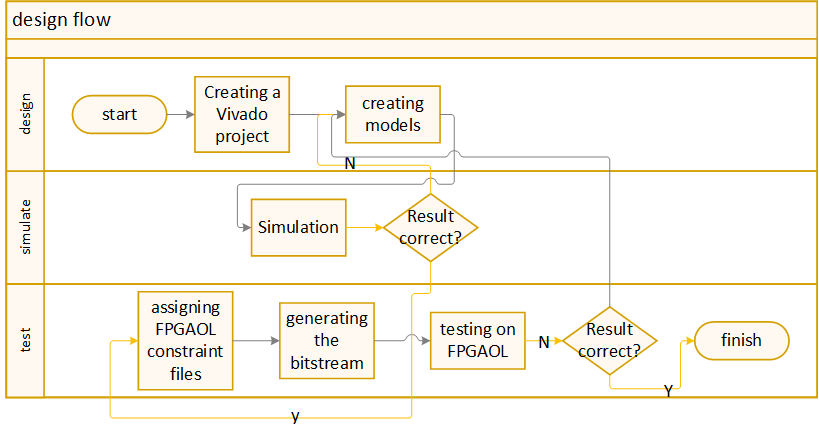
**实验8**

**FPGAOL与上板**

# **实验介绍**

先前，我们已经学习了使用 Vivado 进行 RTL 分析与仿真的流程。分析与仿真过程可以帮助我们检查设计中潜在的逻辑问题，而最终结果的正确性还需要上板进行验证。本次实验，我们将带大家体验使用 Vivado 创建项目并在 FPGAOL 上运行的完整过程。

从本次实验开始，我们的设计流程就可以按照下面的流程图进行：

[](https://soc.ustc.edu.cn/Digital/lab3/figs/design_flow.png)

从上到下可以划分为三个阶段：设计（Design）、仿真（Simulation）和测试（Test）。设计流程包括创建 Vivado 项目、创建模型、分配 FPGAOL 约束文件、选择性地运行行为仿真、生成比特流，最后在 FPGAOL 网站上验证自己的功能。

# **实验目标**

1. 熟悉 FPGAOL 网站的使用；
2. 掌握使用 Vivado 开发项目并在 FPGAOL 上运行的流程。

# 实验内容

## 1 FPGAOL

### 1.1 简介

[FPGAOL（FPGA Online）](https://fpgaol.ustc.edu.cn/)（https://fpgaol.ustc.edu.cn/）是中国科学技术大学计算机教学实验中心组织开发的、基于 Web 端的线上硬件实验平台。用户可以远程访问平台部署好的 FPGA（Nexys 4 DDR）集群，上传本地生成好的比特流文件，并交互地控制 FPGA，实时获得 FPGA 的输出。需要指出的是，该结果是基于 FPGA 实际运行而非仿真产生的，所以可以确保结果与线下操作 FPGA 开发板的结果相同。同时，由于线上设备具备出色的采样性能，平台能够精确发现人眼难以观察到的信号变化，从而为用户快速调试程序提供便利。

单击上面的链接后，用户即可进入登录界面。科大校内学生请选择统一身份认证登录。

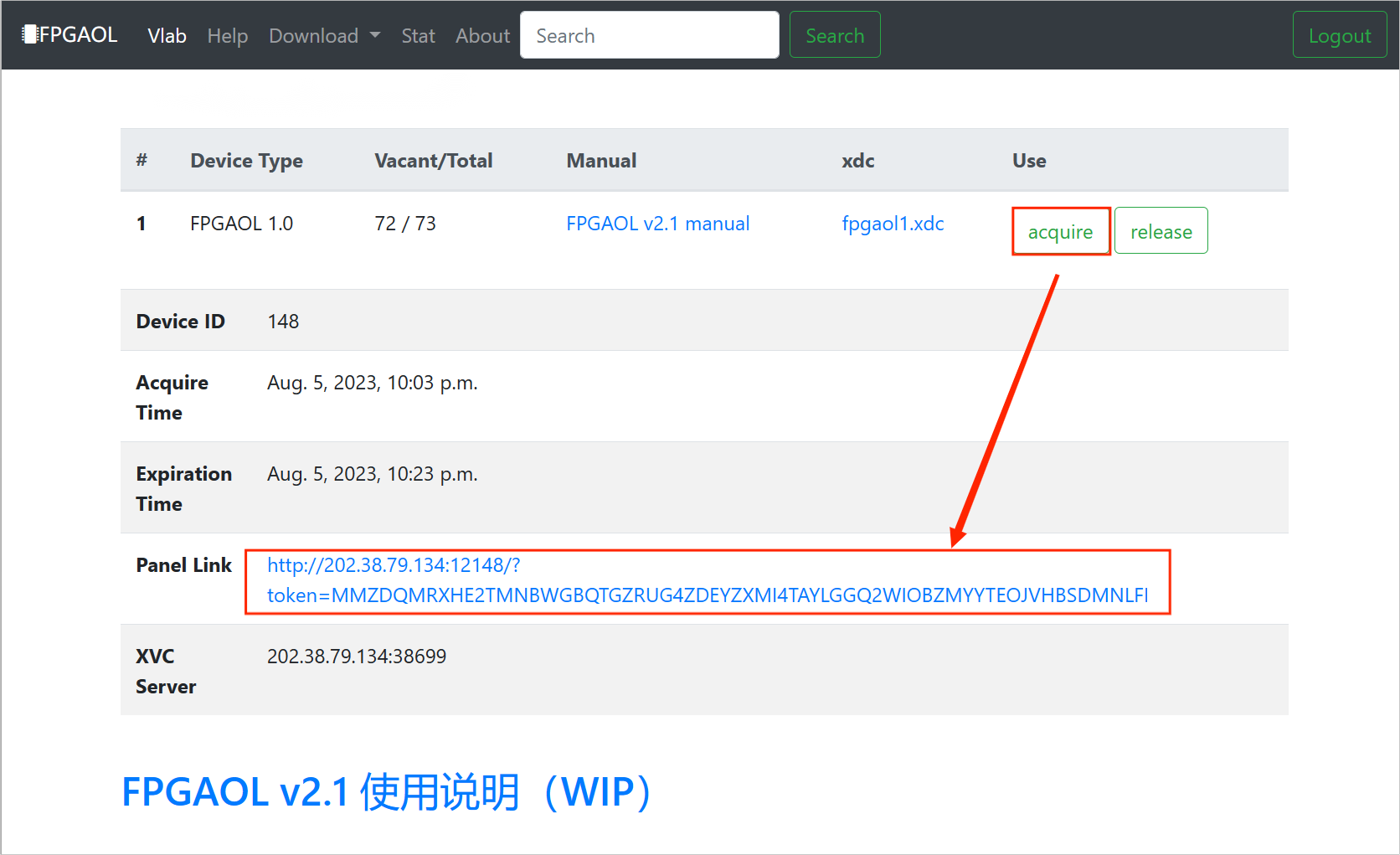
[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/fpgaol.png)

登录完成后，我们将进入下图所示的操作界面。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/fpgaol_base.png)

操作界面上，最为常用的是便是设备请求功能了。由于硬件资源有限，平台采用了限时请求的方式满足各位同学的使用需要。每名同学一次可申请 20 分钟的节点使用资源，超过时间后资源将被自动释放，继续使用则需要重新申请资源。考虑到我们在 FPGAOL 平台上主要是进行功能验证，因此 20 分钟的使用限时并不会带来较大影响。

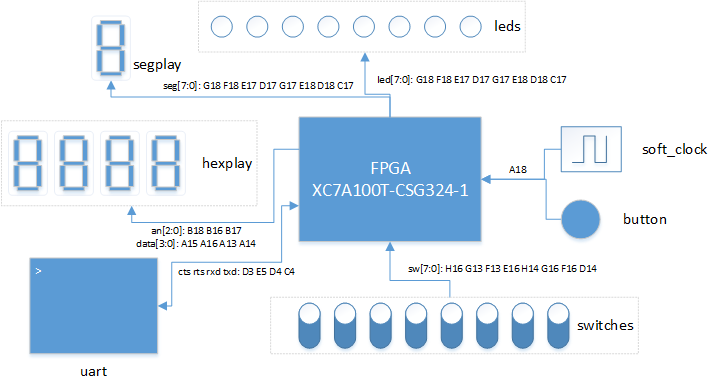
点击 acquire 后，平台便会自动分配一个可用节点，并返回使用链接。用户单击链接即可跳转到相应的操控界面。如果已经使用完成，可以点击 release 释放已申请的节点。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/fpgaol_aquire.png)

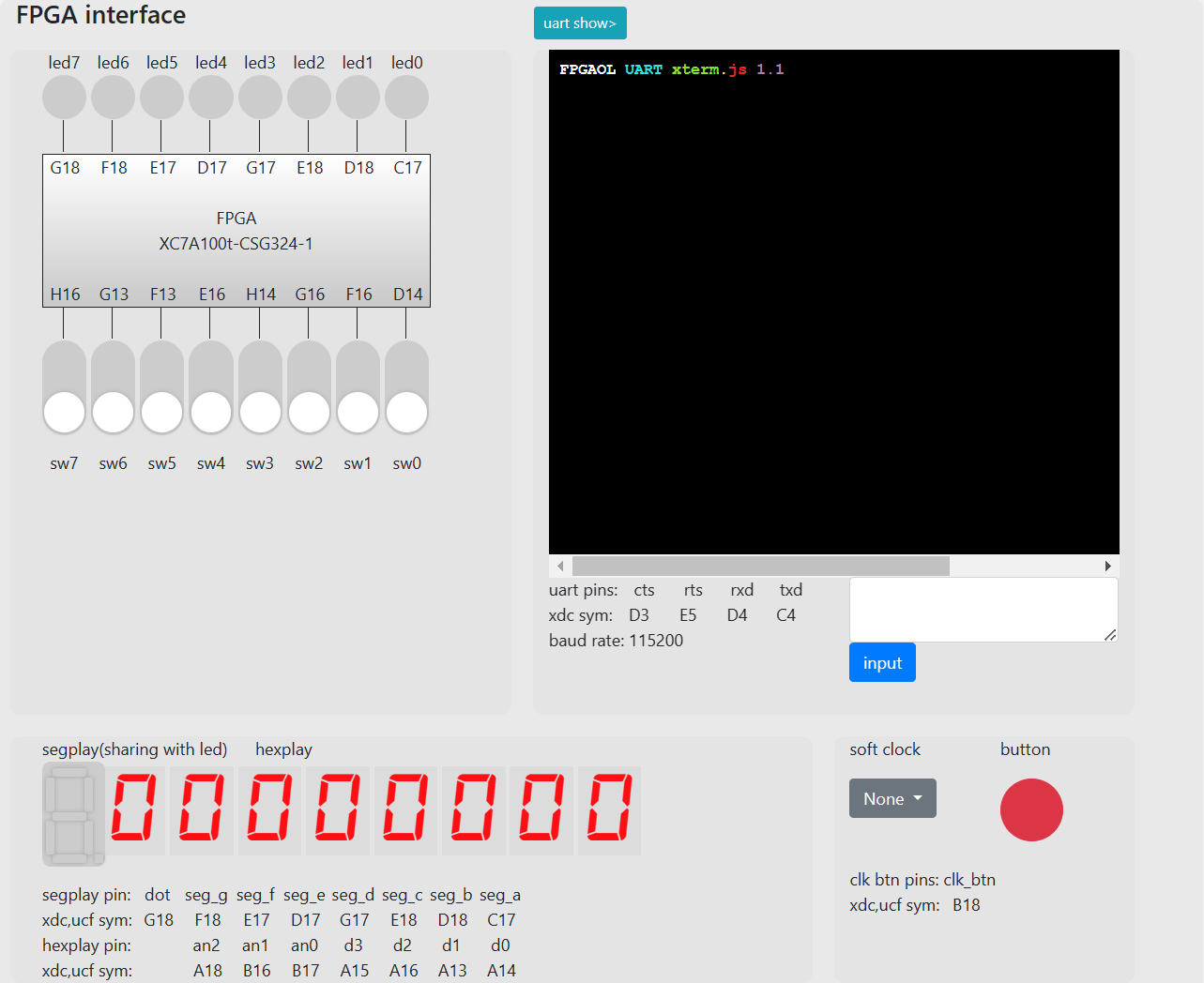
### 1.2 外设

外设，又称外部设备，是指连在计算机主机以外的硬件设备。外设对数据和信息起着传输、转送和存储的作用，是计算机系统中的重要组成部分。

我们的 FPGA 开发板也连接着一些外设资源。这些资源通过特定的端口与 FPGA 芯片连接，实现高效的数据传输。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/peripherals.png)

申请好节点后，我们会进入下图所示的界面。这里是我们与外设直接交互的渠道，按照类别可以分为如下内容：按钮、开关、数码管、LED 灯以及串口。我们将一一进行介绍。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/fpga.png)

#### 1.2.1 按钮

FPGAOL 平台提供了一个按钮。按钮按下时向 FPGA 芯片输入高电平信号，松开时输入低电平信号。默认情况下，按钮处于松开状态。

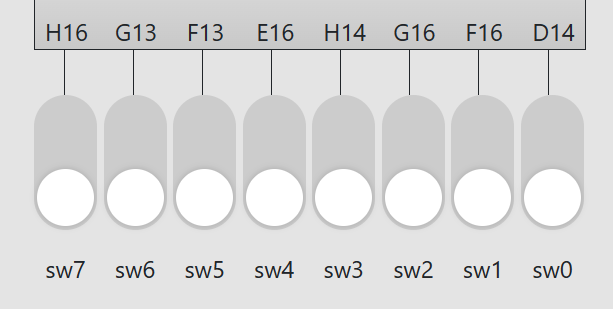
我们一般使用一个 1bit 位宽的信号 btn 代表按钮。因此，正常情况下 btn 的值为 0，当按钮按下时，btn 的值为 1。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/button.png)

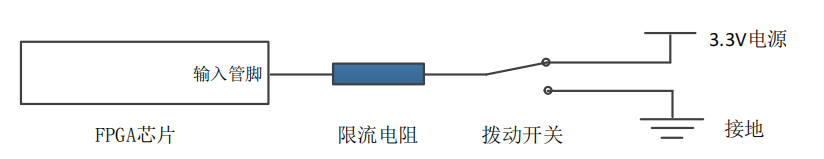
为了方便用户操作，这里的按钮实际上是虚拟的。用户在网页端按下按钮后，平台会产生相应的信号送入 FPGA，而不是按下对应开发板上真正的机械按钮。

#### 1.2.2 开关

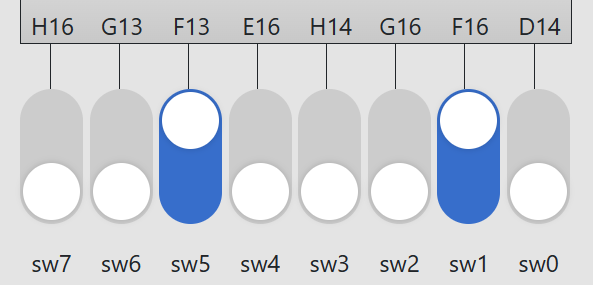
FPGAOL 提供了八个不同的拨码开关。当开关被拨上去时，对应的管脚输出高电平；开关被拨下来时，对应的管脚输出低电平。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/Sw.png)

其工作原理如下图所示：

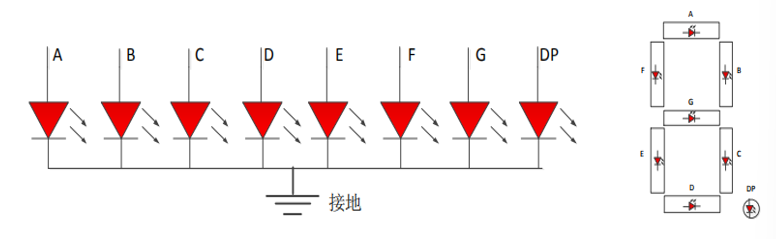
[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/sw_2.png)

我们一般使用一个 8bits 位宽的信号 sw 代表开关输入，其中 sw[i] 对应编号为 i 的开关。例如：下图状态下的开关对应 sw 的值为 8'b00100010。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/sw_example.png)

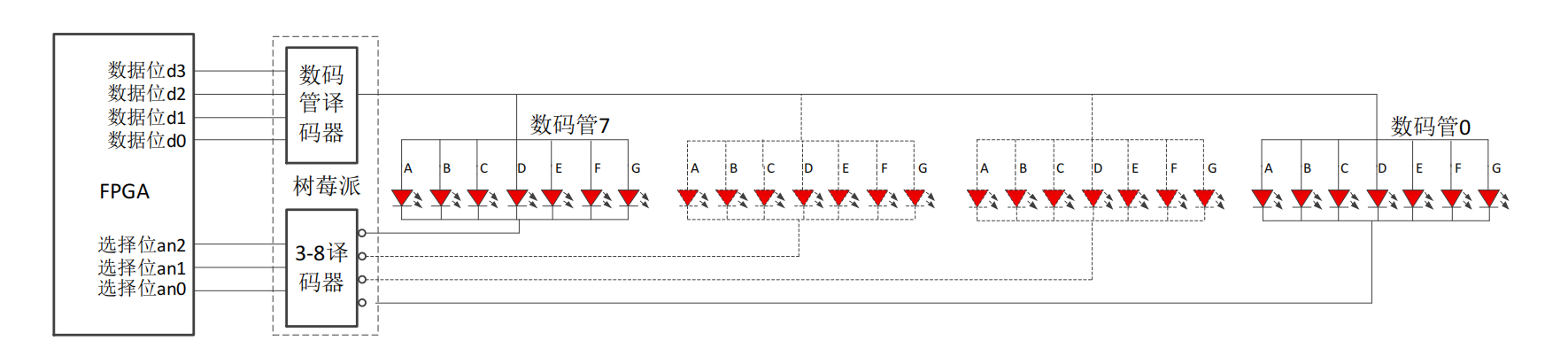
#### 1.2.3 七段数码管

七段数码管本质上是由 8 个 LED（发光二极管）构成，其中 7 个 LED 组成数字本身，1 个 LED 组成小数点。所有 LED 的阴极共同连接到一端并接地，而阳极分别由 FPGA 的 8 个输出管脚控制。当输出管脚为高电平时，对应的 LED 亮起。如下图所示，通过控制 8 个 LED 的亮灭情况，七段数码管便能显示出不同的字符，例如，当 A ∼ F 的 6 个 LED 亮起，而 G、DP 两个 LED 熄灭时，数码管显示的便是字符 "0"。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/seg_1.png)

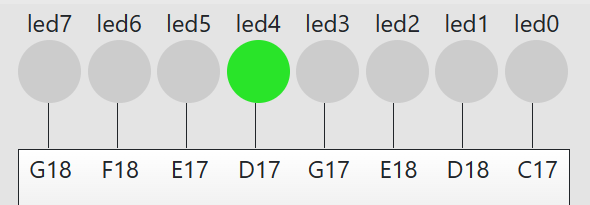
在有多个数码管的情况下，我们通常采用分时复用的方式轮流点亮每个数码管，并保证在同一时间只会有一个数码管被点亮。对于当前点亮的数码管，我们会只传输其应当显示的内容。分时复用的扫描显示利用了人眼的视觉暂留特性，如果公共端的控制信号刷新速度足够快，人眼就不会区分出 LED 的闪烁，从而认为这些数码管是同时点亮的。一般而言，我们建议数码管的扫描频率为 50Hz，也就是说，如果要驱动 8 个数码管，需要一个 400Hz 的时钟。

平台上共有 8 个数码管，最左侧的编号为 7，最右侧的编号为 0。由于实验平台上的管脚数量有限，我们对数码管的显示方式进行了一定的简化：在使能方面，仅使能由 AN[2:0] 所表示的二进制数对应的数码管；在显示的数字方面，直接显示 D[3:0] 对应的 16 进制数。例如，若 AN = 3'b010，D = 4'b1010，则数码管在下标为 2 的数码管上显示字符 "A"。

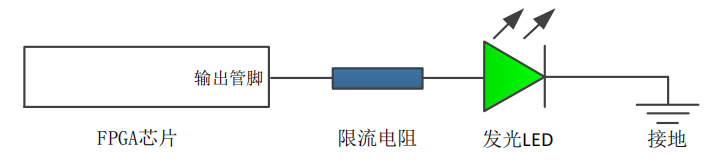
[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/seg.png)

#### 1.2.4 LED

除了上面提到的数码管，我们还提供了 8 个 独立的 LED 灯供用户使用。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/LED.png)

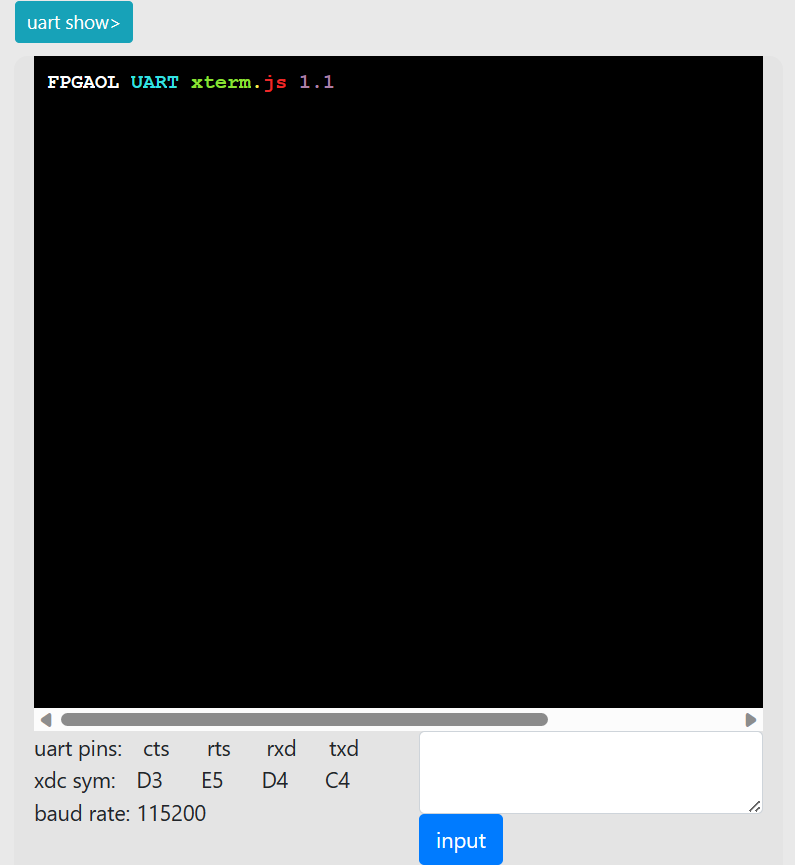
这些 LED 灯可被独立控制。当对应的 FPGA 管脚为高电平时 LED 点亮，为低电平时则熄灭，其原理图如下图所示：

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/LED_1.png)

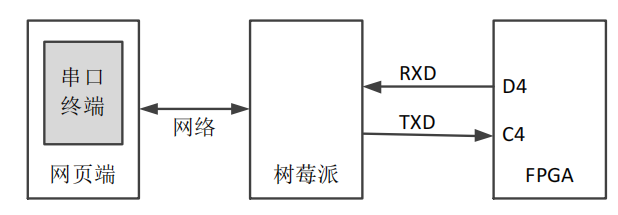
方便起见，我们通常使用统一的 8bits 信号 led 控制这些 LED，led 信号的每一位与对应编号的 LED 管脚相连。例如当 led=8'b0101\_0101 时，编号为偶数的 LED 灯便会亮起。

#### 1.2.5 串口界面

界面右侧的黑色屏幕为串口显示区域。若烧写的程序使用了串口功能，在下方 input 框的输入将作为 FPGA 串口的输入，单击 input 后就会发送；同时，此界面也将接收串口的输出数据，并将其显示在显示屏上。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/uart.png)

FPGA 芯片上的两个 IO 端口直接与树莓派上的串口相连。如果在 FPGA 端编写程序，将 RXD 信号直接与 TXD 信号直连，然后在网页端的串口终端发送数据，便能够实时接收到从 FPGA 侧环回的数据了。串口通信的原理图如下：

[](https://soc.ustc.edu.cn/Digital/lab3/figs/FPGAOL/uart_2.png)

目前的实验中我们不会使用到串口功能，关于串口通信的内容我们将在后面的实验中进行说明。

## 2 综合与实现

下面，我们将通过一个完整的项目带大家体验 Vivado 开发流程。还记得我们在第一次课上演示的猜数字游戏吗？从现在开始，我们就能够运用所学的知识逐步实现这个游戏了。在这个项目里，我们将使用开发板上的 LED 制作一个简易流水灯。

### 2.1 代码设计

要如何实现上面的效果呢？我们可以思考下面的问题：

1. 如何让指定的 LED 灯亮起？很简单，只需要令 led 变量为我们期望的 8bits 数值即可。
2. 如何实现流水的效果？在我们的设计里，亮起的 LED 灯以一定的速度向左平移，如果已经到了最左侧的 7 号灯则会循环回到最右侧的 0 号灯。我们可以使用一个信号 shift 作为标志：当 shift 信号发出时，就让亮起的 LED 灯向左移动一格。怎么实现循环移动的效果呢？或许我们可以这样做：让 led 信号的最高位挪到最低位，其余位顺次左移。这样就可以写出下面的代码：



1. 如何控制流水灯的速度？现在我们只剩下 shift 信号没有确定了。我们可以每间隔一定的时间就发出一次 shift 信号，改变时间间隔就可以实现不同速度的流水灯。

考虑完这三个问题，流水灯模块的 Verilog 代码也就呼之欲出了。

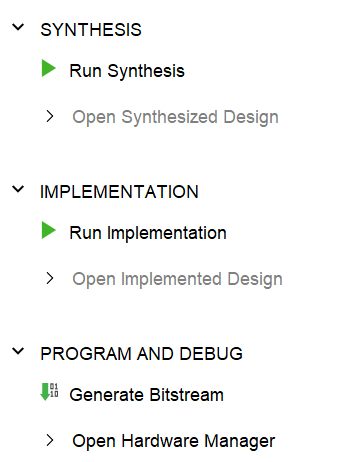


这里我们令计数器的上限值为 5×10^7，是因为开发板上的时钟频率是 100MHz。这样计数器每间隔 0.5s 就会发出一次信号，对应流水灯也向左移动一格。

接下来，请打开 Vivado，创建一个新的项目，随后创建一个新的设计文件 LED.v，写入上面的 Verilog 代码。你可以选择 Open Elaborated Design，以查看流水灯的 RTL 电路图。

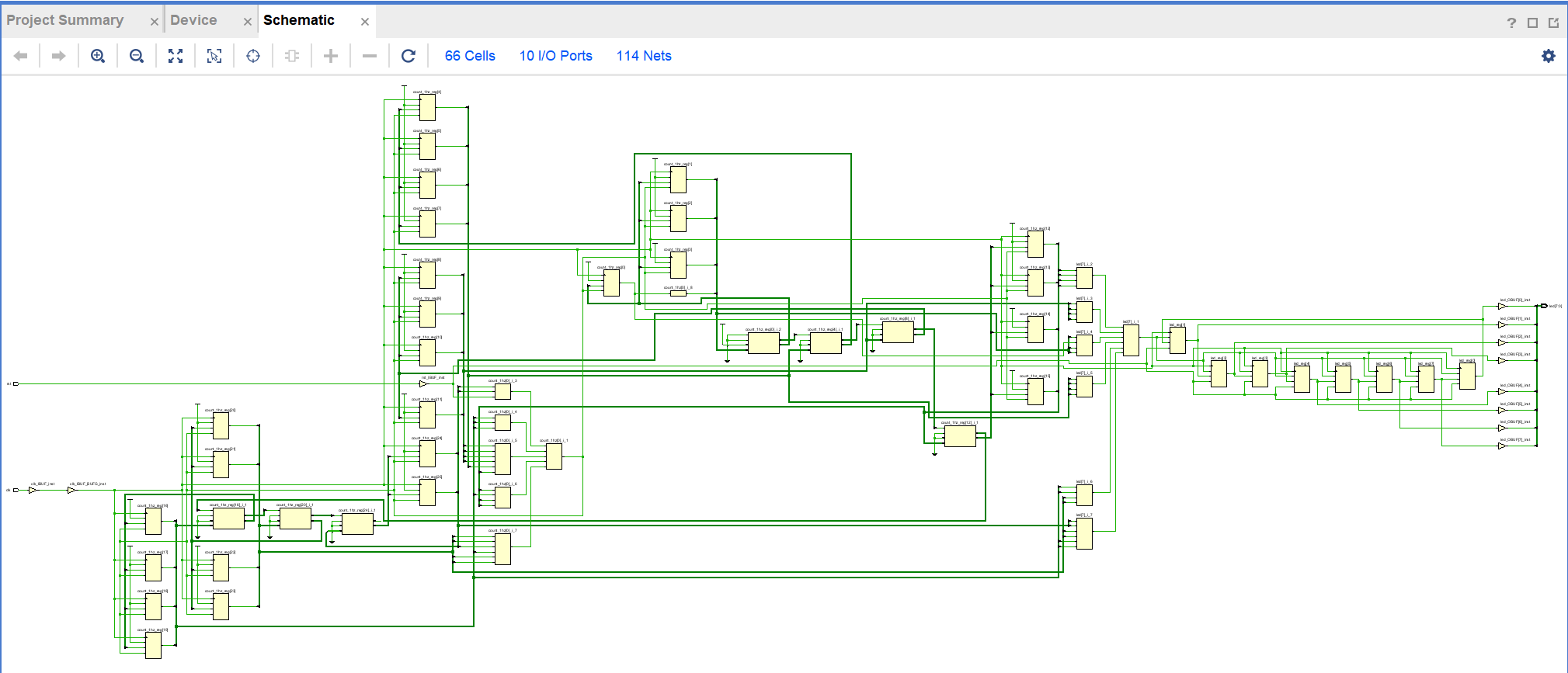
### 2.2 综合与实现

前面的实验中，我们已经介绍了如何使用 Vivado 进行 RTL 分析与仿真。接下来我们将进入综合与实现的环节。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/vivado/vivado.png)

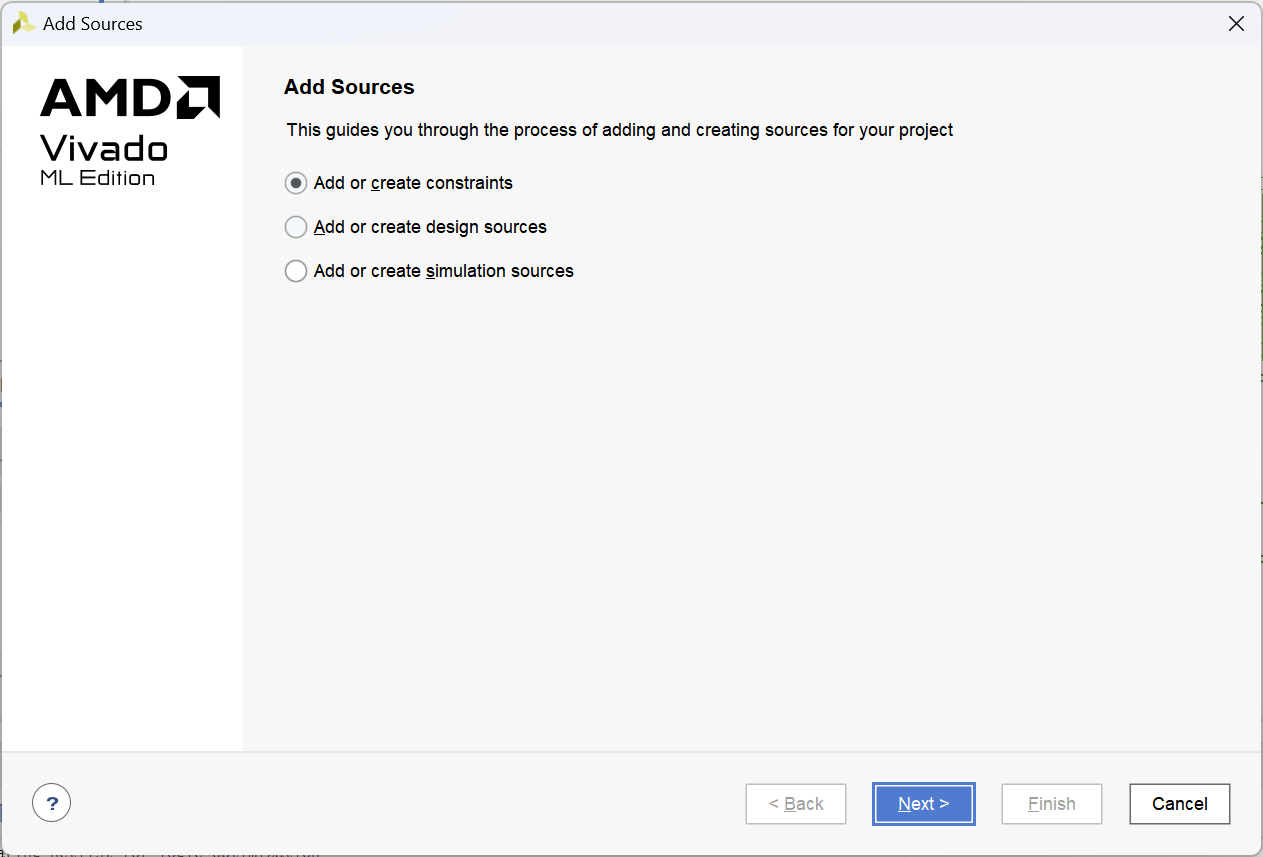
单击左侧的 Run Synthesis 即可开始综合（Synthesis）。在这一步，我们可以找到更多 warnings、critical warnings 和 errors，例如逻辑环路（自己的输出直接接到自己的输入）、多驱动（一个变量由多个 always 块或多个输入修改）等。

综合过程中也可以查看电路图，但这时的电路图大部分以查找表的形式出现，其结构与规模都和 RTL 电路有着较大的不同，因此很难找出问题。此外，为了更优的电路性能，综合出的电路也有可能把某一模块的元件安置到另一个模块里，所以在使用综合出的电路图查找问题时要关注这些细节。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/vivado/syn_circuit.png)

在实现（Implementation）之前，我们需要一个额外的约束文件，用来指示模块输入输出端口和开发板端口之间的对应关系。由于开发板型号、配置不同，因此对应的约束文件也不同。本课程使用的约束文件可以在[这里](https://fpgaol.ustc.edu.cn/download/FPGAOL.xdc)下载。

得到约束文件后，将其放入项目文件夹中，并在 Vivado 中添加设计文件 Constraints。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/vivado/add_con.png)

使用文件时，将需要连接接口的注释符号 # 去掉，改为对应模块接口的名字即可。例如，我们这次使用的约束文件如下：

## This file is a general .xdc for FPGAOL\_BOARD (adopted from Nexys4 DDR Rev. C)

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## 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 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] }];

## FPGAOL HEXPLAY

#set\_property -dict { PACKAGE\_PIN A14 IOSTANDARD LVCMOS33 } [get\_ports { hexplay\_data[0] }];

#set\_property -dict { PACKAGE\_PIN A13 IOSTANDARD LVCMOS33 } [get\_ports { hexplay\_data[1] }];

#set\_property -dict { PACKAGE\_PIN A16 IOSTANDARD LVCMOS33 } [get\_ports { hexplay\_data[2] }];

#set\_property -dict { PACKAGE\_PIN A15 IOSTANDARD LVCMOS33 } [get\_ports { hexplay\_data[3] }];

#set\_property -dict { PACKAGE\_PIN B17 IOSTANDARD LVCMOS33 } [get\_ports { hexplay\_an[0] }];

#set\_property -dict { PACKAGE\_PIN B16 IOSTANDARD LVCMOS33 } [get\_ports { hexplay\_an[1] }];

#set\_property -dict { PACKAGE\_PIN A18 IOSTANDARD LVCMOS33 } [get\_ports { hexplay\_an[2] }];

## FPGAOL BUTTON & SOFT\_CLOCK

set\_property -dict { PACKAGE\_PIN B18 IOSTANDARD LVCMOS33 } [get\_ports { btn }];

##USB-RS232 Interface

#set\_property -dict { PACKAGE\_PIN C4 IOSTANDARD LVCMOS33 } [get\_ports { UART\_TXD\_IN }]; #IO\_L7P\_T1\_AD6P\_35 Sch=uart\_txd\_in

#set\_property -dict { PACKAGE\_PIN D4 IOSTANDARD LVCMOS33 } [get\_ports { UART\_RXD\_OUT }]; #IO\_L11N\_T1\_SRCC\_35 Sch=uart\_rxd\_out

#set\_property -dict { PACKAGE\_PIN D3 IOSTANDARD LVCMOS33 } [get\_ports { UART\_CTS }]; #IO\_L12N\_T1\_MRCC\_35 Sch=uart\_cts

#set\_property -dict { PACKAGE\_PIN E5 IOSTANDARD LVCMOS33 } [get\_ports { UART\_RTS }]; #IO\_L5N\_T0\_AD13N\_35 Sch=uart\_rts

完成添加后，单击 Run Implementation 即可开始实现。此时可能会出现更多更难以解决的问题，比如时序不满足（逻辑电路过长）等。但同时我们也可以看到更多资源使用（utilization）、功耗（power）、时序（timing）等方面的信息。

### 2.3 在FPGAOL上运行

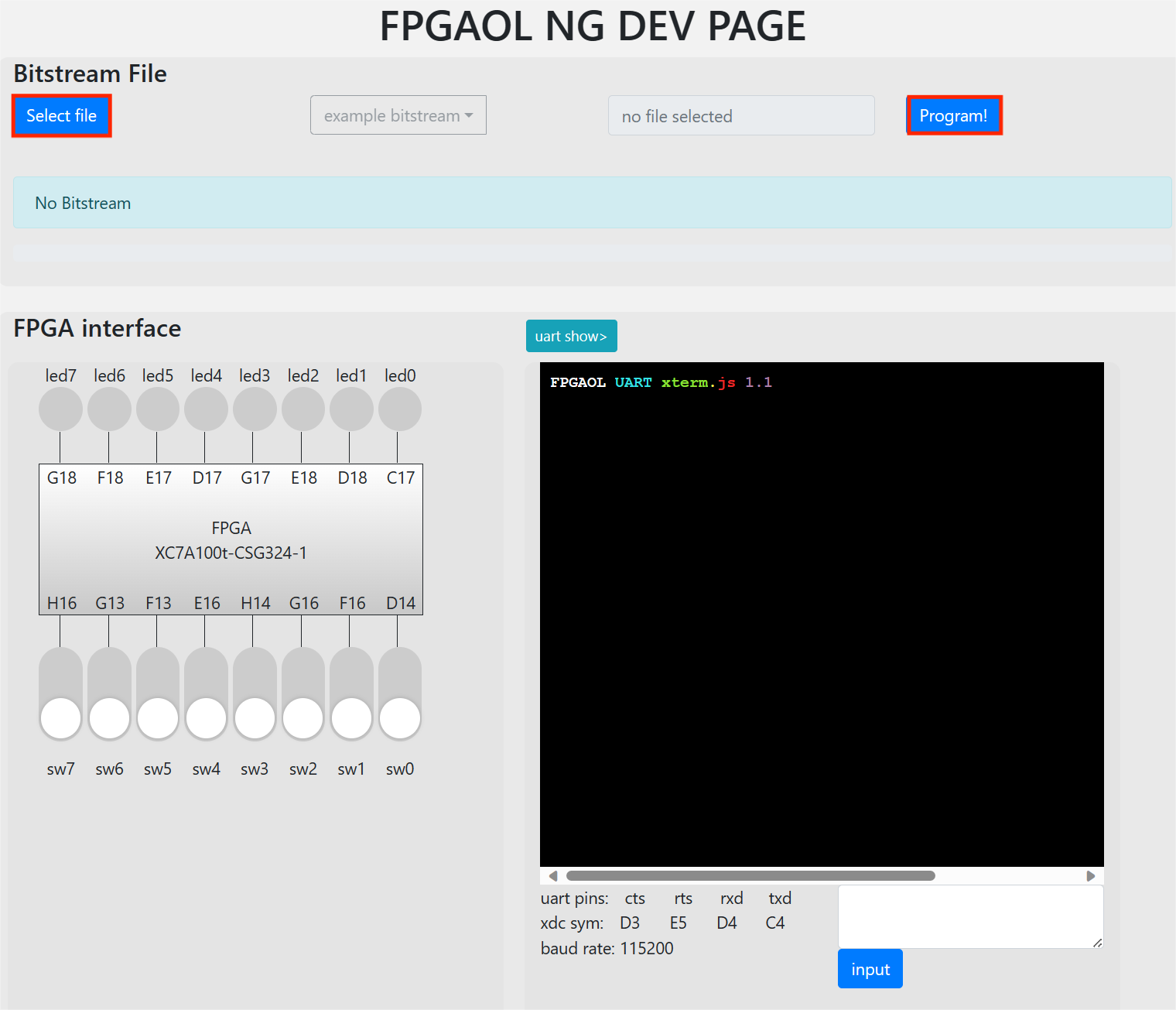
到这里，我们开发的过程就基本结束了，接下来需要在 FPGAOL 平台上进行验证。

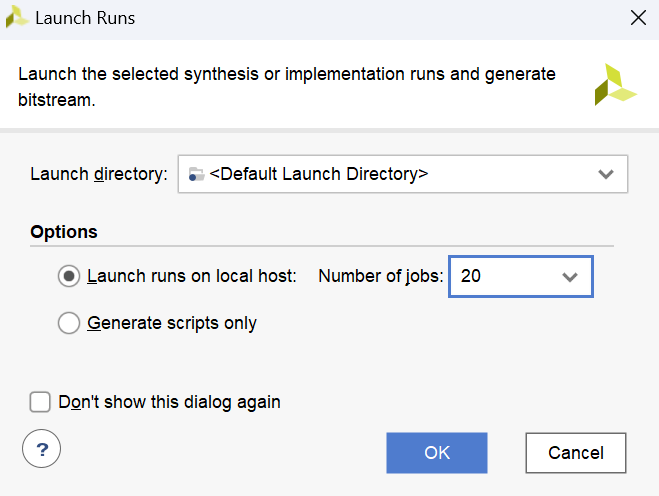
点击左侧最下方的 Generate Bitstream 即可生成比特流文件。在生成比特流文件之前，你可以不进行仿真、综合与实现的步骤（假定你的设计没有问题），Vivado 工具会自动完成综合、实现、布局布线等过程，并最终生成比特流文件。

生成比特流文件的过程一般较为漫长，且取决于所用设备的性能。助教在自己电脑上生成一次比特流耗时大约在三分钟，如果使用 VLAB 平台则可能是五分钟。你可以在开始生成前选择本次生成所使用的核心数，进而加速生成过程。

生成的文件一般存放在 工程目录/工程名.runs/impl\_1/ 目录下，为 顶层模块名.bit。我们建议大家每次生成后将其拷贝到特定的目录，因为下次综合时会清除掉原先的比特流文件。完成后，点击 Cancel 按钮关闭弹出的对话框。

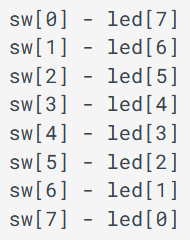
得到比特流文件后，在申请的在线开发板中单击 Select File，将文件上传至服务器。上传完成后点击 Program! 就可以在线进行测试了。

[](https://soc.ustc.edu.cn/Digital/lab3/figs/vivado/online_test.png)

[](https://soc.ustc.edu.cn/Digital/lab3/figs/vivado/gen.png)

# 思考与练习

1. 请编写 Verilog 代码，要求当拨动开关时，对应的 LED 灯便会亮起/关闭。开关与 LED 灯的对应关系如下所示：



你的代码应当能在 FPGAOL 平台上运行。下面是我们提供的代码框架：

