串口收发器和存储器的使用实验报告

BobAnkh

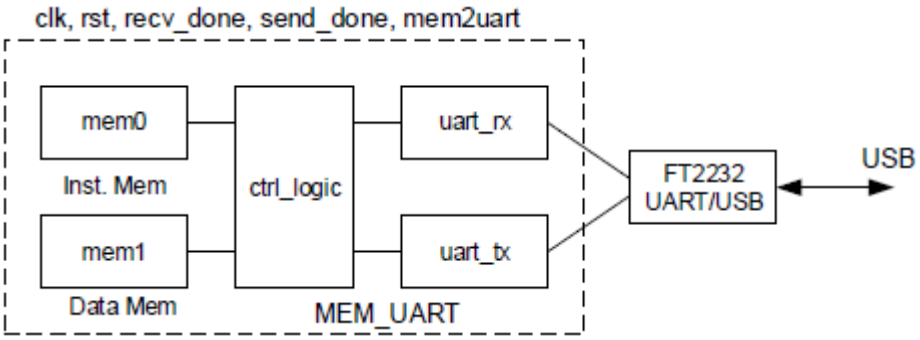
1、实验目的：

了解和掌握 UART 的工作原理，进一步熟悉仿真验证方法，为后续设计做准备。

2、设计方案：

串口接收器包括发送器和接收器两个模块，从串口通过接收器模块接受外部数据并存储至存储器中，数据接收完成后拉高 recv_done 信号。当 mem2uart 信号为高电平时，将从存储器中将数据通过发送器模块自串口向外部发送，发送完毕后，拉高 send_done 信号。

示意图如下：



在实际编写 testbench 进行仿真过程中，除了例化了整个串口接收器以外，额外添加了一个 uart_rx 用来接收串口输出数据，并将之保存到内存中，从而便于和参照数据进行比对。

3、关键代码：

由于代码量较多，这里仅给出核心 testbench 代码，其余部分可以见对应附件。将数据读取后通过串口接收器接收到内存中，完成接收后将 mem2uart 置为 1，向串口外发送数据，并通过另一个 uart_rx 来进行接收并存储，与参考数据进行比对。

```
// read and receive data
initial
begin
    $readmemh("C:/Users/lenovo/send.txt", send_data);
    $readmemh("C:/Users/lenovo//ref.txt",ref_data);
    for (count = 0; count<13'd4096 ; count = count+1'd1) begin
        Rx_Serial = 0;
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=send_data[count][0];
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=send_data[count][1];
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=send_data[count][2];
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=send_data[count][3];
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=send_data[count][4];
```

```
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=send_data[count][5];
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=send_data[count][6];
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=send_data[count][7];
        #(PERIOD*CLKS_PER_BIT) Rx_Serial=1;
        #(PERIOD*CLKS_PER_BIT);
    end
end

//Recieve done and change to send data
always @(posedge recv_done) begin
    $display("Receive Done!");
    mem2uart = 1;
end

always @(posedge s_Rx_DV) begin
    if (cnt<13'd2048) begin
        output_data[cnt] = s_Rx_Byte;
        cnt = cnt+1'd1;
    end
end

//Validate
always @(posedge send_done) begin
    $display("Send done!");
    for(count=0; count<13'd2048; count=count+1) begin
        if(ref_data[count]!=output_data[count])begin
            $display("The %d is not correct!",count);
            $display("Ref:%d",ref_data[count]);
            $display("Output:%d",output_data[count]);
            correct = 0;
        end
    end
    if (correct == 0) begin
        $display("Test not pass!Something is wrong!");
    end
    else begin
        $display("Test pass!All is correct!");
    end
    #(PERIOD*8000000)
    $stop;
end
endmodule
```

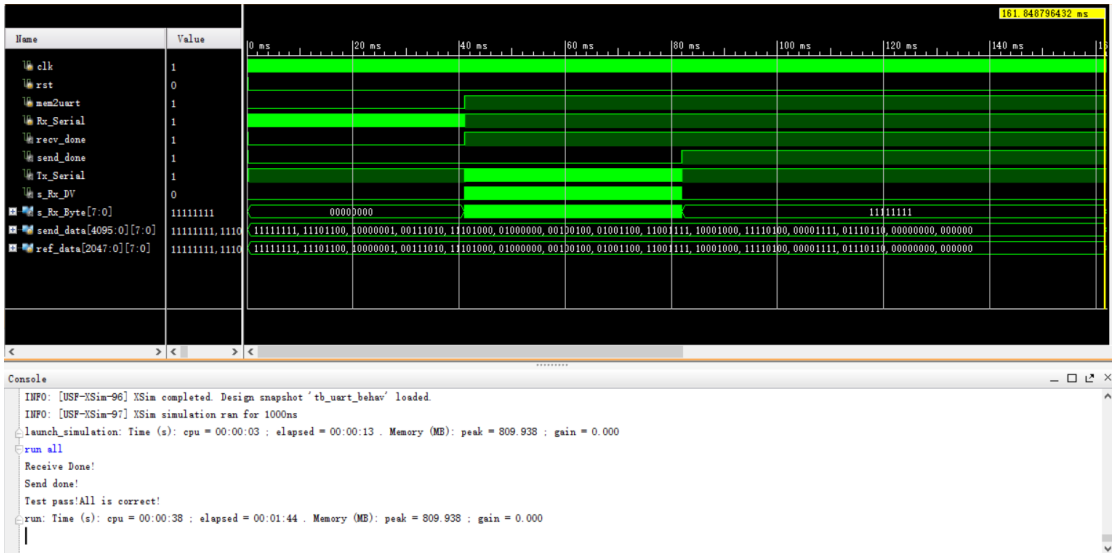
4、文件清单：

因为本次实验只是自己编写 testbench 进行仿真验证，所以随报告仅附上 testbench 文件：tb_uart.v

5、仿真结果及分析：

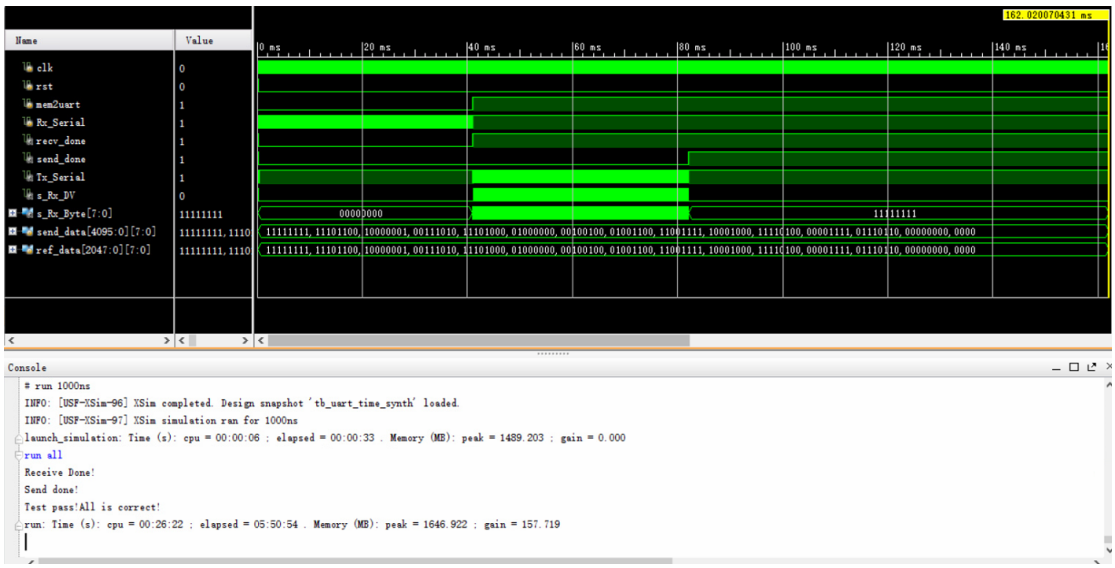
在本实验中，由于仿真性能和时间等原因，将波特率设置成了 1M，正常完成了仿真工作。

前仿（行为仿真）：



可以看到顺利实现了相关功能，tcl 控制台给出了验证正确的判断，说明串口输出的数据和参考数据是一致的，没有错误，符合预期设计。

时序仿真：

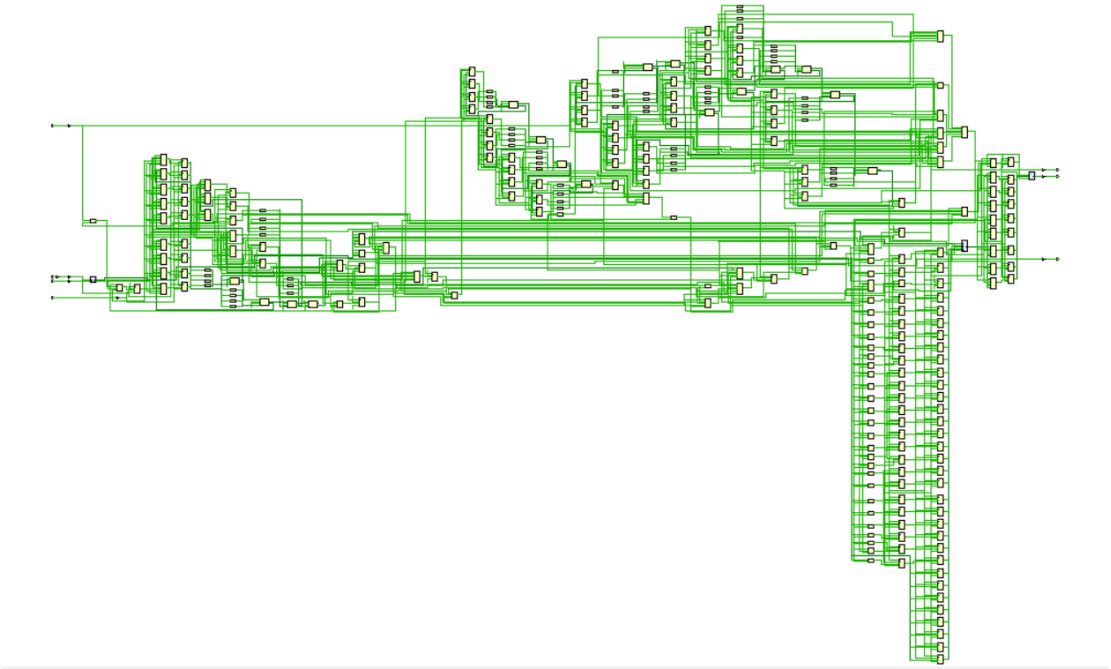


可以看到顺利实现了相关功能，tcl 控制台给出了验证正确的判断，说明串口输出的数据和参考数据是一致的，没有错误，符合预期设计。

6、综合情况：

查看整体综合之后的电路原理图如下：

在进行综合时选择将 flatten_hierarchy 属性置成了 none，以便顺利后仿，综合后对应的电路原理图为：



综合后 FPGA 逻辑资源情况：

Name	Slice LUTs* (20800)	Slice Registers (41600)	F7 Muxes (16300)	Block RAM Tile (50)	Bonded IOB (210)	BUFGCTRL (32)
UART_MEM	248	198	2	0.5	7	1
mem1 (mem)	0	0	0	0.5	0	0
uart_rx_inst (uart_rx)	78	33	1	0	0	0
uart_tx_inst (uart_tx)	41	31	1	0	0	0

可以看到，UART_MEM 模块使用了 248 个 LUT 和 198 个 Register，使用量较大，应该是因为没有让 vivado 进行优化的原因。

7、 时序性能

整体的时序性能：

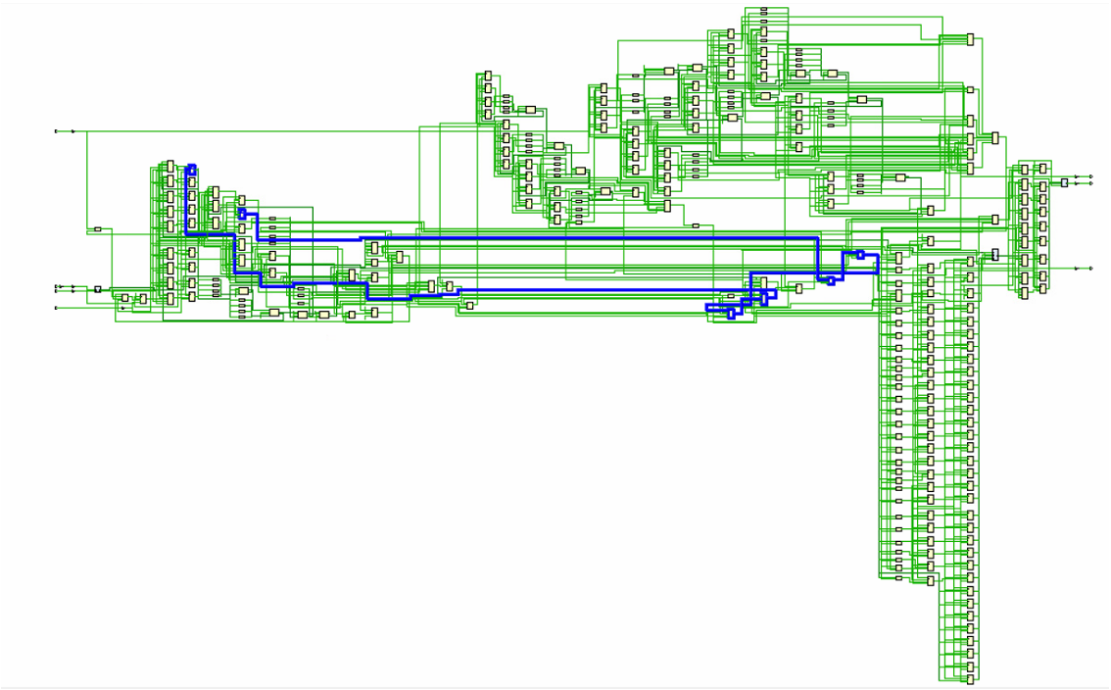
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.606 ns	Worst Hold Slack (WHS):	0.252 ns	Worst Pulse Width Slack (WPWS):	4.460 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	435	Total Number of Endpoints:	435	Total Number of Endpoints:	201

All user specified timing constraints are met.

可以看到，建立时间约束最坏的 slack 为 5.606ns，保持时间约束最坏的 slack 为 0.252ns

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Source Clock	Destination Clock	Exce
Path 1	5.606	4	34	addr1_reg[10]/C	addr1_reg[0]/CE	4.150	1.163	2.987	clk	clk	
Path 2	5.606	4	34	addr1_reg[10]/C	addr1_reg[10]/CE	4.150	1.163	2.987	clk	clk	
Path 3	5.606	4	34	addr1_reg[10]/C	addr1_reg[11]/CE	4.150	1.163	2.987	clk	clk	
Path 4	5.606	4	34	addr1_reg[10]/C	addr1_reg[12]/CE	4.150	1.163	2.987	clk	clk	
Path 5	5.606	4	34	addr1_reg[10]/C	addr1_reg[13]/CE	4.150	1.163	2.987	clk	clk	
Path 6	5.606	4	34	addr1_reg[10]/C	addr1_reg[14]/CE	4.150	1.163	2.987	clk	clk	
Path 7	5.606	4	34	addr1_reg[10]/C	addr1_reg[15]/CE	4.150	1.163	2.987	clk	clk	
Path 8	5.606	4	34	addr1_reg[10]/C	addr1_reg[1]/CE	4.150	1.163	2.987	clk	clk	
Path 9	5.606	4	34	addr1_reg[10]/C	addr1_reg[2]/CE	4.150	1.163	2.987	clk	clk	
Path 10	5.606	4	34	addr1_reg[10]/C	addr1_reg[3]/CE	4.150	1.163	2.987	clk	clk	

从 10 条 setup 的 slack 最短的路径中，在原理图上找出其中时间最短的一条路径：



查看它的具体信息如下：

Summary

Name	Path 1
Slack	5.606ns
Source	addr1_reg[10]/C (rising edge-triggered cell FDRE clocked by clk {rise@0.000ns fall@5.000ns period=10.000ns})
Destination	addr1_reg[0]/CE (rising edge-triggered cell FDRE clocked by clk {rise@0.000ns fall@5.000ns period=10.000ns})
Path Group	clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	10.000ns (clk rise@10.000ns - clk rise@0.000ns)
Data Path Delay	4.150ns (logic 1.163ns (28.024%) route 2.987ns (71.976%))
Logic Levels	4 (LUT4=2 LUT5=1 LUT6=1)
Clock Path Skew	-0.040ns
Clock Uncertainty	0.035ns

Source Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clk rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000	Site: P17	clk
net (fo=0)	0.000	0.000		clk
			Site: P17	clk_IBUF_inst/I
IBUF (Prop ibuf I O)	(r) 1.478	1.478	Site: P17	clk_IBUF_inst/O
net (fo=1, unplaced)	0.800	2.278		clk_IBUF
				clk_IBUF_BUFG_inst/I
BUFG (Prop bufg I O)	(r) 0.096	2.374		clk_IBUF_BUFG_inst/O
net (fo=200, unplaced)	0.800	3.174		clk_IBUF_BUFG
				addr1_reg[10]/C

Data Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
FDRE (Prop fdre C Q)	(f) 0.496	3.670		addr1_reg[10]/Q
net (fo=2, unplaced)	0.752	4.422		n_0_addr1_reg[10]
				addr1[15]_i_6/I1
LUT4 (Prop lut4 I1 O)	(f) 0.295	4.717		addr1[15]_i_6/O
net (fo=2, unplaced)	0.676	5.393		n_0_addr1[15]_i_6
				addr1[15]_i_4/I0
LUT4 (Prop lut4 I0 O)	(f) 0.124	5.517		addr1[15]_i_4/O
net (fo=20, unplaced)	0.509	6.026		n_0_addr1[15]_i_4
				wdata1[31]_i_1/I0
LUT5 (Prop lut5 I0 O)	(r) 0.124	6.150		wdata1[31]_i_1/O
net (fo=34, unplaced)	0.522	6.672		n_0_wdata1[31]_i_1
				addr1[15]_i_1/I3
LUT6 (Prop lut6 I3 O)	(r) 0.124	6.796		addr1[15]_i_1/O
net (fo=16, unplaced)	0.528	7.324		n_0_addr1[15]_i_1
FDRE				addr1_reg[0]/CE
Arrival Time		7.324		

Destination Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clk rise edge)	(r) 10.000	10.000		
	(r) 0.000	10.000	Site: P17	clk
net (fo=0)	0.000	10.000		clk
			Site: P17	clk_IBUF_inst/I
IBUF (Prop ibuf I O)	(r) 1.408	11.408	Site: P17	clk_IBUF_inst/O
net (fo=1, unplaced)	0.760	12.168		clk_IBUF
				clk_IBUF_BUFG_inst/I
BUFG (Prop bufg I O)	(r) 0.091	12.259		clk_IBUF_BUFG_inst/O
net (fo=200, unplaced)	0.760	13.019		clk_IBUF_BUFG
				addr1_reg[0]/C
clock pessimism	0.116	13.134		
clock uncertainty	-0.035	13.099		
FDRE (Setup fdre C CE)	-0.169	12.930		addr1_reg[0]
Required Time		12.930		

由以上信息可以分析其 slack 是怎么得出的：数据通路上的到达时间 Arrival Time 为 7.324ns，又由其要求的时间 Required Time 为 12.930ns，由 Required Time 减去 Arrival Time 即可得到这条路径的建立时间的 slack，为 5.606ns