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：FPGA 原理及 Vivado 综合

学生姓名：张郑飞扬

学生学号：PB21071416

完成日期：2022.11.17

计算机实验教学中心制

2020 年 09 月

【实验题目】

FPGA 原理及 Vivado 综合

【实验目的】

了解 FPGA 工作原理

了解 Verilog 文件和约束文件在 FPGA 开发中的作用

学会使用 Vivado 进行 FPGA 开发的完整流程

【实验环境】

VLAB 平台： vlab.ustc.edu.cn

FPGAOL 实验平台： fpgaol.ustc.edu.cn

Logisim

Vivado 工具

【实验过程】

Step1.初识 FPGA

通过阅读实验手册，了解了 FPGA 的主要功能，FPGA 是目前最主流的可编程逻辑芯片，支持在电路层次进行编程。

Step2.FPGA 基本结构

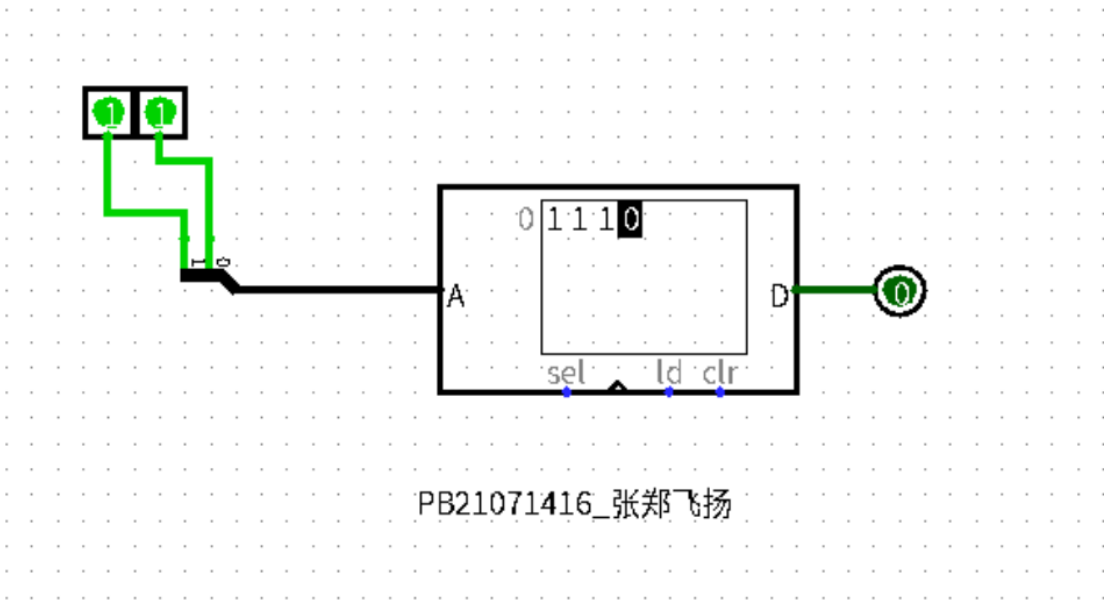
通过阅读实验手册，初步了解了 FPGA 的结构。FPGA 内部包含了大量的可编程逻辑单元（ CLB，Configurable Logic Blocks），通过可编程的交叉互连矩阵(Switch Matrix)连接在一起。每个 CLB 内部包含两个 Slice， Slice 是 FPGA 的最小单元。每个 Slice 都是由查找表（ LUT， Look-Up Table）和触发器（FF,Flip-Flop）构成。

查找表本质上来说就是一个 1bit 位宽的静态随机存储器（SRAM）。通过向 SRAM 加载编程数据，便能实现各种逻辑功能，可编程的互连资源能够将 CLB 级联起来，实现更为复杂的逻辑功能。

此外， FPGA 内部还有一些其它资源，如 BlockRAM 用于满足各类数据存储需求， DLL 用于生成不同频率的时钟信号， IOB 用于控制引脚的输入输出状态，这些资源也都是可编程的。

Step3.可编程逻辑单元

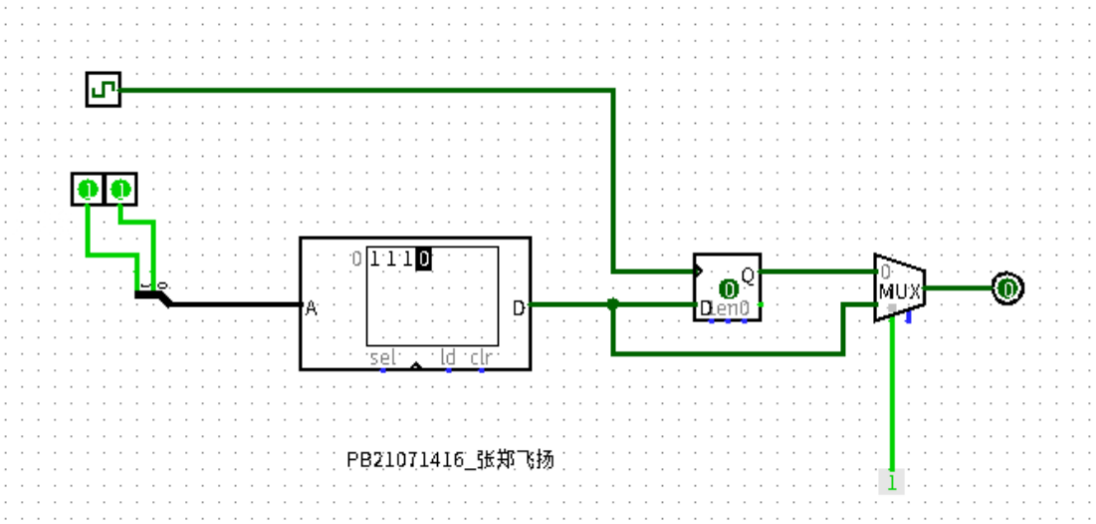
首先在 Logisim 中使用 RAM 搭建如下图所示的电路。在 RAM 属性内将地址宽度设为 2，数据位宽设为 1。在 RAM 模块上单击鼠标右键，选择“Edit Contents”修改 RAM 的初始值为“1 1 1 0”



这个电路和与非门是逻辑等效的。

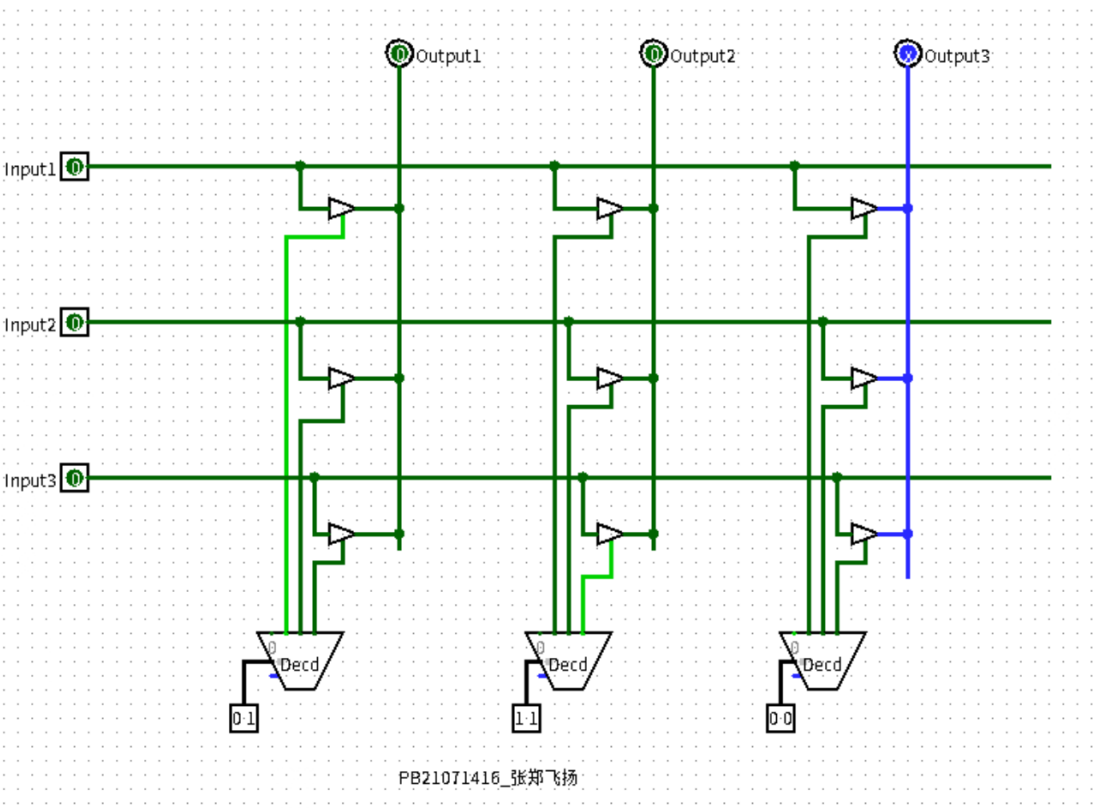
在 RAM 的输出后面添加一个触发器和选择器，并添加一个时钟信号，如下图所示，便实现了对组合逻辑和时序逻辑的支持，通过选

择器选择输出信号是否被寄存。

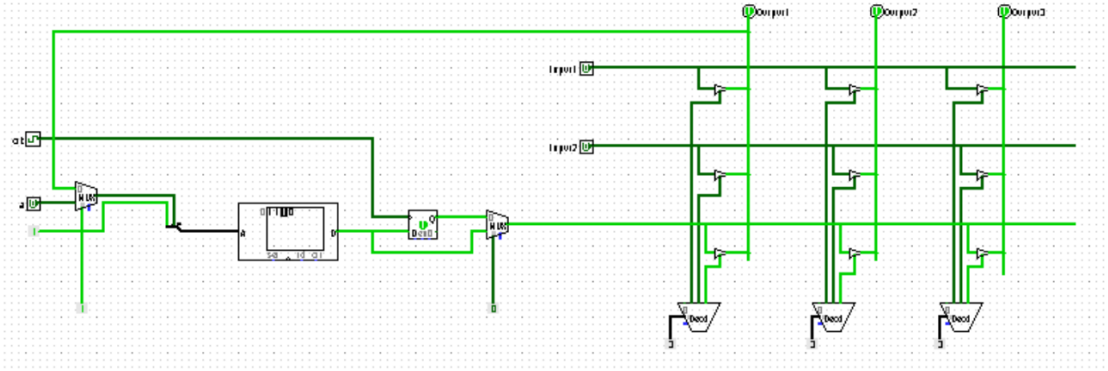


Step4.交叉互联矩阵

在 Logisim 中画出如下电路图，该电路中包含三个输入和三个输出，我们可以发现，通过修改选择器的选择信号，可以将任一输入信号输出到任意输出端口上去，也可以不输出到任何端口。



将可编程逻辑单元与交叉互连矩阵相连接，便能够实现信号的可编程逻辑单元的功能扩展和信号反馈。



Step5.Vivado 综合

首先新建一个 Vivado 工程， 建立工程时，应确保选择的芯片型号的“xc7a100tcsg324-1” ，接着在工程中加入 Verilog 设计代码，如下图所示：

```
module test(
input  clk,
input  rst,
input  [7:0] sw,
output reg [7:0] led
    );
always@(posedge clk or posedge rst)
begin
   if(rst)
      led <= 8'haa;
   else
      led <= {sw[0],sw[1],sw[2],sw[3],sw[4],sw[5],sw[6],sw[7]};
end
endmodule
```

然后在工程中加入约束文件，直接引用群中上传的 FPGAOL 的 xdc 约束文件，将注释删除，并进行适当修改。

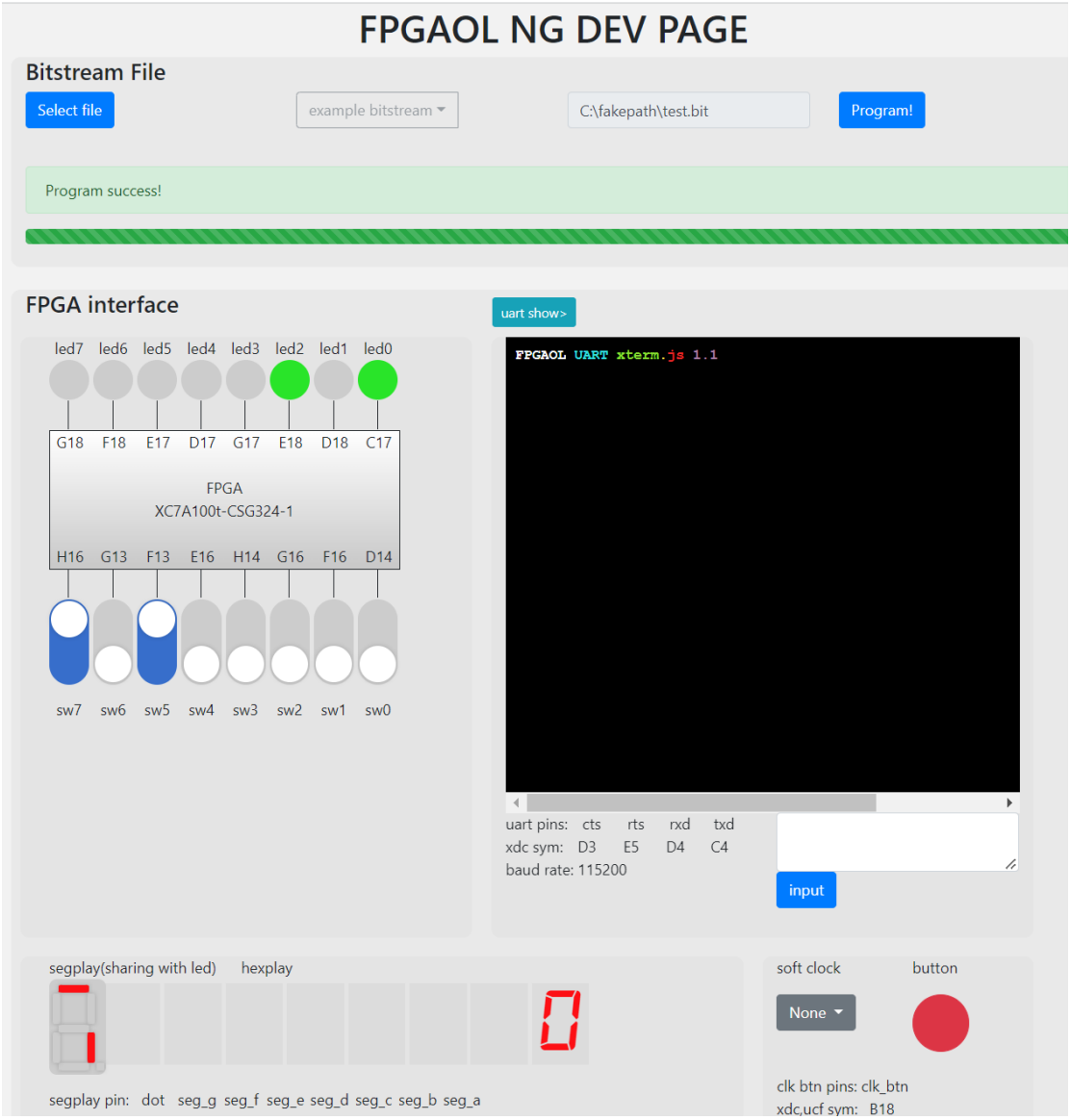
完成代码和约束文件输入后， 保存工程，并点击“ Generate Bitstream”， Vivado 工具会自动完成综合、实现、布局布线等过

程，并最终生成 bit 文件。

Step6.烧写 FPGA

生成 bit 文件后，需要将其烧写到 FPGA 内。首先登陆 FPGAOL 实验平台网站： fpgaol.ustc.edu.cn， 申请一个设备节点， 并通过生成的链接进入设备页面。

在设备页面中点击“select file”按钮， 选择前面生成的 bit 文件，然后点击“Program”按钮进行烧写。

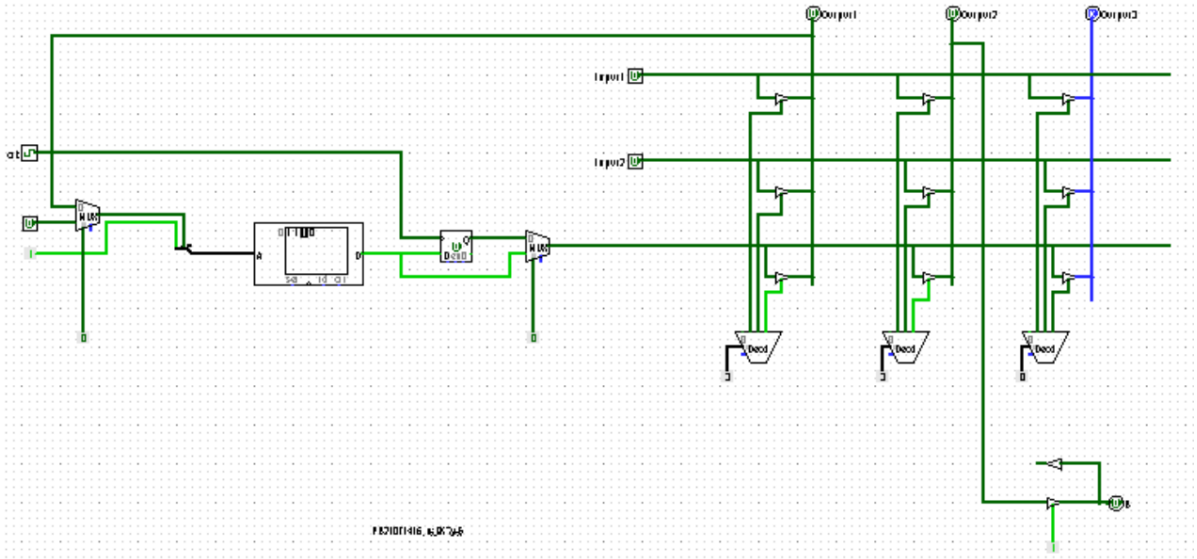


可见开关和 LED 的对应关系是相反的，即最左侧的开关控制最右侧的 LED，最右侧的开关控制最左侧的 LED。

【实验练习】

题目 1.

电路截图如下



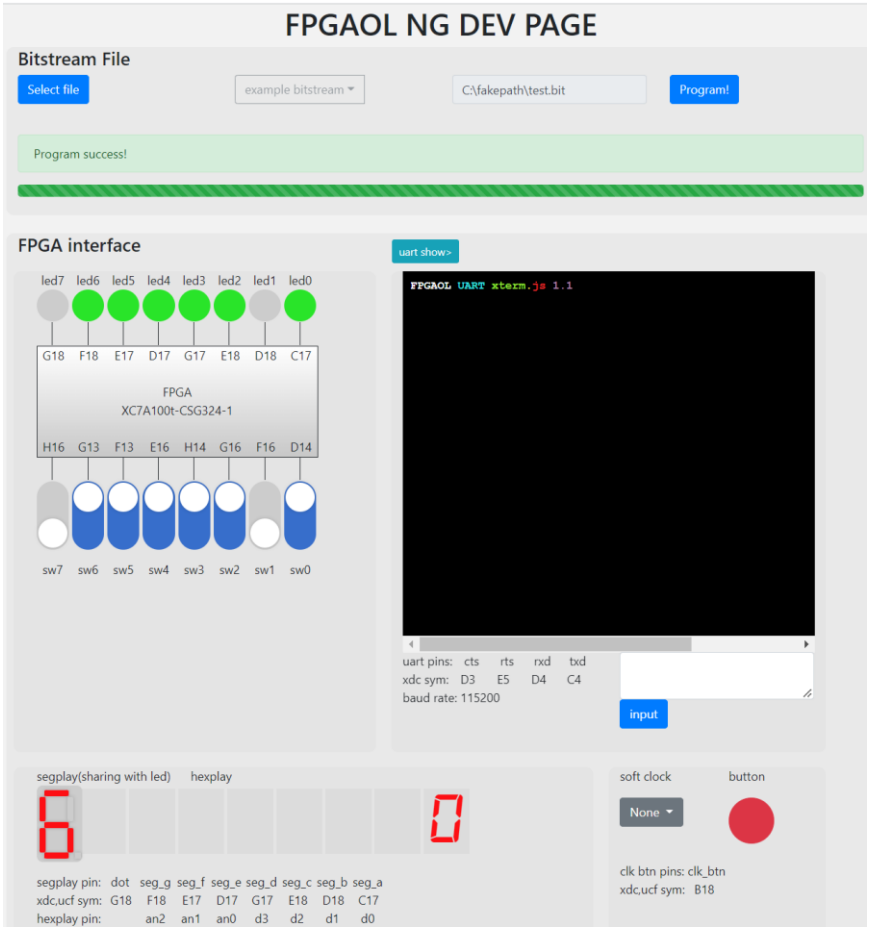
该电路中，可编程逻辑单元功能同两输入与非门，一个输入为 1，另一个输入连接 output1，output1 对应的线用于信号反馈。output2 接出引脚 B，用于输出结果，input1，input2，output3 是不使用的，故第一、二个 Decoder 置 2'b10，第三个 Decoder 置 2'b00。

题目 2.

XDC 文件中，把约束语句中 sw[i]改为 sw[7-i]即可，如图：

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk    }];
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { rst    }];
set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { sw[7] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { sw[6] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { sw[5] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }];
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
```

烧写到 FPGA 上测试如图，可见成功使开关和 LED 一一对应：

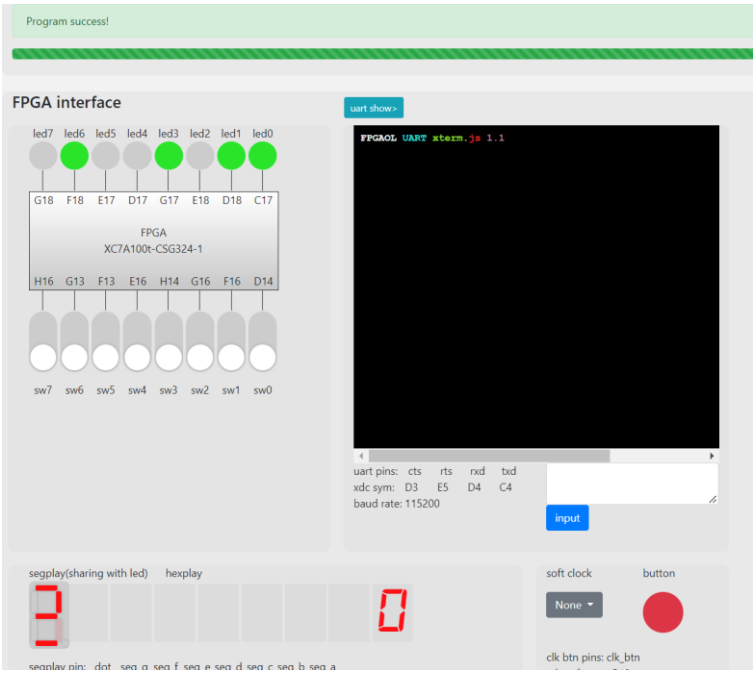


题目 3.

①30 位的计数器代码如下

```
module CNT(
input  clk,
input  rst,
output reg [7:0]led
    );
reg [29:0]out;
always@(posedge clk or posedge rst)
begin
   if(rst)
   begin
      out <= 0;
      led <= 0;
      end
   else
     if(out == 30'b111111111111111111111111111111)
     begin
      out <= 0;
      led <= 0;
     end
     else
      begin
            out <= out+1;
            if(out[22] == 1) led[0] <= 1;  else led[0] <= 0;
            if(out[23] == 1) led[1] <= 1;  else led[1] <= 0;
            if(out[24] == 1) led[2] <= 1;  else led[2] <= 0;
            if(out[25] == 1) led[3] <= 1;  else led[3] <= 0;
            if(out[26] == 1) led[4] <= 1;  else led[4] <= 0;
            if(out[27] == 1) led[5] <= 1;  else led[5] <= 0;
            if(out[28] == 1) led[6] <= 1;  else led[6] <= 0;
            if(out[29] == 1) led[7] <= 1;  else led[7] <= 0;
      end
end
endmodule
```

烧写到 FPGA 上效果为低位的 LED 灯闪烁频率高，高位的闪烁频率低，这与数字的进位原理符合。



①32 位的计数器代码如下

```
module CNT(
input  clk,
input  rst,
output reg [7:0]led
    );
reg [31:0]out;
always@(posedge clk or posedge rst)
begin
   if(rst)
   begin
      out <= 0;
      led <= 0;
   end
   else
     if(out == 32'b11111111111111111111111111111111)
     begin
      out <= 0;
      led <= 0;
     end
     else
      begin
            out <= out+1;
            if(out[24] == 1) led[0] <= 1;  else led[0] <= 0;
            if(out[25] == 1) led[1] <= 1;  else led[1] <= 0;
            if(out[26] == 1) led[2] <= 1;  else led[2] <= 0;
            if(out[27] == 1) led[3] <= 1;  else led[3] <= 0;
            if(out[28] == 1) led[4] <= 1;  else led[4] <= 0;
            if(out[29] == 1) led[5] <= 1;  else led[5] <= 0;
            if(out[30] == 1) led[6] <= 1;  else led[6] <= 0;
            if(out[31] == 1) led[7] <= 1;  else led[7] <= 0;
      end
end
endmodule
```

烧写到 FPGA 上效果为低位的 LED 灯闪烁频率高，高位的闪烁频率低，但相比 30 位的计数器，整体闪烁频率慢了，这也与数字的进位原理符合。

在该过程中，时钟信号每激励一次，计数器就+1，而由于 FPGA 的时钟信号是 100HZ，对于低位来说，意味着变化频次非常快，于是我们用高八位来观测计数器的进位过程才比较清晰。

【总结与思考】

在本次实验中，我主要了解了 FPGA 工作原理，并了解 Verilog 文件和约束文件在 FPGA 开发中的作用，第一次将文件烧写到了 FPGA 上进行观测，这令我收获颇丰。