DUT 说明

DUT 本身是一个路由器，它的输入 din 为 16 位宽，请注意：每一位对应一个输入，即 din[0]的数据给图中的 00 输入端口，din[1]的数据给图中的 01 输入端口…din[15]的数据给图中的 15 号输入端口，（也就是说数据实际上是串行的，每次对相应的端接口只输入一位数据）。

同理，输出端口也是和 dout 中的每位对应。

请注意：DUT 模块代码中，所有带有_n 尾缀的信号均是低（0）有效。

每次开始传输时，首先应该通过 din 端口的特定位传输地址，地址宽度均为 4 位宽。如源地址 sa=3，目的地址 da 为 7，即通过 3 号输入端口给 7 号输出端口传输数据。

参考写法：

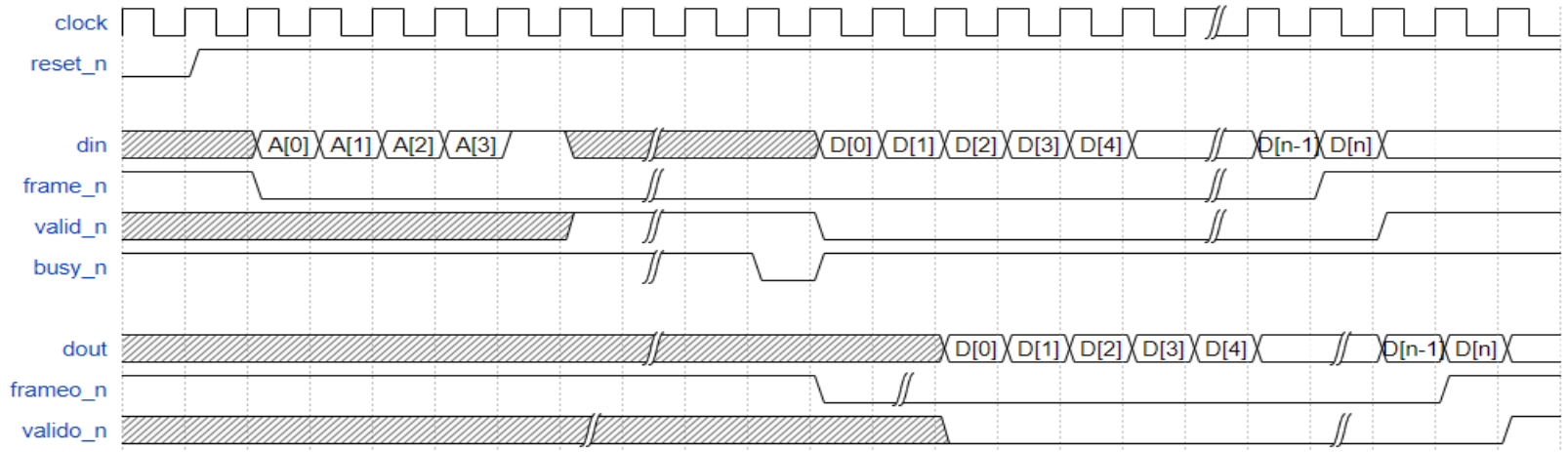
```
sa = 3;
da = 7;
payload.delete(); //clear previous data
repeat($urandom_range(4,2))
  payload.push_back($urandom);
```

请注意：每次开始传输，frame_n 同时拉低，处于有效状态，直到最后一个数据时，（即传输完地址，padding 数据，数据后），frame_n 才能再次拉高，期间必须一直为低。再连续的两次传输之前，frame_n 必须至少有一个周期的间隔是处于高电平的。

各信号名的一样如下表中所示：

信号名	意义
reset_n	复位信号，低有效
clock	输入的时钟信号
frame_n	当输入数据包时，本信号一定为低，当最后一个数据到达的同时，把 frame_n 拉高，暗示这是最后一个输入数据
valid_n	输入数据有效信号，低有效
din	输入数据包，含有地址，pad 以及数据
dout	输出数据
busy_n	当其为低时，说明当前连接处于 busy 状态
valido_n	输出数据有效信号，低有效
frameo_n	开始输出数据包时，本信号为低，当输出最后一个数据的同时，本信号拉高，暗示这是最后一个输出数据

本 DUT 的时序图如下所示：



必须明确：地址输入是 4 位宽的，数据串行输入，地址输入后，必须至少在下一个时钟周期输入一个 1，然后中间处于填充状态，这个状态的长度自定，然后开始串行发送数据。

必须明确：只有等到 busy_n 为高才能输入数据。

本次实验，我们不关注 DUT 内部是如何处理数据的，但必须明确：在 DUT 收到数据有效输入后，必须经过两个周期的延迟才会得到输出。

下面我们以 exercise1 中的内容具体讲解下本 DUT 的时序问题：

```
rtr_io.cb.frame_n[sa] <= 1'b0; //start of packet
for(int i=0; i<4; i++) begin
  rtr_io.cb.din[sa] <= da[i]; //i'th bit of da
  @rtr_io.cb;
end
```

开始传输，首先拉低 frame_n，发送 4 位地址信号。

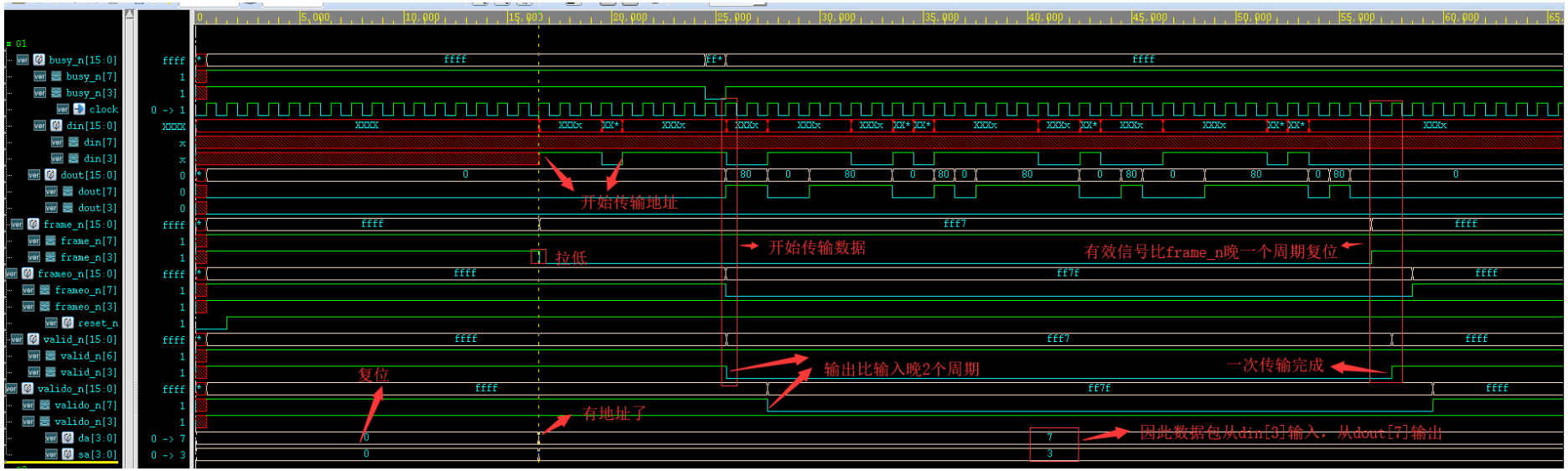
```
    //Define send_pad() task to drive some padding bits into router
task send_pad();

rtr_io.cb.frame_n[sa] <= 1'b0;
rtr_io.cb.din[sa] <= 1'b1;
rtr_io.cb.valid_n[sa] <= 1'b1;
repeat(5) @rtr_io.cb;

endtask: send_pad
```

发送填充数据，其中第一个数据必须为 1.

```
foreach(payload[index])
  for(int i=0; i<8; i++) begin
  rtr_io.cb.din[sa] <= payload[index][i];
  rtr_io.cb.valid_n[sa] <= 1'b0; //driving a valid bit
  rtr_io.cb.frame_n[sa] <= ((i == 7) && (index == (payload.size() - 1)));
  @rtr_io.cb;
  end
rtr_io.cb.valid_n[sa] <= 1'b1;
endtask: send_payload
```

发送数据。

仿真波形如图所示：



更多内容，请查看 testbench。