# 远程实验 01_通过串口与外设通信

## 简介

本次实验中，我们将利用在线实验平台上的串口作为主要交互接口，实现对开关、LED 等基本外设的控制。

## 实验目的

了解在线实验平台结构

了解串口工作原理

学会通过串口对各类外设进行控制

## 实验环境

PC 一台（网络流畅）

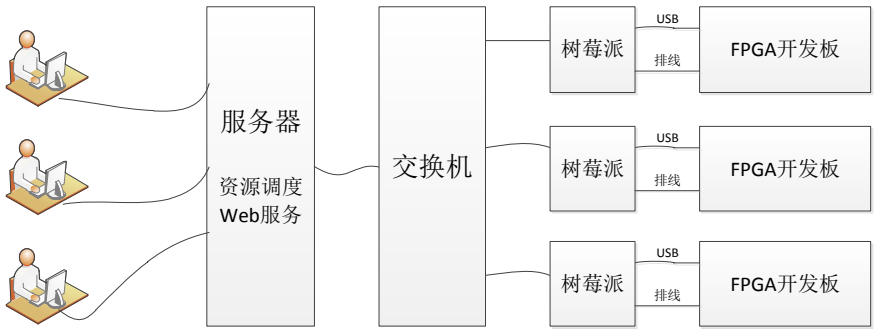
远程桌面环境（vlab）

FPGA 远程实验平台（FPGAOL）

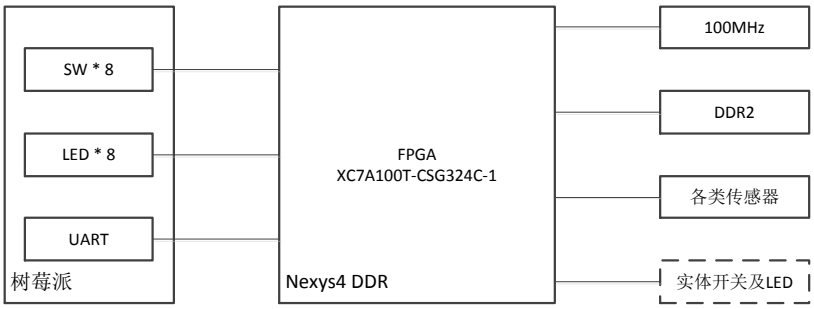
## 实验步骤

### Step1： 了解 FPGAOL 在线平台结构

用户登录 FPGAOL 平台网址之后，可申请获取一个设备节点的使用权，每个设备节点包含一个树莓派和一个 FPGA 开发板，其中树莓派对用户透明，用户在网页端操作将通过树莓派作用到 FPGA 板卡的相应管脚上，同时 FPGA 板卡的特定输出也会经树莓派采集并显示在网页端的界面上，因此可以认为树莓派是用户与 FPGA 板卡进行交互的一个代理，但从本质上来说，与用户直接操作 FPGA 板卡并没有太大的区别(由于增加了中间环节,用户体验和稳定性可能会有所差别)。




树莓派与 FPGA 之间主要有三类虚拟接口：开关、LED、串口，此外，FPGA 板卡自带的各类外设也可正常使用，由于是远程连接，无法实际操控实体开关，也无法观察实体 LED 的变化，其结构示意图如下：




## Step2： 串口环回测试

在 Vivado 中建立一个新的工程，编写串口环回测试代码，其设计代码如下：

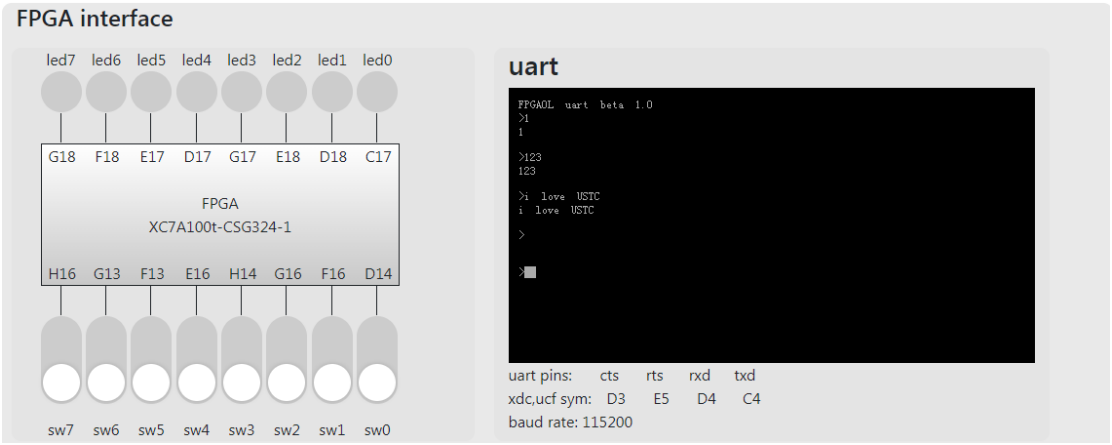
```
module top (
input       rx,
output  tx);
    assign  tx = rx;
endmodule
```

可以看出，代码非常简单，电路功能为：把从 rx 端口接受到的数据立即通过 tx 端口发送回去。

管脚约束文件为：

```
set_property -dict { PACKAGE_PIN C4     IOSTANDARD LVCMOS33 } [get_ports { rx }];
set_property -dict { PACKAGE_PIN D4     IOSTANDARD LVCMOS33 } [get_ports { tx }];
```

对工程进行综合，最后将生成的 bit 文件烧写到 FPGAOL 平台，并通过串口终端进行测试。可以发现，通过串口发送的数据又原封不动的显示在了终端上。




## Step3：通过串口与开关和 LED 通信

在该步骤中，我们设计一个串口控制器，将从树莓派发过来的最后一个非“回车”字符，以二进制形式显示在 8bit 位宽的 LED 上。此外，在接收到字符的同时，读取 8 个拨动开关的状态，通过串口发送给上位机，最终以 ASCII 码的形式显示在串口终端上。

串口的工作原理和发送、接受电路的实现代码我们已经在之前的数电电路实验课程中学习过了，这里直接复用之前的代码，如下所示：

接收端电路源代码：

```
module rx(
    input               clk,rst,
    input               rx,
    output  reg         rx_vld,
    output  reg  [7:0]  rx_data
    );
parameter   DIV_CNT    = 10'd867;
parameter   HDIV_CNT   = 10'd433;
parameter   RX_CNT     = 4'h8;
parameter   C_IDLE     = 1'b0;
parameter   C_RX       = 1'b1;
```

```
reg         curr_state;
reg         next_state;
reg [9:0]   div_cnt;
reg [3:0]   rx_cnt;
reg
rx_reg_0,rx_reg_1,rx_reg_2,rx_reg_3,rx_reg_4,rx_reg_5,rx_reg_6,rx_reg_7;
//reg [7:0]   rx_reg;
wire        rx_pulse;
always@(posedge clk or posedge rst)
begin
    if(rst)
        curr_state  <= C_IDLE;
    else
        curr_state  <= next_state;
end
always@(*)
begin
    case(curr_state)
        C_IDLE:
            if(div_cnt==HDIV_CNT)
                next_state  = C_RX;
            else
                next_state  = C_IDLE;
        C_RX:
            if((div_cnt==DIV_CNT)&&(rx_cnt>=RX_CNT))
                next_state  = C_IDLE;
            else
                next_state  = C_RX;
    endcase
end

always@(posedge clk or posedge rst)
begin
    if(rst)
        div_cnt <= 10'h0;
    else if(curr_state == C_IDLE)
    begin
        if(rx==1'b1)
            div_cnt <= 10'h0;
        else if(div_cnt < HDIV_CNT)
            div_cnt <= div_cnt + 10'h1;
        else
            div_cnt <= 10'h0;
    end
```

```
    else if(curr_state == C_RX)
    begin
        if(div_cnt >= DIV_CNT)
            div_cnt <= 10'h0;
        else
            div_cnt <= div_cnt + 10'h1;
    end
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        rx_cnt  <= 4'h0;
    else if(curr_state == C_IDLE)
        rx_cnt  <= 4'h0;
    else if((div_cnt == DIV_CNT)&&(rx_cnt<4'hF))
        rx_cnt  <= rx_cnt + 1'b1;
end
assign rx_pulse = (curr_state==C_RX)&&(div_cnt==DIV_CNT);
always@(posedge clk)
begin
    if(rx_pulse)
    begin
        case(rx_cnt)
            4'h0: rx_reg_0 <= rx;
            4'h1: rx_reg_1 <= rx;
            4'h2: rx_reg_2 <= rx;
            4'h3: rx_reg_3 <= rx;
            4'h4: rx_reg_4 <= rx;
            4'h5: rx_reg_5 <= rx;
            4'h6: rx_reg_6 <= rx;
            4'h7: rx_reg_7 <= rx;
        endcase
    end
end
always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        rx_vld  <= 1'b0;
        rx_data <= 8'h55;
    end
    else if((curr_state==C_RX)&&(next_state==C_IDLE))
    begin
        rx_vld  <= 1'b1;
```

```
        rx_data                                                    <=
{rx_reg_7,rx_reg_6,rx_reg_5,rx_reg_4,rx_reg_3,rx_reg_2,rx_reg_1,rx_reg_0};
    end
    else
        rx_vld  <= 1'b0;
end
endmodule
```

发送端源代码为：

```
module tx(
    input           clk,rst,
    output  reg     tx,
    input           tx_ready,
    output  reg     tx_rd,
    input   [7:0]   tx_data
);
parameter   DIV_CNT   = 10'd867;
parameter   HDIV_CNT  = 10'd433;
parameter   TX_CNT    = 4'h9;
parameter   C_IDLE    = 1'b0;
parameter   C_TX      = 1'b1;

reg         curr_state,next_state;
reg [9:0]   div_cnt;
reg [4:0]   tx_cnt;
reg [7:0]   tx_reg;
always@(posedge clk or posedge rst)
begin
    if(rst)
        curr_state  <= C_IDLE;
    else
        curr_state  <= next_state;
end
always@(*)
begin
    case(curr_state)
        C_IDLE:
            if(tx_ready==1'b1)
                next_state  = C_TX;
            else
                next_state  = C_IDLE;
        C_TX:
            if((div_cnt==DIV_CNT)&&(tx_cnt>=TX_CNT))
                next_state  = C_IDLE;
```

```
            else
                next_state  = C_TX;
    endcase
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        div_cnt <= 10'h0;
    else if(curr_state==C_TX)
    begin
        if(div_cnt>=DIV_CNT)
            div_cnt <= 10'h0;
        else
            div_cnt <= div_cnt + 10'h1;
    end
    else
        div_cnt <= 10'h0;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        tx_cnt  <= 4'h0;
    else if(curr_state==C_TX)
    begin
        if(div_cnt==DIV_CNT)
            tx_cnt <= tx_cnt + 1'b1;
    end
    else
        tx_cnt <= 4'h0;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        tx_rd   <= 1'b0;
    else if((curr_state==C_IDLE)&&(tx_ready==1'b1))
        tx_rd   <= 1'b1;
    else
        tx_rd   <= 1'b0;
end
always@(posedge clk or posedge rst)
begin
    if(rst)
        tx_reg  <= 8'b0;
    else if((curr_state==C_IDLE)&&(tx_ready==1'b1))
```

```
        tx_reg  <= tx_data;
end

always@(posedge clk or posedge rst)
begin
    if(rst)
        tx  <= 1'b1;
    else if(curr_state==C_IDLE)
        tx  <= 1'b1;
    else if(div_cnt==10'h0)
    begin
        case(tx_cnt)
            4'h0:   tx  <= 1'b0;
            4'h1:   tx  <= tx_reg[0];
            4'h2:   tx  <= tx_reg[1];
            4'h3:   tx  <= tx_reg[2];
            4'h4:   tx  <= tx_reg[3];
            4'h5:   tx  <= tx_reg[4];
            4'h6:   tx  <= tx_reg[5];
            4'h7:   tx  <= tx_reg[6];
            4'h8:   tx  <= tx_reg[7];
            4'h9:   tx  <= 1'b1;
        endcase
    end
end
endmodule
```

顶层代码为：

```
module top(
input               clk,rst,
input               rx,
output              tx,
output reg  [7:0]   led,
input       [7:0]   sw);
wire                tx_ready;
wire        [7:0]   tx_data;
wire        [7:0]   rx_data;
rx                  rx_inst(
.clk                (clk),
.rst                (rst),
.rx                 (rx),
.rx_vld             (rx_vld),
.rx_data            (rx_data)
);
```

```
tx                  tx_inst(
.clk                (clk),
.rst                (rst),
.tx                 (tx ),
.tx_ready           (tx_ready),
.tx_rd              (tx_rd),
.tx_data            (tx_data)
);
assign  tx_ready    = rx_vld;
assign  tx_data     = sw;
always@(posedge clk or posedge rst)
begin
    if(rst)
        led <= 8'h0;
    else if((rx_vld)&&(rx_data!=8'h0a))
        led <= rx_data;
end
endmodule
```

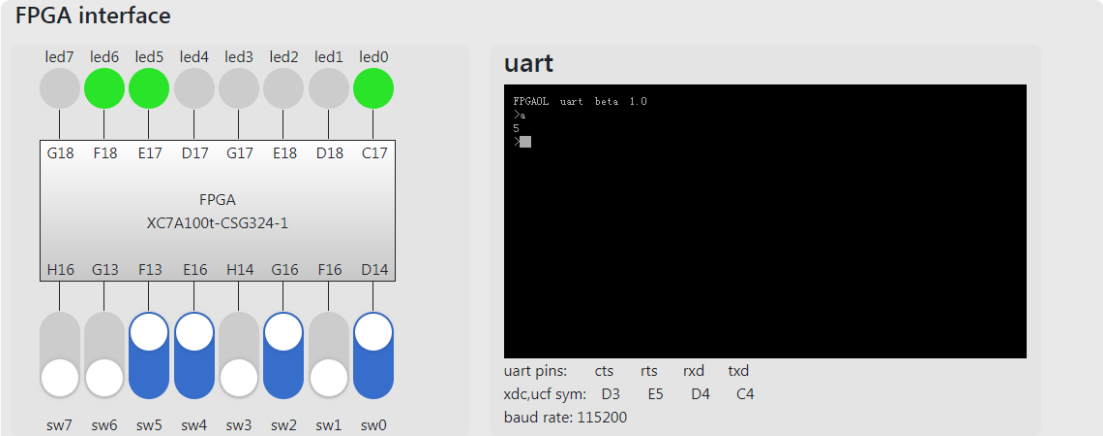
其管脚约束文件为：

```
#clk,rst
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { rst }];
#uart
set_property -dict { PACKAGE_PIN C4    IOSTANDARD LVCMOS33 } [get_ports { rx }];
set_property -dict { PACKAGE_PIN D4    IOSTANDARD LVCMOS33 } [get_ports { tx }];

#Switch
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { sw[7] }];
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { sw[6] }];
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { sw[5] }];
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
#Led
set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
```

```
set_property -dict { PACKAGE_PIN C17    IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
```

将生成的 bit 文件烧写进 FPGAOL，通过串口终端输入字符‘a’，可以发现，led 灯的状态变成了 8'b0110_0001（字符‘a’的 ASCII 码），同时将开关状态作为 ASCII 码所对应的字符（8'b0011_0101 对应字符‘5’）输出到终端，如下图所示。




## step4:实现命令解析功能

该步骤要求在串口协议基础上，实现一个读写命令解析功能，如下表所示，功能电路接收以 ASCII 码格式发来的命令，并根据命令类型做出合适的响应。

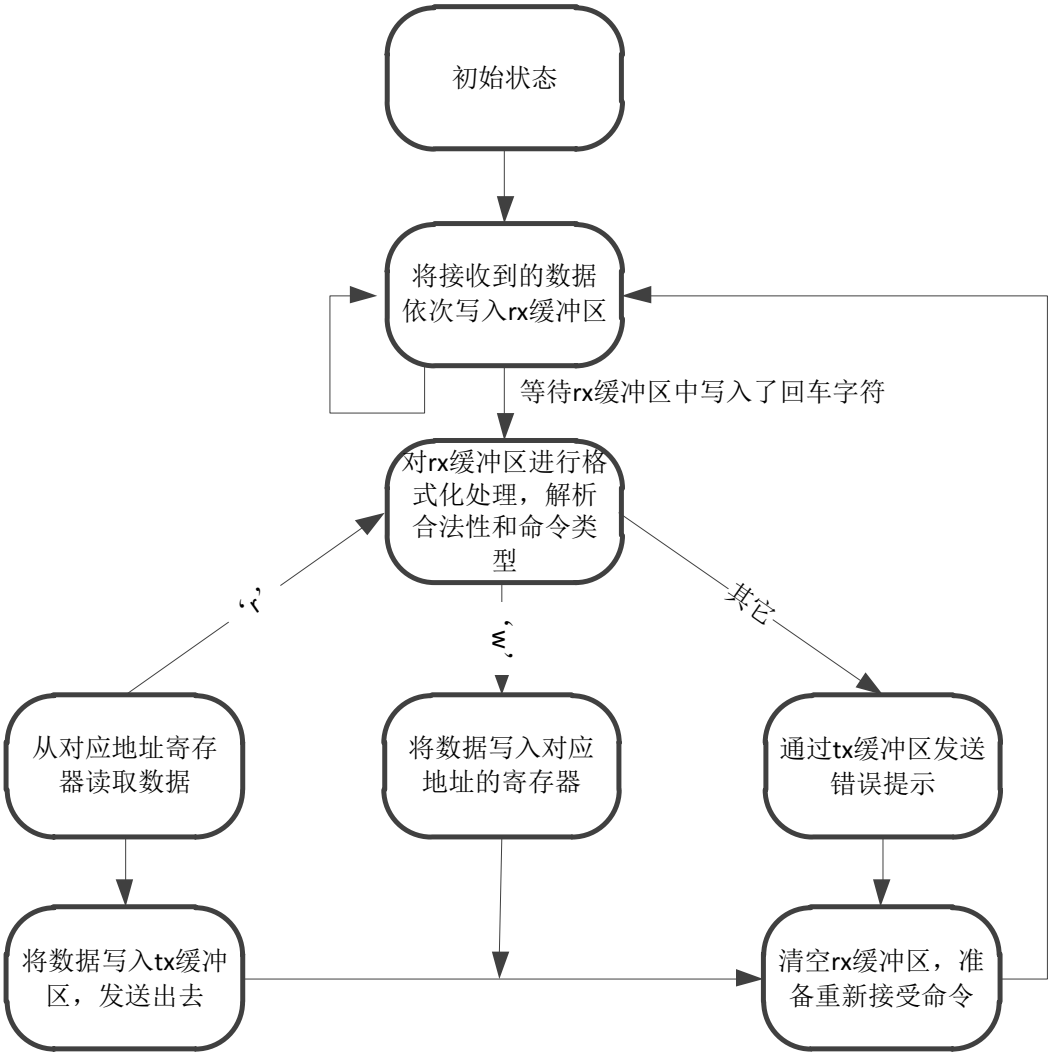
| 命令 | 格式 | 示例 | 说明 |
|---|---|---|---|
| 写命令 | w addr data | >w 0 f0 | 将数据写入指定地址，无返回值。 |
| 读命令 | r addr | >r 0<br>f0 | 读取指定地址的数据，并显示在终端中。 |
| 其它命令 | xxx xxx xx | >x y<br>E! | 除读写命令外，均为非法命令，返回“E！” |

为简化设计，我们只支持读和写两个命令，同时对命令格式进行

了严格的限制，凡不符合格式的均为非法命令。同时本设计只支持两个寄存器，如下表所示：

| 寄存器地址 | 说明 |
| --- | --- |
| 0 | 对应 led 灯的状态，该寄存器可读、可写 |
| 1 | 对应拨动开关状态，该寄存器只读，写无效 |

提示：可通过如下所示的有限状态机来实现电路功能




示例运行结果如下所示：

| | |
|---|---|
| >w 0 f0 | >向地址0写入数据f0 |
| >r 0 | >从地址0读取数据 |
| f0 | 返回地址0的数据（f0） |
| >w 1 ff | >向地址1写入数据ff |
| >r 1 | >从地址1读取数据 |
| 35 | 返回地址1的数据（35） |
| >test | >其它命令 |
| E! | 返回错误提示：E! |

注意：网页端的串口终端在收到回车键‘\n’后才会向 FPGA 发送整串的 ASCII 码数据

说明：有兴趣的同学可尝试完成该步骤的编码和上板验证工作，我们后续会争取提供一个可用的 bit 流文件和参考示例代码。

## 总结与思考

1.请总结本次实验的收获

2.请评价本次实验的难易程度

3.请评价本次实验的任务量

4.请为本次实验提供改进建议