数字逻辑与处理器基础实验
第四次作业

无 81 马啸阳 2018011054

2020 年 5 月 17 日

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 实验原理

2.1 串口原理

串口（UART，Universal Asynchronous Receiver/Transmitter）是一种全双工异步通信接口。其时序示意图如图 1所示，发送一个字节时，先发送一个起始位逻辑 0，随后是从 LSB 至 MSB 的 8 个数据位，最后为 1，1+1/2 或 2 个停止位逻辑 1。

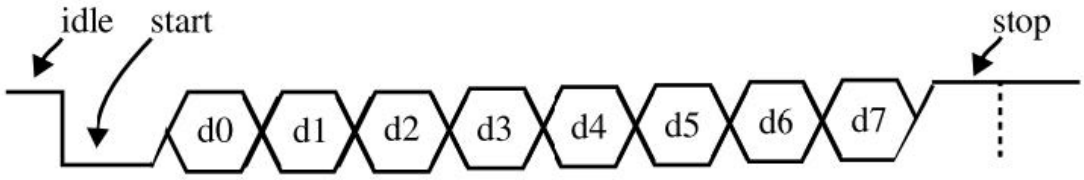

图 1: 串口时序示意图

每位数据持续时间不变，由发送器时钟控制，每秒发送的数据位个数称为波特率。要保证正常收发，需要发送与接收端波特率相差不过大。对于 9600 的波特率，发送一个数据位时间位 1/9600 秒。由于是异步通信，接收器与发送器时钟信号不同，二者采样间隔不同，接收器设计不好可能会导致采样错误。

2.2 实验设计

串口收发器如图 2所示，含发送器和接收器两个模块。接收器模块从外部串行接收数据，发送器模块串行向外发送数据。控制模块实现从串口到存储器与从存储器到串口的数据流控制。

串口到存储器方向，从串口读取 512*32*2bit 的数据到 FPGA 中的指令存储器和数据存储器，其中指令存储空间为 512*32bit，数据存储空间为 512*32bit。数据接收完成，拉高 recv_done 信号。

存储器到串口方向，当 mem2uart 为高电平时，FPGA 从数据存储器中读取数据，并通过串口发送到上位机。数据发送完成，拉高 send_done 信号。

串口接收器与发送器各通过状态机控制，接收器接收串行输入信号，每接收一个字节拉高 o_Rx_DV 一个时钟周期，并将接收到的字节并行输出。发送器反之，并行输入待发送信号，比接收器多一个使能输入和使能输出，串行输出信号，每发送一个字节拉高 o_Tx_Done 一个时钟周期。二者的状态机中以 r_Clock_Count 计数时钟周期以分频得到波特率，状态机其余部分按串口协议实现。基本状态机如图 3所示（此图为简略版本），需要其它辅助计数值帮助计数时钟周期并进行采样，接收器与发送器类似。

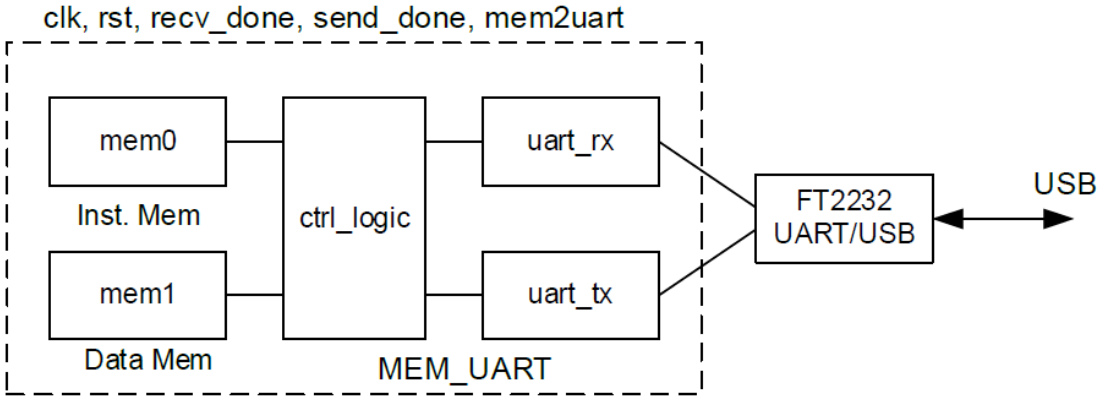

图 2: 串口收发器模块框图

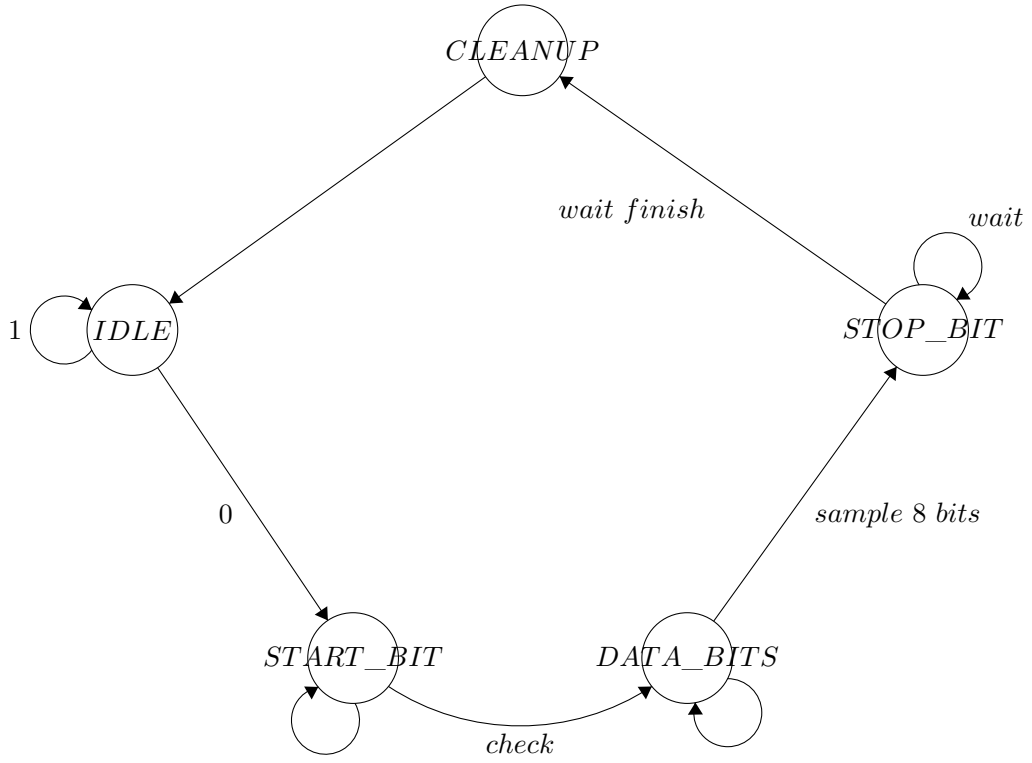

图 3: 串口状态机

对于控制模块，接收部分设置计数器计数写入地址与字中字节号，将由串口接收器进行接收，每接收到 4 个字节写入存储器，写满指令存储器开始写数据存储器，接收完成拉高 recv_done；发送部分在 mem2uart 位高电平时依次读数据存储器，由串口发送器将数据发送。

对于逐字取反码的附加功能，增加 invert 控制端，当它为 1 时，屏蔽所有先前收发功能，将数据存储器所有内容读出，求反写回。每个字使用四个周期。第一个周期，地址加一。第二个周期开始时读到数据，将数据求反。第三个周期写使能置一。第四个周期开始时写入，检查地址是否达到最高，若是则拉高 recv_done，结束过程。

3 实验代码

3.1 文件清单

```
.
└── mem.v ..... 存储器模块
```

3.2 串口收发器测试模块

测试模块中使用的是原先的串口收发器模块代码。实例化后串口收发器后，测试模块中还实例化了一个串口发送器和串口接收器用以向串口收发器发送测试数据并接收回来。测试模块中波特率（CLKS_PER_BIT）与存储器大小（MEM_SIZE）作为参数可调，但在门级仿真中需要与实现的串口收发器模块参数保持一致。测试模块先从文件读入发送的数据，通过串口发送器发送，在串口收发器发出接收完毕的信号后，拉高 mem2uart 开始接收串口收发器发送的数据存储器数据，接收完毕后写入文件进行比较。

```
1  `timescale 1ns/1ps
2  `define PERIOD 10
3
4  module UART_MEM_tb;
5  reg    clk;
6  reg    rst;
7  reg    mem2uart;
8  wire   Rx_Serial;
9  wire   recv_done;
10 wire   send_done;
11 wire   Tx_Serial;
12
13 wire   Rx_DV;
14 wire   [7:0] Rx_Byte;
15 wire   Rx_Serial_R;
16
17 reg    Tx_DV;
18 reg    [7:0] Tx_Byte;
19 wire   Tx_Active;
20 wire   Tx_Done;
```

```
21 wire  Tx_Serial_T;
22
23 parameter CLKS_PER_BIT = 16'd100;
24 parameter MEM_SIZE = 2;
25
26 integer i, j;
27 reg [7:0] send_mem [8*MEM_SIZE:1];
28 reg [7:0] recv_mem [4*MEM_SIZE:1];
29
30 assign Rx_Serial = Tx_Serial_T;
31 assign Rx_Serial_R = Tx_Serial;
32
33 UART_MEM #(.CLKS_PER_BIT(CLKS_PER_BIT), .MEM_SIZE(MEM_SIZE)) UART_MEM_inst
34          (.clk(clk),
35           .rst(rst),
36           .mem2uart(mem2uart),
37           .Rx_Serial(Rx_Serial),
38           .recv_done(recv_done),
39           .send_done(send_done),
40           .Tx_Serial(Tx_Serial)
41          );
42
43 uart_rx #(.CLKS_PER_BIT(CLKS_PER_BIT)) uart_rx_inst
44         (.i_Clock(clk),
45          .i_Rx_Serial(Rx_Serial_R),
46          .o_Rx_DV(Rx_DV),
47          .o_Rx_Byte(Rx_Byte)
48         );
49
50 uart_tx #(.CLKS_PER_BIT(CLKS_PER_BIT)) uart_tx_inst
51         (.i_Clock(clk),
52          .i_Tx_DV(Tx_DV),
53          .i_Tx_Byte(Tx_Byte),
54          .o_Tx_Active(Tx_Active),
55          .o_Tx_Serial(Tx_Serial_T),
56          .o_Tx_Done(Tx_Done)
57         );
58
59 initial begin
```

```
60      $readmemh("send.txt", send_mem);
61      i = 1;
62      j = 1;
63      clk = 0;
64      rst = 1;
65      mem2uart = 0;
66  end
67
68  initial fork
69      forever
70          #(`PERIOD/2) clk <= ~clk;
71      #200 rst <= 0;
72      #300 Tx_DV <= 1;
73      #300 Tx_Byte <= send_mem[1];
74      #300 i <= i + 1;
75  join
76
77  always @(posedge Tx_Done) begin
78      if (i <= 8 * MEM_SIZE) begin
79          Tx_Byte = send_mem[i];
80          i = i + 1;
81      end
82      else begin
83          Tx_DV = 0;
84      end
85  end
86
87  always @(posedge recv_done) begin
88      mem2uart = 1'b1;
89  end
90
91  always @(posedge Rx_DV) begin
92      if (j <= 4 * MEM_SIZE) begin
93          recv_mem[j] = Rx_Byte;
94          if (j == 4 * MEM_SIZE) begin
95              $writememh("recv.txt", recv_mem);
96              $finish;
97          end
98      end
```

```
99      j = j + 1;
100 end
101
102 endmodule
```

3.3 串口收发器模块

修改原代码，得到可取反的串口收发器模块。增加 invert 输入端，当它为 1 时，屏蔽所有先前收发功能，将数据存储器所有内容读出，求反写回。每个字使用四个周期。第一个周期，地址加一。第二个周期开始时读到数据，将数据求反。第三个周期写使能置一。第四个周期开始时写入，检查地址是否达到最高，若是则拉高 recv_done，结束过程。

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer: THU EE
5  //
6  // Create Date: 2019/05/08 17:29:33
7  // Design Name:
8  // Module Name: UART_MEM
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module UART_MEM(
24     input clk,       // 100MHz
25     input rst,       // BTNU
26     input mem2uart, // SW0
27     input invert,
```

```
28      /*----------------------------MEM--------------------------------*/
29      output reg recv_done,    // led 0
30      output reg send_done,    // led 1
31      /*----------------------------UART-------------------------------*/
32      input Rx_Serial,
33      output Tx_Serial
34      );
35
36      parameter CLKS_PER_BIT = 16'd100;  // 100M/9600
37      parameter MEM_SIZE = 2;
38
39
40      /*---------------------------------UART RX-------------------------*/
41      wire Rx_DV;
42      wire [7:0] Rx_Byte;
43
44      uart_rx #(.CLKS_PER_BIT(CLKS_PER_BIT)) uart_rx_inst
45          (.i_Clock(clk),
46           .i_Rx_Serial(Rx_Serial),
47           .o_Rx_DV(Rx_DV),
48           .o_Rx_Byte(Rx_Byte)
49           );
50
51      /*---------------------------------UART TX-------------------------*/
52      reg Tx_DV;
53      reg [7:0] Tx_Byte;
54      wire Tx_Active;
55      wire Tx_Done;
56
57      //assign Tx_DV = Rx_DV;
58      //assign Tx_Byte = Rx_Byte+1'b1;
59
60      uart_tx #(.CLKS_PER_BIT(CLKS_PER_BIT)) uart_tx_inst
61          (.i_Clock(clk),
62           .i_Tx_DV(Tx_DV),
63           .i_Tx_Byte(Tx_Byte),
64           .o_Tx_Active(Tx_Active),
65           .o_Tx_Serial(Tx_Serial),
66           .o_Tx_Done(Tx_Done)
```

```
67          );
68
69      /*----------------------------------MEM-----------------------------*/
70      // instruction memory
71      reg [15:0] addr0;
72      reg rd_en0;
73      reg wr_en0;
74      wire [31:0] rdata0;
75      reg [31:0] wdata0;
76
77      mem #(.MEM_SIZE(MEM_SIZE)) mem0 (
78          .clk(clk),
79          .addr(addr0),
80          .rd_en(rd_en0),
81          .wr_en(wr_en0),
82          .rdata(rdata0),
83          .wdata(wdata0)
84          );
85
86
87      // data memory
88      reg [15:0] addr1;
89      reg rd_en1;
90      reg wr_en1;
91      wire [31:0] rdata1;
92      reg [31:0] wdata1;
93
94      mem #(.MEM_SIZE(MEM_SIZE)) mem1 (
95          .clk(clk),
96          .addr(addr1),
97          .rd_en(1'b1),
98          .wr_en(wr_en1),
99          .rdata(rdata1),
100         .wdata(wdata1)
101         );
102
103     reg [1:0] invert_state;
104
105     /*---------------------------------MEM Control--------------------------*/
```

```
106
107     reg [2:0] byte_cnt;
108     reg [31:0] word;
109     //reg recv_done;
110     reg [31:0] cntByteTime;
111     //reg send_done;
112
113     always@(posedge clk)begin
114         if(rst)begin
115             addr0 <= 16'd0;
116             rd_en0 <= 1'b0;
117             wr_en0 <= 1'b0;
118             wdata0 <= 32'd0;
119             addr1 <= 16'd0;
120             rd_en1 <= 1'b0;
121             wr_en1 <= 1'b0;
122             wdata1 <= 32'd0;
123             byte_cnt <= 3'd0;
124             word <= 32'd0;
125             recv_done <= 1'b0;
126             cntByteTime <= 32'd0;
127
128             Tx_DV <= 1'b0;
129             Tx_Byte <= 8'd0;
130             send_done <= 1'b0;
131
132             invert_state <= 2'b11;
133         end
134         else if (~invert)
135         begin
136             // uart to memory
137             if(Rx_DV)begin
138                 // receive a word = 4Byte
139                 if(byte_cnt == 3'd3)begin
140                     byte_cnt <= 3'd0;
141
142                     // receive instruction
143                     if(addr0 < MEM_SIZE)begin
144                         addr0 <= addr0+1'b1;
```

```
145                         wr_en0 <= 1'b1;
146                         wr_en1 <= 1'b0;
147                         wdata0 <= {Rx_Byte,word[23:0]};
148                     end
149                     // receive data
150                     else begin
151                         if(addr1 < MEM_SIZE)begin
152                             addr1 <= addr1+1'b1;
153                             wr_en0 <= 1'b0;
154                             wr_en1 <= 1'b1;
155                             wdata1 <= {Rx_Byte,word[23:0]};
156 
157                         end
158                     end
159                 end
160                 else begin
161                     byte_cnt <= byte_cnt+1'b1;
162 
163                     if(byte_cnt==3'd0) word[7:0] <= Rx_Byte;
164                     else if(byte_cnt==3'd1) word[15:8] <= Rx_Byte;
165                     else if(byte_cnt==3'd2) word[23:16] <= Rx_Byte;
166                     else;
167 
168                     wr_en0 <= 1'b0;
169                     wr_en1 <= 1'b0;
170                 end
171 
172 
173             end
174             else begin
175                 wr_en0 <= 1'b0;
176                 wr_en1 <= 1'b0;
177 
178                 if(addr1 == MEM_SIZE && recv_done == 1'b0 && mem2uart==1'b0)begin
    // receive done
179                     recv_done <= 1'b1;
180                     addr1 <= 16'd0;
181                     byte_cnt <= 3'd0;
182                 end
```

```
183             end
184
185             // memory to uart
186             if(mem2uart==1'b1)begin
187                 if(cntByteTime == CLKS_PER_BIT*20 && send_done==1'b0)begin  // 1Byte time
188                     cntByteTime <= 32'd0;
189
190                     if(addr1 != 16'd0) Tx_DV <= 1'b1;
191                     if(byte_cnt==3'd0) Tx_Byte <= rdata1[7:0];
192                     else if(byte_cnt==3'd1) Tx_Byte <= rdata1[15:8];
193                     else if(byte_cnt==3'd2) Tx_Byte <= rdata1[23:16];
194                     else if(byte_cnt==3'd3) Tx_Byte <= rdata1[31:24];
195                     else;
196
197                     if(byte_cnt == 3'd3)begin
198                         byte_cnt <= 3'd0;
199
200                         if(addr1 < MEM_SIZE)begin
201                             addr1 <= addr1+1'b1;
202                         end
203                         else begin
204                             send_done <= 1'b1;
205                         end
206
207                     end
208                     else begin
209                         byte_cnt <= byte_cnt+1'b1;
210                     end
211
212                 end
213                 else begin
214                     cntByteTime <= cntByteTime+1'b1;
215                     Tx_DV <= 1'b0;
216                 end
217             end
218         end
219         else begin
220             wdata1 <= ~rdata1;
221             case(invert_state)
```

```
                2'b00: begin
                    wr_en1 <= 1'b0;
                    invert_state <= 2'b01;
                end
                2'b01: begin
                    wr_en1 <= 1'b1;
                    invert_state <= 2'b10;
                end
                2'b10: begin
                    wr_en1 <= 1'b1;
                    if(addr1 == MEM_SIZE && recv_done == 1'b0) begin  // invert done
                        recv_done <= 1'b1;
                    end
                    invert_state <= 2'b11;
                end
                2'b11: begin
                    addr1 <= addr1 + 1'b1;
                    wr_en1 <= 1'b0;
                    invert_state <= 2'b00;
                end
            endcase
        end
    end
endmodule
```

3.4 取反测试模块

与先前测试模块基本一致，区别在于，接收完毕后不拉高 mem2uart，而是拉高 invert 开始取反，取反完成后拉高 mem2uart 开始输出。

```
`timescale 1ns/1ps
`define PERIOD 10

module UART_MEM_invert_tb;
reg   clk;
reg   rst;
reg   mem2uart;
reg   invert;
wire  Rx_Serial;
```

```
10 wire  recv_done;
11 wire  send_done;
12 wire  Tx_Serial;
13
14 wire  Rx_DV;
15 wire  [7:0] Rx_Byte;
16 wire  Rx_Serial_R;
17
18 reg   Tx_DV;
19 reg   [7:0] Tx_Byte;
20 wire  Tx_Active;
21 wire  Tx_Done;
22 wire  Tx_Serial_T;
23
24 parameter CLKS_PER_BIT = 16'd100;
25 parameter MEM_SIZE = 2;
26
27 integer i, j;
28 reg [7:0] send_mem [8*MEM_SIZE:1];
29 reg [7:0] recv_mem [4*MEM_SIZE:1];
30 reg inverted;
31
32 assign Rx_Serial = Tx_Serial_T;
33 assign Rx_Serial_R = Tx_Serial;
34
35 UART_MEM #(.CLKS_PER_BIT(CLKS_PER_BIT), .MEM_SIZE(MEM_SIZE)) UART_MEM_inst
36          (.clk(clk),
37           .rst(rst),
38           .mem2uart(mem2uart),
39           .invert(invert),
40           .Rx_Serial(Rx_Serial),
41           .recv_done(recv_done),
42           .send_done(send_done),
43           .Tx_Serial(Tx_Serial)
44          );
45
46 uart_rx #(.CLKS_PER_BIT(CLKS_PER_BIT)) uart_rx_inst
47         (.i_Clock(clk),
48          .i_Rx_Serial(Rx_Serial_R),
```

```
          .o_Rx_DV(Rx_DV),
          .o_Rx_Byte(Rx_Byte)
        );

uart_tx #(.CLKS_PER_BIT(CLKS_PER_BIT)) uart_tx_inst
        (.i_Clock(clk),
          .i_Tx_DV(Tx_DV),
          .i_Tx_Byte(Tx_Byte),
          .o_Tx_Active(Tx_Active),
          .o_Tx_Serial(Tx_Serial_T),
          .o_Tx_Done(Tx_Done)
        );

initial begin
    $readmemh("send.txt", send_mem);
    i = 1;
    j = 1;
    clk = 1'b0;
    rst = 1'b1;
    mem2uart = 1'b0;
    invert = 1'b0;
        inverted = 1'b0;
end

initial fork
    forever
        #(`PERIOD/2) clk <= ~clk;
    #200 rst <= 0;
    #300 Tx_DV <= 1;
    #300 Tx_Byte <= send_mem[1];
    #300 i <= i + 1;
join

always @(posedge Tx_Done) begin
    if (i <= 8 * MEM_SIZE) begin
        Tx_Byte = send_mem[i];
        i = i + 1;
    end
    else begin
```

```
88          Tx_DV = 1'b0;
89      end
90  end
91
92  always @(posedge recv_done) begin
93      if (invert) begin
94          inverted <= 1'b1;
95                  invert <= 1'b0;
96          mem2uart <= 1'b1;
97                  rst <= 1'b1;
98                  #100 rst <= 1'b0;
99      end
100     else if(~inverted) begin
101         invert <= 1'b1;
102         rst <= 1'b1;
103         #100 rst <= 1'b0;
104     end
105 end
106
107 always @(posedge Rx_DV) begin
108     if (j <= 4 * MEM_SIZE) begin
109         recv_mem[j] = Rx_Byte;
110         if (j == 4 * MEM_SIZE) begin
111             $writememh("recv.txt", recv_mem);
112             $finish;
113         end
114     end
115     j = j + 1;
116 end
117
118 endmodule
```

4　仿真结果与分析

4.1　串口收发器

串口收发器的行为级仿真设置波特率 100k，存储器大小 512，仿真结果如图 4、5、6、7所示。输出的文件经简单脚本处理与理论输出文件对比，验证正确，如图 8所示。

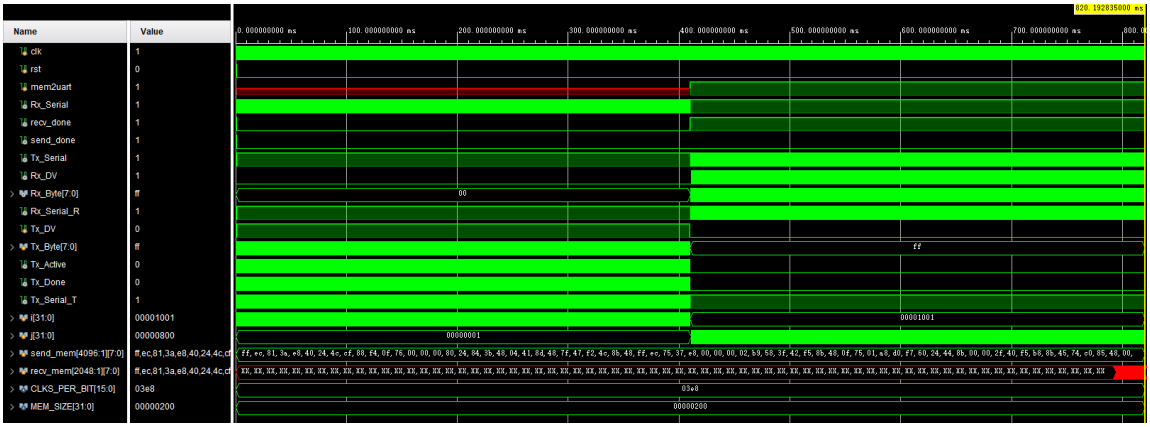

图 4: 串口收发器行为级仿真

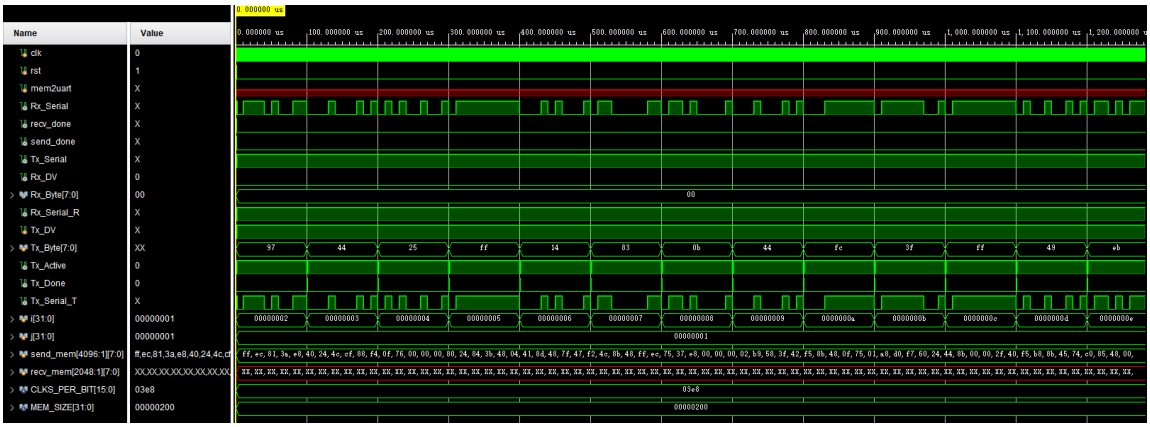

图 5: 串口收发器行为级仿真前部放大

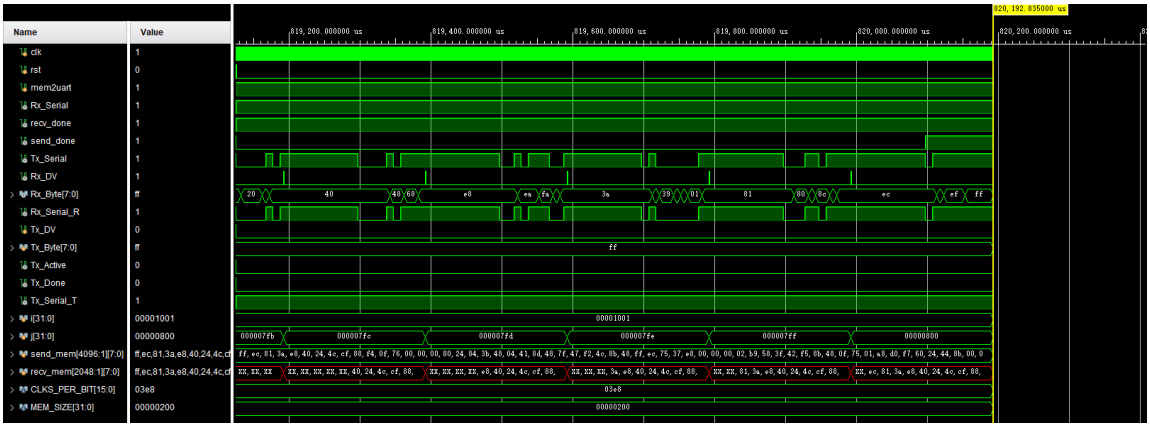

图 6: 串口收发器行为级仿真后部放大

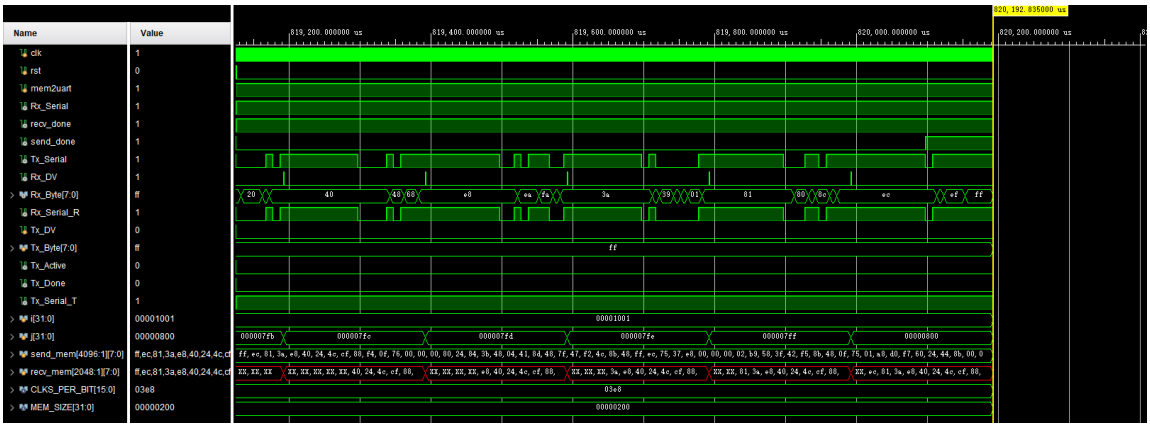

图 7: 串口收发器行为级仿真中部放大

```
$ tr -d '\r\n' < recvn.txt > recvl.txt && diff -i -w recv.txt recvl.txt
```

图 8: 仿真输出结果比较

实现后时序要求全部满足，如图 9所示。面积占用如图 10所示（此处资源情况没有修改波特率和存储器容量，仍为 9600 与 512），实现后由于指令存储器没有起到作用，因而实现时被忽略。

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.323 ns	Worst Hold Slack (WHS):	0.097 ns	Worst Pulse Width Slack (WPWS):	3.750 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:	457	Total Number of Endpoints:	457	Total Number of Endpoints:	244

All user specified timing constraints are met.

图 9: 串口收发器时序性能

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Bonded IOB (106)	BUFGCTRL (32)
UART_MEM	168	211	83	136	32	8	1
mem1 (mem)	32	32	8	0	32	0	0
uart_rx_inst (uart_rx)	41	33	20	41	0	0	0
uart_tx_inst (uart_tx)	21	22	8	21	0	0	0

图 10: 串口收发器面积占用

门级仿真由于运行时间较长，故采用较高的波特率与较小的数据量。波特率设为 1M，存储器容量均设为 2（总共传 4 个字，2 个进入指令寄存器，2 个进入数据寄存器）。实现后的功能仿真如图 11所示（时序仿真无法正确运行，但综合后的时序可以正确运行，原因不明）。读出的数据结果经比较也正确。

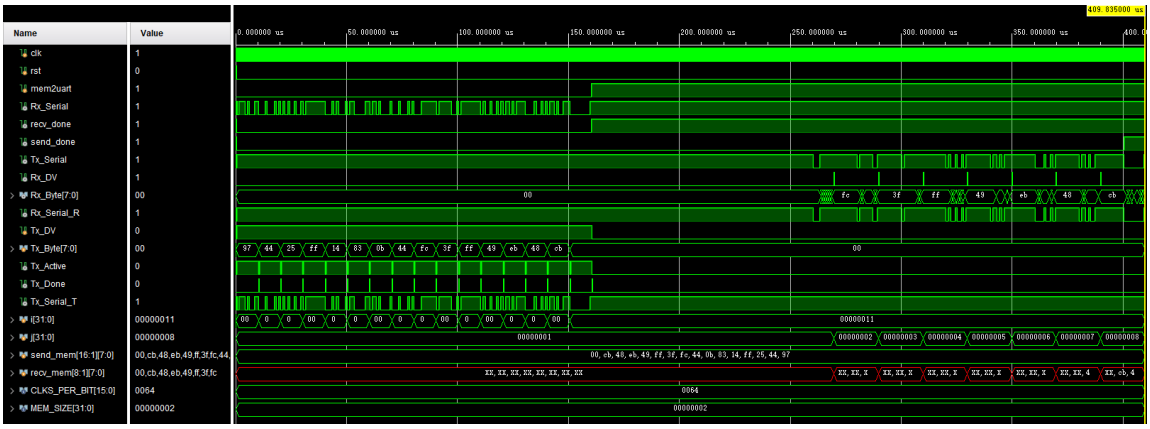

图 11: 串口收发器门级仿真

4.2 可取反串口收发器

实现后时序要求全部满足，如图 12所示。面积占用如图 13所示（此处资源情况没有修改波特率和存储器容量，仍为 9600 与 512）。

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.323 ns	Worst Hold Slack (WHS):	0.097 ns	Worst Pulse Width Slack (WPWS):	3.750 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:	457	Total Number of Endpoints:	457	Total Number of Endpoints:	244

All user specified timing constraints are met.

图 12: 可取反串口收发器时序性能

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Bonded IOB (106)	BUFGCTRL (32)
UART_MEM	168	211	83	136	32	8	1
mem1 (mem)	32	32	8	0	32	0	0
uart_rx_inst (uart_rx)	41	33	20	41	0	0	0
uart_tx_inst (uart_tx)	21	22	8	21	0	0	0

图 13: 可取反串口收发器面积占用

同上，将波特率设为 1M，存储器容量均设为 2。实现后的功能仿真（与先前相同，时序仿真无法正确运行，但综合后的时序可以正确运行）如图 14所示。写入数据存储器的数据为 FC、3F、FF、49、EB、48、CB、00，最终读出结果为 03、C0、00、B6、14、B7、34、FF，正确取反。

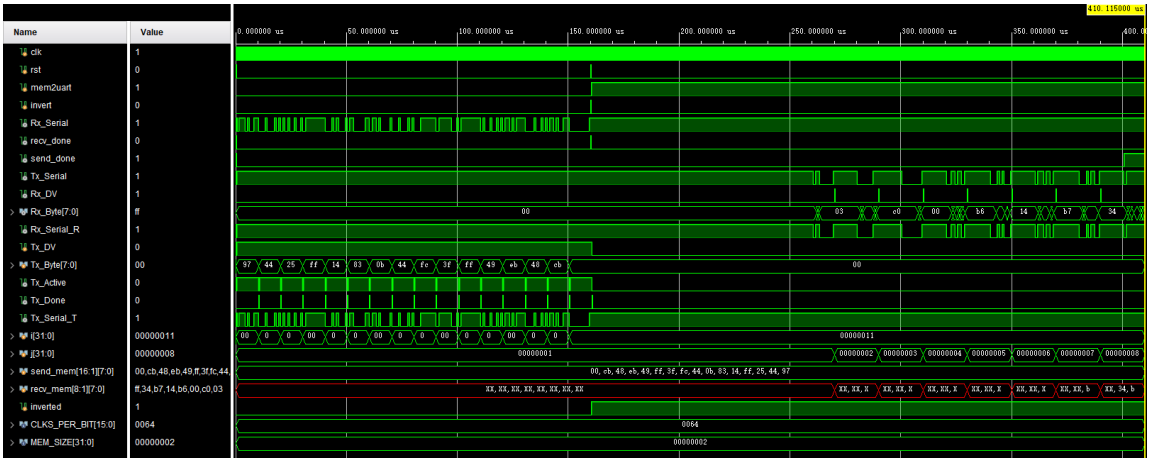

图 14: 带取反串口收发器门级仿真