

CH569 HSPI 时序及 HSPI 简明 FPGA 示例

一、简述

CH569 的高速并行接口（High Speed Parallel Interface，后面简称 HSPI）支持 8 位、16 位和 32 位并行数据传输，最高传输速度可达 3.8Gbps。本文档内容主要包括 HSPI 的接收和发送时序以及简明的 FPGA 收发示例和解析，有助于快速搭建 CH569 与 FPGA 的传输平台。

二、HSPI 时序图

1. HSPI 接收时序:

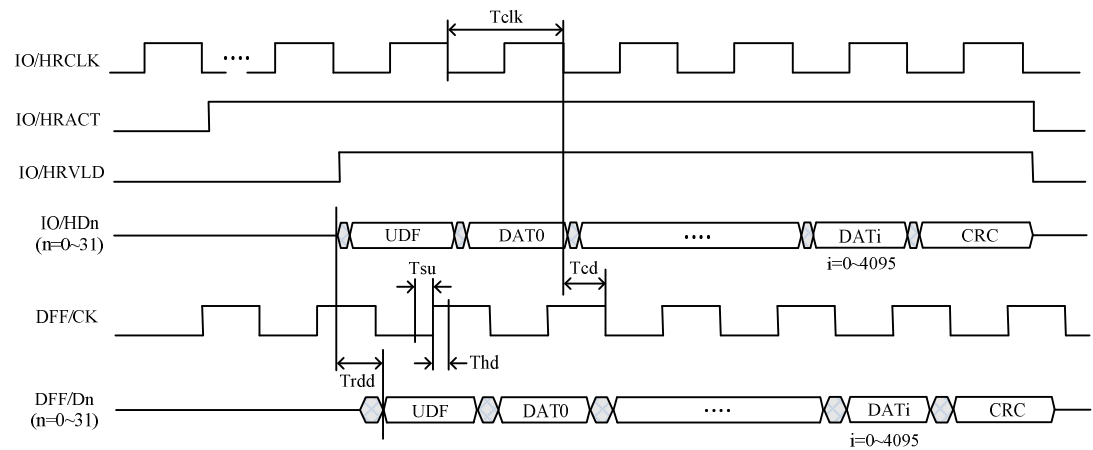


图 1. HSPI 接收时序图

上图为 HSPI 接收时序图，图中 IO/*表示 CH569 的 I/O 上的信号波形，DFF/*表示最终作用在内部触发器上的信号波形。

时间发面，时钟信号经路径延迟 Tcd 后达到触发器 CK 端，多位并行数据信号经路径延迟 Tdd 后达到稳定，可以被时钟信号正确采样。同时须满足触发器基本的建立时间 Tsd 和保持时间 Thd 要求，避免出现亚稳态。

数据方面，接收的内容依次为 32 比特的用户自定义字段、最多 4096 字节的有效数据和 CRC 字段。需要注意的是，在 8 位和 16 位模式下，对数据进行 CRC16 校验，CRC 字段为 2 字节的 CRC 校验结果，而 32 位模式下，对数据进行 CRC32 校验，CRC 字段为 4 字节的 CRC 校验结果

2. HSPI 发送时序:

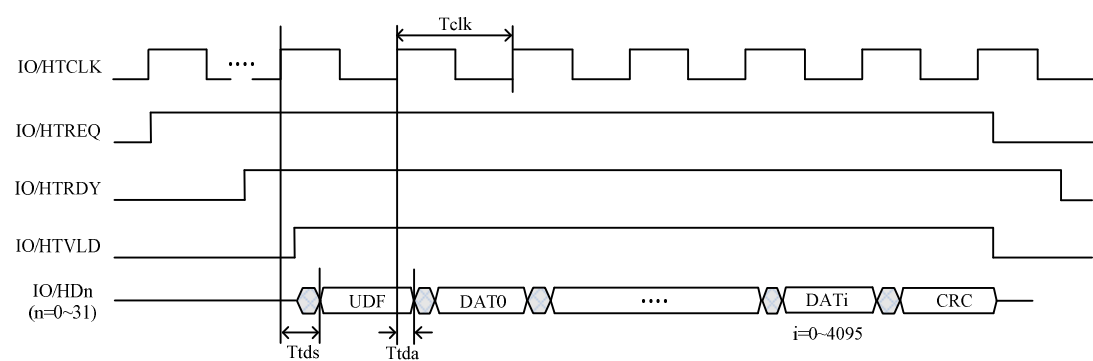


图 2. HSPI 发送时序图

上图为 HSPI 的发送时序图，内部触发器的时序不再赘述，主要关注时钟信号和数据信号经过不同路径后到达 CH569 的 I/O 的延迟。图中，时钟有效沿到数据稳定的延迟为 T_{tds} ，时钟有效沿对数据进行有效采样的裕量为 T_{tda} 。发送方亦可设置时钟极性进行数据发送。

3. 时间参数

参数	说明	最小值	最大值	单位
Tclk	接口时钟周期	8.33		ns
Tsu	触发器建立时间		0.2	ns
Thd	触发器保持时间		0.1	ns
Tcd	时钟信号路径延迟，包括 I/O 延迟、传输路径延迟和时钟树延迟等。		2.46	ns
Trdd	接收数据路径延迟包括 I/O 延迟和传输路径延迟。		2.92	ns
Ttda	发送数据有效采样裕量	1.757		ns
Ttds	时钟上升沿到数据稳定的延迟		2.5	ns

三、FPGA 程序实例及解析

附件程序 EXM_HSPI.v 为 HSPI 底层控制的 FPGA 实现，可为 FPGA 用户提供使用参考。该例程可与 CH569 的 HSPI 进行 32 位模式的数据发送与接收，8 位与 16 位模式可根据参考实例进行修改实现。下面将对程序进行解析，有助于开发实现。

1、接口定义

```
module EXM_HSPI (  
    //tx signal  
    HTCLK,  
    HTREQ,  
    HTRDY,  
    HTVLD,  
    HTD,  
    HTOE,  
    tx_act,  
    //rx signal  
    HRCLK,  
    HRACT,  
    HRVLD,  
    HTACK,  
    HRD,  
    //global signal  
    rstn,  
    clk,  
    HSPI_MODE,  
    dat_mod,  
    //RAM signal  
    ram_clk,  
    ram_csn,  
    ram_wen,  
    ram_addr,  
    ram_rdata,  
    ram_wdata  
);
```

EXM_HSPI 接口主要有 HSPI 收发接口、全局信号接口和 RAM 接口。其中，HSPI 接口与 CH569 引脚封装相匹配，可参考手册进行连接。全局信号接口为时钟、复位和一些配置信号，还有一些配置信息，在程序中以参数的形式做了定义，用户可根据自身设计进行参考和修改。RAM 接口只是简单的数据存储，用户可根据自身设计更换为相应的存储接口，或者是将数据经缓冲后通过其他外设接口进行转移。

2、发送冲突处理

```
always @(posedge clk) reg_rx_active_sync <= HRACT;          //sync HRACT

always @(posedge clk or posedge wire_rst) begin
    if (wire_rst) reg_tx_ack <= 1'b0;
    else if (reg_rx_active_sync) begin
        if (reg_tran_req) reg_tx_ack <= ~HSPI_MODE;        //up side first
        else reg_tx_ack <= 1'b1;
    end
    else reg_tx_ack <= 1'b0;
end

assign HTACK = reg_tx_ack;

always @(posedge clk or posedge wire_rst) begin            //tx request
    if (wire_rst) reg_tran_req <= 1'b0;
    else if (wire_tran_dat_end) reg_tran_req <= 1'b0;
    else if (tx_act & !reg_rx_active_sync) reg_tran_req <= 1'b1;
end
```

若对方已经发出发送请求时，则自身的发送触发将会被忽略，不再产生发送行为。如果出现上下两端同时发出发送请求的情况，则只有配置为上端的一方才能获得发送 **ready** 信号进行数据发送。

3、数据格式

```
assign wire_tran_hd_dat = {TX_LEN[1:0], reg_tran_num, HSPI_UDF}; //content of header
always @(*) begin //tx data select
    if (wire_tran_hd_sel) wire_send_data = wire_tran_hd_dat;    //tx UDF
    else if (wire_crc_sel) wire_send_data = wire_tx_crc_res32;  //tx CRC
    else wire_send_data = ram_rdata;                             //tx data payload
end
```

数据发送总共分为 3 段：依次为数据包头、数据负载和 CRC 字段的发送。其中，数据包头由发送长度值的最低 2 比特、4 比特发送序列号和 26 比特用户自定义字段构成。

用户自定义字段可用于应用扩展，用于自定义信息的传输。可填充上层的协议信息等内容，实现信息交互。

发送序列号可作为接收方判断是否产生丢包的判断依据，保证收发过程的连续性。

发送长度值的最低 2 比特用于接收方判断有效数据的总字节数。例如，以 32 位数据模式传输，但是传输长度非 4 字节对齐，则最后的 32 位数据中的有效字节数可以通过接收到的发送长度最低 2 位来判定。

4、接收状态信息

```
assign wire_dat32_mod = dat_mod[1] == 1'b1; //dat_mod = 2'b1x -> 32bits mode
assign wire_recv_check = !reg_rx_active & reg_rx_act_d1; //rx status check
always @(posedge HRCLK or posedge wire_reset_sync) begin
    if (wire_reset_sync) reg_rx_crc_err <= 1'b0;
    else if (wire_rx_end) reg_rx_crc_err <= 1'b0; //clear last CRC flag before new CRC result coming
    else if (wire_recv_check) begin
        if (wire_dat32_mod) reg_rx_crc_err <= wire_recv_crc_out32 != 32'hC704DD7B; //32bit CRC check
        else reg_rx_crc_err <= wire_recv_crc_res != 16'h800d; //8bit/16bits CRC check
    end
end

always @(posedge HRCLK or posedge wire_reset_sync) begin //tx and rx sequence match check
    if (wire_reset_sync) reg_num_mismatch <= 1'b0;
    else if (wire_recv_check) reg_num_mismatch <= reg_recv_num != reg_recv_udf[29:26];
end

always @(posedge HRCLK) reg_recv_check <= wire_recv_check;
always @(posedge HRCLK or posedge wire_reset_sync) begin //rx sequence count
    if (wire_reset_sync) reg_recv_num <= 4'd0;
    else if (reg_recv_check & !reg_num_mismatch & !reg_rx_crc_err) reg_recv_num <= reg_recv_num + 1'b1;
end
```

接收状态信息包括接收 CRC 校验结果、接收序列号匹配结果，二者直接体现传输的可靠性与正确性。

需要注意的是，上述程序中，在判断 CRC 校验是否正确时，根据数据模式配置的不同，

CRC 的比较值是不同的。在 8 位或 16 位模式下，采用的是 CRC16，最终的 CRC 结果应该是 16`h800D，否则判定 CRC 错误；在 32 位模式下，采用的是 CRC32，最终的 CRC 结果应该是 32`hC704DD7B，否则判定 CRC 错误。

当接收方判定 CRC 正确，并且收发序列号匹配时，接收序列号才会自增，否则保持不变，以此确保传输过程中不存在漏包的情况产生。