实验九：LC3 系统整体仿真

1. 实验目的：

1.1. 掌握 Boot 模块的工作原理

1.2. 掌握 LC3 顶层 Top 模块的设计

1.3. 能够对 LC3 系统进行整体仿真

2. 实验内容：

2.1. 学习 Boot 模块的功能和工作机制

2.2. 学习 LC3 系统顶层的 Top 模块设计

2.3. 学习 LC3 整体仿真的相关知识

2.4. 输出 LC3 仿真波形以及查看波形

3. 实验步骤：

3.1. Boot 模块介绍

在之前的几次实验中，已经介绍了 LC3 系统中各主要模块的相关功能逻辑，在这次实验中，需要将所有的模块合并起来，形成一个完整的系统，并对其进行整体的仿真，不同于之前的单元测试，整体仿真需要所有的模块互相配合，真正地运行 LC3 指令集的程序。在仿真之前，LC3 系统中还有最后的一些模块需要介绍。

Boot 模块在整个 LC3 系统中负责启动时的程序动态加载功能，换言之就是告诉 LC3 系统应该执行什么程序。当 LC3 系统启动后，首先需要一个起始地址，将其视为程序的开头，从起始地址存储的指令开始，不断向后执行一条条的指令。但是当系统刚启动时，内存中的数据都是随机的，LC3 并不知道自己要从哪里开始执行指令，也不知道要执行的指令有哪些，因此需要一个模块，在 LC3 真正开始运行前，对内存进行初始化，将需要运行的程序写入内存，并且告诉 LC3 从哪个地址开始执行程序，这便是 Boot 模块的主要功能。

如表 9.1 所示，列举了 Boot 模块的接口以及各自的作用。我们希望 LC3 系统能够运行不同的程序，而不是固定的某一个程序，因此在 LC3 系统运行前，可以先通过 UART 协议，将 LC3 编译器编译生成的 obj 文件传输给 LC3，Boot 将程序指令初始化进内存中，这样就可以动态的指定 LC3 需要运行的程序，因此 Boot 模块需要接收 UART 模块的输入，对应的就是 uartRx 接口的作用。

在 LC3 的程序文件中，前 2Byte 的数据默认是整个程序的起始地址，Boot 模块先读取 2Byte 的数据，之后依据这个数据作为程序起始地址，向内存中顺序写入指令，这就是 initMem 接口的作用，DataPath 也需要获取程序的起始地址为 pc 寄存器做初始化，这就是 initPC 接口的作用。

在内存初始化完成前，LC3 系统是没有运作的，因此在内存初始化完成后，Boot 模块需要给 LC3 的 Controller 模块一个信号，告诉它可以开始运行程序了，这就是 work 接口的作用。

表 9.1 Boot 模块接口及作用

接口名称	作用
uartRx	从 UART 模块接收需要运行的程序指令
work	告诉 LC3 系统初始化是否完成

initPC	告诉 DataPath 程序的起始地址
initMem	向内存中写入需要运行的程序

3.2. LC3 系统 Top 模块设计

如图 9.1 所示，是 LC3 系统顶层 Top 模块的示意图，Top 模块在 Top.scala 文件中定义，是包括了 LC3 所有相关模块的顶层模块，负责连接其中的各大主要模块。

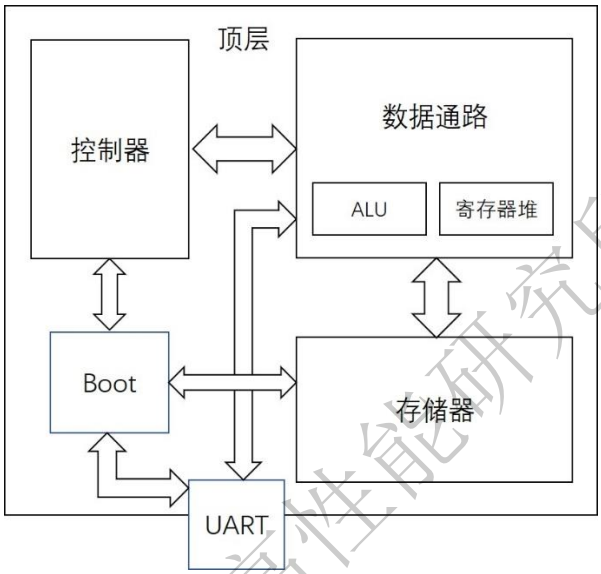

图 9.1 LC3 系统顶层模块图

在 Top.scala 中有一个配置选项 FPGAPlatform，用于配置 LC3 项目是否生成 FPGA 运行的 verilog 文件，为 true 时生成用于 FPGA 的 veirlog 文件，为 false 时生成用于仿真的 verilog 文件。这影响到内存和 UART 是使用仿真手段实现还是使用真实的物理设计实现。

Top 模块的接口非常的简单，只有两根 1bit 的信号线。因为 UART 协议是 LC3 接收输入输出的唯一手段，因此 Top 的接口只需要一根接收 UART 信号的线和一根传输 UART 信号的线即可。

Top 中定义的模块如图 9.1 所示，主要是 Controller（控制器），DataPath（数据通路），Memory（存储器），Boot，Uart_Tx 和 Uart_Rx 模块。其中 Uart_Tx 和 Uart_Rx 串口端分别连接 Top 模块的两根接口信号，Uart_Tx 数据端直接连接到 DataPath 中的 UART 传输接口上即可，而 Uart_Rx 的数据端需要根据情况来选择，在系统刚启动时，它需要与 Boot 模块连接，接收需要运行的程序指令存入内存中，当系统初始化完成，正式开始运行程序时，它需要与 DataPath 连接，作为程序中的常规输入通道来使用。

之后其他的模块如图 9.1 中的箭头所示，各个模块之间同名的接口相互连接，在 Chisel 语法中，这种情况可以方便地直接使用模块.io<>模块.io 来将所有同名信号相互连接。

3.3. LC3 系统整体仿真

在 chisel_lc3 目录下运行"make emu"命令，可以看到如图 9.2 所示的输出，

其中在 Mem init finished, LC3 start work 之前，属于内存初始化阶段，在这段话输出之后，代表 LC3 正式开始运行，在不添加其他参数的情况下，默认会运行一个简单的 hello 程序，就是通过 UART 端口输出 hello 字符串，在输出完成，程序结束后，就可以按 Ctrl+C 终止仿真。（由于在终端中显示 UART 输出的功能没有做性能优化，因此可能运行的很慢）

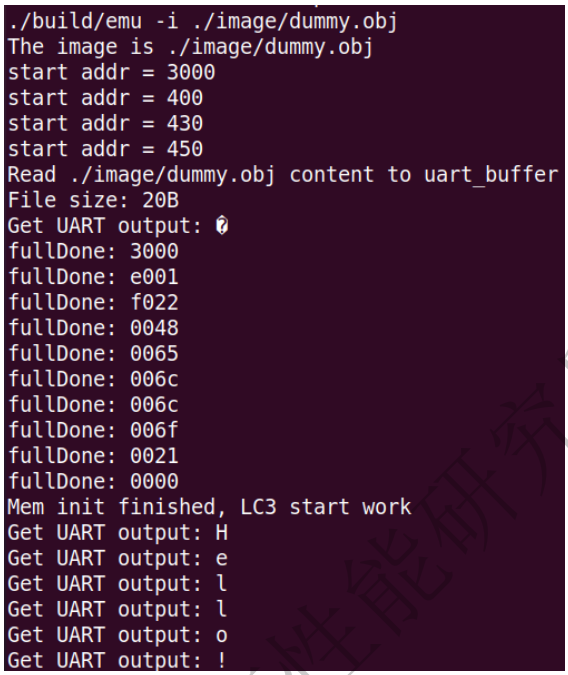

图 9.2 LC3 仿真输出 Hello

在 chisel_lc3 项目的 image 目录下，有许多文件，其中后缀名为*.asm 的文件是 LC3 的汇编文件，可以直接用编辑器打开，而后缀名为*.trap 的文件是实现 LC3 中 trap 功能的程序的可执行文件，主要是 GETC，OUT，PUTS 三个指令，其功能如表 9.2 所示，这些指令其实也是一个个汇编程序，感兴趣的可以查看同名的 asm 文件，学习如何实现这些基础指令。

表 9.2 LC3 TRAP 指令及作用

指令	作用
GETC	接收 1Byte 的数据输入，将其存入寄存器 R0 低 8 位中
OUT	将寄存器 R0 的低 8 位输出
PUTS	输出一个字符串，字符串的起始地址存在 R0 中

Image 目录下还有一个 lc3as 文件，这是一个终端的 LC3 编译器，可以将汇编文件编译成后缀名为*.obj 的汇编文件，例如在 image 目录下运行" lc3as dummy.asm"命令，就可以看到如图 9.3 所示的输出，在 image 目录下会生成 dummy.sym 和 dummy.obj 2 个文件，其中 dummy.obj 就是 LC3 可以直接执行的程序文件。

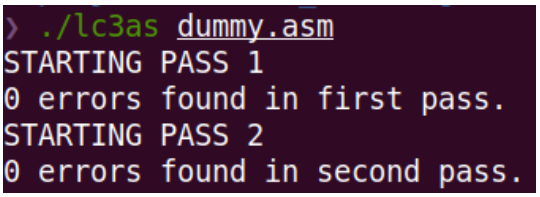

图 9.3 编译 dummy.asm

dummy.asm 就是在执行" make emu"时 LC3 默认执行的程序，在 image 下还给出了 echo.asm 程序，这个程序会不断等待用户输入字符串，并且将字符串原样输出，通过添加 IMAGE 参数即可运行，尝试运行" make emu IMAGE=echo"，在输出 Mem init finished, LC3 start work 后，尝试输入一些字符串，然后等待 LC3 的输出，如图 9.4 所示。如果想要运行自己写的程序，也可以将 asm 文件放到 image 目录下，使用 IMAGE 参数指定程序名仿真即可。

实验九-任务一：仿真运行 LC3，成功执行 dummy 程序，在终端中看到如图 9.2 所示的输出。也可以尝试自己编写一个带输入输出的汇编程序（例如输入一个数，将其加 1 后输出），然后在 LC3 上仿真运行

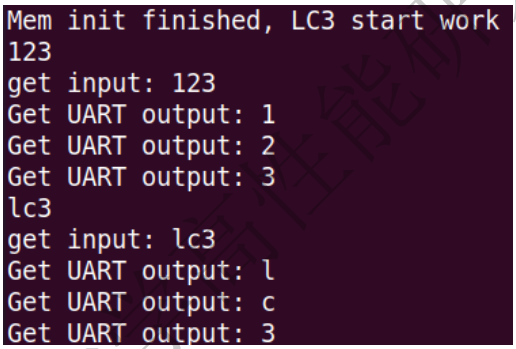

图 9.4 LC3 仿真运行 echo 程序

仿真运行指定程序的命令非常简单，接下来要介绍 IMAGE 这个参数是如何实现指定程序的，首先打开 chisel_lc3 项目的 Makefile 文件，虽然看起来比实验二的 makefile 复杂，但实际上主要内容就是类似的，相关的语法实验二已经介绍过了。主要看 IMAGE 参数，通过传入 IMAGE 参数，Makefile 会找到对应的 obj 文件，并且将 obj 文件的路径通过仿真文件的-i 参数传入，obj 文件的路径会传入到 Verilator 的顶层仿真函数中，也就是 src/test/csrc/wrapper.cpp 文件中的 main 函数，在 main 函数中调用 parse_args 函数，将传入的 obj 文件路径存入一个 image 字符数组中，最后调用仿真中的 init_ram 函数，将这个 obj 文件中的指令初始化进内存中执行。因此虽然仿真时只有一个参数，但是实际上它涉及到了 Makefile 传参，C++传参和 Chisel 的内存初始化 3 个部分。

3.4. 仿真波形输出与查看

在仿真时程序运行顺利，就能够在终端看到期望的输出结果，但是当设计中出现错误时，就需要调试修改设计代码，在 C 语言中可以使用 Virtual Stuio 或者 GDB 等调试工具来调试，但是在硬件设计的仿真中，却没有这类方便好用的工具，甚至没办法在出错时指出错误的明确位置。在硬件设计中，最主要的调试方式还

是查看波形 (waveform)，即在电路设计仿真时，每个时钟周期将电路中所有信号的值都保存下来，然后查看电路中每个数据信号的值，判断是哪部分模块功能没有如预期的工作。

LC3 系统同样支持在仿真时输出波形，通过仿真时添加 TRACE 参数，即可开启波形，即执行"make emu TRACE=-t"命令，在仿真过程中会将波形存储在./build/emu.vcd 文件中。不过需要注意的是，由于波形文件需要保存电路设计中所有的信号，因此在仿真出现错误，或者仿真结束后，要及时停止仿真，否则可能会由于波形文件过大，而占满所有磁盘的存储空间。

波形文件可以直接用编辑器打开，但是由于信号太多，因此仍然需要使用专门查看波形的工具，本实验使用 GTKWave，这是一个开源免费的波形文件查看器，支持 Verilog VCD/EVCD 文件格式，提供了友善易用的图形界面，支持 Windows/Linux/Mac 多平台。在 Linux 环境下想要安装 GTKWave 非常的简单，只要运行"sudo apt install gtkwave"即可，在安装完成后，在终端中输入"gtkwave ./build/emu.vcd"，之后，会弹出如图 9.5 所示的窗口，这就是 GTKWave 的主要界面。

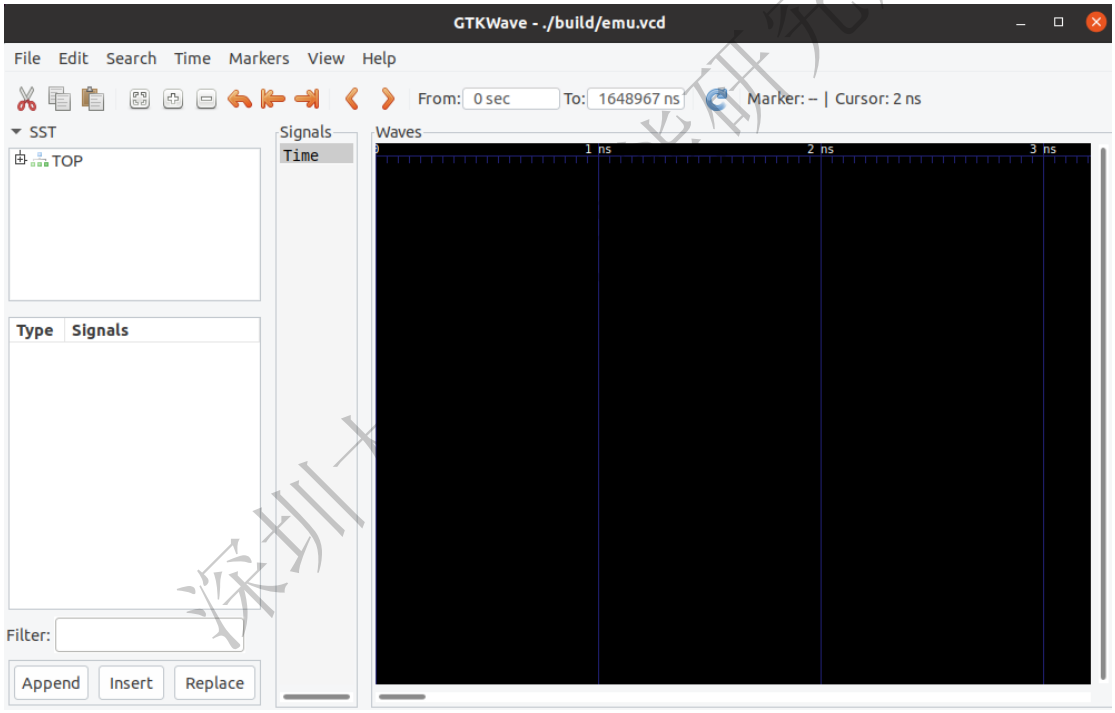

图 9.5 GTKWave 主要界面

GTKWave 的主界面可以分为 5 个部分，如图 9.6 所示，其中红色框的部分是模块结构窗口，点击模块左边的加号按钮，可以展开查看这个模块内部的子模块，在这里可以清晰地看到 LC3 系统模块之间的结构。在模块窗口中左键选中一个模块后，在蓝色框框选的范围内，可以看到被选中模块的所有接口信号和内部信号的名称，下方还提供了一个搜索框，可以在 Filter 搜索框中搜索想要的信号，来快速找到需要的信号。在信号选中窗口选中想要查看波形的信号，直接按住左键，将其拖动到绿色窗口中，这个信号就会出现在绿色框框选的窗口中，同时会在黄色的波形窗口中显示出这个信号的波形，即在仿真过程中值的变化。通常在一个信号刚被添加进查看信号窗口中时，波形查看窗口中信号的显示比例是不正常的，

需要点击紫色框工具栏里从左往右第 4 个按钮，将波形调整到一个适合查看的比例，然后就能够开始进行调试，检查电路设计中的错误了。

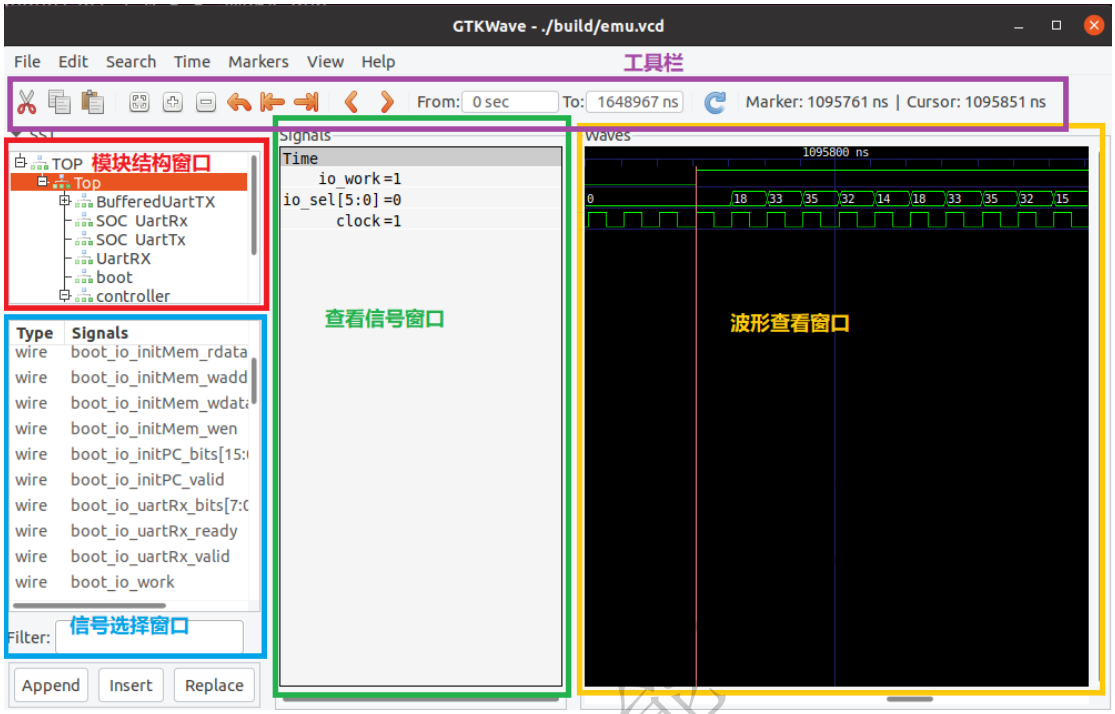

图 9.6 GTKWave 窗口介绍

接下来使用 GTKWave 打开仿真运行 dummy 程序生成的波形文件，并于汇编文件和 LC3 的状态机相比较，查看波形中 LC3 系统的控制器状态变化情况。实验时最终看到的波形应该与图 9.6 相同。首先在模块结构窗口中选中最顶层的 TOP 模块，在信号选择窗口中找到 clock 信号，拖动到查看信号窗口中，正如实验二中所说，Chisel 中所有 Module 类型的模块，都会有一个隐式的 clock 和 reset 接口信号，如果是 RawModule 类型则没有。拉出 clock 信号可以在查看其他波形是更加直观的看到数值随着周期的变化。接下来找到 Controller 模块，找到 state 信号，将其拖入到信号查看窗口中，首先可以看出，在波形中前面很大一部分时间里，state 的值都是 0，这是由于在程序真正执行前，需要初始化内存的原因，为了验证这一说法，找到 Boot 模块中的 io_work 信号，可以看到，在 work 信号为 0 时，表示系统在内存初始化阶段，当 work 信号变为 1 后，LC3 系统开始工作，Controller 模块中的 state 寄存器也开始改变自己的状态。不过此时会发现 state 寄存器的状态值与 LC3 的状态转移图对不上，这是由于默认波形值是使用十六进制表示的，而 LC3 的状态转移图是十进制，因此需要在信号查看窗口中选中 state 信号，在右键菜单中选择 Data Format->Decimal，state 信号就会转为十进制表示。

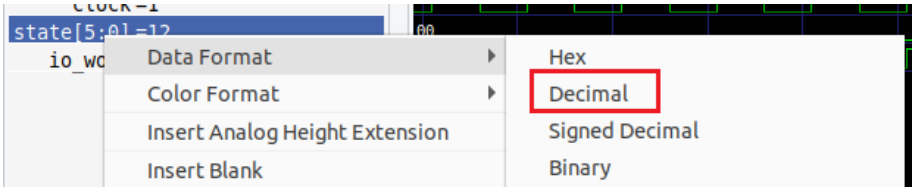

图 9.7 修改信号值进制

实验九-任务二：查看 UART 信号的波形，找到通过 UART 传输"hello!"字符串的过程