**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**

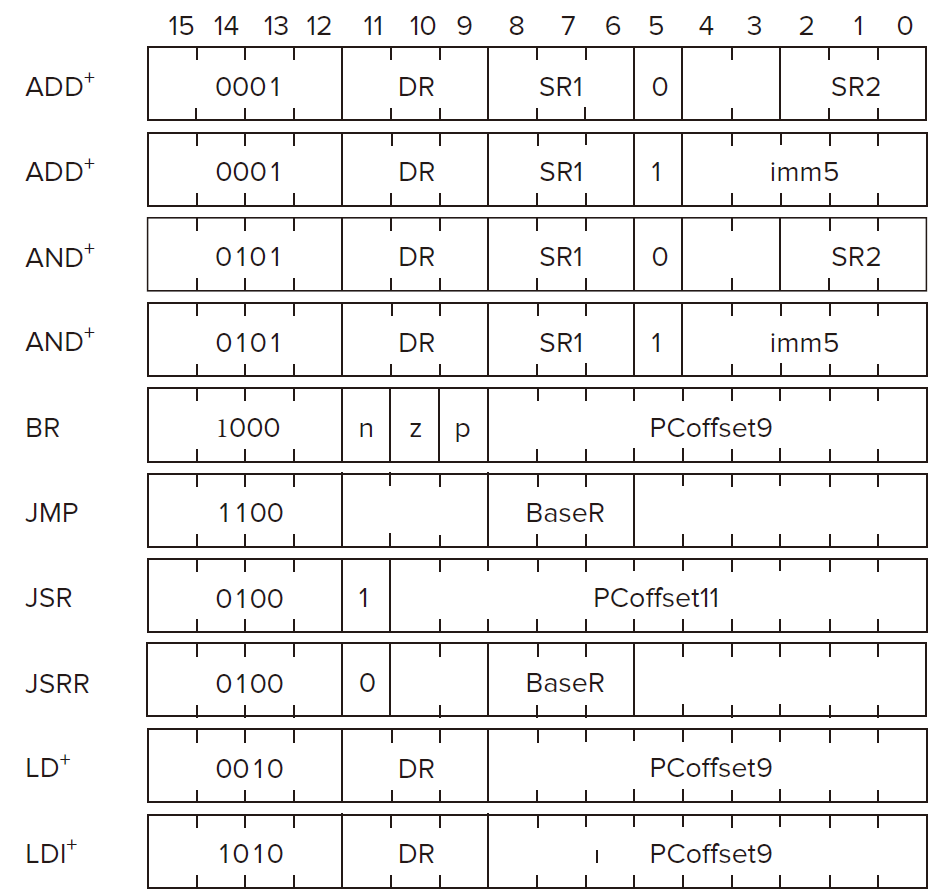
**【设计概述】**

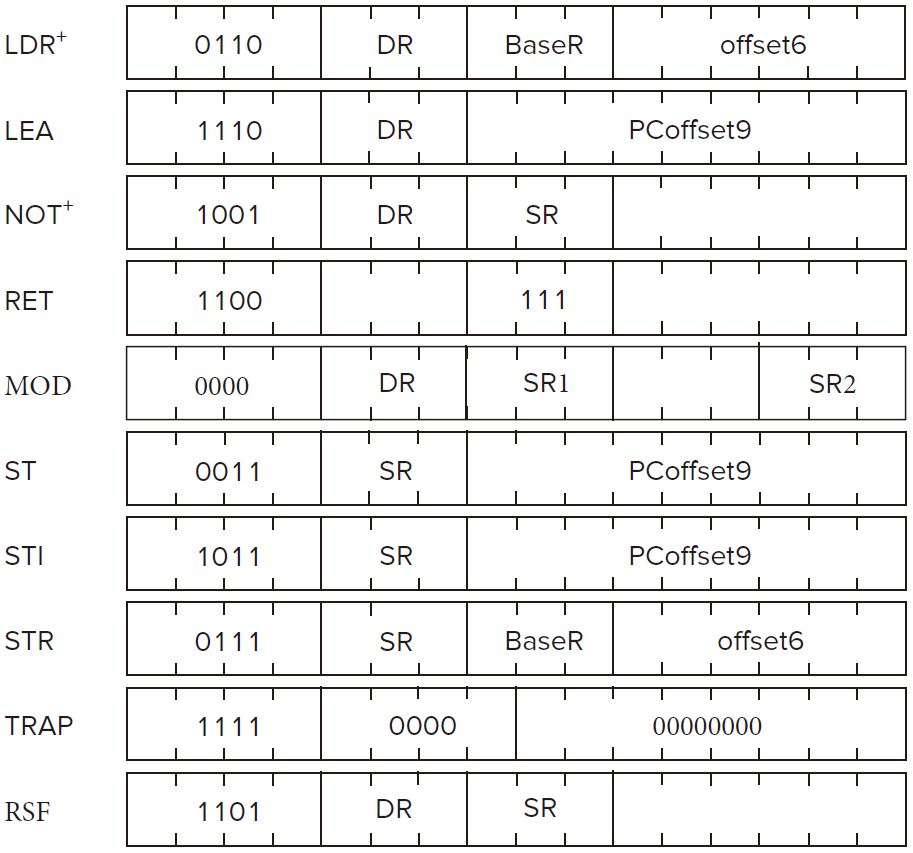
在这个实验中，我将依托fpgaol在线平台，以有限状态自动机的方式实现一个支持编程的myISA指令集。用户可以通过查阅myISA指令表来操作内存和临时变量寄存器的值来进行编程。

myISA给出了内存组织方式、寄存器组、指令集（包括操作码、数据类型、寻址模式）等信息。myISA的可寻址空间大小是64，寻址基本单位是16位。myISA和大多数的机器一样，提供了临时存储空间（R[7:0]）。

myISA中的一条指令分为两个部分：操作码（做什么）和操作数（对谁操作）。所有指令可以分为三类：运算（operate）、数据搬移（data movement）和控制（control）。运算类指令负责处理信息；数据搬移类指令则负责在内存和寄存器之间以及内存/寄存器和IO设备之间转移信息；控制类指令负责改变指令执行的顺序，即它们能让程序随时跳转至另一个地方继续执行（而不是常规的顺序向下执行）。

**myISA指令表**

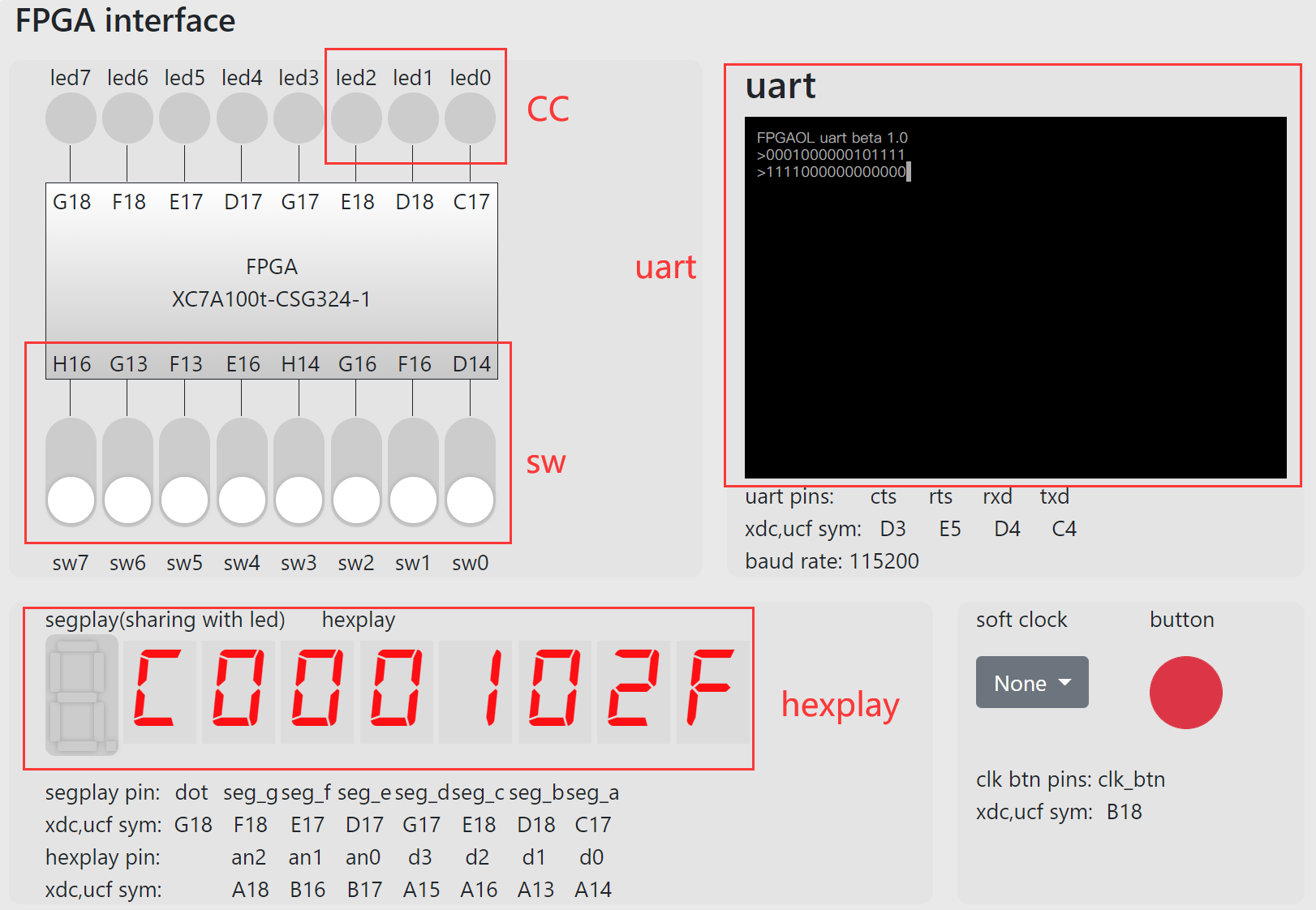




**每条指令的用途和示例**

|  |  |  |
| --- | --- | --- |
| INST\_ADD | 0001 001 001 0 XX 001  0001 001 001 1 11111 | 将R1与R1相加，结果放在R1中  将R1与立即数-1相加，结果放在R1中 |
| INST\_AND | 与INST\_ADD类似，功能是相与 | |
| INST\_BR | 1000 010 000000011 | 检测条件码是否为CC = 3’b010，如果是，则PC <= PC + 3 |
| INST\_JMP | 1100 XXX 101 XXXXXX | 令PC <= R[5] |
| INST\_JSR | 0100 1 00000000001  0100 0 XX 101 XXXXXX | 把当前的PC存到R7，同时令PC <= PC + 1  把当前的PC存到R7，同时令PC <= R[5] |
| INST\_LD | 0010 011 000000011 | 把mem[PC + 3]的值存到R[3]里 |
| INST\_LDI | 1010 011 000000011 | 把mem[mem[PC + 3]]的值存到R[3]里 |
| INST\_LDR | 0110 011 010 000011 | 把mem[R[2] + 3]的值存到R[3]里 |
| INST\_LEA | 1110 111 111111111 | 把PC – 1的值存到R[7]里 |
| INST\_NOT | 1001 111 000 XXXXXX | 把~R[0]存到R[7]里 |
| INST\_MOD | 0000 111 110 XXX 101 | 把R[6]模R[5]的值存到R[7]里 |
| INST\_ST | 0011 111 111111111 | 把R[7]的值存到mem[PC - 1]处 |
| INST\_STI | 1011 111 111111111 | 把R[7]的值存到mem[mem[PC - 1]]处 |
| INST\_STR | 0111 111 110 111111 | 把R[7]的值存到mem[R[6] - 1]处 |
| INST\_HALT | 1111 000000000000  1111 111111111111 | 程序运行到此结束  机器开始从mem[0]执行所有指令 |
| INST\_RSF | 1101 111 110 XXXXXX | 把R[6]右移一位的值存到R[7] |

**FPGAOL各功能区使用说明**



CC：条件码显示区，显示当前条件码（高电平有效）。

hexplay：当sw[7:0]全为0时，显示上一条输入进内存的指令（以低4位16进制呈现）；当sw[7:0]有1时，显示为1的最低位为编号的寄存器内容。

sw：开关，控制hexplay的显示内容。

uart：串口输入，输入指令。

**【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

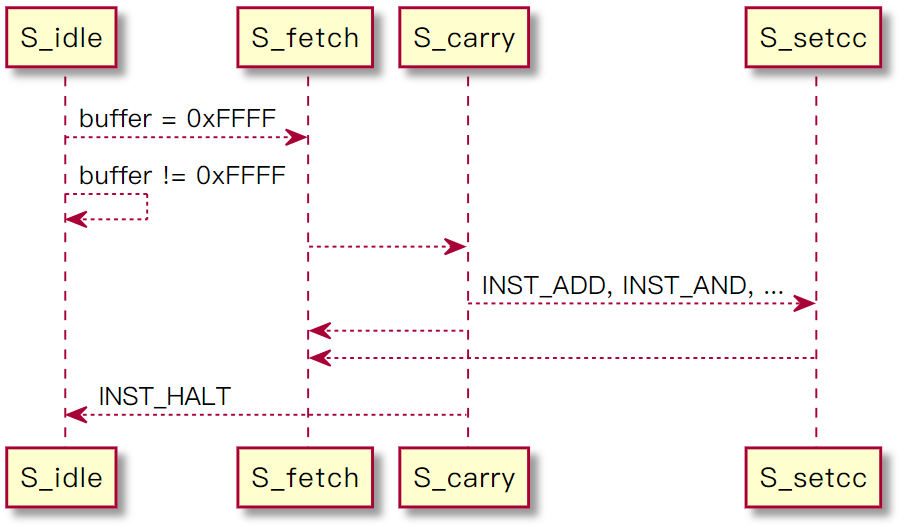
用户通过串口在静止状态输入指令，指令格式如下：

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；将在此不会用到的INST\_RTI重定义为INST\_MOD，可以方便地取余数。各指令具体的实现操作如下所示。

      INST\_ADD:begin

          if(IR[5] == 1'b1)begin

              R[IR[11:9]] <= R[IR[8:6]] + {{11{IR[4]}}, IR[4:0]};

          end

          else begin

              R[IR[11:9]] <= R[IR[8:6]] + R[IR[2:0]];

          end

      end

      INST\_AND:begin

          if(IR[5] == 1'b1)begin

              R[IR[11:9]] <= R[IR[8:6]] & {{11{IR[4]}}, IR[4:0]};

          end

          else begin

              R[IR[11:9]] <= R[IR[8:6]] & R[IR[2:0]];

          end

      end

      INST\_BR:begin

          if((IR[11] & CC[2]) | (IR[10] & CC[1]) | (IR[9] & CC[0])) begin

              PC <= PC + IR[5:0];

              R[3'b110] <= 16'hFFFF;

          end

      end

      INST\_JMP:begin

          PC <= R[IR[8:6]][5:0];

      end

      INST\_JSR:begin

          R[7] <= PC;

          if(IR[11]) PC <= PC + IR[5:0];

          else PC <= R[IR[8:6]][5:0];

      end

      INST\_LD:begin

          R[IR[11:9]] <= mem[PC + IR[5:0]];

      end

      INST\_LDI:begin

          R[IR[11:9]] <= mem[mem[PC + IR[5:0]][5:0]];

      end

      INST\_LDR:begin

          R[IR[11:9]] <= mem[R[IR[8:6]][5:0]+IR[5:0]];

      end

      INST\_LEA:begin

          R[IR[11:9]] <= PC + IR[5:0];

      end

      INST\_NOT:begin

          R[IR[11:9]] <= ~R[IR[8:6]];

      end

      INST\_ST:begin

          mem[PC + IR[5:0]] <= R[IR[11:9]];

      end

      INST\_STI:begin

          mem[mem[PC + IR[5:0]][5:0]] <= R[IR[11:9]];

      end

      INST\_STR:begin

          mem[R[IR[8:6]][5:0] + IR[5:0]] <= R[IR[11:9]];

      end

      INST\_RSF:begin

          R[IR[11:9]] <= {1'b0, R[IR[8:6]][7:1]};

      end

      INST\_TRAP:begin

          //PC <= ENDL;

      end

      INST\_MOD:begin

          R[IR[11:9]] <= R[IR[8:6]] % R[IR[2:0]];

      end

* 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。（上电时，所有临时寄存器R[7:0]的值被初始化为0）

**（1）简单加法**

串口输入：0001000000101111 // R0 = R0 + 0xF

0001001001100111 // R1 = R1 + 0x7

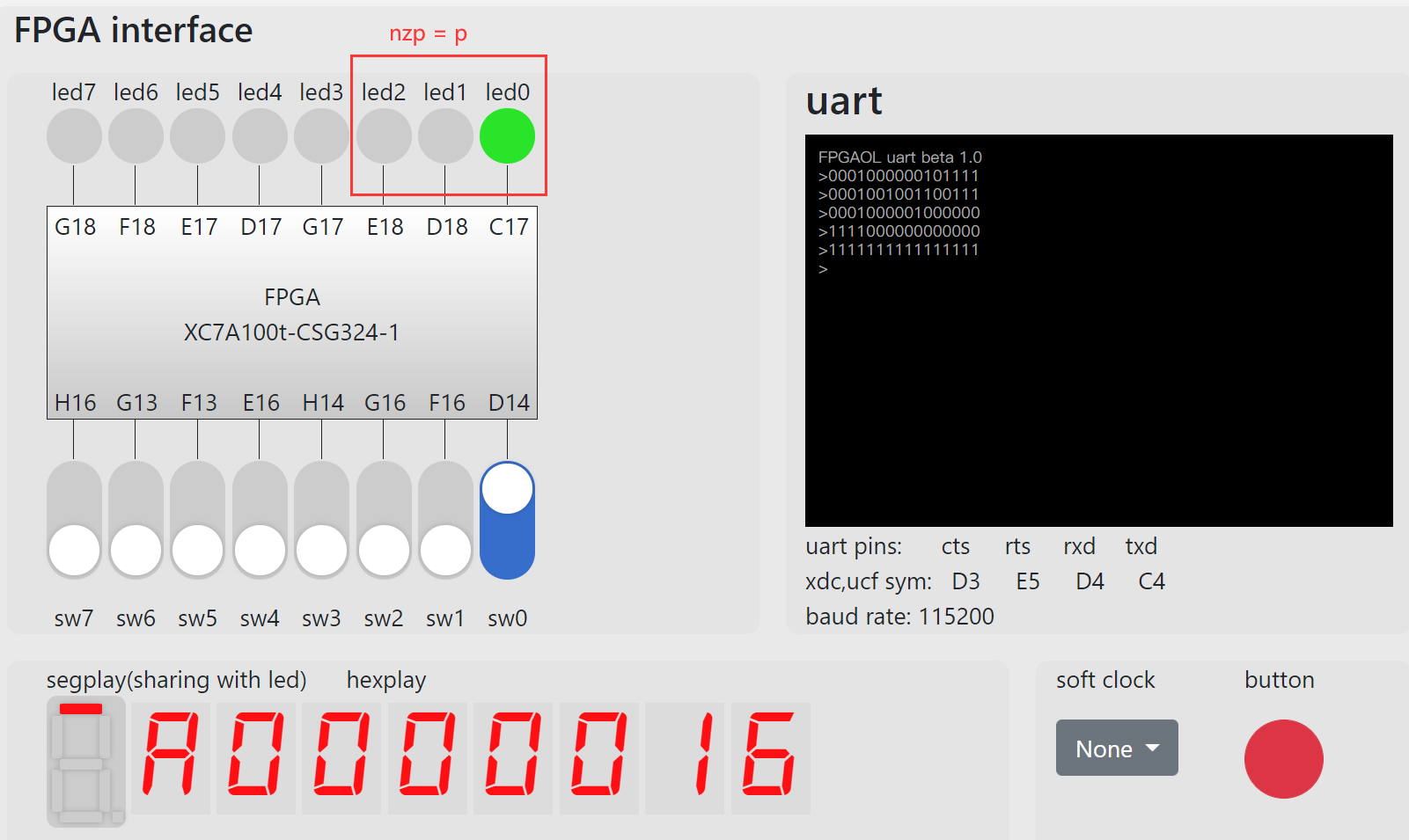
0001000001000000 // R0 = R1 + R0

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 = R0 + 0xF

0001001001100111 // R1 = R1 + 0x7

1001001001111111 // R1 = ~R1

0001001001100001 // R1 = R1 + 1

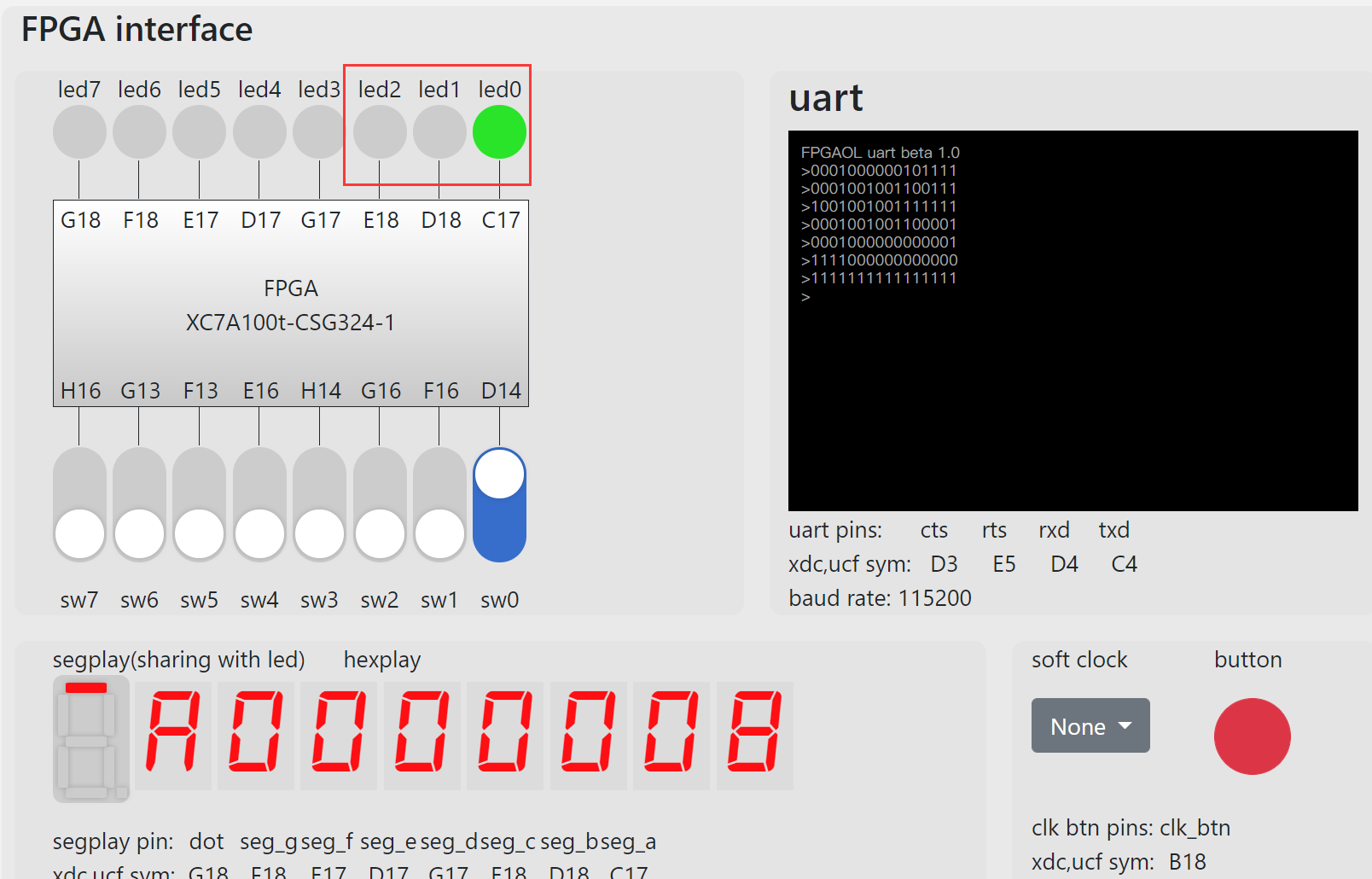
0001000000000001 // R0 = R0 + R1

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 = R0 + 0xF

0001001001100111 // R1 = R1 + 0x7

0001111000000111 // R7 = R0 + R7(\*)

0001001001111111 // R1 = R1 – 1

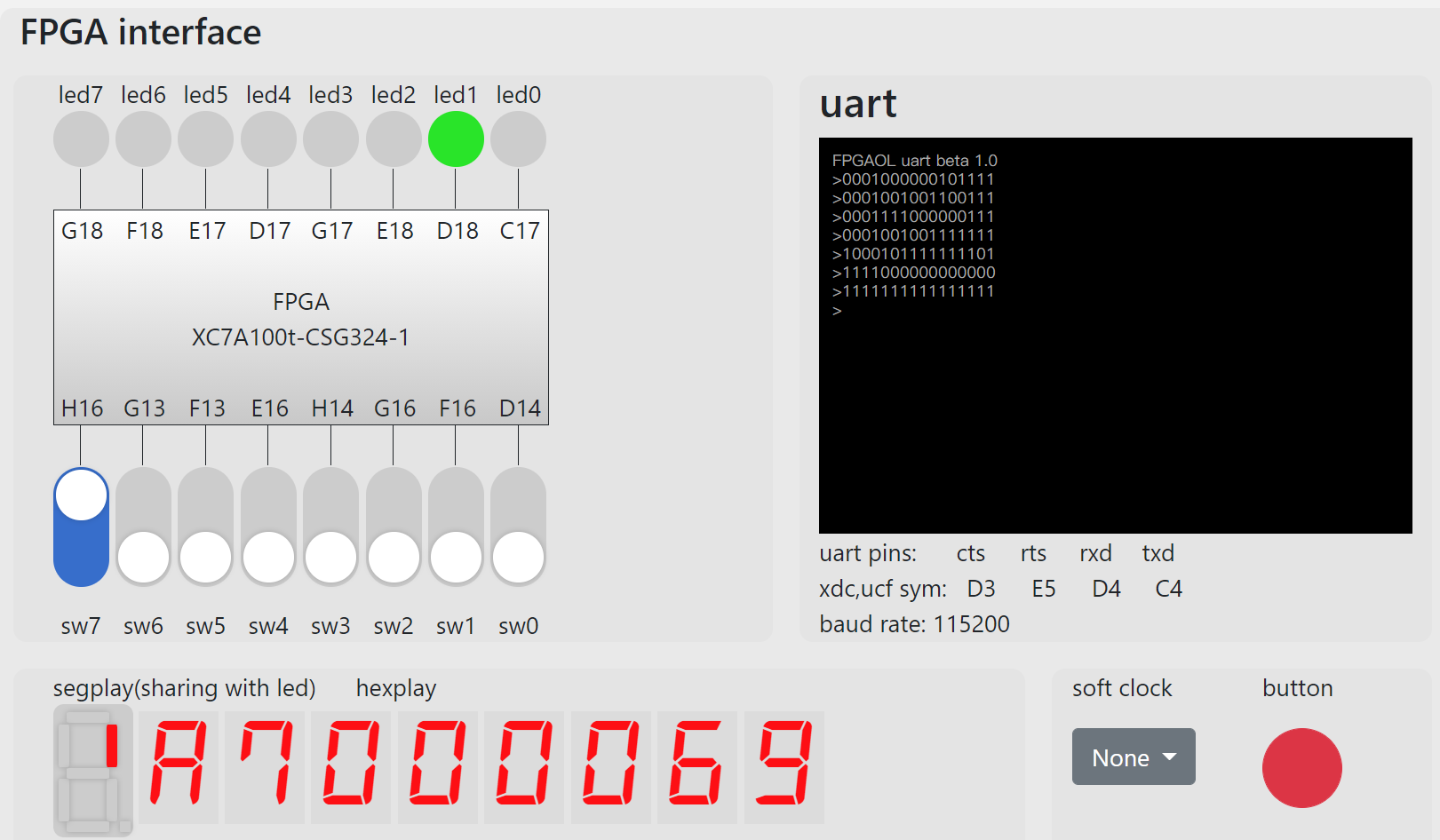
1000101111111101 // if(R1 != 0) goto (\*)

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 // 用LD指令读入#288

0100100000001000 // 用JSR指令表示进入函数

0101010001100111

0001001010000100

0001000001111001

1000001111111011

0001000001111001

1000100000000001

0001001001111001

1111000000000000

0101010010100000

0101011011100000

0101100100100000

0001010010100001

0001011011101000

0101101011000001

1000010000000001

0001100010000100

0001010010000010

0001011011000011

1000101111111010

1100000111000000

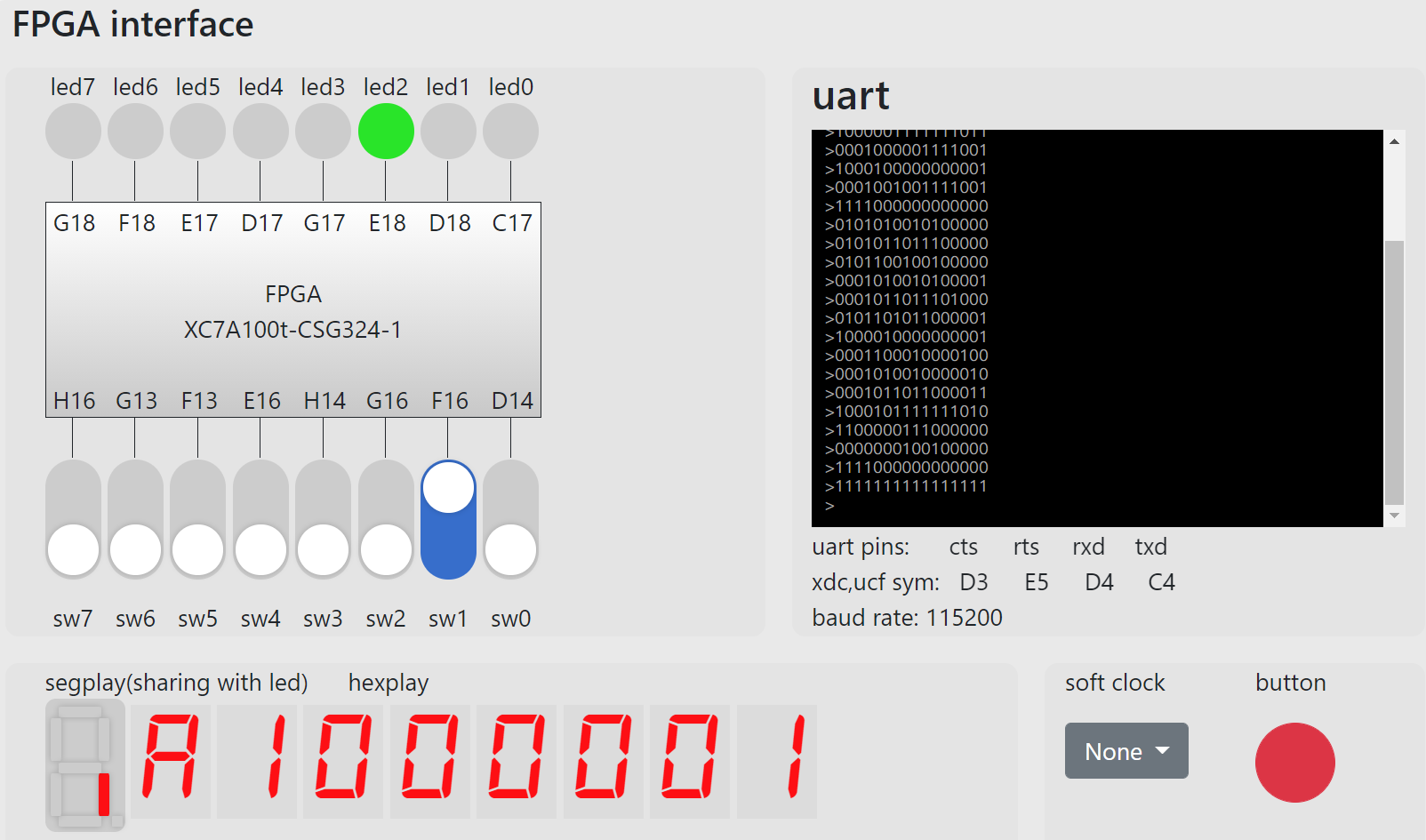
0000000100100000 // #288

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 = R0 + 0xF

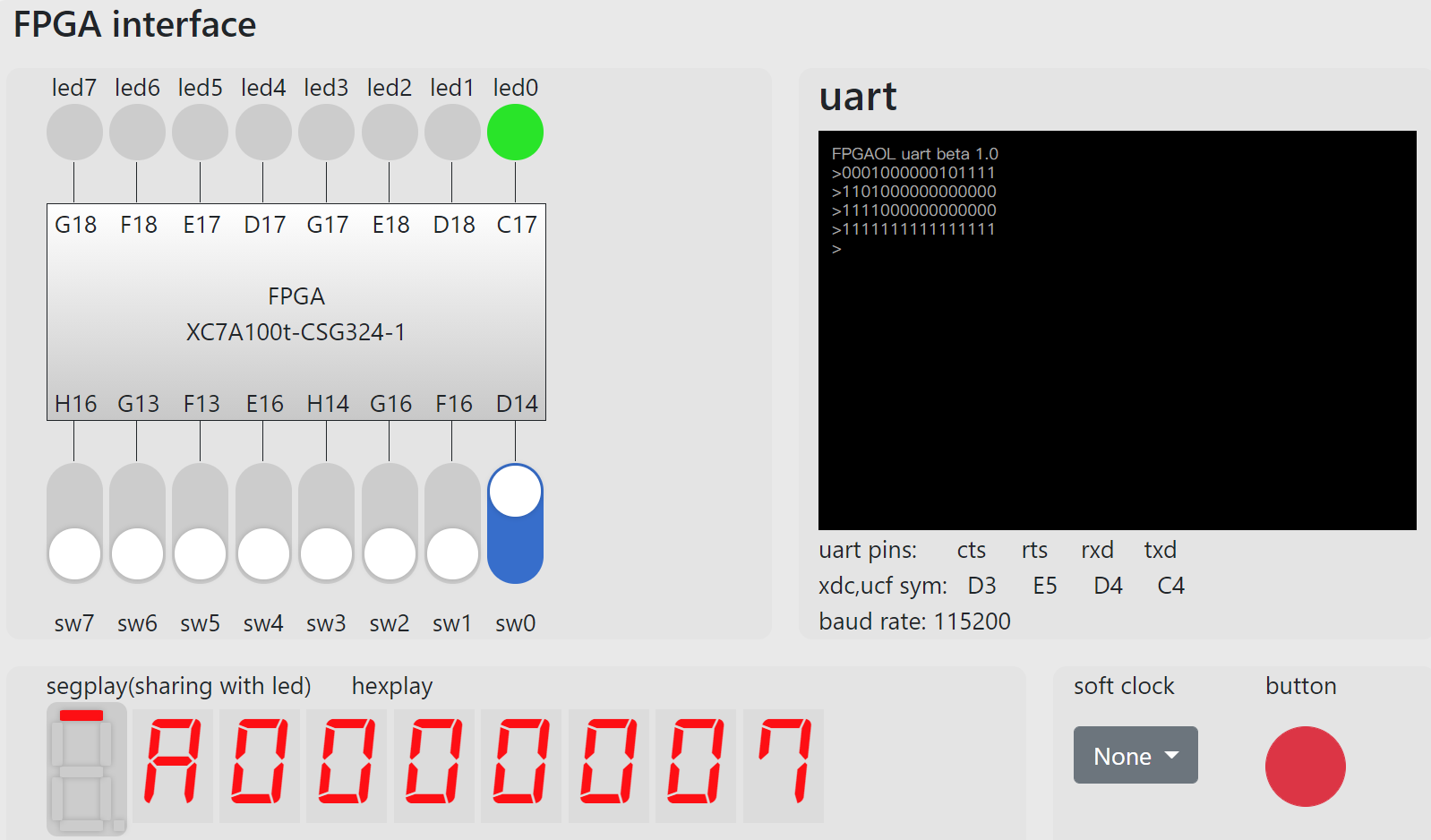
1101000000000000 // R0 = R0 >> 1

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