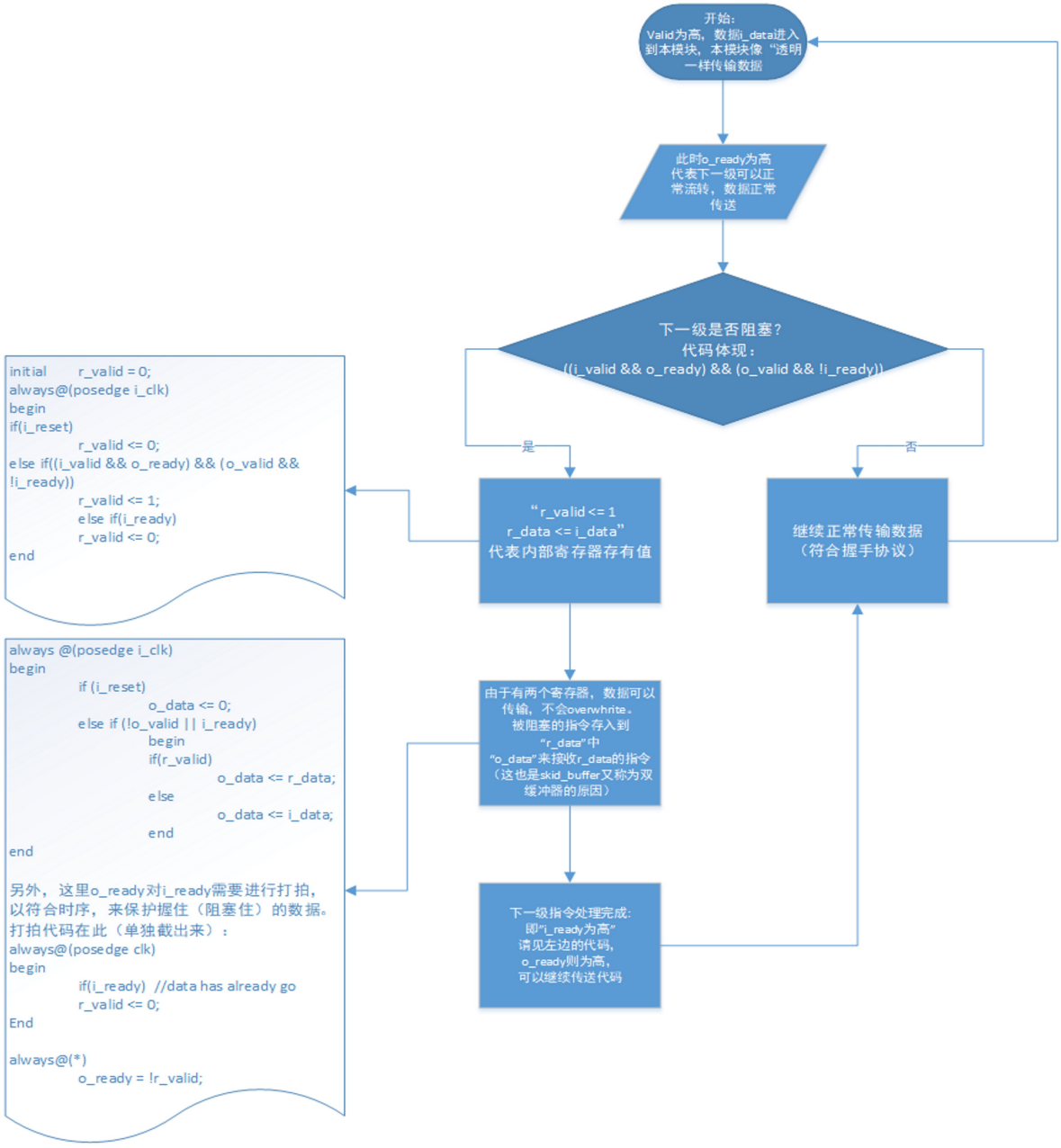
文字完整描述（全文皆由崔洋本人撰写）

（清晰的流程图单独列出，在本文件夹下）

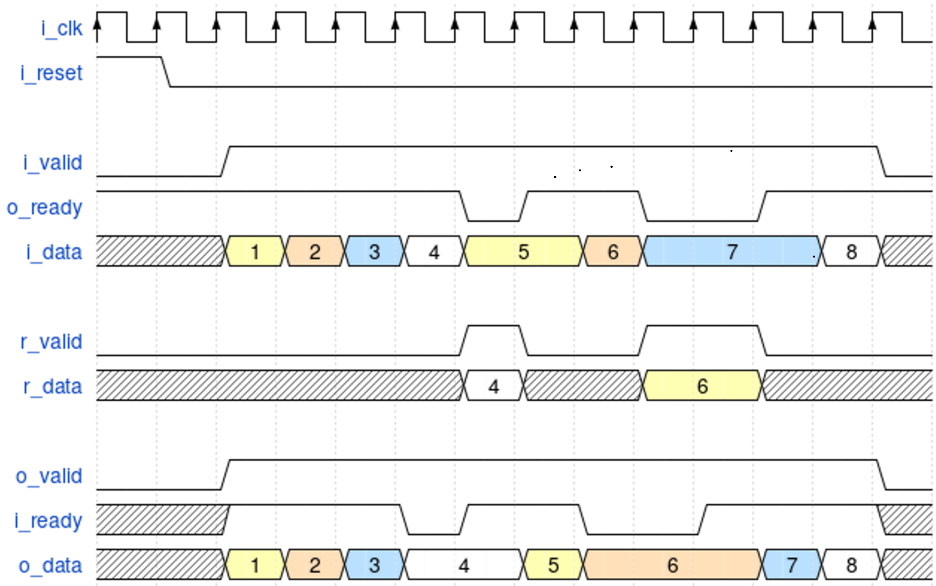
- 代码概述



- 怎样通过寄存器打拍，来满足数据传输的时序？
  - valid与data打拍情况

当下一级的模块还在处理指令，未准备好接收时，即i_ready此时为低电平，skid_buffer通过寄存器将data, valid握住，此时输出一直为寄存器存入的（老）data，无论本模块接收到的输入怎么样，都不变。此时时序图中，r_valid保持为高，且r_data为之前的数据。

这就是valid打拍。




```
/*
always@(posedge i_clk)
begin
if(i_reset)
        r_valid <= 0;
else if((i_valid && o_ready) && (o_valid && !i_ready))    //have incoming data, but the output     is stalled
        r_valid <= 1;
else if(i_ready)  //data has already go
        r_valid <= 0;
end
*/
```

而由于此时skid_buffer可以暂存两个数据（防滑缓存器又称为双缓冲器）。

- 如果只有一组寄存器的情况？！
  如果只有一组寄存器，会导致下一个传进来的数据将阻塞的数据复写，丢失了阻塞的数据！只有双缓冲器的情况下，才不会丢失数据，而是将阻塞住的数据握住，等待下一级处理完成

- ready打拍情况

```
else if(i_ready)  //data has already go
        r_valid <= 0;
end
always@(*)
        o_ready = !r_valid;
```

o_ready对i_ready打一拍，内置的寄存器存入上一个时钟周期的指令(r_data)。下一级阻塞的情况下，在两个周期内数据输入不会间断，满足时序要求

- 握手协议体现在哪里?
  观察时序图可知，当i_valid与o_ready不同时为高，则输入数据不刷新（刹车）。
  即只有当i_valid与o_ready同时为1，才能正常接收数据。

```
//////////////////////
always@(*)
        o_ready = !r_valid;

always @(posedge i_clk)
begin
        if (i_reset)
                o_data <= 0;
        else if (!o_valid || i_ready)        //reg_out //"!(o_valid && !i_ready)"
                begin
```

```
            if(r_valid)
                o_data <= r_data;
            else
                o_data <= i_data;
            end
end
```

r_valid在阻塞住的情况下保持为高，同时防滑寄存器不接受上一级指令。在这行代码中，o_ready接收取反后的r_valid信号，这是因为，o_ready要告诉上一级信号已阻塞