实验七：LC3 系统-存储器模块设计

1. 实验目的

1.1. 掌握 LC3 系统中存储器的功能和原理

1.2. 掌握存储器仿真时需要初始化的原因

1.3. 了解存储器仿真时初始化的解决方法

2. 实验内容

2.1. 阅读 LC3 存储器实验指南和源代码

3. 实验步骤

本实验为阅读性材料，由于存储器中的主要内容在于如何对其中的内容进行初始化，保证一些必要的功能，不属于 LC3 微结构的范畴，更接近具体的工程实现，因此不做要求。

在构建完整的数据通路前，需要先考虑存储器模块的实现，也就是所谓的内存，内存是一个在程序运行时，保存所有程序指令，以及一些必需数据的存储单元，通常使用静态随机存取存储器（Static Random-Access Memory，SRAM）实现，SRAM 的特点在于访问速度相比于硬盘来说更快，但是在电力供应停止后，里面所存储的数据就会小时，这种特性称为易失性。就设计上来说，存储器就是一个单纯的保存数据的模块，没有其他多余的功能，甚至在之后将 LC3 在 FPGA 上启动时，存储器直接使用了一个第三方的 IP 核。

下面的代码实现了一个最简单的存储器模块，其中接口的含义如下表所示，其中的实现非常的简单，第 11 行定义了一组 32K*16bits 的寄存器堆 data_array，当然这个代码只是参考，现实实现中 data_array 不能够用寄存器搭建，需要使用 SRAM 来替换。第 13 行就是找到 data_array 中读地址位置的数据，再下一拍将其传出。第 15 行就是在写使能有效时，将需要写入的数据写入 data_array 中，这就是完整的存储器模块逻辑了。

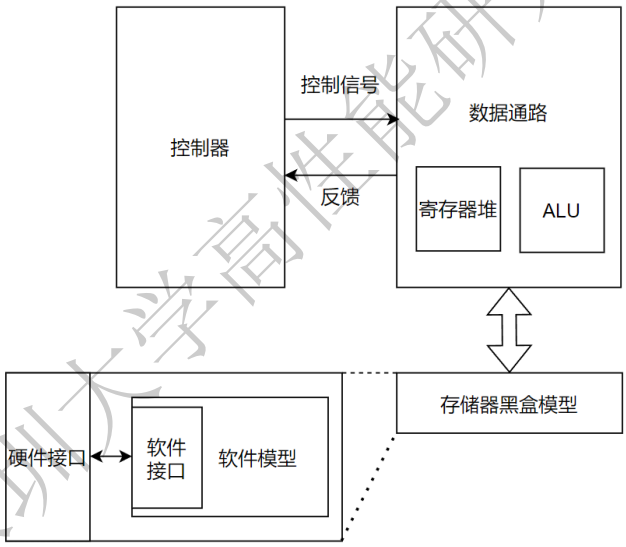
```
1 class lut_mem extends Module {
2     val io = IO(new Bundle {
3         val clk = Input(Clock())
4         val rIdx = Input(UInt(16.W))
5         val rdata = Output(UInt(16.W))
6         val wIdx = Input(UInt(16.W))
7         val wdata = Input(UInt(16.W))
8         val wen = Input(Bool())
9         })
10
11    val data_array = Mem((1<<15), UInt(16.W))
12
13    io.rdata := RegNext(data_array(io.rIdx))
14
15    when (io.wen) { data_array(io.wIdx) := io.wdata }
16 }
```

lut_mem 接口

接口名称	含义
clk	时钟，用于同步存储器中的状态
rIdx	读地址
rdata	从存储器中读出的数据
wIdx	写地址
wdata	需要写入存储器中的数据
wen	写使能，为 true 时才会将 wdata 写入存储器

可以看出，存储器的工作逻辑十分简单，在 LC3 中实现时，主要的难点在于如何对其初始化。在 LC3 系统刚开始仿真运行时，存储器中的所有数据都是 0，但只有指定 LC3 系统需要运行的程序，以及需要运行的程序在存储器中的起始地址后，LC3 才能真正的运行。

在仿真用存储器模块的实现原理上，采用软件模型代替硬件模块以便于初始化。 其软硬件模型如下图所示：



下图定义了存储器模块的接口，端口信号描述和代码定义如下：

端口名称	方向	位宽
读地址1	输入	16 bits
读数据1	输出	16 bits
写地址	输入	16 bits
写数据	输入	16 bits
写使能	输入	1 bits
访存完成	输出	1 bits
内存映射	输入	1 bits

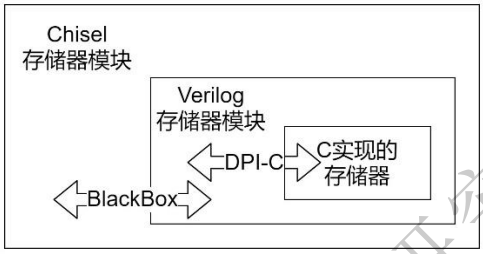
```scala
// src/main/scala/LC3/Memory.scala (49-57)

class MemIO extends Bundle {
  val raddr = Input(UInt(16.W))
  val rdata = Output(UInt(16.W))
  val waddr = Input(UInt(16.W))
  val wdata = Input(UInt(16.W))
  val wen = Input(Bool())
  val R = Output(Bool())
  val mio_en = Input(Bool())
}
```

接下来介绍一些需要用到的知识：DPI-C 和 BlackBox。其中 DPI-C 是 Verilog 为了与其他语言 (C/C++)进行交互而设计的机制，可以实现 Verilog 中调用 C 函数。

而 BlackBox 是 Chisel 语言中的一个类，也就是黑盒，顾名思义，就是一个只知道接口和其主要作用的模块，但是其内部的具体实现可以不用关心。它可以将用 Verilog 语言编写的模块导入进 Chisel 编写的电路中使用。

简单来说，DPI-C 可以使用 C 语言来仿真实现 Verilog 中某个模块的功能，而 BlackBox 可以在 chisel 文件中使用 Verilog 编写的模块。因此，在仿真中，存储器模块的实现如下图所示，使用 C 来模拟实现一个存储器，这样的话就可以在 LC3 仿真开始前，用 C 对齐进行初始化，然后使用 DPI-C，把 C 实现的存储器模块用 Verilog 包裹起来，再使用 BlackBox 将这个 Verilog 模块导入到 Chisel 中使用。注意这种实现方法只能在仿真中使用，之后会介绍在 FPGA 上运行 LC3 时如何对真实的存储器进行初始化。



在我们编写好 LC3 的汇编程序后，需要把程序转换成二进制码，然后将二进制码放入内存的指定位置地址。由于仿真中的存储器模块底层实际是使用 C 语言实现的，其具体的实现代码在 src/test/csrc/ram.cpp 文件中，如下图所示，其实现也非常的简单，主要是在这个文件中定义了一个 C 语言静态的全局变量 ram，ram 本质上是一个一维数组，其中的每个元素是一个 16bits 的无符号整数。这样就可以在 Verilator 的顶层仿真文件中，调用 init_ram 函数，确保在 LC3 仿真正式开始前，存储器中已经初始化完毕，已经写入了需要运行的程序。

```
// src/test/csrc/ram.cpp

#define RAMSIZE 65536           // 存储器大小

typedef uint16_t paddr_t;       // 存储器宽度为16

static paddr_t ram[RAMSIZE];    // 定义存储器数组

// ...

extern "C" void ram_helper(paddr_t rIdx, paddr_t *rdata, paddr_t wIdx, paddr_t wdata, uint8_t wen) {
    int rIdxReg = rIdx;
    *rdata = ram[rIdxReg];          // 存储器读
    if (wen) ram[wIdx] = wdata;     // 存储器写
    //printf("[debug] rIdx=%4x, *rdata=%4x, wIdx=%4x, *wdata=%4x, wen=%x\n", rIdx,  *rdata, wIdx, wdata, wen);
}
```

在我们编写好 LC3 的汇编程序后，需要把程序转换成二进制码，然后将二进制码放入内存的指定位置地址。对于硬件存储器，其初始值均为零，初始化过程就是将二进制码一行一行写进存储器中。在硬件仿真中，可以采用另一种方法——像软件一样把存储器初始化，把存储器看成一个大数组，将二进制码赋值到数组对应地址中。

存储器的读写功能在一个名为 ram_helper 的 C 函数中表达。软件黑盒每周期接收硬件端口的信号转化成 ram_helper 函数的参数，然后执行下列语句。

*rdata = ram[rIdxReg] 将存储器中的值读出来作为硬件端口 rdata 的输出。

If(wen) ram[wIdx] = wdata 当读使能时，将 wdata 赋值到存储器 wIdx 地址中。

接下来在 Chisel 中定义软件黑盒接口，新建一个类 RAMHelper 继承 BlackBox，其中 clk 为时钟信号，剩下五个信号对应硬件前五个端口，硬件软口中的 R 和 mio_en 则用内部硬件逻辑描述。因此用这五个信号即可完成存储器的读写功能

```
// src/main/scala/LC3/Memory.scala
...
class RAMHelper() extends BlackBox {
  val io = IO(new Bundle {
    val clk = Input(Clock())
    val rIdx = Input(UInt(16.W))
    val rdata = Output(UInt(16.W))
    val wIdx = Input(UInt(16.W))
    val wdata = Input(UInt(16.W))
    val wen = Input(Bool())
  })
}
```

最后，编写黑盒模型将上述硬件接口、软件接口连起来，代码如下：

```
// src/main/scala/LC3/Memory.scala
val mem = Module(new RAMHelper())
mem.io.clk := clock
mem.io.rIdx := io.raddr
io.rdata := mem.io.rdata
mem.io.wIdx := io.waddr
mem.io.wdata := io.wdata
mem.io.wen := io.wen
```

至此，用于仿真的 LC3 存储器以及定义完成。这个存储器虽然在硬件设计电路中作为存储器模块，可以用于和其他硬件电路交互，但是其内部是用 C 实现的软件逻辑，这类代码只能够用于仿真，不能够真正制作成物理电路。之后的实验中，将 LC3 在 FPGA 上运行时，会使用第三方的 SRAM IP 核作为存储器模块，同时实验材料中会给出用于初始化存储器模块的文件，FPGA 会将这个初始化文件写入 Flash 中，FPGA 启动时依靠这个初始化文件来初始化存储器模块。