Lab 7 报告

学号: 2021K8009908004、2021K8009910001、2021K8009908024

姓名: 王致力、徐畅杰、张修梁

箱号: 34

一、实验任务

1. 将原有 CPU 访问 SRAM 的接口调整为类 SRAM 总线接口

2. 设计实现一个 “类 SRAM-AXI” 的转接桥，拼接类 SRAM 总线接口的 CPU，运行 AXI 固定延迟验证

3. 完善并完成 AXI 随机延迟验证

二、实验设计

（一）总体设计思路

1、出于简化设计的考虑进行的协议简化

实际上类 SRAM 可以为 CPU 提供相当大的性能的优化，但是为了内部控制的简洁考虑，对于类 SRAM 协议进行了一定的简化：

仅仅当下一流水级有能力接收请求的时候才会发送请求，这是为了避免 if 级可能因为分支跳转导致请求发送但是尚未接收的较为尴尬的状态，尤其是 addr_ok，data_ok 可能还恰好在此时握手导致的额外复杂性，所以在本设计中暂时不考虑这种情况。

但是访存级显然是不会有以上的问题的，首先没有分支预测捣乱，其次 wb 级的 allowin 还是总是拉高的，那么我们能否采取充分利用类 SRAM 的设计呢，暂时也没有如此的考虑，这是因为读写通道在 AXI 上是分开的，也就是说甚至有可能会有读写请求相互“超车”的情况，但是若想要避免这种问题就需要引入一个类似于 raid_buffer 的设计来将完成的请求重新排序，以防止请求的提交 (addr_ok) 并不顺序的问题（详见后续 bug)。出于设计简洁性的考虑没有采取这样的设计。

2、CPU 的类 SRAM 握手

在取值与进行内存相关操作时需要使用类 SRAM 的握手协议进行相关的操作。该协议支持多个读写端口进行相关握手，但是充分开发其并行性对于内部状态的控制与管理有着较为严格的限制，例如在 CPU 的预取指与取指级中，可能因为分支或者中断的到来导致之前取到的指令无效，但是这难以简单地通过控制 `valid`，`readygo`，`allowin` 进行控制，要想进行控制将会不得不增加额外的寄存器以存储相关状态，故暂时不采取这种设计，目前仅仅当下一流水级处于 `allowin` 时才会发送请求，也就是说只要请求被接收下一个流水级一定可以处理，该流水级才会发送相关请求。

3、类 SRAM 部分

与原 CPU 较为相似，除了信号 `size`、`addr_ok` 和 `data_ok` 外，其余信号都与原有的 SRAM 接口信号对应。对于新增的握手信号 `addr_ok` 和 `data_ok`，需将其作为流水线控制信号生成逻辑的一部分。在握手信号不满足条件时，由于协议的自由性，可以根据当前的状态选择更改发送的内容，例如 pre_if 级在碰到分支同时请求没有被接收时会更改请求的地址值为分支的地址值，这提供了一定的性能保障。

4、AXI 部分

主要模块即类 SRAM-AXI 的转接桥,此处我们将原来的 SRAM 接口 `mycpu_top` 模块改为 `mycpu_core`，并在新的 AXI 接口 `mycpu_top` 中实例化。该转接桥需要负责选择使用哪个内存通道来读取数据，根据当前状态和输入信号来决定是否读取指令数据或数据存储器。此外，它为 AXI 协议层与类 SRAM 协议层提供了一层抽象，即类 SRAM 协议层可以按照类 SRAM 协议自由地更改请求，而且认为有两个读写请求通道，而 AXI 仅仅有一个读请求通道与两个写请求通道，而两个读请求通道实际上需要为同一次写事务服务，实际上

也就认为只有一个写请求通道。

5、类 SRAM 转 AXI

该部分主要负责将类 SRAM 协议转换成 AXI 协议，需要注意而者的两点不同，首先类 SRAM 协议允许在握手之前发送的请求改变，但是 AXI 是不允许的，故引入了快照机制，即当握手完成时，会记录下信号的内容并缓存直到单次请求完成，以防止 AXI 出现问题。而类 SRAM 认为有两个通道都可以发送读请求，但是类 SRAM 仅仅有一个可以发送读请求的通道，故需要添加一个选择器进行仲裁，同样由于单次读写请求的时候不允许改变，所以选择器也必须缓存结果直到单次读请求完成。同时由于类 SRAM 仅仅需要将读写位更改即可说明是否为读写，但是 AXI 中却需要通过不同的通道发送请求，所以需要一定的机制进行分流。

（二）重要模块 1 设计：pre_if-if 握手模块

由于更改为类 SRAM 接口，需要额外的握手机制从而确保功能的正常。

1、工作原理

通过握手信号与 `ID` 级的 `allowin` 产生 `req` 与 `readygo` 信号，从而完成指令请求的发送与接收，经过一定的封装使得这一流水级的更改对于其他流水级不可见。

2、接口定义

仅仅展示较为重要的新增外部接口与内部接口。

表 1: 接口定义表

接口名称	方向	位宽	功能描述
pre_if_allowin	pre_if->pc_src	1	表明 pc 能否自加
pre_if_readygo	pre_if->if	1	pre_if 的发送是否已经完成
inst_sram_req	pre_if->top	1	pre_if 发出请求
inst_sram_addr_ok	top->pre_if	1	cpu_top 收到请求
inst_sram_addr	pre_if->top	32	请求的地址
if_allowin	if->id	1	if 级是否允许 pre_if 发送下一条指令
if_readygo	if->id	1	if 级是否已经完成
inst_sram_rdata	top->if	32	转接桥发送的数据
inst_sram_data_ok	top->if	1	转接桥发送的数据有效

3、功能描述

该部分的主要设计目的是为了确保 if 级别发送的请求与向 id 级提供的指令完全正确，为此就需要进行一定的简化，首先是对于 sram 的协议，仅仅当 if 级有能力接收下一条指令时 pre_if 才会发送请求。同时由于某些情况下 if 级的指令无法在读到时就被接收，所以需要在这种情况下暂存指令，同时若一条指令已经被发射，就应该通过某些方式将其阻塞或者无效化以防止重复执行。

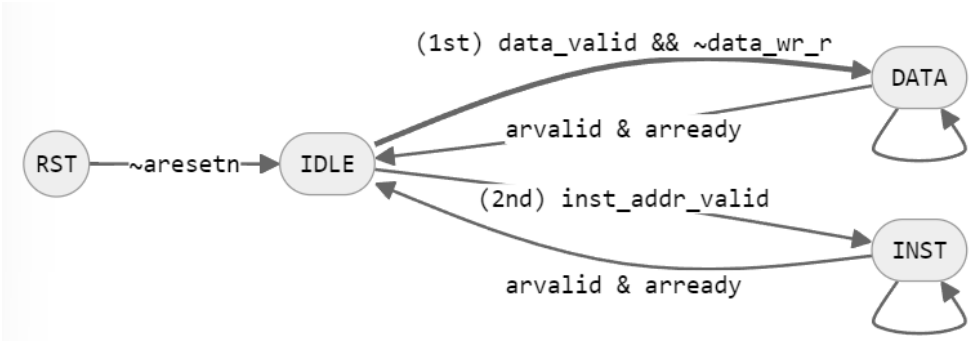
相似的 exe 与 mem 级也可以使用类似的设计。

（三）重要模块 2 设计：AXI 转接桥仲裁模块

由于数据和指令读取可能发生冲突，必须在转接时选择其中一方（此处为数据）优先读取，并在处理读请求时阻塞其他读请求。

1、工作原理

使用一个 `selecter` 寄存器记录下图中的状态，在仲裁器 IDLE 时，若同时有数据和指令读请求，状态机将选择数据读优先处理（图中较粗的一支转移通路）



2、接口定义

同上图，图中信号为该模块输入，输出为 `selecter` 寄存器。

（四）重要模块 3 设计：AXI 转接桥快照模块

由于类 SRAM 协议与 AXI 协议的不同，需要将请求记录以防止握手之后请求改变以便于 AXI 端的发送。

1、工作原理

该部分的请求已经得到执行时，拉高 addr_ok，若之后成功接收到 req，则进行快照以保存完整的请求。

2、接口定义

表 2: 接口定义表

接口名称	方向	位宽	功能描述
inst_addr	IN	32	包含其他相关信息，暂时不重复列举
data_addr	IN	32	包含其他相关信息，暂时不重复列举
inst_req	IN	1	指令请求，可能随时改变，认为仅仅当握手的那一拍的数据是有效的
inst_addr_ok	OUT	1	实际上认为是转接桥有能力处理这一条指令请求
data_req	OUT	1	收到数据请求
data_addr_ok	IN	1	实际上认为是转接桥有能力处理这一条数据请求
inst_valid	OUT	1	指令有效，即待处理
data_valid	OUT	1	数据有效，即待处理
data_add_r	IN	32	提供给 AXI 的一层抽象化后的请求，在一次总线事务中不会改变

3、功能描述

为将类 SRAM 总线转化为 AXI 总线提供一层抽象，即类 SRAM 总线在协议上只要没有握手就可以改变请求，但是 AXI 总线由于其高速的需求并非这样，所以需要一层封装以使得协议能够顺利转换，实际上也可以通过限制发送请求的一方来达到这一目的。

（五）重要模块 4 设计：AXI 握手模块

将部分信号进行了封装，使得状态机的运转逻辑得到简化。

1、工作原理

将写的两个通道与读的一个通道和 selecter 封装为 data_addr_handled，而读通道与 delecter 封装为 inst_addr_handlded，而从到主的回应直接解包发送给对应的 data_ok，避免使用较为无用的状态机。

2、接口定义

表 3: 接口定义表

接口名称	方向	位宽	功能描述
inst_addr_handled	OUT	1	指令的请求得到解决，inst 快照可以接收下一条请求了
data_addr_handled	OUT	1	数据的请求得到解决，data 快照可以接收下一条请求了
arready	IN	1	ar 通道上从机就绪
arvalid	OUT	1	ar 通道上请求有效
wvalid	OUT	1	w 通道请求有效
wready	IN	1	w 通道从机就绪
bvalid	OUT	1	b 通道请求有效
bready	IN	1	b 通道从机就绪
cpu_inst_sram_data_ok	OUT	1	inst 数据有效
cpu_data_sram_data_ok	OUT	1	data 数据有效

3、功能描述

提供了上下层的抽象，将具体的握手信号经过一定的处理与等待转化为了较为简单的信号，从而使得各个状态机与握手信号得到简化。

三、实验过程

（一）实验流水账

2023.11.6 19：00 - 23：00 完成 exp14 并完成上板验证。

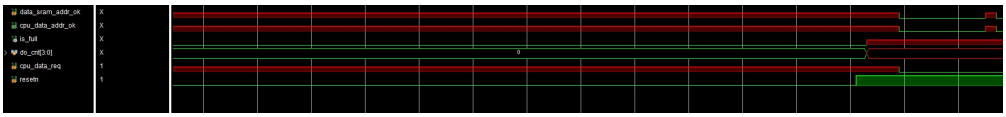
2023.11.20 13：00 - 22：00 完成 exp15 并上板验证，发现 exp16 问题。

2023.11.25 15：00 - 21：00 完成 exp16 并且通过上板验证。

（二）错误记录

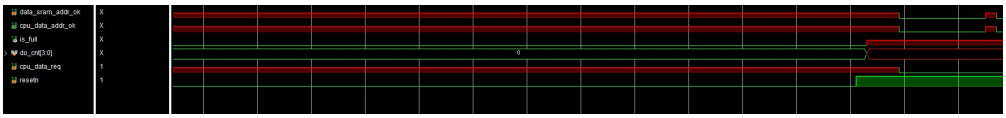
1、错误 1：因为没有进行信号复位导致发出无效请求

（1）错误现象



因为发送了无效的请求（X）导致了类 SRAM 的封装器返回的 addr_ok 均为不定态（X）。

（2）分析定位过程



查看通过查看上层封装器的源代码发现其内部有一个状态机，而该状态机的运转依赖于 req。

（3）错误原因

之前所诉的状态机的运转依赖于 req，若发送不定态请求会导致该状态机的运转为不定态，导致出现不定态。

（4）修正效果

在开始时复位所有的信号，以防止发送不定态请求。

（5）归纳总结

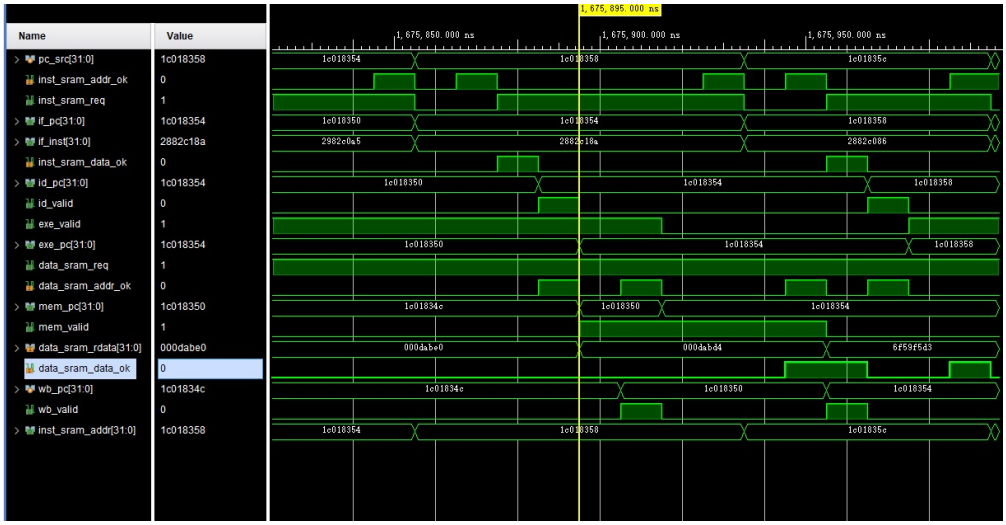
在接口上一定要小心，避免不定态信号的产生，以避免导致其他并不熟悉的结构功能错乱。

2、错误 2：未考虑 st 指令在 mem 级的等待

（1）错误现象

ld 指令发出后在数据真正返回前就接收到了 data_ok，导致其提前带着错误的数据信息流到写回级发生错误。

（2）分析定位过程



查看发现 1c018354 接收到了一个 data_ok，但是其对应的 data_ok 和 data 却在下一次才到达。查看发现上一条写指令并未等待 data_ok 导致了错误。

（3）错误原因

```
// valid signals
assign mem_ready_go      = (~mem_res_from_mem | data_sram_data_ok) & ~mem_gone | (|mem_exc_rf_reg);
assign mem_allowin       = ~mem_valid | mem_ready_go & wb_allowin | cancel_exc_ertn | mem_gone;
```

已知 1c018350 为一条写指令，但是其在 mem 级却没有等待 data_ok 就流到了下一流水级，导致了错误。

阅读源码文件发现确实没有进行读相关的等待

（4）修正效果

```
assign mem_ready_go      = (~mem_res_from_mem & ~mem_we | data_sram_data_ok) & ~mem_gone | (|mem_exc_rf_reg);
assign mem_allowin       = ~mem_valid | mem_ready_go & wb_allowin | cancel_exc_ertn | mem_gone;
```

为 mem 级增加写指令的等待。

（5）归纳总结

应当熟悉相关的信号的含义以此确保在握手上不会出现问题。

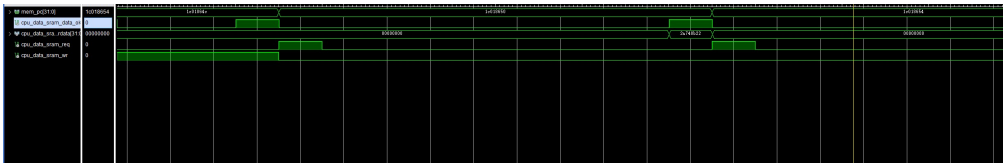
3、错误 3：转接桥通道的速率不一致导致读写顺序错乱

（1）错误现象

进行读的时候读到了下一个应该到来的数据。

```
[2470867 ns] Error!!!
    reference: PC = 0x1c018654, wb_rf_wnum = 0x0a, wb_rf_wdata = 0x2a748b22
    mycpu    : PC = 0x1c018654, wb_rf_wnum = 0x0a, wb_rf_wdata = 0x000d7164
```

（2）分析定位过程



```
1c01864c:   29a3c084    st.w    $r4,$r4,-1808(0x8f0)
1c018650:   29a3c0a5    st.w    $r5,$r5,-1808(0x8f0)
1c018654:   28a3c18a    ld.w    $r10,$r12,-1808(0x8f0)
1c018658:   28a3c086    ld.w    $r6,$r4,-1808(0x8f0)
1c01865c:   28a3c0a4    ld.w    $r4,$r5,-1808(0x8f0)
1c018660:   28a3c0a6    ld.w    $r6,$r5,-1808(0x8f0)
```

分析波形与代码发现是因为指令之前的写指令收到了本条读指令的数据并且流到了下一流水级，直到这条读指令收到再下一条读指令，写指令的握手仍然没有返回。

（3）错误原因

由于 AXI 各个通道的速率不一致，导致了哪怕先发送的写请求也可能被后发送的读请求“超车”导致错误。

（4）修正效果

```
assign data_sram_req    = (exe_res_from_mem | mem_we) & ~(mem_ale | mem_exc_flush)
        & mem_allowin & ~mem_handled & exe_valid;//(|mem_exc_rf[6:0]))
```

限制 AXI 协议，仅当 mem 级有能力接收 exe 发出的请求才会发送 exe 级的请求。

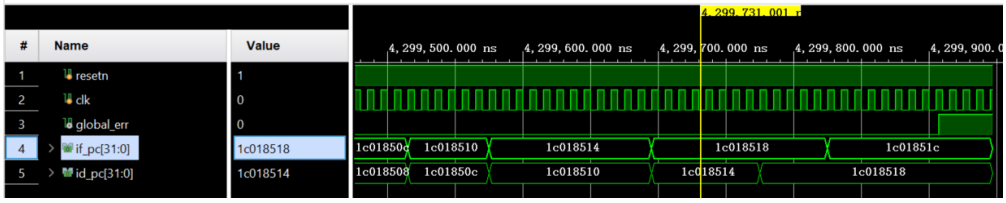
（5）归纳总结

实际上这一错误也可以通过插 buffer 避免，但是需要考虑较为多的情况，暂时采用限制协议的方法规避。

4、错误 4：流水线 PC 错误

（1）错误现象

在 IF 级的 PC 还是 0x1c018518 时，ID 已经拿到了该 PC，造成流水级的错误。



（2）分析定位过程

分析波形与代码发现是因为 AXI 转接部分向 CPU 返回的 SRAM 握手信号定义错误。

（3）错误原因

误在转接桥正在处理 CPU 读请求的状态当成了已经处理完成的状态，写入了类 SRAM 的握手信号。

当 AXI 转接桥向 CPU 发送了错误的握手信号时，会导致 CPU 误以为 IF 级已经取到了指令，将错误的指令码（实际上还是上一条指令）放进 ID 级，造成流水线中出现了一条多余的指令，导致错误。

（4）修正效果

换为转接桥已经处理完 CPU 读请求的状态即可，此时 CPU 的流水线恢复正常。

（5）归纳总结

此问题涉及对两种协议的理解，需要注意两者之间的区别和联系。

四、实验总结（可选）

本实验涉及到两种常用的总线协议，其中 AXI 作为广泛应用的一种协议，对信号的依赖关系和时序关系有较强的限制。在进行转接桥实验时，我们曾经为了满足 AXI 的一些要求造出了组合回路，在修改（其实是部分重构）后才得以通过最终的随机延迟实验，令人印象深刻（