实验 10 报告

蔡润泽、付轶凡
箱子号：45

一、实验任务（10%）

在 lab10 的实验环境中，完成:

a) 带握手的类 SRAM 接口到 AXI 接口的转换桥 RTL 代码编写。

b) 通过简单的读写测试。

二、实验设计（40%）

（一）总体设计思路

读、写状态机如下：

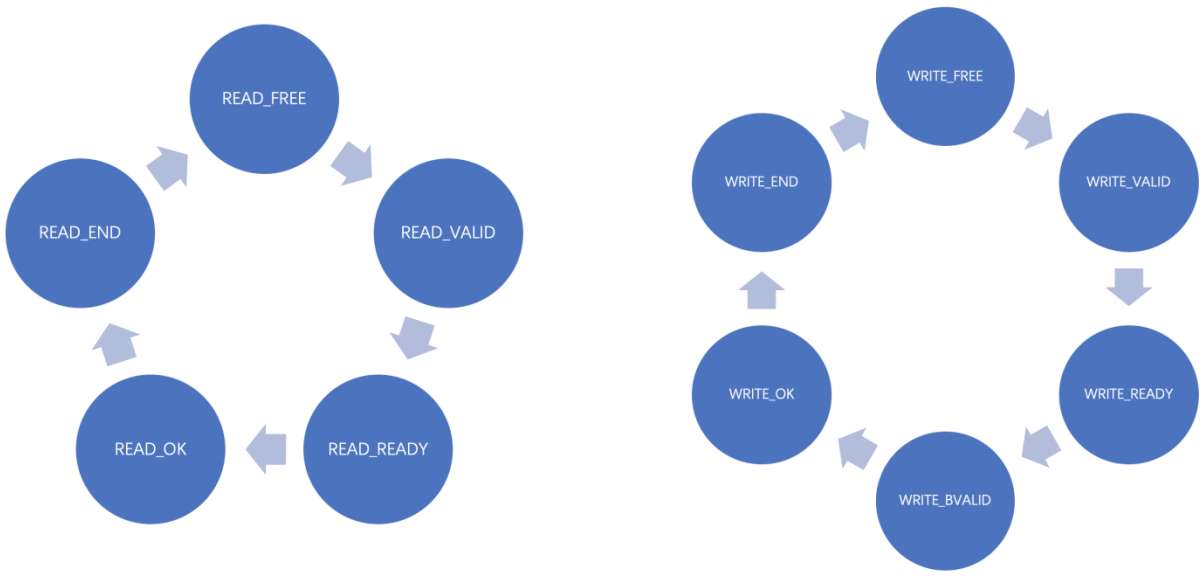


图 1.1 AXI 读、写状态机设计图

如图 1.1 的状态机示意图，在代码设计中，AXI 对于读事务和写事务分别由两个状态机。

1、读事务

对于读事务，一共存在五个状态，FREE、VALID、READY、OK、END：

FREE:表示读操作请求为空，可以接受读请求。

VALID:表示接收到了读请求，于是从 FREE 跳转到了 VALID 状态。在 VALID 状态下发出 arvalid、addr_ok 以及有效的 arsize、araddr。

READY:表示接收到了 arready，于是从 VALID 跳转到了 READY 状态。在 READY 状态下接收并存储 rdata，并发出 rready。

OK：表示接收到了 rvalid，于是从 READY 跳转到了 OK 状态，控制 data_ok 不会与 addr_ok 错误对应。

END：如果从 OK 状态跳到 END 状态，则表示可以发出 data_ok 信号，一个读事务结束。

2、写事务

对于写事务，一共存在六个状态，FREE、VALID、READY、BVALID、OK、END：

FREE:表示写请求为空，可以接受写请求。

VALID:表示接收到了写请求，于是从 FREE 跳转到了 VALID 状态。在 VALID 状态下发出 awvalid、addr_ok 以及有效的 awsize、awaddr。获得正确 wstrb 并在下一拍发出。

READY:表示接收到了 awready，于是从 VALID 跳转到了 READY 状态。在 READY 状态下发出 wdata，并发出 wvalid。

BVALID：表示接收到了 wready，于是从 READY 跳转到了 BVALID 状态。在 BVALID 状态下发出 bready 信号。

OK：表示接收到了 bvalid，于是从 BVALID 跳转到了 OK 状态，控制 data_ok 不会与 addr_ok 错误对应。

END：如果从 OK 状态跳到 END 状态，则表示可以发出 data_ok 信号，一个写事务结束。

（二）重要模块 1 设计：指令类 SRAM(inst sram-like)

1、工作原理

inst_addr_ok 信号用来和 req 信号一起完成读写请求的握手。只有在 clk 的上升沿同时看到 inst_req 和 inst_addr_ok 都为 1 的时候才是一次成功的请求握手。inst_data_ok 对应读事务的时候，它是数据返回的有效信号;对应写事务的时候，它写响应有效信号。当握手成功时，传递数据。

2、接口定义

```
input            inst_req,
input            inst_wr,
input    [ 1:0]  inst_size,
input    [31:0]  inst_addr,
input    [31:0]  inst_wdata,
output   [31:0]  inst_rdata,
```

```
output          inst_addr_ok,
output          inst_data_ok
```

3、功能描述

接收指令的读写请求，并在握手成功时传递数据。

（三）重要模块 2 设计：数据类 SRAM(data sram-like)

1、工作原理

data_addr_ok 信号用来和 req 信号一起完成读写请求的握手。只有在 clk 的上升沿同时看到 data_req 和 data_addr_ok 都为 1 的时候才是一次成功的请求握手。 data_data_ok 对应读事务的时候，它是数据返回的有效信号;对应写事务的时候，它写响应有效信号。当握手成功时，传递数据。

2、接口定义

```
input           data_req,
input           data_wr,
input   [ 1:0]  data_size,
input   [31:0]  data_addr,
input   [31:0]  data_wdata,
output  [31:0]  data_rdata,
output          data_addr_ok,
output          data_data_ok
```

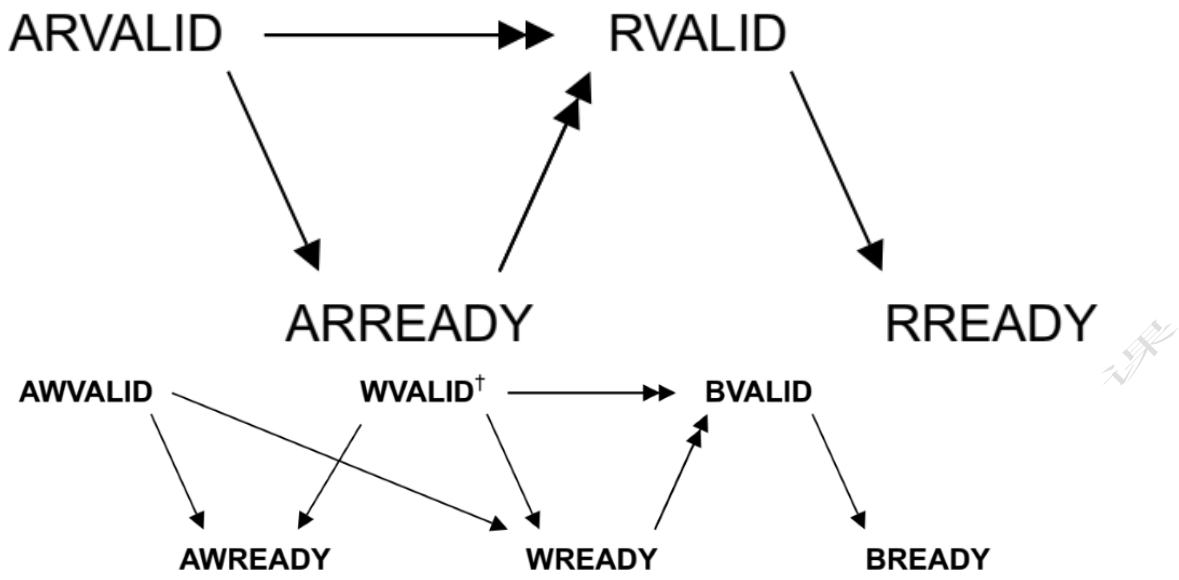
3、功能描述

接收数据的读写请求，并在握手成功时传递数据。

（四）重要模块 3 设计：AXI 模块

1、工作原理

总线的两端可以分为主方 master 和从方 slave，对于读操作来说，主方提出请求，从方接收请求返回数据；对于写操作来说，主方提出请求并发出写数据，从方接收请求和数据，AXI 总线协议采用握手机制完成主方与从方的交互，AXI 总线有五个通道，写地址请求、写数据请求、写响应、读地址请求、读数据请求，每一个通道都有一对 valid 和 ready 信号用来实现握手。这些通道的 valid 和 ready 依赖关系如下图：



† Dependencies on the assertion of **WVALID** also require the assertion of **WLAST**

2、接口定义

```
1.    //ar
2.      output  [ 3:0]  arid,
3.      output  [31:0]  araddr,
4.      output  [ 7:0]  arlen,
5.      output  [ 2:0]  arsize,
6.      output  [ 1:0]  arburst,
7.      output  [ 1:0]  arlock,
8.      output  [ 3:0]  arcache,
9.      output  [ 2:0]  arprot,
10.     output          arvalid,
11.     input           arready,
12.     //r
13.     input   [ 3:0]  rid,
14.     input   [31:0]  rdata,
15.     input   [ 1:0]  rresp,
16.     input           rlast,
17.     input           rvalid,
18.     output          rready,
19.     //aw
20.     output  [ 3:0]  awid,
21.     output  [31:0]  awaddr,
22.     output  [ 7:0]  awlen,
23.     output  [ 2:0]  awsize,
24.     output  [ 1:0]  awburst,
25.     output  [ 1:0]  awlock,
26.     output  [ 3:0]  awcache,
27.     output  [ 2:0]  awprot,
28.     output          awvalid,
29.     input           awready,
30.     //w
31.     output  [ 3:0]  wid,
32.     output  [31:0]  wdata,
33.     output  reg[ 3:0]   wstrb,
34.     output          wlast,
35.     output          wvalid,
36.     input           wready,
37.     //b
38.     input   [ 3:0]  bid,
39.     input   [ 1:0]  bresp,
40.     input           bvalid,
41.     output          bready
```

3、功能描述

接收 CPU 的读、写指令和数据请求，并将请求传递给类 SRAM，当相应的指令和数据准备好时。完成对应的读事务和写事务操作。

三、实验过程（50%）

（一）实验流水账

11 月 16 日 20：00-20：30 阅读讲义

11 月 16 日 20：30-22：00 设计代码

11 月 17 日 10：00-11：30 继续实现代码

11 月 17 日 14：00-15：30 调试 bug

11 月 18 日 19：30-22：30 撰写实验报告

（二）错误记录

1、错误 1：data_ok 和 addr_ok 不对应。

（1）错误现象

原来的版本里运行仿真通过，上板后修改随机种子发现有不通过的情况，在 vivado 上重现该错误，随机种子修改为 16'h030f，运行仿真，出现了如图 3.1 的两个读 data_sram_like 的错误：

```
[   3195 ns] OK!!read data 0
[   3405 ns] Fail!!!read from data sram-like, ref_data[23:16]=8'h11, my_data[23:16]=22
[   3525 ns] Fail!!!read from data sram-like, ref_data[15: 0]=16'h0000, my_data[15: 0]=3311
[   3605 ns] OK!!!read data 3
```

图 3.1 read data 1 和 read data 2 测试错误

（2）分析定位过程

通过查看波形，发现 data_data_ok 与 data_addr_ok 握手有误，如图 3.2 的波形所示，data_addr_ok 在 read 状态机的 valid 阶段发出 addr_ok，在 write 状态机的 valid 阶段也发出了一个 addr_ok，也就是先有读 data 的请求，再有写 data 的请求，由于该随机种子下是长延迟，由波形图可以看出 rvalid 延迟了很晚才来，而 wready 很快就来了，就导致 write 的状态机比 read 的状态机跳转更快，于是 write 先到了 end 态发出了 data_ok，read 后到 end 态发出 data_ok，所以就成了 write 发出的 data_ok 对应 read 发出的 addr_ok，read 发出的 data_ok 对应 write 发出的 addr_ok，导致读写响应的错误：

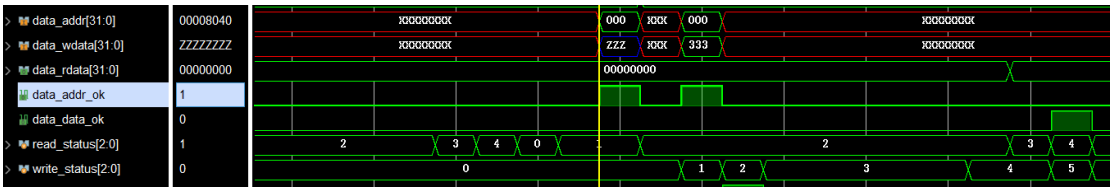


图 3.2 相关波形

（3）错误原因

造成这样错误的原因是 read 先发 addr_ok 后 write 也发 addr_ok 且 write 的 data_ok 比 read 更早到达，所以需要在 read 发出 addr_ok 后将 write 的状态机阻塞住。

（4）修正效果

我们采用了一种较为粗暴的方式阻塞状态机，如图 3.3 的代码，只要 read 进入了其他的状态，就将 write 阻塞在了 free 态不允许它发 addr_ok，这样就不会有错误的对应了。

```
always @(posedge clk) begin
    if (!resetn) begin
        write_status <= `WRITE_FREE;
    end
    else if (write_status==`WRITE_FREE && ((inst_req && inst_wr && read_status==`READ_FREE) || (data_req &
        write_status <= `WRITE_VALID;
    end
    else if (write_status==`WRITE_VALID && awready  && (inst_addr_ok || data_addr_ok)) begin
        write_status <= `WRITE_READY;

    end
    else if (write_status==`WRITE_READY && wready) begin
        write_status <= `WRITE_BREADY;
    end
    else if (write_status==`WRITE_BREADY && bvalid) begin
        write_status <= `WRITE_OK;
    end
    else if (write_status==`WRITE_OK && inst_ok_flag!=2 && data_ok_flag!=2)begin
        write_status <= `WRITE_END;
    end
    else if (write_status==`WRITE_END) begin
        write_status <= `WRITE_FREE;
    end
end
```

图 3.3 增加阻塞后的状态机跳转

修改后，该随机种子下的测试顺利通过。

```
Test end!
----PASS!!!
$finish called at time : 4675 ns : File "D:/share/UCAS_CDE/cpu_axi_ifc_dev/testbench/axi_ifc_tb.v" Line 288
```

图 3.4 测试通过

综合、布局布线、生成 bit 文件，然后上板验证，我们一个一个地拨了 256 个随机种子，全部正常显示。

四、实验总结（可选）

本次实验较前几次实验难度突然增大，一是因为上学期组成原理实验没有设计总线，二是因为讲义比之之前的部分更加难懂，我们这次设计虽然通过了测试，但仍然认为有可以改进的地方，甚至有些地方的理解可能仍然是错误的，只是顺利通过了测试而已。我们之前版本的代码在进行测试时，总是随机种子仿真时可以通过，但是上板采样随机种子时总会不成功，为此一遍一遍地修改设计，大概上板有 4 到 5 次才最终成功地通过了所有的 256 个随机种子采样，但是核心的错误只有上面所述的一个，就是读写的 data_ok 和 addr_ok 不能对应的问题，这恰好就是随机延迟带来的，我们想了很多个修改的办法，最后才确定了用这种很“粗暴”的阻塞方案，之后如果有时间，也会考虑重新修改这一部分的设计。