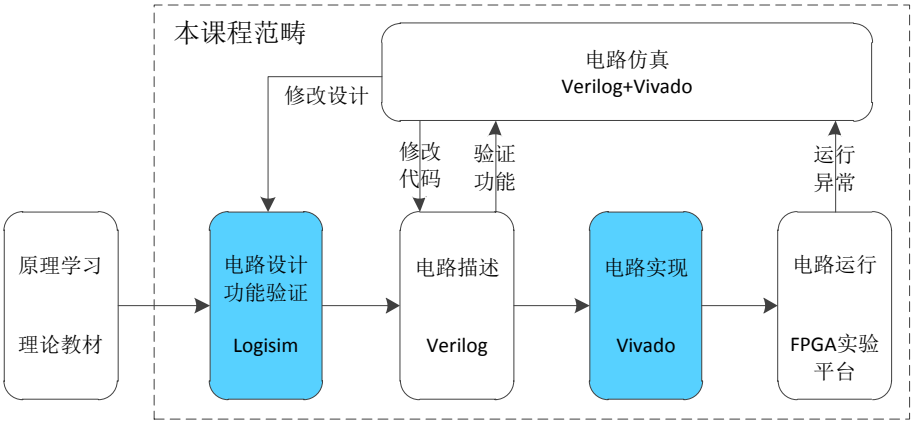
实验 06 FPGA 原理及 Vivado 综合

简介



前文说到过，电路设计的最终目标是运行，本课程所设计的电路最终运行载体是 FPGA 实验平台，FPGA 中文名为现场可编程逻辑门阵列，是目前最为流行的一种可编程逻辑器件，本次实验我们将学习 FPGA 的基本工作原理，并通过 Logisim 搭建一个最基本的 FPGA 原型，力图使读者明白为何这种器件能够实现硬件编程。

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

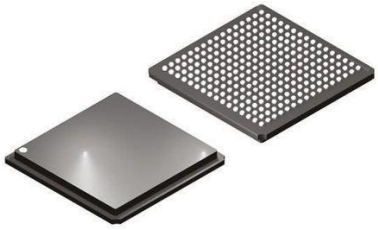
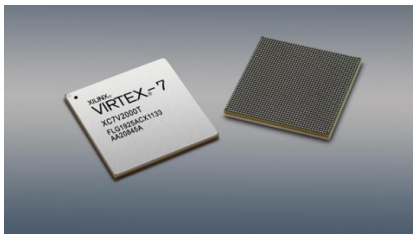
Logisim

Vivado 工具

实验步骤

Step1. 初识 FPGA

FPGA 是一种可编程逻辑器件，其外形一般如下图所示，根据具体封装的不同，芯片引脚分布在底面或者芯片的四周。从外观上看，与其它种类的芯片没有明显差异。

但从功能上来说，可编程逻辑芯片与其它芯片有着本质的不同，如单片机、CPU、内存芯片等都可以称为专用集成电路（ASIC），每种芯片都有专门的用途，如内存芯片用于存储、单片机芯片用于控制，这些芯片的电路结构都是确定的，无法进行更改，虽然有些芯片也支持编程，但都是在软件层面上完成的，不支持对电路结构的编程。可编程逻辑器件则不然，这类器件支持在电路层次进行编程。

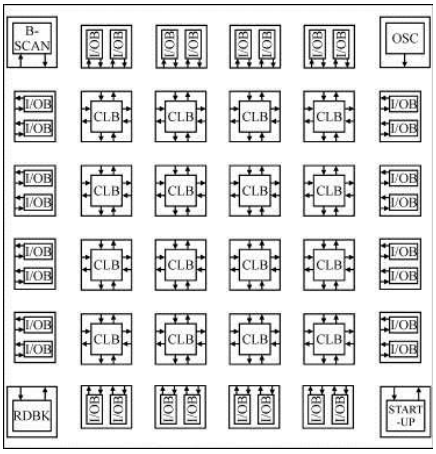
通过专门的 EDA 工具，用户可以将一颗可编程逻辑芯片变成一个加法器，也可以将其变成一个多路选择器或者一个计数器，如果芯片资源充足，用户甚至可以将其变成一个嵌入式片上系统，并在上面运行操作系统。

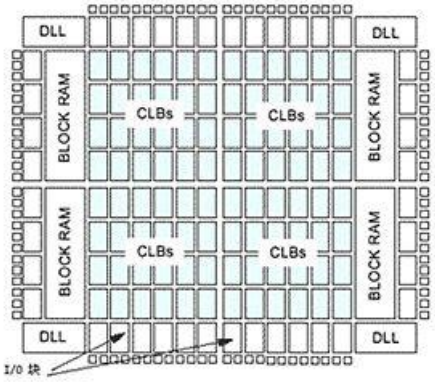
FPGA 是目前最主流的可编程逻辑芯片，除此之外还有 CPLD、PAL、GAL 等，使用的并不广泛。现在 FPGA 芯片的最大生产厂商是美国的赛灵思公司（Xilinx），占据了全球一半以上的市场份额，公司创始人之一 Ross Freeman 于 1984 年发明了 FPGA，从此开创了一个全新的产业。2020 年 10 月 27 日，AMD 公司宣布以 350 亿美金收购 Xilinx。

全球第二大FPGA厂商是Altera，已经在2015年以167亿美金被Intel收购，成为Intel公司史上最大金额的收购案。

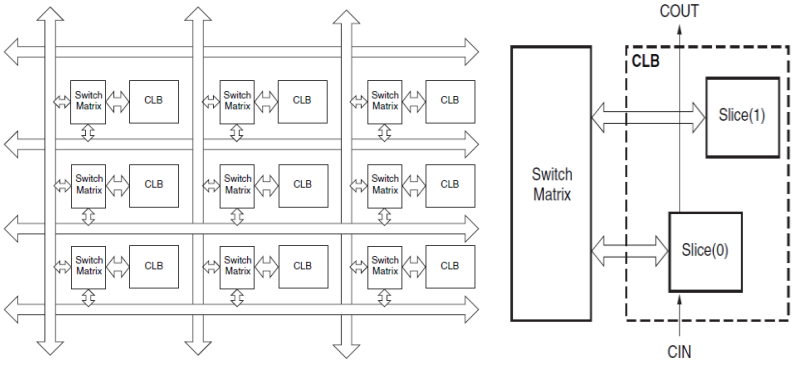
Step2. FPGA基本结构

如下图所示，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都是可配置的，不同的配置便能实现不同的电路功能。交叉互连矩阵也是可编程的，可以根据用户需求将各CLB的输入输出信号进行连接。每个FPGA芯片内都包含了大量的CLB和交叉互连矩阵资源。



每个CLB内部包含两个Slice，Slice是Xilinx公司FPGA的最小单元，下图是从Xilinx官方提供的用户手册上截取的Slice结构图

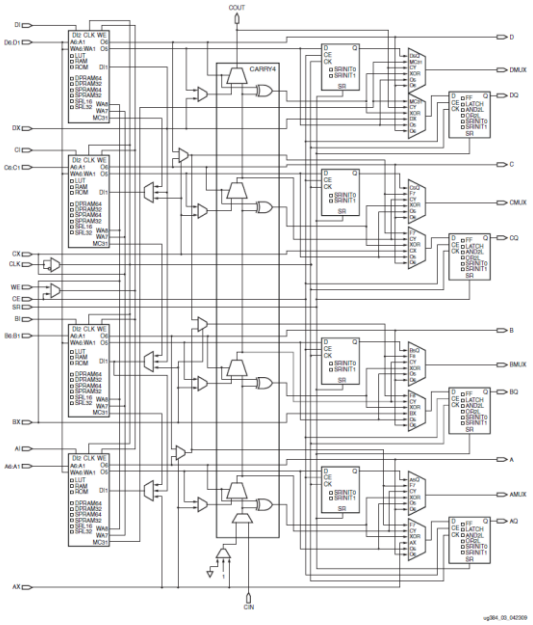


Figure 3: **Diagram of SLICEM**

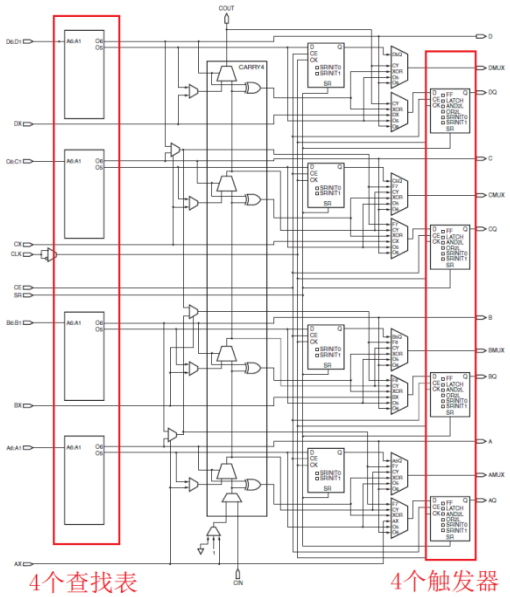


Figure 4: **Diagram of SLICEL**

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。理论上来讲，只要有足够多的 CLB 和互联资源，便能够实现所有需要的组合或时序逻辑功能电路。

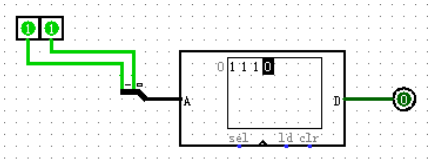
此外，FPGA 内部还有一些其它资源，如 BlockRAM 用于满足各类数据存储需求，DLL 用于生成不同频率的时钟信号，IOB 用于控制引脚的输入输出状态，这些资源也都是可编程的。正是由于 FPGA 的各种可编程特性，使其应用领域越来越广。

Step3. 可编程逻辑单元

从前面的结构图可以看出，Slice 的内部结构还是比较复杂的，我们将通过一个简化的模型对其进行讲解。

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”



通过改变输入信号的值，我们可以发现，输出端（读端口）显示的数据始终是 RAM 内以输入端（地址）信号为编号的数值，通过改变输入端的值，便可以立即访问到存储在 RAM 内部任意地址的数据，这也是为什么称其为随机访问存储器。

上述电路包含有两个输入，一个输出，其真值表为

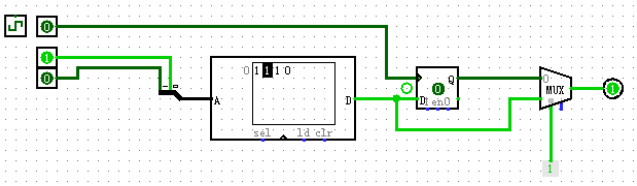
输入	输出
0 0	1
0 1	1
1 0	1
1 1	0

可以很容易看出，这与两输入与非门的真值表完全一样，因此该电路与与非门行为特性也完全一致，它们在逻辑上是等效的。可以认为这是与非门的另外一种实现形式。

如果我们修改 RAM 内的内容，将其改成“1 0 0 0 ”，该电路又变成了或非门的等效电路。不难发现，通过修改 RAM 的内容，便可以实现所有的两输入组合逻辑。如下格式的电路功能都可以实现。

```
assign o = fun(a,b);
```

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。



通过配置 RAM 的内容和选择器的选择信号，以下两种语法格式的电路都可以支持：

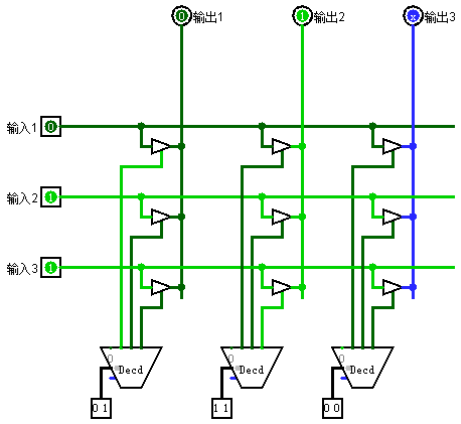
1) assign o = fun(a,b); //组合逻辑
2) always@(posedge clk) //时序逻辑
o <= fun(a,b);

查找表和触发器共同构成了可编程逻辑单元。是 FPGA 中最核心最基本的单元。

Step4. 交叉互连矩阵

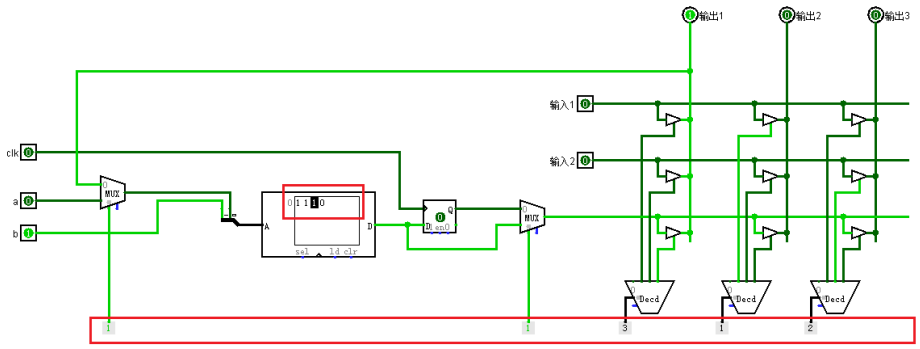
当输入变量个数超出 LUT 输入数，或者需要进行信号反馈时，单靠可编程逻辑单元无法实现，这时候需要借助交叉互连矩阵的功能。FPGA 内部的这种连线资源非常复杂，在此我们也对其进行简化，仅作原理性的讲解。

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。



上图便是交叉互连矩阵的基本原理，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。如下图所示，其中红色方框为配置区域。

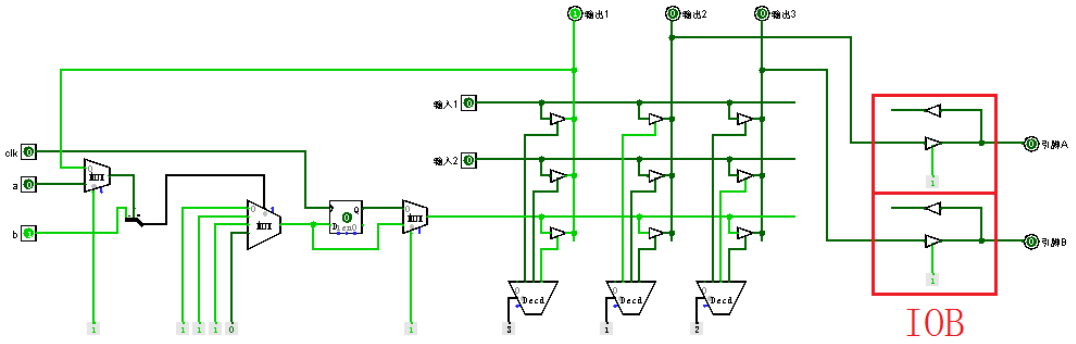


该电路能够支持如下格式的电路

1) *assign o = fun(a,b,输入 1，输入 2)；//更多变量的组合逻辑*
2) *always@(posedge clk)　　//带反馈信号的时序逻辑*
a = fun(a,b);

不难发现，只要有足够的可编程逻辑单元和交叉互连矩阵资源，便可以实现前面学习到的所有逻辑电路，而用户需要做的事情就是在上图中的红色方块标识的配置区域内填入合适的数值。在使用 Xilinx 的 FPGA 芯片时，这项工作通过 Vivado 来完成，用户根据设计需求，完成 Verilog 代码的编写，Vivado 工具会对代码进行综合，生成相应的配置数据。

我们知道，FPGA 芯片上有很多的通用管脚，这些管脚都是可编程的，每个通用管脚都可以被设置为输入信号或输出信号（还可以是输入输出双向信号），也可以被分配给模块的任意一个端口信号，这一点是如何做到的呢？实际上 FPGA 芯片的每一个通用管脚内部都有一个称为 IOB（Input/Output Block）的控制模块，该模块可以控制管脚的输入输出状态，连接到交叉互连矩阵上，便能够实现将任意模块端口分配到该管脚的功能，如下图所示。

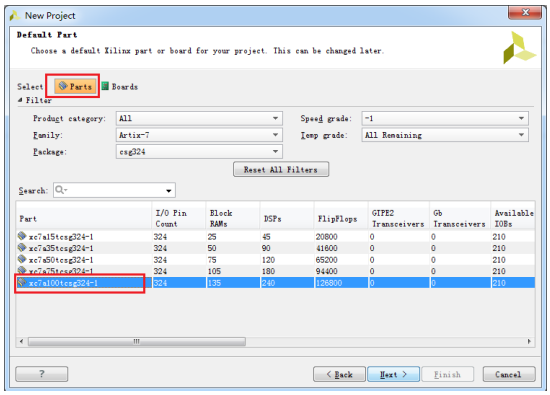


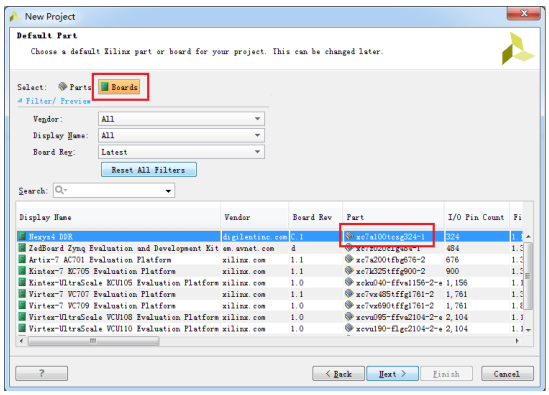
综上所述，通过配置可编程逻辑单元、交叉互连矩阵、IOB 便可以在 FPGA 上实现各种电路功能，其中可编程逻辑单元和交叉互连矩阵的配置数据与 Verilog 代码相对应。那 IOB 所需的配置数据又是从何而来呢？放在 Verilog 代码中肯定不合适，因为这种管脚分配信息与具体使用的 FPGA 芯片有关，如果放在 Vrilog 代码中便降低了代码的可移植性。在 Vivado 综合工具中，除了 Verilog 代码外，还需要一种后缀为 xdc 的约束文件，就是专门用于进行管脚分配的。

Step5. Vivado 综合

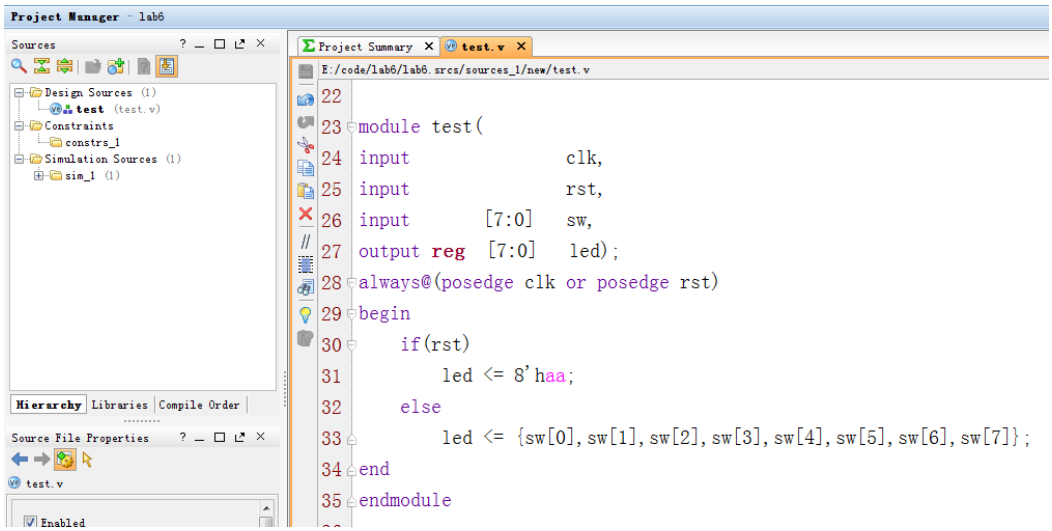
在 Vivado 中建立工程，选定 FPGA 型号，并加入 Verilog 设计文件和 XDC 约束文件，经过综合、实现、布局布线等一系列过程后，会生成一种后缀为 bit 的配置文件，通过烧写工具（Vivado 自带，也可自行开发）将 bit 文件烧写到 FPGA 芯片上，一切顺利的话，电路就能跑起来了。下面我们通过一个实际的例子进行练习。

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”（通过“parts”或“boards”都可以）



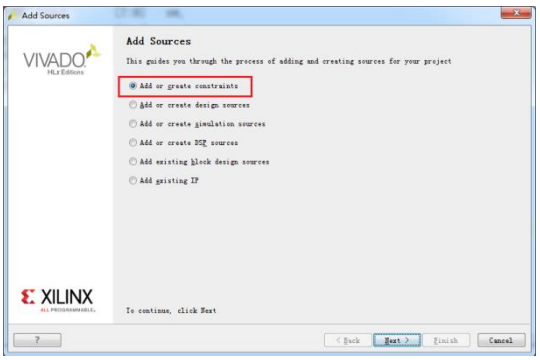


接着在工程中加入 Verilog 设计代码，如下图所示



```
22
23  module test(
24  input               clk,
25  input               rst,
26  input       [7:0]   sw,
27  output reg  [7:0]   led);
28  always@(posedge clk or posedge rst)
29  begin
30      if(rst)
31          led <= 8'haa;
32      else
33          led <= {sw[0],sw[1],sw[2],sw[3],sw[4],sw[5],sw[6],sw[7]};
34  end
35  endmodule
```

然后在工程中加入约束文件，如下图所示，点击“Add source”，选择“Add or create constraints”



如用户使用的是 FPGAOL 实验平台，请在新建的约束文件中添加如下管脚约束信息。

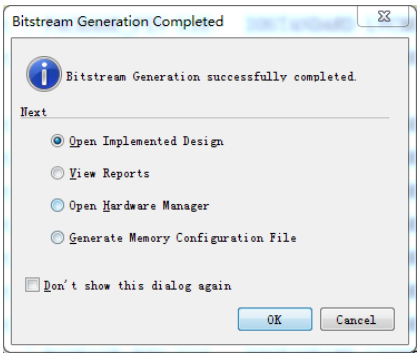
```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
## FPGAOL BUTTON & SOFT_CLOCK
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { rst }];
## FPGAOL LED (signle-digit-SEGPLAY)
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] }];

## FPGAOL SWITCH
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }];
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { sw[5] }];
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { sw[6] }];
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { sw[7] }];

```

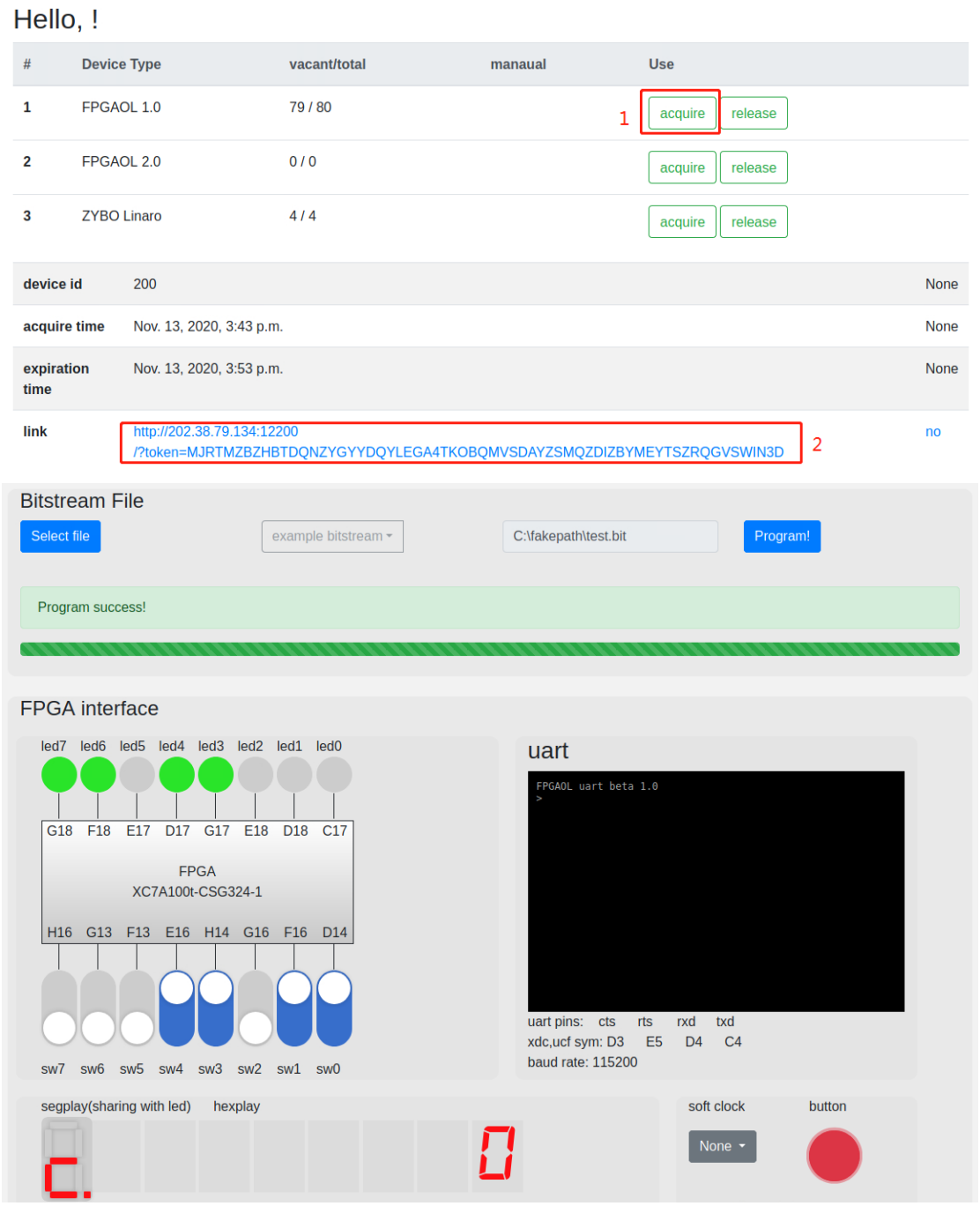
可以看出，管脚约束文件的语法非常简单，“PACKAGE_PIN”后面跟的是 FPGA 芯片的管脚编号，“get_ports”后面跟的是 Verilog 设计文件顶层模块的端口信号。通过该文件，模块的端口信号便与 FPGA 管脚一一对应起来了。

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，生成的 bit 文件一般存放在“工程目录/工程名.runs/impl_1/”内。选择“取消”按钮关闭弹出的对话框。

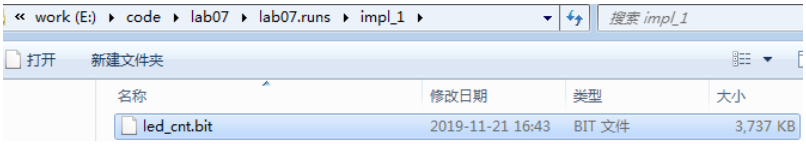


Step6. 烧写 FPGA

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。Bit 文件的路径位于“工程目录/xxx.runs/impl_1/yyy.bit”，其中 xxx 为工程名，yyy 为顶层模块名，如下图所示。



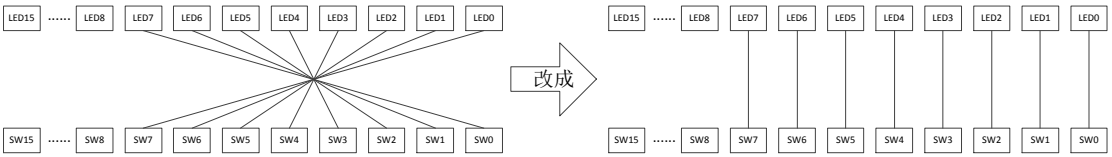
将 bit 文件烧写到 FPGAOL 平台的设备节点上，拨动 8 个虚拟开关，或者按下 button 按键，观察 LED 运行结果。并试着将实验结果与前面的 Verilog 设计文件和 xdc 约束文件联系起来，看看它们是如何影响电路行为的。

实验练习

题目 1. 请通过实验中给出的可编程逻辑单元、交叉互连矩阵及 IOB 电路图，实现如下代码，并将其输出到引脚 B 上。给出配置数据和电路截图。

```
module test(input clk,output reg a);
always@(posedge clk)
    a <= a ^ 1'b1;
endmodule
```

题目 2. 实验中的开关和 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，请修改实验中给出的 XDC 文件，使开关和 LED 一一对应（最左侧的开关控制最左侧的 LED），如下图所示。



题目 3. 设计一个 30 位计数器，每个时钟周期加 1，用右侧的 8 个 LED 表示计数器的高 8 位，观察实际运行结果。将该计数器改成 32 位，将高 8 位输出到 LED，与前面的运行结果进行对比，分析结果及时钟信号在其中所起的作用。

总结与思考

1.请总结本次实验的收获

2. 请评价本次实验的难易程度

3. 请评价本次实验的任务量

4. 请为本次实验提供改进建议