实验四：总线接口设计与实现

朱梓涵 学号：24325356

2025 年 12 月 21 日

1 实验目的

本实验的主要目的是：

1. 理解 AXI4-Lite 总线协议的基本原理和通信机制。
2. 学习使用状态机实现总线协议。
3. 掌握主从设备之间的握手通信过程。
4. 理解 MMIO（Memory-Mapped I/O）的工作原理。
5. 将总线协议集成到流水线 CPU 中。

2 实验环境

- **操作系统**: Windows 11
- **开发工具**: IntelliJ IDEA
- **构建工具**: SBT
- **仿真与测试**: Verilator, chiseltest

3 实验原理

3.1 AXI4-Lite 协议概述

AXI4-Lite 是总线协议的简化版本。它包含 5 个独立的通道：

- **读地址通道（AR）**：主机发送读地址。
- **读数据通道（R）**：从机返回读取的数据。

- **写地址通道（AW）**：主机发送写地址。
- **写数据通道（W）**：主机发送写数据。
- **写响应通道（B）**：从机返回写操作响应。

3.2 通信框架

本实验采用的通信框架如下：

- **CPU 侧**：通过 `AXI4LiteMasterBundle` 接口发起读写请求。
- **AXI4LiteMaster**：将简单的读写请求转换为符合 AXI4-Lite 协议的信号。
- **AXI4LiteChannels**：5 个通道的信号线，符合 AXI4-Lite 规范。
- **AXI4LiteSlave**：接收 AXI4-Lite 协议信号，转换为设备可理解的读写操作。
- **设备侧**：通过 `AXI4LiteSlaveBundle` 接口响应读写请求。

3.3 握手机制

AXI4-Lite 协议采用 VALID/READY 握手机制：

- 发送方通过 `VALID` 信号表示数据有效。
- 接收方通过 `READY` 信号表示准备接收。
- 只有当 `VALID` 和 `READY` 同时为高时，握手完成，数据传输成功。

4 模块实现与分析

4.1 状态机设计

本实验使用状态机实现 AXI4-Lite 协议。定义了以下状态：

```
object AXI4LiteStates extends ChiselEnum {
  val Idle, ReadAddr, ReadDataWait, ReadData,
      WriteAddr, WriteData, WriteResp = Value
}
```

Listing 1: 状态定义

4.2 AXI4LiteMaster 实现

4.2.1 主机状态机逻辑

1. **Idle 状态**：等待来自 CPU 的读写请求。收到读请求时，保存地址并转到 ReadAddr 状态；收到写请求时，保存地址和数据并转到 WriteAddr 状态。

2. **ReadAddr 状态**：拉高 ARVALID，发送读地址 ARADDR。等待从机 ARREADY 信号，握手完成后，拉高 RREADY，转到 ReadData 状态。

3. **ReadData 状态**：保持 RREADY 为高，等待从机 RVALID。收到 RVALID 时，锁存 RDATA，拉高 read$_v alid\Gamma{*}h\emptyset$fiæ$CPU$ıfiffiIdle$\Delta\Theta$

3. WriteAddr 状态：拉高 AWVALID，发送写地址 AWADDR。等待从机 AWREADY 信号，握手完成后，拉高 WVALID，转到 WriteData 状态。

4. WriteData 状态：保持 WVALID 为高，发送 WDATA 和 WSTRB。等待从机 WREADY 信号，握手完成后，拉高 BREADY，转到 WriteResp 状态。

5. WriteResp 状态：保持 BREADY 为高，等待从机 BVALID。收到 BVALID 时，拉高 write$_v alid\Gamma{*}h\emptyset$fiffiIdle$\Delta\Theta$

4.2.2 关键代码实现

```
switch(state) {
  is(AXI4LiteStates.Idle) {
    when(io.bundle.read) {
      addr := io.bundle.address
      state := AXI4LiteStates.ReadAddr
      ARVALID := true.B
    }.elsewhen(io.bundle.write) {
      addr := io.bundle.address
      write_data := io.bundle.write_data
      write_strobe := io.bundle.write_strobe
      state := AXI4LiteStates.WriteAddr
      AWVALID := true.B
    }
  }
}
```

Listing 2: 主机状态机核心代码

此外，当主机不在 Idle 状态时，busy 信号为高，拒绝新的请求：

```
io.bundle.busy := state =/= AXI4LiteStates.Idle
```

且 valid 信号在状态机执行前清零，确保只持续一个周期：

```
when(read_valid) { read_valid := false.B }
when(write_valid) { write_valid := false.B }
```

4.3 AXI4LiteSlave 实现

4.3.1 从机状态机逻辑

1. **Idle 状态**：清除所有控制信号。优先响应读请求（ARVALID），收到时保存地址，拉高 ARREADY，转到 ReadAddr。收到写请求时，保存地址，拉高 AWREADY，转到 WriteAddr。

2. **ReadAddr 状态**：拉低 ARREADY，拉高 read 信号通知设备读取数据，转到 ReadData 状态。

3. **ReadData 状态**：保持 read 为高，等待设备 read$_valid\Theta 60\Delta X$

3. WriteAddr 状态：拉低 AWREADY，等待主机 WVALID。收到写数据后，锁存数据和写选通，拉高 WREADY 和 write，转到 WriteData。

4. WriteData 状态：拉低 WREADY 和 write，拉高 BVALID，转到 WriteResp。

5. WriteResp 状态：保持 BVALID 为高，等待主机 BREADY。握手完成后返回 Idle 状态。

4.3.2 关键代码实现

```
switch(state) {
  is(AXI4LiteStates.Idle) {
    when(io.channels.read_address_channel.ARVALID) {
      addr := io.channels.read_address_channel.ARADDR
      ARREADY := true.B
      state := AXI4LiteStates.ReadAddr
```

```
    }.elsewhen(io.channels.write_address_channel.AWVALID) {
      addr := io.channels.write_address_channel.AWADDR
      AWREADY := true.B
      state := AXI4LiteStates.WriteAddr
    }
  }
}
```

Listing 3: 从机状态机核心代码

使用寄存器保证 RDATA 在 RVALID 为高时保持稳定：

```
val rdataReg = RegInit(0.U(dataWidth.W))
io.channels.read_data_channel.RDATA := rdataReg
```

4.4 性能优化

本实现采用了以下优化策略：

1. **流水化握手**：在地址握手完成后立即准备数据握手，减少等待周期。
2. **优先级处理**：从机优先响应读请求，提高取指效率。
3. **信号稳定性**：使用寄存器锁存关键数据，避免毛刺。

5 CSR 指令与总线交互

CSR 指令在总线协议中的交互主要体现在 MMIO（Memory-Mapped I/O）上。CPU 通过地址映射访问 CSR 寄存器或外设寄存器：

- 读写 CSR 时，控制单元发出相应的读写请求。
- AXI4-Lite Master 接收请求，将地址和数据转换为总线事务。
- AXI4-Lite Slave 根据地址将请求路由到具体的 CSR 模块或外设。
- 通过握手机制，确保数据传输的正确性和稳定性。

6 测试结果与分析

6.1 测试原理

`BusTest.scala` 包含多个测试用例，验证 AXI4-Lite 实现的正确性：

- **FunctionalTest**：创建 `TestBox` 模块，模拟主从机忙碌状态，验证读写事务的地址、数据、选通信号及 `valid` 信号时序。
- **连续事务测试**：随机生成 1000 个读写事务，模拟从机忙碌状态，验证高负载下的总线稳定性。
- **其他测试**：包括 `TimerTest`（定时器）、`MemoryTestF`（内存）和 `ROMLoaderTestF`（ROM 加载）。

6.2 分析

1. **正确性验证**：所有测试用例通过，说明实现符合 AXI4-Lite 协议规范。数据传输正确，地址、数据、选通信号及 `valid` 信号时序均符合预期。
2. **性能分析**：单次读写事务的握手周期符合设计预期（约 3-4 周期）。连续事务测试证明了总线在高负载下的可靠性。

7 改进建议

1. **建议：提供更多调试案例和方法指导。**

   建议增加具体的调试案例，例如如何追踪一条指令在总线中的完整传输过程，如何分析波形图定位握手失败问题等。

2. **建议：增加可视化工具。**

   建议提供或推荐一些工具，能够将总线上的信号交互以图形化方式展示，辅助理解握手过程。

8 实验结论

通过本次实验，我深入理解了 AXI4-Lite 总线协议的工作原理，掌握了使用状态机实现复杂通信协议的方法。我成功实现了：

- 符合 AXI4-Lite 规范的主机和从机模块。
- 基于 VALID/READY 握手机制的通信流程。
- 完善的测试用例，验证了总线的正确性和稳定性。

本次实验不仅提升了我的硬件设计能力，也让我对计算机系统中各模块间的互连和通信有了更深刻的认识，为后续更复杂的系统设计打下了坚实基础。