计算机组成原理
实验 3: SRAM 实验 & 实验 4: UART 串口实验
实验报告

李祥泽
2018011331
lee_johnson@qq.com

实验目的

1. 熟悉 THINPad 教学计算机存储器和串行接口的配置及与总线的连接方式;
2. 掌握 THINPad 教学机内存 (RAM) 和串口 UART 的访问时序和方法;
3. 理解总线数据传输的基本原理.

实验内容

使用教学计算机上的 FPGA 芯片, 设计内存和串口的读写逻辑, 完成以下功能:

1. 将拨码开关设置为起始地址 (记作 Addr), 地址的单位是 32 位 (即 1 个字).
2. 单击 RST 按键.
3. 从 UART 串口自动读取 10 个字节的数据, 将其依次存储在 BaseRAM 以 Addr 开始的 10 个字中.
4. 将收到的数据按原顺序自动发送给 UART 串口.
5. 再设置拨码开关, 单击 RST 按键, 能够重复以上过程.

实验过程

设计了用于联合控制 BaseRAM 和 UART 串口的控制器 `MemController`. 在该控制器中有 3 个状态机, 分别控制: 1. 控制器当前的读写状态, 2. 读串口-写内存的节拍, 3. 读内存-写串口的节拍. 另外, 还有一组用来暂存数据的寄存器 `data`; 如果总线没有被设置成高阻态 (由信号 `dataHiZ` 控制), 就将从这个寄存器中取数据.

> 从将来复用该设计的角度来说, 我认为应该设计 5 个状态机, 其中 1 个是主控, 剩下 4 个分别控制串口的读, 串口的写和内存的读, 写. 但是, 在本实验中, 串口的读总是紧跟着内存的写, 反之亦然, 因此状态机可以简化成 3 个.
>
> 我认为这样的简化还是有意义的, 因为同时控制多个状态机的代码实在是太复杂了 (当然更可能是我学艺不精, 没搞懂多个状态机互相交互的方法).

在 RST 被按下时, 控制器从拨码开关 (`dip_sw`) 获取地址设定, 存入一组寄存器 `baseAddr`. 这将作为内存寻址的基址, 与主控状态机中的计数器相加后传给 BaseRAM. 同时, 所有状态机都将被复位, 所有使能信号都设置为**不使能**.

读串口-写内存状态机有 7 个状态, 为 (均以 `READ_` 开头, 以转移顺序排列):

1. `IDLE`: 空置状态, 为初始状态, 也是完成一轮操作后进入的状态. 该状态不会自动转出, 而是由主控状态机控制它转移到下面一个状态.
2. `STDBY`: 预备状态. 从该状态转出时, `dataHiZ` 将被拉高 (使数据总线进入高阻态).
3. `WAIT_UART`: 等待数据到来. 该状态只在 `uart_dataready` 为高时自动转移到下一个状态. 转出时将 `uart_rdn` 拉低以开始读取.
4. `READ_UART`: 从 UART 串口读数据. 从该状态转出时, 数据将被存入数据寄存器 `data`; 并且 `uart_rdn` 被拉高.

5. `BUBBLE`: 一个空节拍. 从该状态转出时 `dataHiZ` 拉低, 数据从寄存器进入总线.
6. `WAIT_RAM`: 也类似于空节拍, 确保数据总线上的数据达到 RAM 所要求的建立时间. 转出时 `ram_we_n` 拉低以开始写入.
7. `WRITE_RAM`: 写入内存. 转移的下一个状态是 `IDLE`. 转出时 `ram_we_n` 拉高.

读内存-写串口状态机有 12 个状态, 为 (均以 `WRITE_` 开头, 以转移顺序排列, 合并了):

1. `IDLE`: 与上一个类似.
2. `STDBY`: 类似, 转出时将 `dataHiZ` 拉高以准备读.
3. `WAIT_RAM`: 一个空节拍, 转出时将 `ram_oe_n` 拉低开始读内存.
4. `READ_RAM`: 读内存, 使数据从内存中进入总线. 转出时将总线数据装入 `data`, 并将 `ram_oe_n` 拉高.
5. `BUBBLE`: 类似的空节拍. 转出时 `dataHiZ` 拉低.
6. `LOAD_UART`: 仍然是空节拍, 确保数据在总线上经过足够长时间. 转出时将 `uart_wrn` 拉低开始将数据装入串口的寄存器.
7. `LOAD_BUB_1`, `LOAD_BUB_2`, `LOAD_BUB_3`: 将数据装入串口的空节拍. 经验主义的产物, 应该算是为了解决串口和控制器之间的时钟跨域问题而加进去的.
8. `WRITE_UART`: 写串口. 转出时将 `uart_wrn` 拉高, 使串口开始向外写.
9. `WAIT_TBRE`, `WAIT_TSRE`: 等待这两个信号拉高, 也即等待串口完成写. 完成后转 `IDLE`.

主控状态机只在其余两个状态机都处于 `IDLE` 状态时才工作. 它有 4 个状态, 为 (以 `MAIN_` 开头):

1. `STDBY`: 初始状态. 转出时将计数器清零 (虽然之前应该已经是 0 了, 但是保险起见).
2. `READ`: 读串口-写内存状态. 如果计数器到所需数值, 将计数器清零并转下一状态; 否则计数器加 1, 控制读串口-写内存状态机进入 `READ_STDBY` 状态开始操作.
3. `WRITE`: 读内存-写串口状态. 与上一个类似, 可以控制读内存-写串口状态机进入 `WRITE_STDBY` 状态.
4. `IDLE`: 全部完成后的空置状态. 不会自动转移, 等待复位按钮被按下.

实验结果

本实现通过了 ThinPAD-Cloud 自动评测. 评测号 9906.

思考题

1. 静态存储器的读和写各有什么特点?

我们使用的是 *异步静态随机存取存储器*. 它无需外接时钟信号, 而是由输出使能 `OE_N` 和写入使能 `WE_N` 两个信号来控制.

在读取时, 将地址线设置好, 输出使能 `OE_N` 拉低, 经过一定延时, 相应的数据就会送到总线. 如果保持使能而更改地址, 原来输出会维持一定时间, 然后输出一定时间混乱数据, 最终稳定为新的正确数据.

在写入时, 将地址线和总线设置好, 输入使能 `WE_N` 拉低, 经过一定延时, 相应的数据就会写入内存. 如果保持使能, 写入也会持续发生.

当然, 读取和写入也可以用片选 `CE_N` 或字节使能 `BE_N` 控制, 其区别在于每种控制方式所要求的数据建立时间, 延时和数据保持时间各不相同.

2. 什么是 RAM 芯片输出的高阻态? 它的作用是什么?

高阻态是三态逻辑门所拥有的特殊状态. 在这种状态中, 器件对端口表现出较高的阻抗, 因而不对该端口的电平产生贡献. 如果单独测量一个处于高阻态的引脚, 其表现就类似于开路; 如果高阻态的引脚与其他输出相连, 则该线的电平完全由其他输出决定.

三态门通常成对用于同时进行数据的输出与输入的总线上; 一个三态门以设备为输入, 向总线输出; 另一个则相反. 当输出门为高阻态, 输入门为正常态时, 总线不受设备输出的影响, 而设备能接收总线的状态; 反之则设备不受总线的影响, 而总线受设备输出的控制. 还可以将两个三态门都设为高阻态, 则相当于设备从总线上断开. 但是一般不能将两个门同时设为正常态.

具体到 RAM 芯片, 其高阻态应用在数据 IO 线上. 当需要 RAM 向外发送数据时, 外部设为高阻态, 打开 RAM 的输出, 将数据线的控制权交给 RAM; 反之, RAM 输出设为高阻态 (通过设置输出不使能), 输入打开, 将数据线控制权交给外部.

3. 如果希望将 BaseRAM 和 ExtRAM 作为一个统一的 64 位存储器访问, 该如何进行?

让两者共享相同的地址线和控制信号. 然后约定 ExtRAM 为高 32 位, BaseRAM 为低 32 位, 在需要数据操作时做拼合和拆解即可. 代码可能形如:

```
assign external_64b_io = { ext_ram_data, base_ram_data };
assign { base_ram_addr, ext_ram_addr } = 2{external_addr};
assign { ext_ram_be_n, base_ram_be_n } = external_be_n;
assign { base_ram_we_n, ext_ram_we_n } = 2{external_we_n}; // same for oe_n and ce_n
```

4. 总结教学机上 UART 串口和普通的串口芯片 8251 的异同.

相同点

都只接受 8 位数据输入/输出.

都通过一组移位寄存器完成串并转换.

都支持异步通信, 并且以一组信号来标识异步读写状态.

不同点

UART 不支持同步通信, 8251 支持同步通信.

UART 没有调整设置的功能; 8251 有配置字寄存器, 通过向该寄存器写入数据可以控制 8251 的工作模式.

UART 不提供除传输的数据以外的信息; 8251 有状态字寄存器, 从其中读取数据可以获得 3 个额外的错误标识位, 分别表示数据校验失败, 接收器缓存溢出, 异步通信帧错误.

5. 如果要求将 PC 发过来的数据存入 BaseRAM 的某个单元, 然后将其读出, 加 1, 再送回 PC; 该如何操作?

事实上本次实验 3 和实验 4 联合进行完成的就是类似的操作 (只不过没有加 1).

让写内存状态机紧随读串口状态机之后启动, 写串口状态机紧随读内存状态机之后启动 (之间加 1 个节拍用来把数据加 1) 即可.