error 排除，到 Pin Planner 中检查复位按键电平极性（如低电平有效带来的常态死锁），再到利用上位机现象倒推硬件 FIFO 残留问题。这种“宏观现象 $\rightarrow$ 微观协议 $\rightarrow$ 寄存器时序”的逆向调试能力。

最后，感谢本学期所有实验过程中，我的小组队友贺子珏、助教哥哥对我实验过程中的帮助！