基于FPGA的GMII与RGMII接口相互转换（包含源工程文件）

Xilinx相关原语讲解导航页面，想要了解更多原语，可以点击该链接跳转。

这段时间通过FPGA把ARP、ICMP、UDP协议全部通过FPGA实现了一遍，本来本文打算记录一下arp协议的，但在此之前应该先解决RGMII接口与GMII接口的转换问题。

经过前文讲解，开发板上使用的以太网PHY芯片是88R1518，原理图如下所示，留给用户的是RGMII双沿传输数据，时钟频率125MHz。在FPGA内部都是单沿处理数据，所以需要通过一个模块将外部输入的双沿信号转换为单沿信号，另一个模块可以把内部输出的单沿信号转换为双沿信号。

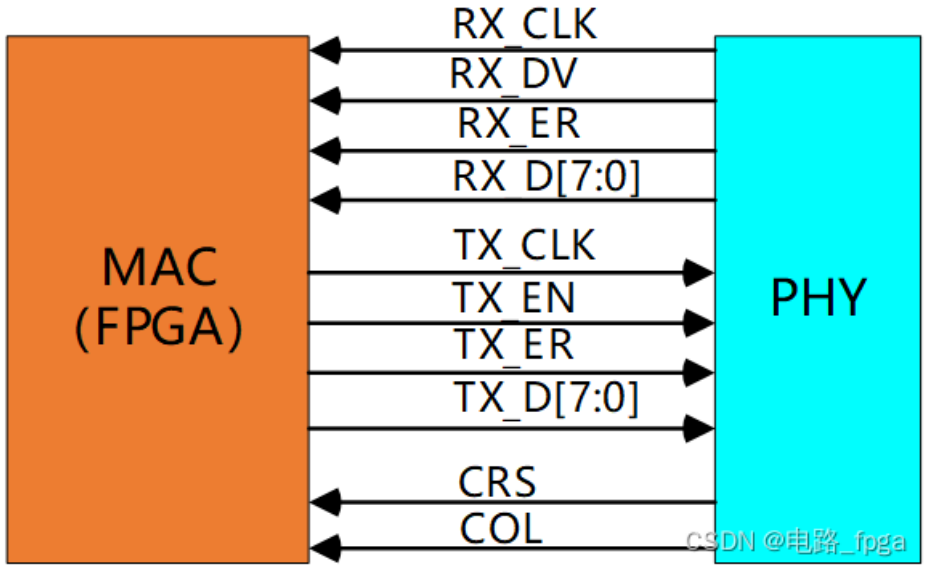

图1 RGMII接口信号

125MHz时钟下单沿传输数据的时序与GMII接口保持一致，双沿传输数据的时序与RGMII接口保持一致，所以需要设计一个GMII时序与RGMII时序转换的模块。GMII接口时序和RGMII接口时序前文已经进行详细讲述，此处只做简要回顾。

GMII与RGMII相互转换的信号流向如下图所示。

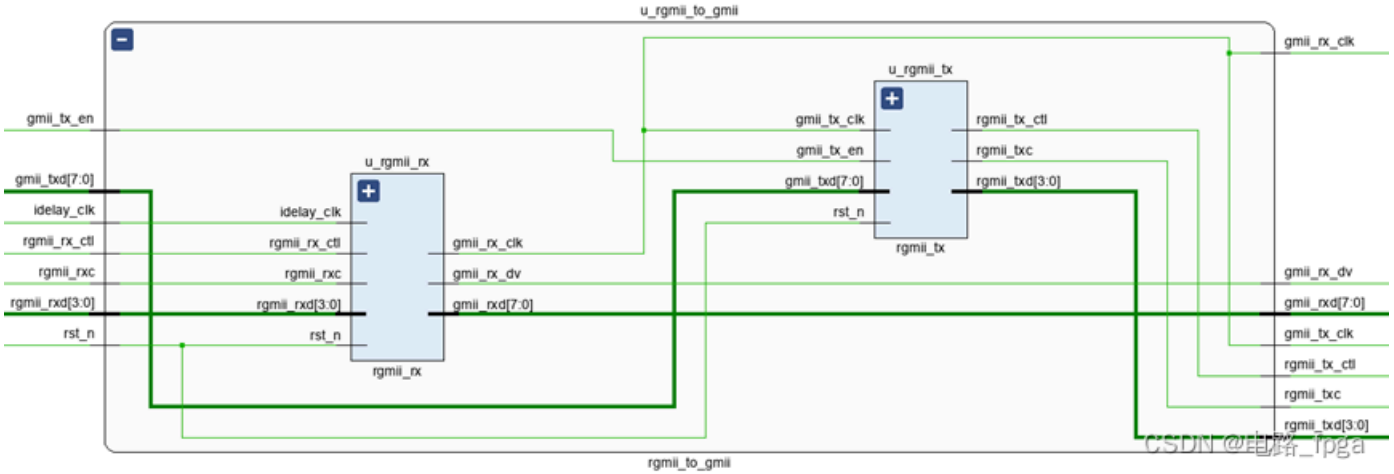

图2 gmii与rgmii转换

1、RGMII、GMII接口

FPGA的RGMII接口接收数据的时序如下图所示，在时钟的上升沿和下降沿均会传输4bit数据，整个时钟周期等价传输1Byte数据。

同时还有一个控制信号RX_CTL，在时钟上升沿输出RXD数据有效指示信号，下降沿输出数据有效指示信号异或错误信号，所以只有当RX_CTL在时钟上升沿和下降沿均为高电平时，RXD传输的数据才是有效的。

需要注意采集信号的时钟应该延后数据和控制信号2ns，让时钟的上升沿和下降沿与数据和控制信号的中部对齐，数据和控制信号处于稳定状态，采集时才能够减小错误。

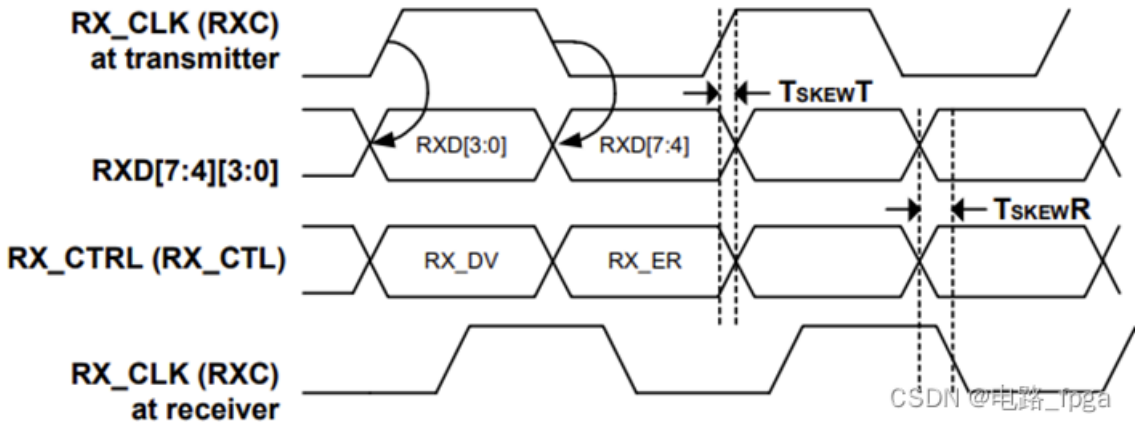

图3 RGMII接收数据时序图

而PHY芯片GMII接口接收数的时序如下图所示，数据只在时钟的上升沿输出，但是数据线是8bit的，所以每个时钟依旧传输一个字节数据。

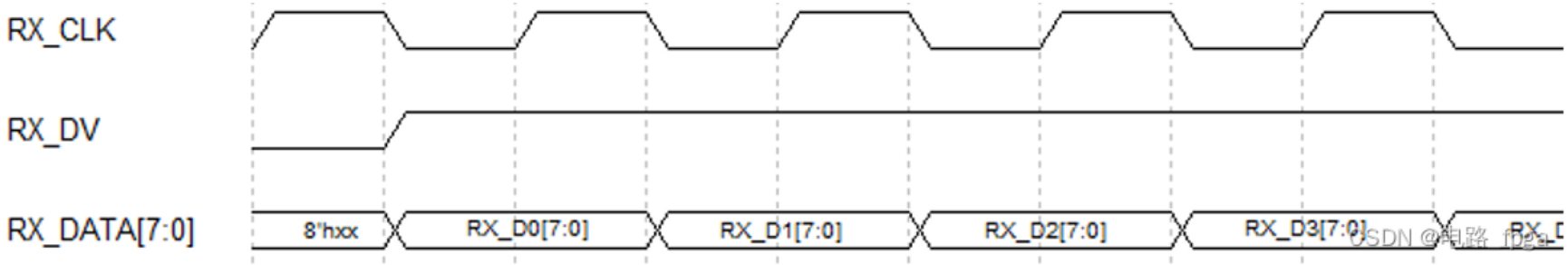

图4 GMII接收数据时序图

当FPGA接收到PHY芯片的RGMII时序时，通过一个模块转换为GMII时序，把转换后的GMII数据进行解码等操作。可以借助IDDR把在时钟双沿采集数据转换为在时钟上升沿采集数据，关于IDDR的使用已经在前文进行了详细讲解，本文不在赘述。

下图是FPGA通过RGMII接口发送时序，与发送时序相差不大，在时钟的双沿对数据和控制信号进行采集。FPGA输出的时钟TX_CLK应该滞后数据和控制信号变化2ns，使时钟的双沿与数据、控制信号的中部对齐。

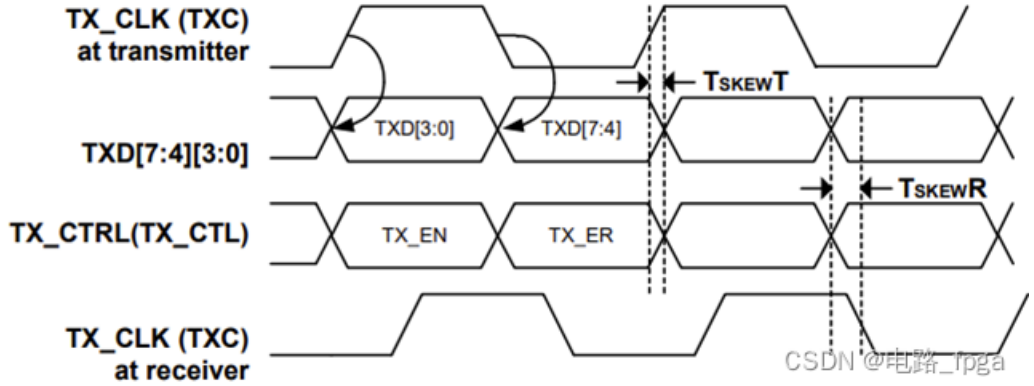

图5 RGMII发送时序

FPGA的GMII发送数据时序如下图所示，在时钟的上升沿输出数据和使能信号，每个时钟周期输出8位数据。

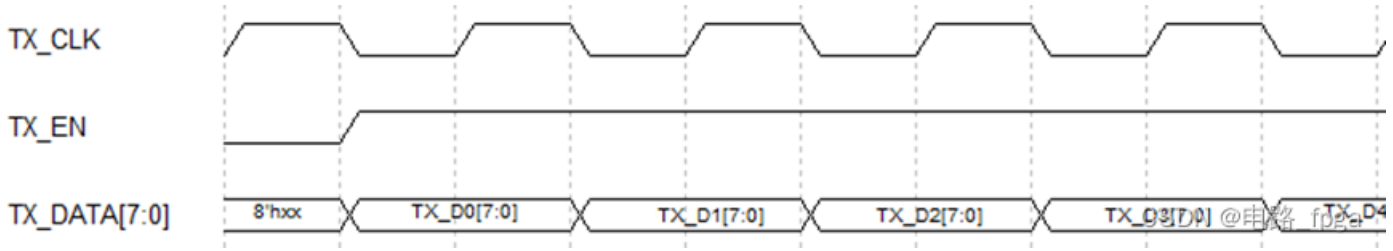

图6 GMII发送时序

FPGA内部通过GMII输出数据，然后通过一个模块把GMII时序转换为RGMII输出 FPGA芯片 ，GMII转RGMII模块主要借助ODDR实现。

图2需要把接收到的时钟信号延迟2ns，然后才能给FPGA作为时钟信号去采集数据和控制信号，延时时钟2ns可以使用IDELAYE2实现，但是一般PHY可以自己把时钟信号延迟2ns后输出，应该是便于CPU使用，此时FPGA就可以直接使用该信号作为时钟了，不需要在进行延时。

不同PHY芯片设置时钟延时的方式不同，YT8531C可以通过上拉引脚进行设置，而本文使用的88R1518是通过内部寄存器设置是否启用2ns时钟延时。

如下图所示，第2页第21号寄存器的bit5设置为1时，输出的时钟信号就是经过延时的，所以该芯片输出的时钟信号默认滞后数据和控制信号2ns，FPGA可以直接使用该时钟信号。

Table 109: MAC Specific Control Register 2
Page 2, Register 21

Bits	Field	Mode	HW Rst	SW Rst	Description
15	Reserved	RW	0x0	0x0	0
14	Copper Line Loopback	RW	0x0	0x0	1 = Enable Loopback of MDI to MDI 0 = Normal Operation
13	Default MAC interface speed (LSB)	RW	0x0	Update	Changes to these bits are disruptive to the normal operation; therefore, any changes to these registers must be followed by software reset to take effect. Also, used for setting speed of MAC interface during MAC side loopback. Requires that customer set both these bits and force speed using register 0 to the same speed. MAC Interface Speed during Link down. Bits 6,13 00 = 10 Mbps 01 = 100 Mbps 10 = 1000 Mbps
12:7	Reserved		0x20	0x20	Reserved.
6	Default MAC interface speed (MSB)	RW	0x1	Update	Changes to these bits are disruptive to the normal operation; therefore, any changes to these registers must be followed by software reset to take effect. Also, used for setting speed of MAC interface during MAC side loopback. Requires that customer set both these bits and force speed using register 0 to the same speed. MAC Interface Speed during Link down. Bits 6, 13 00 = 10 Mbps 01 = 100 Mbps 10 = 1000 Mbps
5	RGMII Receive Timing Control	RW	0x1	Update	Changes to these bits are disruptive to the normal operation; therefore, any changes to these registers must be followed by software reset to take effect. 1 = Receive clock transition when data stable 0 = Receive clock transition when data transitions
4	RGMII Transmit Timing Control	RW	0x1	Update	Changes to these bits are disruptive to the normal operation; therefore, any changes to these registers must be followed by software reset to take effect. 1 = Transmit clock internally delayed 0 = Transmit clock not internally delayed
3	Block Carrier	RW	0x0	Retain	1 = Enable Block Carrier Extension

	Extension Bit				0 = Disable Block Carrier Extension
2:0	Reserved	RW	0x6	0x6	Reserved.

图7 88R1518寄存器

上图中bit5为高电平时，88R1518会将接收到的时钟信号延时后作为采集数据和控制信号的时钟信号。所以在默认情况下FPGA输出的时钟信号沿与数据和控制信号变化沿对齐即可。

手册中可以查到，当21_2.5（第2页21号寄存器的第5位）为1时，PHY芯片发送时钟信号与数据信号的时序如下图所示，时钟沿滞后数据和控制信号的变化沿一段时间，与数据和控制信号的中部稳定部分对齐，可以直接在时钟沿采集数据和控制信号，这种是比较常用的。

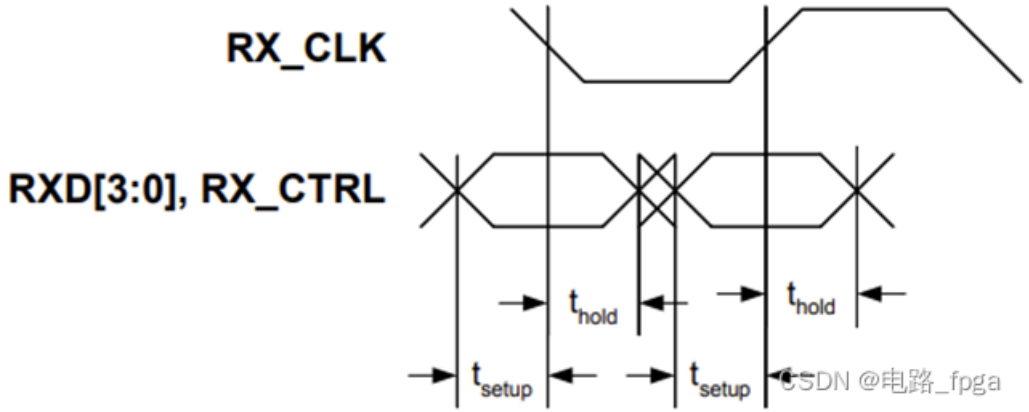

图8 88R1518发送时序

当21_2.5（第2页21号寄存器的第5位）为0时，PHY芯片发送时钟信号与数据信号的时序如下图所示，时钟沿与数据和控制信号的变化沿对齐，此时FPGA就不能直接利用该时钟采集数据，需要先将接收的时钟信号延迟2ns，也就是四分之一时钟周期，才能作为采集数据的时钟信号。

如果使用这种模式，那么FPGA内部就只能通过 原语　IDLAYE对时钟信号进行延时了，因此这种模式不是很常用。

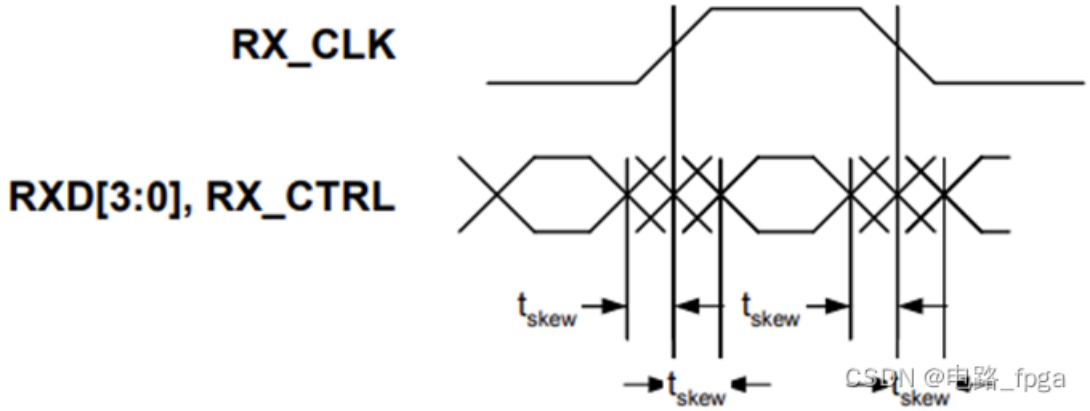

图9 88R1518发送时序

当21_2.4（第2页21号寄存器的第4位）为1时，要求接收时钟信号与数据信号的时序如下图所示，时钟沿与数据变化沿对齐。PHY芯片接收到时钟信号，会在内部把接收的时钟信号延时2ns后作为采集数据和控制信号的时钟信号。所以这个也是最常用的。

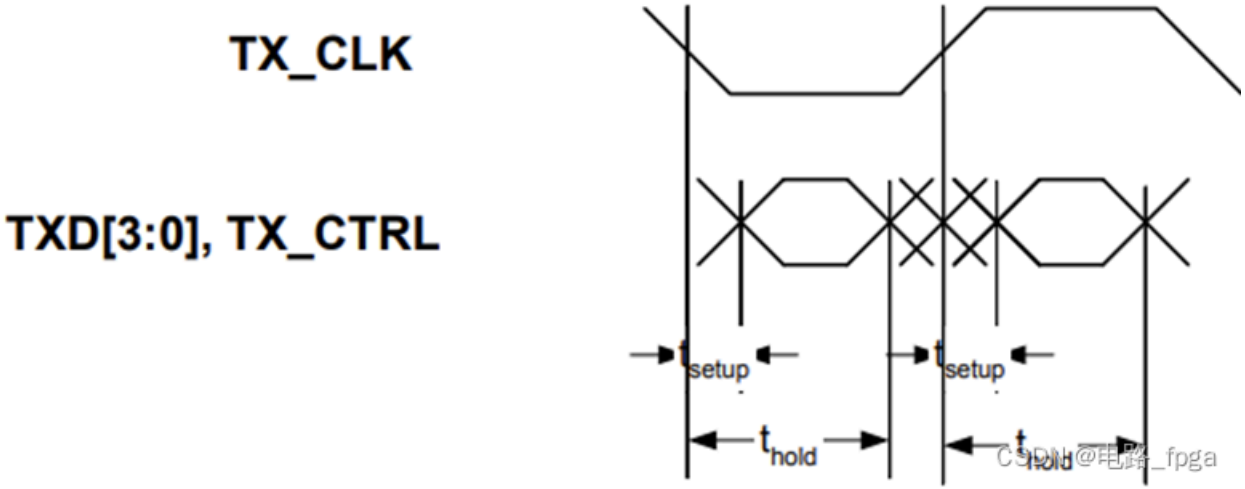

图10 88R1518接收时序

当21_2.4（第2页21号寄存器的第4位）为0时，PHY芯片就没有对接收时钟信号进行延时的功能，要求接收时钟信号与数据信号的时序如下图所示，时钟沿滞后数据变化2ns，这个延时在FPGA内部可以通过锁相环或ODELAYE(K7系列及其以上才有)实现，这个模式不常用。

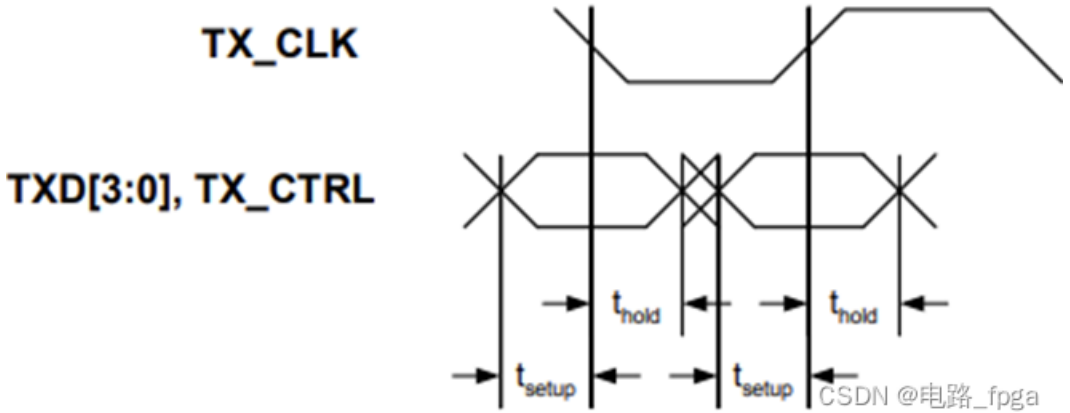

图11 88R1518接收时序

所以PHY芯片默认会把RX_CLK延时2ns后输出，同时也会将TX_CLK延时2ns作为采集TX_DATA、TX_CTL的时钟。FPGA内部不需要使用IDELAYE和ODELAYE结构，但是为了通用，本文还是会使用IDELAYE，只不过将延时系数设置为0而已，后续如果遇到没有延时功能PHY芯片也可以采用该设计思路。

2、RGMII转GMII

单时钟沿采集与双沿采集的转换是利用IDDR和ODDR原语实现，这两个原语的功能和参数含义在前面文章都有详细讲解和仿真，本文不在过多赘述。

输入端RGMII转GMII的框图如下所示，需要使用IDELAYE、IDELAYCTRL、IDDR、IOBUF、BUFG等原语。

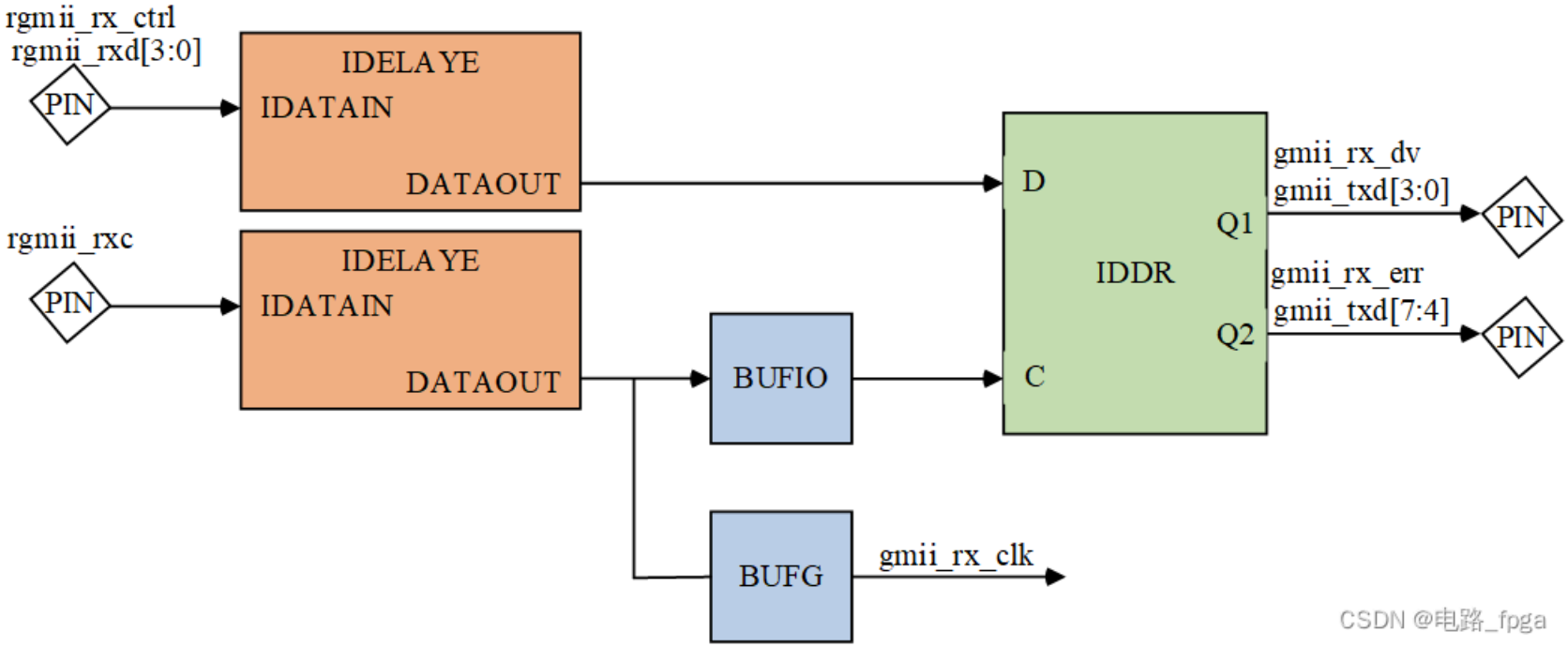

图12 RGMII转GMII框图

通过前文对IDELAYE的讲解可知，即使把延时参数设置为0，输出相对输入也会被延时600ps，这应该是因为信号要穿过IDELAYE所消耗的时间，为了使不加延时的情况下，数据与时钟对齐，故在数据和时钟信号路径上都添加了IDLAYE，rgmii_rx_ctrl和rgmii_rxd的延时系数固定为0，rgmii_rxc的延时系数根据需要延时的时间进行设置。

rgmii_rxc经过IDELAYE后分为两路，一路通过BUFIO去驱动IDDR的时钟端口，另一路经过BUFG驱动FPGA内部CLB和RAM等时序资源的时钟。从时钟管脚到BUFIO延时更小，但是只能驱动IOB中的时钟端口，而全局时钟缓冲器BUFG到达FPGA内部的IOB、CLB块RAM的时钟延迟和抖动非常小，为其他模块提供操作时钟。关于BUFIO和BUFG在FPGA内部的位置，在本文后面进行查看，对比就可以知道BUFIO的位置优势了。

上述原语前文均有详细讲解，此处不再赘述，下面直接给出该模块的核心代码：

```
    assign gmii_rx_clk = rgmii_rxc_bufg;//将时钟全局时钟网络驱动CLB等资源。

    //将时钟信号延时。
    IDELAYE2 #(
        .IDELAY_TYPE        ( "FIXED"           ),//FIXED,VARIABLE,VAR_LOAD,VAR_LOAD_PIPE
        .IDELAY_VALUE       ( IDELAY_VALUE      ),//Input delay tap setting (0-31)
        .REFCLK_FREQUENCY   ( 200.0             ) //IDELAYCTRL clock input frequency in MHz
    )
```

```
    u_delay_rxc (
        .CNTVALUEOUT        (                   ),//5-bit output: Counter value output
        .DATAOUT            ( rgmii_rxc_delay   ),//1-bit output: Delayed data output
        .C                  ( 1'b0              ),//1-bit input: Clock input
        .CE                 ( 1'b0              ),//1-bit input: enable increment/decrement
        .CINVCTRL           ( 1'b0              ),//1-bit input: Dynamic clock inversion
        .CNTVALUEIN         ( 5'b0              ),//5-bit input: Counter value input
        .DATAIN             ( 1'b0              ),//1-bit input: Internal delay data input
        .IDATAIN            ( rgmii_rxc         ),//1-bit input: Data input from the I/O
        .INC                ( 1'b0              ),//1-bit input: Inc/Decrement tap delay
        .LD                 ( 1'b0              ),//1-bit input: Load IDELAY_VALUE input
        .LDPIPEEN           ( 1'b0              ),//1-bit input: Enable PIPELINE register
        .REGRST             ( ~rst              ) //1-bit input: Active-high reset tap-delay
    );

    ///例化全局时钟资源。
    BUFG BUFG_inst (
        .I  ( rgmii_rxc_delay   ),//1-bit input: Clock input
        .O  ( rgmii_rxc_bufg    ) //1-bit output: Clock output
    );

    //全局时钟IO缓存
    BUFIO BUFIO_inst (
        .I  ( rgmii_rxc_delay   ),//1-bit input: Clock input
        .O  ( rgmii_rxc_bufio   ) //1-bit output: Clock output
    );

    //输入延时控制
    (* IODELAY_GROUP = "rgmii_rx" *)
    IDELAYCTRL  IDELAYCTRL_inst (
        .RDY    ( rst           ),//1-bit output: Ready output
        .REFCLK ( idelay_clk    ),//1-bit input: Reference clock input
        .RST    ( ~rst_n        ) //1-bit input: Active high reset input
    );

    //将输入控制信号和数据进行拼接，便于后面好用循环进行处理。
    assign din[4 : 0] = {rgmii_rx_ctl,rgmii_rxd};

    //rgmii_rx_ctl和rgmii_rxd输入延时与双沿采样
    generate

```

```
        genvar i;
        for(i=0 ; i<5 ; i=i+1)begin : RXDATA_BUS
            //输入延时
            (* IODELAY_GROUP = "rgmii_rx" *)
            IDELAYE2 #(
                .IDELAY_TYPE        ( "FIXED"           ),//FIXED,VARIABLE,VAR_LOAD,VAR_LOAD_PIPE
                .IDELAY_VALUE       ( 0                 ),//Input delay tap setting (0-31)
                .REFCLK_FREQUENCY   ( 200.0             ) //IDELAYCTRL clock input frequency in MHz
            )
            u_delay_rxd (
                .CNTVALUEOUT        (                   ),//5-bit output: Counter value output
                .DATAOUT            ( din_delay[i]      ),//1-bit output: Delayed data output
                .C                  ( 1'b0              ),//1-bit input: Clock input
                .CE                 ( 1'b0              ),//1-bit input: enable increment/decrement
                .CINVCTRL           ( 1'b0              ),//1-bit input: Dynamic clock inversion
                .CNTVALUEIN         ( 5'b0              ),//5-bit input: Counter value input
                .DATAIN             ( 1'b0              ),//1-bit input: Internal delay data input
                .IDATAIN            ( din[i]            ),//1-bit input: Data input from the I/O
                .INC                ( 1'b0              ),//1-bit input: Inc/Decrement tap delay
                .LD                 ( 1'b0              ),//1-bit input: Load IDELAY_VALUE input
                .LDPIPEEN           ( 1'b0              ),//1-bit input: Enable PIPELINE register
                .REGRST             ( ~rst              ) //1-bit input: Active-high reset tap-delay
            );

            //输入双沿采样寄存器
            IDDR #(
                .DDR_CLK_EDGE   ( "SAME_EDGE_PIPELINED" ),//"OPPOSITE_EDGE", "SAME_EDGE" or "SAME_EDGE_PIPELINED"
                .INIT_Q1        ( 1'b0                  ),//Initial value of Q1: 1'b0 or 1'b1
                .INIT_Q2        ( 1'b0                  ),//Initial value of Q2: 1'b0 or 1'b1
                .SRTYPE         ( "SYNC"                ) //Set/Reset type: "SYNC" or "ASYNC"
            )
            u_iddr_rxd (
                .Q1             ( gmii_data[i]          ),//1-bit output for positive edge of clock
                .Q2             ( gmii_data[5 + i]      ),//1-bit output for negative edge of clock
                .C              ( rgmii_rxc_bufio       ),//1-bit clock input rgmii_rxc_bufio
                .CE             ( 1'b1                  ),//1-bit clock enable input
                .D              ( din_delay[i]          ),//1-bit DDR data input
                .R              ( ~rst                  ),//1-bit reset
                .S              ( 1'b0                  ) //1-bit set
            );

```

```
    end
endgenerate

//通过拼接生成数据信号和数据有效指示信号。
assign gmii_rxd = {gmii_data[8:5],gmii_data[3:0]};
assign gmii_rx_dv = gmii_data[4] & gmii_data[9];//只有当上升沿和下降沿采集到的控制信号均为高电平时，数据才有效。
```

对应的Testbench在后文给出，与GMII转RGMII模块一起进行仿真，此处直接贴仿真结果，如下图所示，rgmii_rxc的时钟沿与rgmii_rx_ctl和rgmii_rxd的中部对齐，粉红色信号就是rgmii接口输入信号，天蓝色信号就是ODDR转换后信号。

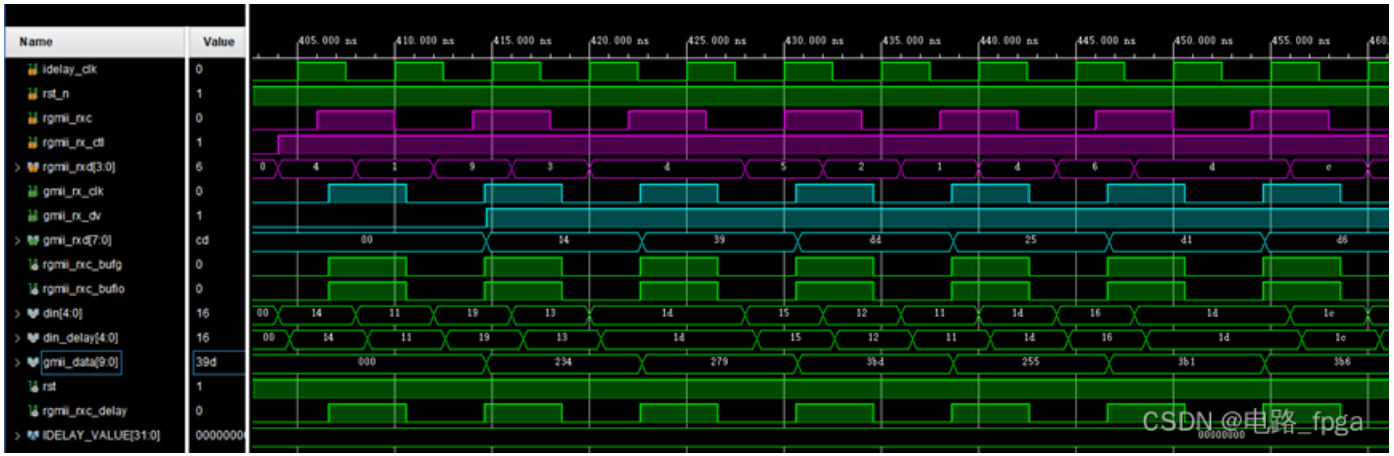

图13 rgmii转gmii时钟有延时仿真

使用TestBench2，即可对输入时钟无延时的RGMII接口信号进行仿真，仿真结果如下图所示。rgmii_rxc的时钟沿与rgmii_rx_ctl和rgmii_rxd变化沿对齐，如下图粉红色信号。

rgmii_rxc的IDELAYE的延时参数设置为25，延时时间为25*78ps=1950ps≈2ns，延时后输出信号为下图橙色时钟，该时钟与rgmii_rx_ctl和rgmii_rxd中部对齐。该时钟经过BUFIO后驱动IDDR，RGMII转换后的信号是下图中天蓝色信号，分析可知，转换成功。

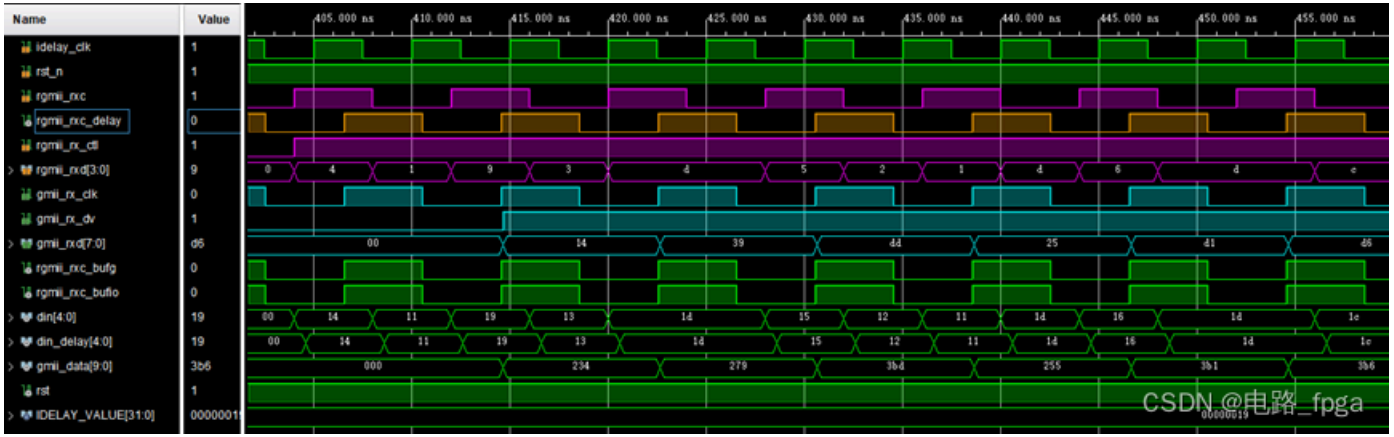

图14 rgmii转gmii时钟无延时仿真

RGMII转GMII模块的设计和仿真分析就到此结束了，后续以太网相关设计均会使用该模块。

3、GMII转RGMII

由于K7器件以下的器件不含ODELAYE，所以这里就不加该原语了。如果需要延时时钟，A7系列可以加锁相环将时钟延时90°实现，K7及以上加ODELAYE即可。

本文直接使用ODDR实现单沿传输转双沿传输，ODDR详细讲解可以参考前文。本文使用ODDR的SAME_EDGE模式，对应时序图如下所示，

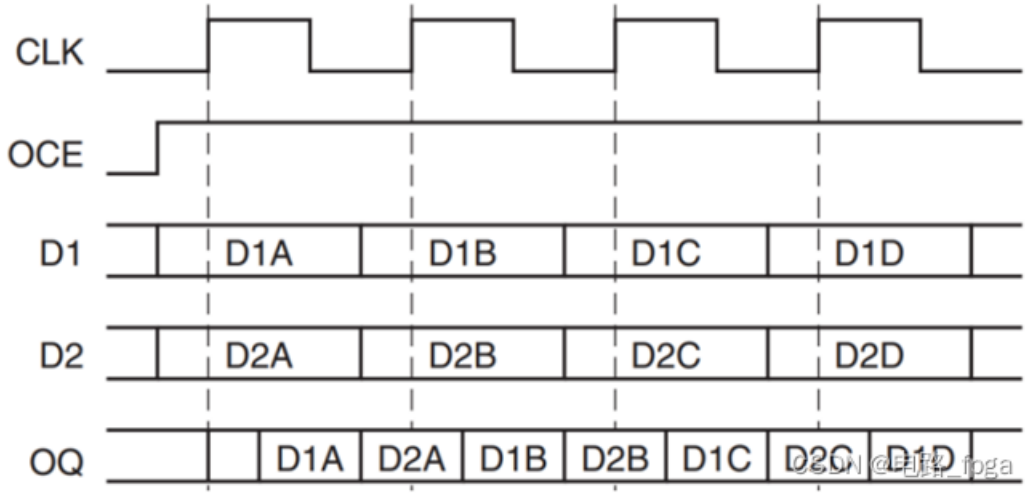

图15 ODDR SAME_EDGE模式时序图

输出时要把GMII时序转换为RGMII时序，转换的框图如下所示。

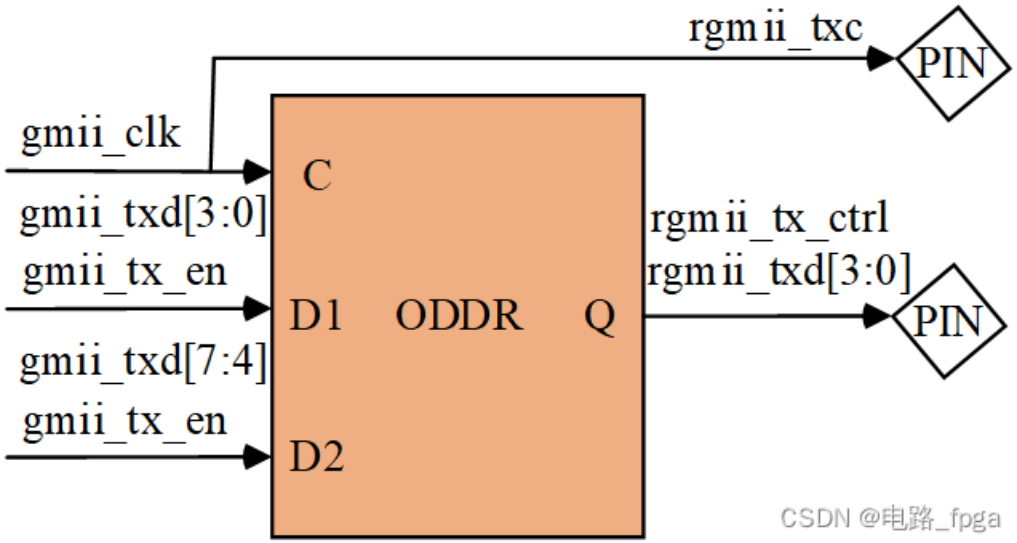

图16 GMII转RGMII框图

数据使能信号gmii_tx_en经过ODDR转换为双沿采样的rgmii_tx_ctrl，8位gmii_txd数据经过ODDR转换成双沿采样的4位rgmii_txd信号。

参考代码如下所示，为简化书写，代码采用generate for语句对ODDR进行多次例化。

```
    assign rgmii_txc = gmii_tx_clk;

    //输出双沿采样寄存器 (rgmii_tx_ctl)
    ODDR #(
        .DDR_CLK_EDGE  ( "SAME_EDGE"    ),//"OPPOSITE_EDGE" or "SAME_EDGE";
        .INIT          ( 1'b0           ),//Initial value of Q: 1'b0 or 1'b1;
        .SRTYPE        ( "SYNC"         ) //Set/Reset type: "SYNC" or "ASYNC";
    )
    ODDR_inst (
        .Q             ( rgmii_tx_ctl   ),//1-bit DDR output
        .C             ( gmii_tx_clk    ),//1-bit clock input
        .CE            ( 1'b1           ),//1-bit clock enable input
        .D1            ( gmii_tx_en     ),//1-bit data input (positive edge)
        .D2            ( gmii_tx_en     ),//1-bit data input (negative edge)
        .R             ( ~rst_n         ),//1-bit reset
        .S             ( 1'b0           ) //1-bit set
    );

```

```
generate
    genvar i;
    for(i=0; i<4; i=i+1)begin : TXDATA_BUS
        //输出双沿采样寄存器 (rgmii_txd)
        ODDR #(
            .DDR_CLK_EDGE  ( "SAME_EDGE"    ),//"OPPOSITE_EDGE" or "SAME_EDGE"
            .INIT          ( 1'b0           ),//Initial value of Q: 1'b0 or 1'b1
            .SRTYPE        ( "SYNC"         ) //Set/Reset type: "SYNC" or "ASYNC"
        )
        ODDR_inst (
            .Q             ( rgmii_txd[i]   ),//1-bit DDR output
            .C             ( gmii_tx_clk    ),//1-bit clock input
            .CE            ( 1'b1           ),//1-bit clock enable input
            .D1            ( gmii_txd[i]    ),//1-bit data input (positive edge)
            .D2            ( gmii_txd[4+i]  ),//1-bit data input (negative edge)
            .R             ( ~rst_n         ),//1-bit reset
            .S             ( 1'b0           ) //1-bit set
        );
    end
endgenerate
```

仿真结果如下所示，粉色信号是输入的GMII相关信号，黄色信号是转换后的RGMII输出信号。gmii_txd在时钟gmii_txc上升沿输入8’h14，rgmii_txd在时钟rgmii_txc的上升沿输出4’h4，在下降沿输出4’h1，实现双沿数据转单沿数据。

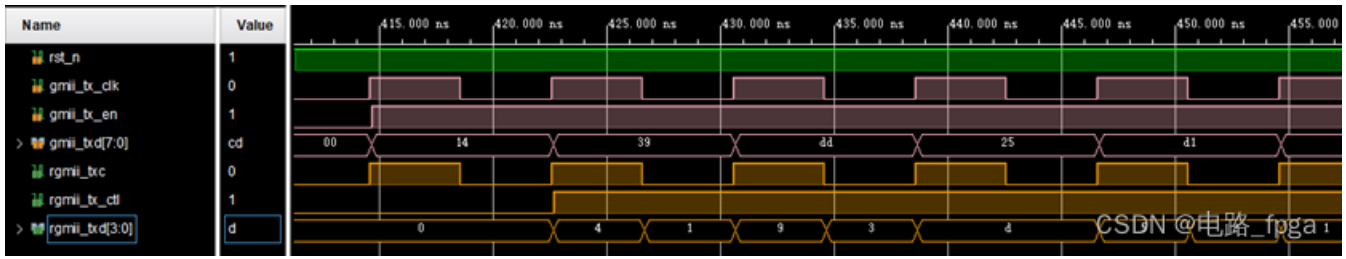

图17 gmii转rgmii仿真

前文用到的test.v分别如下所示。

```
`timescale 1 ns/1 ns
module test();
    localparam      CYCLE       =   10              ;//系统时钟周期，单位ns，默认10ns；
    localparam      PHY_CYCLE   =   8               ;//PHY芯片时钟周期，单位ns，默认8ns；
    localparam      IDELAY_VALUE=   0               ;//输入时钟延时(如果为n,表示延时n*78ps) ,设置为25时延时约2ns.
    localparam      RST_TIME    =   10              ;//系统复位持续时间，默认10个系统时钟周期；

    reg                             clk             ;//系统时钟，默认100MHz；
    reg                             rst_n           ;//系统复位，默认低电平有效；
    reg                             rgmii_rxc       ;
    reg             [3 : 0]         rgmii_rxd       ;
    reg                             rgmii_rx_ctl    ;
    reg                             rgmii_rxc_r     ;

    wire                            gmii_tx_clk     ;
    wire                            gmii_tx_en      ;
    wire            [7 : 0]         gmii_txd        ;
    wire                            rgmii_txc       ;
    wire                            rgmii_tx_ctl    ;
    wire            [3 : 0]         rgmii_txd       ;
    wire                            gmii_rx_clk     ;
    wire                            gmii_rx_dv      ;
    wire            [7 : 0]         gmii_rxd        ;

    assign gmii_tx_clk = gmii_rx_clk;
    assign gmii_tx_en = gmii_rx_dv;
    assign gmii_txd = gmii_rxd;

    //例化top模块；
    top #(.IDELAY_VALUE(IDELAY_VALUE))
    u_top (
        .clk            ( clk           ),//系统时钟信号，100MHz；
        .rst_n          ( rst_n         ),//系统复位，默认低电平有效；
        .rgmii_rxc      ( rgmii_rxc     ),//RGMII接收接口时钟信号；
        .rgmii_rxd      ( rgmii_rxd     ),//RGMII接收接口数据信号；
        .rgmii_rx_ctl   ( rgmii_rx_ctl  ),//RGMII接收接口数据有效指示信号；
        .rgmii_txc      ( rgmii_txc     ),//RGMII发送接口时钟输入信号；
        .rgmii_txd      ( rgmii_txd     ),//RGMII发送接口数据输入信号；
        .rgmii_tx_ctl   ( rgmii_tx_ctl  ),//RGMII发送接口数据输入有效指示信号；
        .gmii_tx_en     ( gmii_tx_en    ),//gmii的数据发送使能信号；
```

```
    .gmii_txd       ( gmii_txd      ),//gmii发送数据；
    .gmii_tx_clk    ( gmii_tx_clk   ),//gmii发送时钟；
    .gmii_rx_clk    ( gmii_rx_clk   ),//gmii接收时钟；
    .gmii_rx_dv     ( gmii_rx_dv    ),//gmii接收数据有效指示信号；
    .gmii_rxd       ( gmii_rxd      ) //gmii接收数据信号；
);

//生成周期为CYCLE数值的系统时钟；
initial begin
    clk = 0;
    forever #(CYCLE/2) clk = ~clk;
end

//用于生成PHY芯片的数据；
initial begin
    rgmii_rxc_r = 0;
    forever #(PHY_CYCLE/2) rgmii_rxc_r = ~rgmii_rxc_r;
end

//生成周期为PHY_CYCLE数值的PHY芯片时钟；
initial begin//延时2ns，使时钟沿与数据沿错开；
    #2;rgmii_rxc = 0;
    forever #(PHY_CYCLE/2) rgmii_rxc = ~rgmii_rxc;
end

//生成复位信号；
initial begin
    #1;
    rgmii_rx_ctl = 0;
    rgmii_rxd = 0;
    rst_n = 1;
    #2;
    rst_n = 0;//开始时复位10个时钟；
    #(RST_TIME*CYCLE);
    rst_n = 1;
    #(30*CYCLE);
    repeat(60)begin//生成60个4位随机数据作为测试；
        @(posedge rgmii_rxc_r);
        rgmii_rxd = {$random} % 16;
        rgmii_rx_ctl = 1'b1;
```

```
            #(PHY_CYCLE/2);
            rgmii_rxd = {$random} % 16;
        end
        @(posedge rgmii_rxc_r);
        rgmii_rx_ctl = 1'b0;
        $stop;//停止仿真；
    end

endmodule
```

```
`timescale 1 ns/1 ns
module test();
    localparam      CYCLE       =   10              ;//系统时钟周期，单位ns，默认10ns；
    localparam      PHY_CYCLE   =   8               ;//PHY芯片时钟周期，单位ns，默认8ns；
    localparam      IDELAY_VALUE=   25              ;//输入时钟延时(如果为n,表示延时n*78ps) ,设置为25时延时约2ns.
    localparam      RST_TIME    =   10              ;//系统复位持续时间，默认10个系统时钟周期；

    reg                             clk             ;//系统时钟，默认100MHz；
    reg                             rst_n           ;//系统复位，默认低电平有效；
    reg                             rgmii_rxc       ;
    reg             [3 : 0]         rgmii_rxd       ;
    reg                             rgmii_rx_ctl    ;

    wire                            rgmii_txc       ;
    wire                            rgmii_tx_ctl    ;
    wire            [3 : 0]         rgmii_txd       ;

    //例化top模块；
    top #(.IDELAY_VALUE(IDELAY_VALUE))
    u_top (
        .clk            ( clk           ),//系统时钟信号，100MHz；
        .rst_n          ( rst_n         ),//系统复位，默认低电平有效；
        .rgmii_rxc      ( rgmii_rxc     ),//RGMII接收接口时钟信号；
        .rgmii_rxd      ( rgmii_rxd     ),//RGMII接收接口数据信号；
        .rgmii_rx_ctl   ( rgmii_rx_ctl  ),//RGMII接收接口数据有效指示信号；
        .rgmii_txc      ( rgmii_txc     ),//RGMII发送接口时钟输入信号；
```

```
        .rgmii_txd      ( rgmii_txd     ),//RGMII发送接口数据输入信号；
        .rgmii_tx_ctl   ( rgmii_tx_ctl  ) //RGMII发送接口数据输入有效指示信号；
    );

    //生成周期为CYCLE数值的系统时钟；
    initial begin
        clk = 0;
        forever #(CYCLE/2) clk = ~clk;
    end

    //生成周期为PHY_CYCLE数值的PHY芯片时钟；
    initial begin//延时2ns，使时钟沿与数据沿错开；
        rgmii_rxc = 0;
        forever #(PHY_CYCLE/2) rgmii_rxc = ~rgmii_rxc;
    end

    //生成复位信号；
    initial begin
        #1;
        rgmii_rx_ctl = 0;
        rgmii_rxd = 0;
        rst_n = 1;
        #2;
        rst_n = 0;//开始时复位10个时钟；
        #(RST_TIME*CYCLE);
        rst_n = 1;
        #(30*CYCLE);
        repeat(60)begin//生成60个4位随机数据作为测试；
            @(posedge rgmii_rxc);
            rgmii_rxd = {$random} % 16;
            rgmii_rx_ctl = 1'b1;
            #(PHY_CYCLE/2);
            rgmii_rxd = {$random} % 16;
        end
        @(posedge rgmii_rxc);
        rgmii_rx_ctl = 1'b0;
        $stop;//停止仿真；
    end

endmodule
```

4、对综合后的信号分布进行分析

分配工程的相关引脚后，对工程进行综合、实现，然后打开器件布局布线图，找到rgmii_rxc时钟引脚，其时钟走线如下图所示。

白线就是该信号的走线，从管脚进入FPGA后，先经过PAD中的IBUF，然后经过IDELAYE模块进行延时，之后一部分经过BUFIO驱动IDDR时钟端口，另一部分需要通过BUFG进入全局时钟网络。

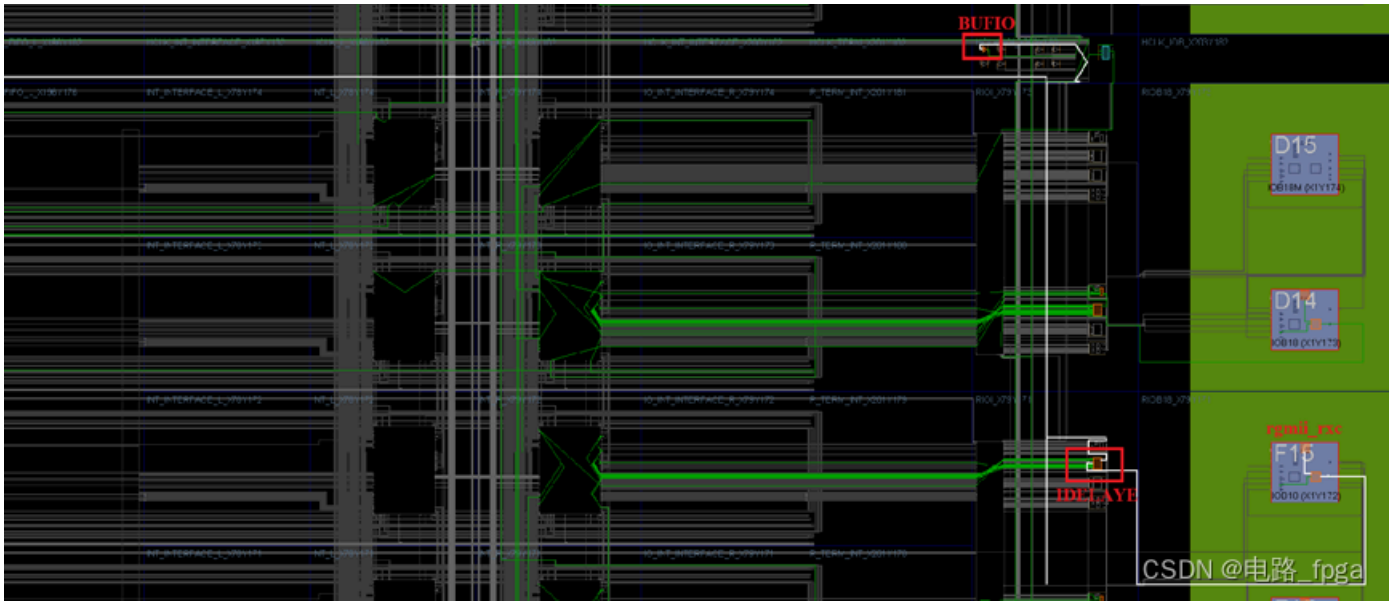

图18 rgmii_rxc时钟走线

从上图中可以看出BUFIO距离时钟管脚还是比较近的，上图看不到BUFG，是因为BUFG距离时钟管脚太远，无法截图，BUFG位置看下图，下图右下角就是BUFG的位置，而BUFIO和时钟管脚、IDDR这些都位于右上角的红框处，白色走线就是时钟rgmii_rxc到达BUFG的走线，走线长度相比BUFIO不知道是多少倍了，所以在采集数据时，一般都是直接使用BUFIO作为IDDR这些IOB时序器件的时钟信号。

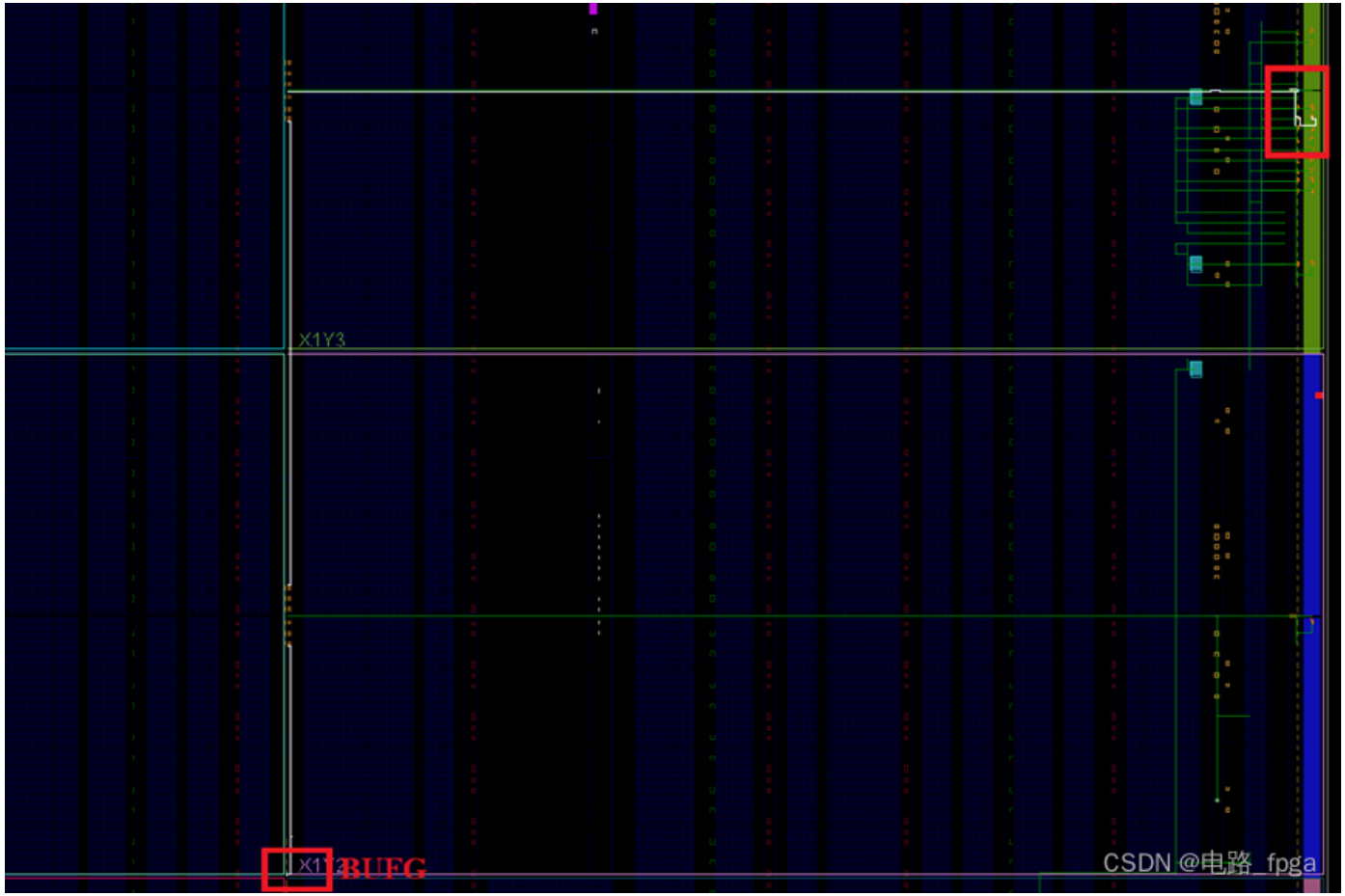

图19 rgmii_rxc到达BUFG的走线

把BUFG放大，如图20所示，前文在介绍BUFG是，FPGA上半部分是有16个BUFG的，缩小该图也是可以看到的，有兴趣的自己看。本工程使用了三个BUFG，一个是rgmii_rxc进入全局时钟网络，另外两个都是锁相环用到的，把100MHz转换为200MHz，锁相环的输出和反馈时钟均通过BUFG，因此也可以看出锁相环的延时还是较大的。

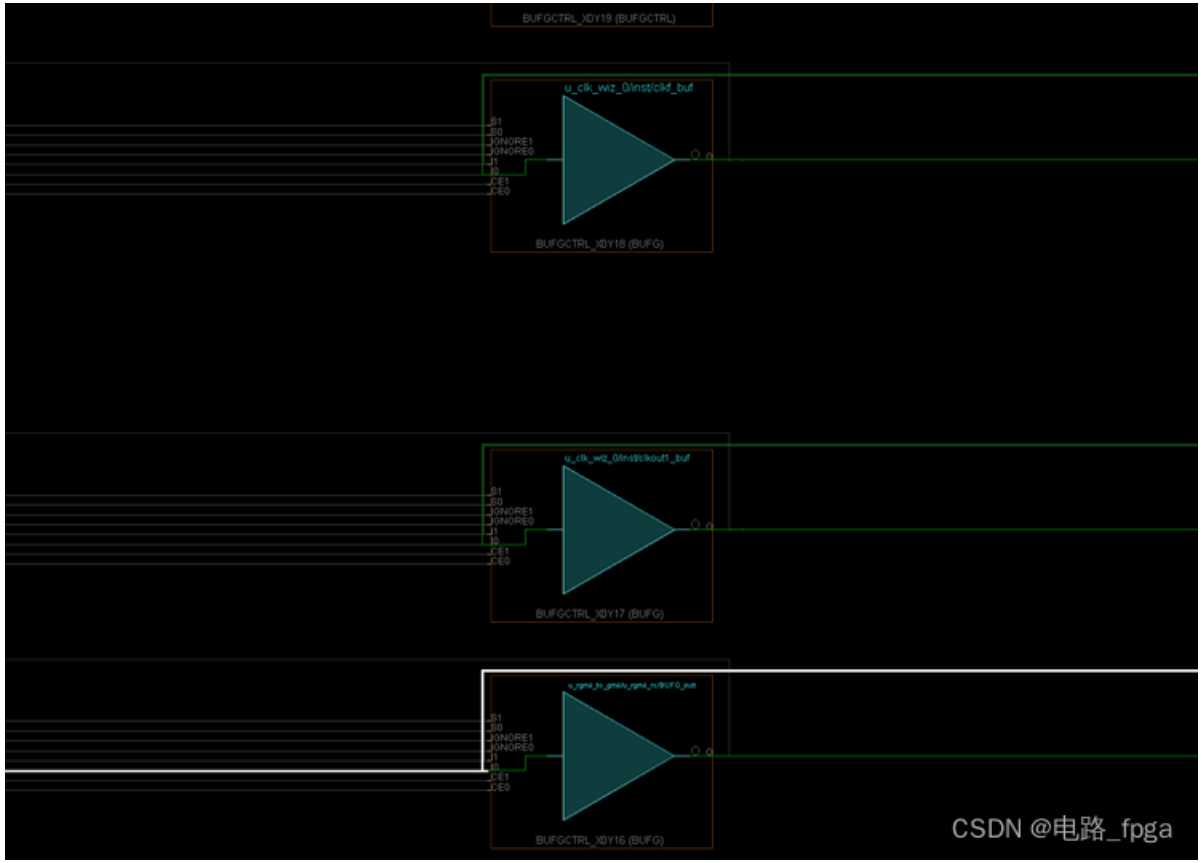

图20 放大BUFG

下图是查看BUFIO的输出，直接驱动IDDR的时钟端口，相比BUFG的输出就会快很多，IDDR的时钟和输入数据之间的延时就会小很多。

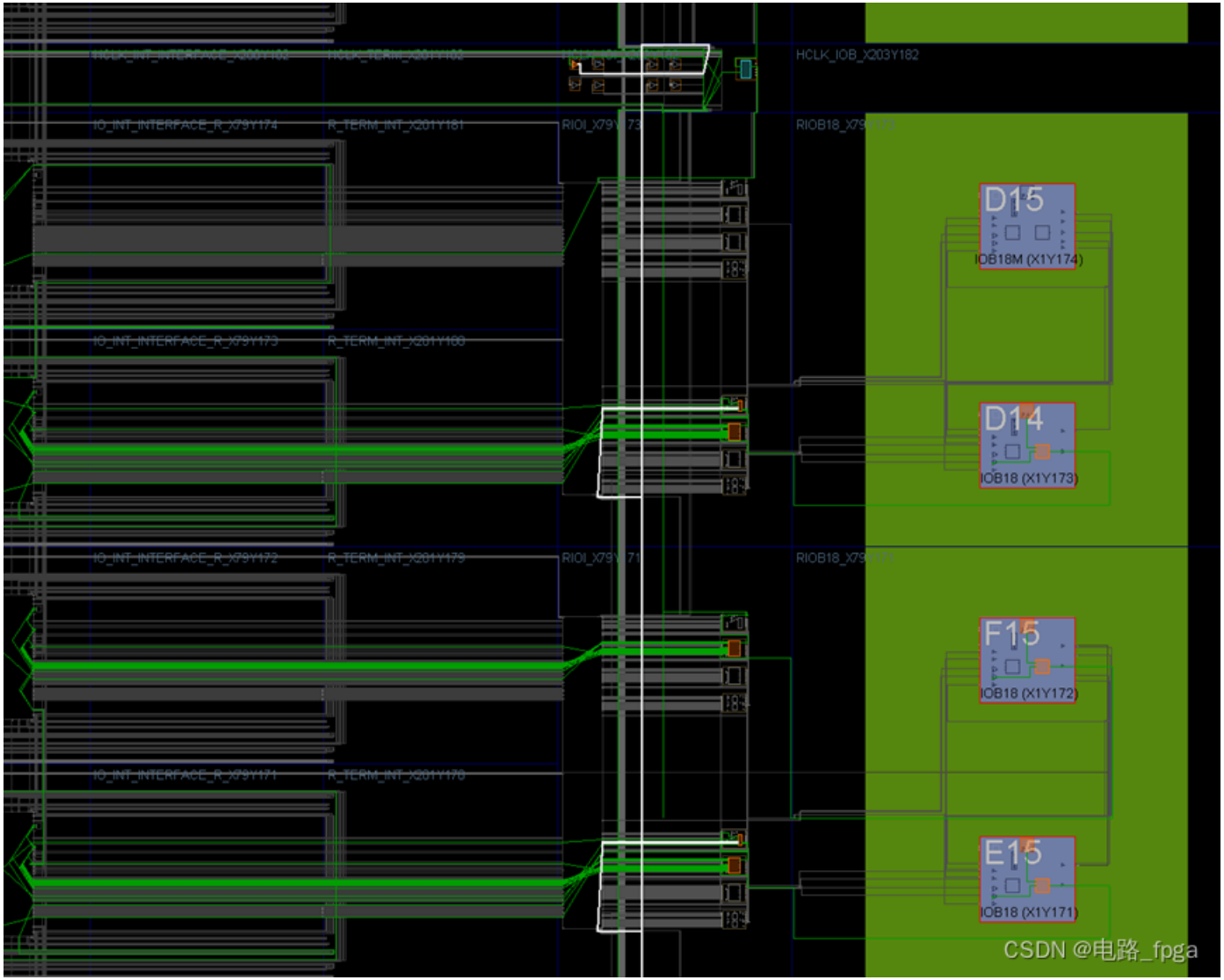

图21 BUFIO输出直接驱动IDDR时钟端口

本小节通过分析FPGA中BUFIO和BUFG的位置，间接分析为什么gmii_rxc会通过BUFIO驱动IDDR，而不是通过BUFG驱动IDDR。

5、总结

本文主要是实现GMII和RGMII接口的相互转换，因为在FPGA内部数据处理时，往往是单沿传输数据，所以需要通过一个模块把双沿传输的数据转换为单沿传输的数据，然后传输给FPGA内部模块进行处理。

主要使用前文详细讲解的一些原语，BUFIO、BUFG、IDDR、ODDR、IDELAYE这些单元在FPGA中都是实际存在的，可以通过vivado综合后的布局布线，来查看物理距离，进而确定为什么使用BUFIO，BUFG这些资源。

一般能够提供RGMII接口的PHY芯片，都能够实现时钟的延时，这是为了方便CPU、ARM使用的，因为CPU、ARM做时钟延时估计不怎么好处理。即使PHY芯片不能对时钟延时，FPGA也能够通过IDELAYE、ODELAYE原语对输入、输出的时钟进行延时。

在公众号后台回复”**gmii转rgmii**”（不包括引号）获取本文工程文件。

您的支持是我更新的最大动力！将持续更新工程，如果本文对您有帮助，还请多多点赞👍、评论💬和收藏⭐！