中国科学技术大学计算机学院
《数字电路实验》报告

实验题目：在 FPGAOL 平台上实现 myISA

学生姓名：黄瑞轩

学生学号：PB20111686

完成日期：2021.12.14

计算机实验教学中心制

2020 年 09 月

实验题目

在 FPGAOL 平台上实现 myISA

实验目的

- 学习有限状态自动机的编写
- 学习 FPGAOL 平台上串口的使用

实验环境

- FPGAOL 实验平台：fpgaol.ustc.edu.cn
- Vivado 工具

实验设计

【设计概述】

在这个实验中，我将依托 fpgaol 在线平台，以有限状态自动机的方式实现一个支持编程的 myISA 指令集。

【详细设计】

本实验设计的状态机有如下 4 个状态：

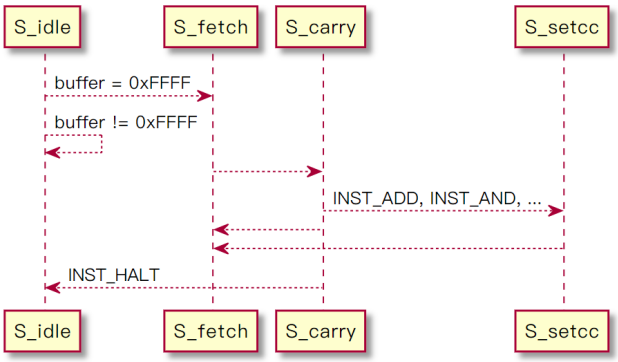
```
S_idle  = 2'd0;      静止状态
S_fetch = 2'd1;      取指令状态
S_carry = 2'd2;      执行指令状态
S_setcc = 2'd3;      设置条件码状态
```

用户通过串口在静止状态输入指令，指令格式如下：

```
01010...01010101     输入一个 16bit 指令，最后一行指令必须是 HALT
1111111111111111     表示包括 HALT 在内的所有指令都输入完毕，开始执行
```

设置一个串口缓冲区寄存器 buffer[16:0]和指针 buffer_ptr 指向下一个待写入的 buffer 位置，buffer_ptr 从 buffer 高位写到低位，即初始状态是 buffer_ptr = 15。每当 rx_data 不是换行符，就向 buffer 中写，否则初始化 buffer_ptr = 15。当读到换行符时，说明这一行指令已经被完全读到 buffer 中，然后需要判断：如果 buffer 的内容是 0xFFFF，表示用户输入结束了，不应该写入内存中；否则需要将 buffer 内容存入 mem[PC]，然后使得 PC 自增。

状态机的第一部分是现态（cs）和次态（ns）的转换部分，遵从如下状态图：



简单解释一下：S_idle 状态下，需要对 buffer 进行判断以切换到 S_fetch 状态，否则保持 S_idle 状态；S_fetch 状态下，无条件切换到 S_carry 状态；S_carry 状态需要判断 IR 中操作码以确定是否跳转到 S_setcc 状态和 S_idle 状态（操作码为 INST_HALT 时），否则跳转到 S_fetch 状态；S_setcc 状态无条件跳转到 S_fetch 状态。

状态机的第二部分是时序状态转换部分。在每个 posedge clk，此部分使 cs 转化为 ns 状态。如果 cs 是 S_idle 而 ns 是 S_fetch，保险起见还要将 buffer 清零。

状态机的第三部分是逻辑处理部分。这一部分指示了在各个状态我们的状态机还要执行的其他的操作。

- **S_idle**

首先是展示寄存器状态。通过开关 sw[7:0]以独热码形式来控制七段数码管显示内容，如 sw[7:0] = 8'b10000000 表示七段数码管应当展示寄存器 R7 的内容，格式为：R700XXXX，XXXX 为 R7 内容的十六进制编码。

其次是完成对 buffer 缓冲区的写入和对内存指令的写入。

- **S_fetch**

将 mem[PC]读入 IR，并使 PC 自增。

- **S_carry**

根据 IR 内容完成相应操作。这里的执行操作以 LC-3（以 *Introduction to Computing Systems: from Bits & Gates to C/C++ & Beyond, 3rd edition* 书中所述为准）为基础，将未定义的 unused 指令定义为 INST_RSF，即右移一位指令（高位补逻辑 0），并修改条件码。

- **S_setcc**

根据 IR 中 DR 寄存器内容设置条件码。

功能测试

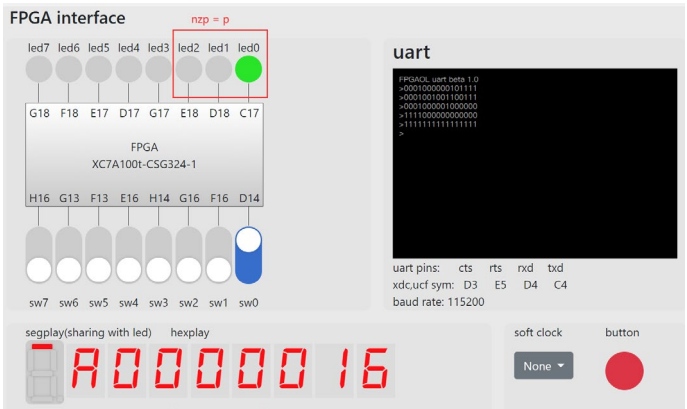
因为这是片上系统的实现，自然用一些示例程序来做测试较为合理，下面列出几种示例程序来测试程序功能。

（1）简单加法

串口输入：0001000000101111 // R0 = 0xF
0001001001100111 // R1 = 0x7
0001000001000000 // R0 = R1 + R0
1111000000000000 // HALT
1111111111111111 // RUN

期望结果：R0 == 0x16，CC == 3'b001

实际结果：

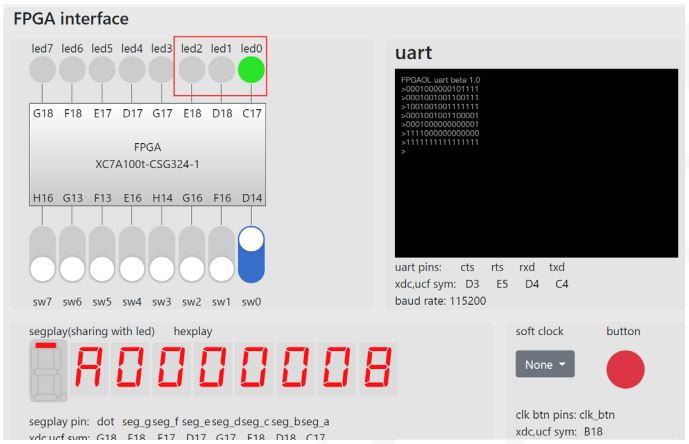


（2）简单减法

串口输入：0001000000101111 // R0 = 0xF
0001001001100111 // R1 = 0x7
1001001001111111 // R1 = ~R1
0001001001100001 // R1 = R1 + 1
0001000000000001 // R0 = R0 + R1
1111000000000000 // HALT
1111111111111111 // RUN

期望结果：R0 == 0x8，CC == 3’b001

实际结果：

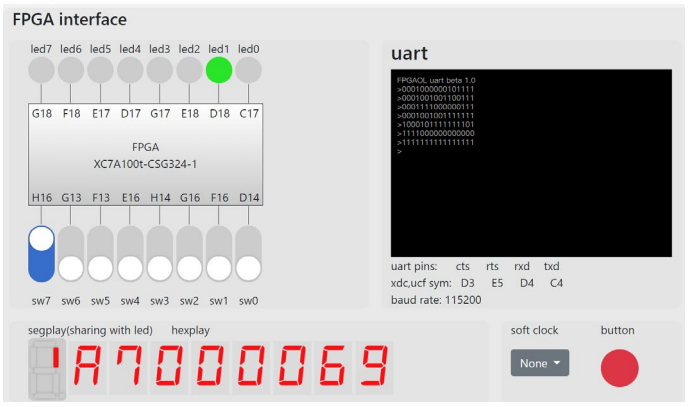


（3）乘法

串口输入：0001000000101111 // R0 = 0xF
0001001001100111 // R1 = 0x7
0001111000000111 // R7 = R0 + R7(*)
0001001001111111 // R1 = R1 – 1
1000101111111101 // if(R1 != 0) goto (*)
1111000000000000 // HALT
1111111111111111 // RUN

期望结果：R7 == 0x69，CC == 3’b010

实际结果：

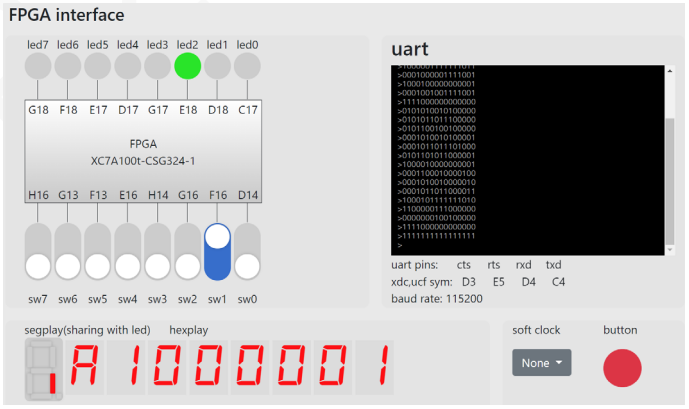


（4）求除以 7 的余数

串口输入：0010001000010101
0100100000001000
0101010001100111
0001001010000100
0001000001111001
1000001111111011
0001000001111001
1000100000000001
0001001001111001
1111000000100101
0101010010100000
0101011011100000
0101100100100000
0001010010100001
0001011011101000
0101101011000001
1000010000000001
0001100010000100
0001010010000010
0001011011000011
1000101111111010
1100000111000000
0000000100100000 // #288
1111000000000000 // HALT
1111111111111111 // RUN

期望结果：R1 == 0x1 [288≡1 (mod 7)]

实际结果：



（5）用 INST_RSF 实现逻辑右移

串口输入：0001000000101111 // R0 = 0xF

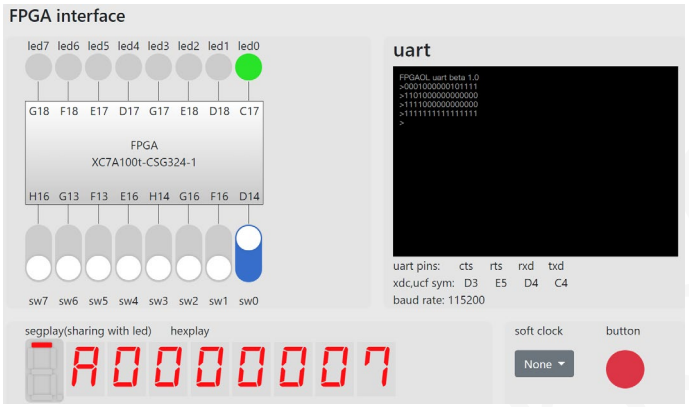
1101000000000000 // R0 = R0 >> 1

1111000000000000

1111111111111111

期望结果：R0 == 0x7

实际结果：



经过以上测试，已经可以大致确认本实验成果具有相当的稳定性和正确性。

总结与思考

- 本次实验中我学会了除了 sw[7:0]、button 之外另一种和 FPGA 交互的方式：串口。并且这种交互方式自由度极大，几乎可以利用其实现任何交互功能。
- 本次实验是在本学期课程《计算系统概论 A》所学内容上的扩充，由于工作量较大、逻辑设计容量较大，导致本次实验的难度较大。
- 本次实验的任务量是视所选题材的具体内容而定的。衡量本实验，需要从创新性和复杂性中找到一种平衡。有的题材虽然任务量很重，但是大多是重复的操作，用任务量来衡量就毫无意义。而本题材灵感取自课本所学，在这之上加以扩充，实现在这门课的编程中较难实现的一些指令，如右移运算。
- 改进建议：详细介绍一下 fpgaol 平台上的串口使用，原使用文档中的串口还是实体外设，和虚拟平台略有出入。

附件

myISA.v //设计文件

myISA-xdc.xdc //约束文件