Lab 3 report

PB2111711 陈昕琪

实验目的与内容

> 1. 学会使用 Vivado 创建项目并在 FPGAOL 上运行的完整过程。
> 2. 了解如何使用 FPGAOL 网站的使用。
> 3. 进一步熟悉 Verilog 语言并了解基础的信号处理技术。

1. 必做内容：开关与LED

> **要求：**
>
> 编写 Verilog 代码，要求当拨动开关时，对应的 LED 灯便会亮起/关闭。
>
> **逻辑实现：**
>
> 定义一个寄存器temp，在sw输入时将sw的各位值分别赋给temp，之后再把temp赋值给led，实现相应的led灯亮起。

代码如下：

```
module Top (
    input   [7:0]                       sw,
    output  [7:0]                       led
);
reg [7:0] temp;
// Write your codes here.
always @(*) begin
    temp[7] = sw[0];
    temp[6] = sw[1];
    temp[5] = sw[2];
    temp[4] = sw[3];
    temp[3] = sw[4];
    temp[2] = sw[5];
    temp[1] = sw[6];
    temp[0] = sw[7];
end

assign led=temp;

endmodule
```

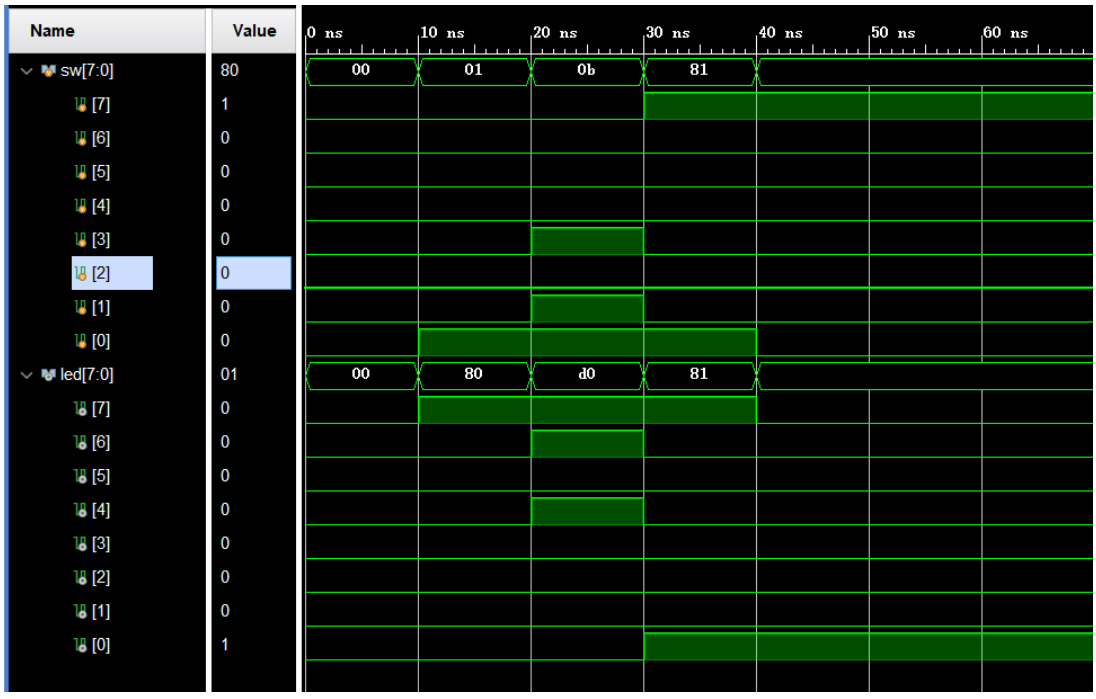
仿真结果与分析

> 根据所写的代码，编写相应的仿真文件验证其正确性。 对输入的sw考虑出现多个1，多个0，或者全为0的情况，用#来分开时间。

仿真文件代码如下：

```
module Top_tb();
reg [7:0] sw;
wire [7:0] led;

initial begin
    sw = 0000_0000;
    #10;
    sw = 0000_0001;
    #10;
    sw = 0000_0011;
    #10;
    sw = 1000_0001;
    #10;
    sw = 1000_0000;
end

Top top(
    .sw(sw),
    .led(led)
);
endmodule
```

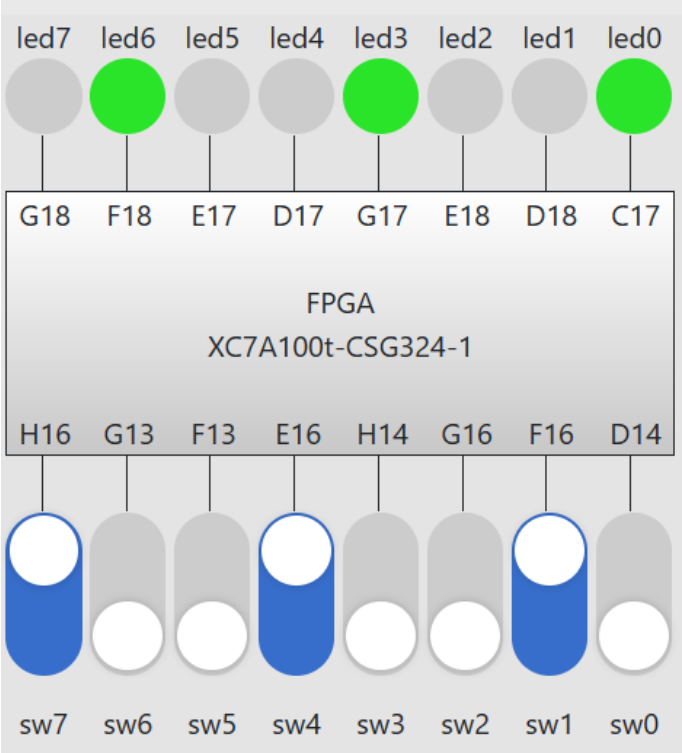
> 仿真得出的波形图如下：



> 对比分析得知程序正确。

测试结果与分析

> 对约束文件，开放相应的led和sw端口，并烧写成bit流文件，上板烧写结果如下。



> 测试运行得知，程序正确。

2. 必做内容：计数器 Pro Plus

> **要求：**
>
> 在FPGAOL上实现一个简易的闪光器，让LED[7:0]以1Hz的频率闪烁(亮0.5s，灭0.5s，以此循环)。
>
> **逻辑实现：**
>
> 参考实验教程中的流水灯代码，在计数器每次达到50_000_000时，将led翻转，即可实现亮灭交替的效果。其中为了仿真测试方便，初始化count_1hz为2，之后每次计数器到达最大值时都是0.5s。

代码如下：

```
module Flash(
    input clk,
    input btn,
    output reg [7:0] led
);

reg [31:0] count_1hz;
wire rst;
parameter TIME_CNT = 50_000_000;

initial begin
    led = 8'b1111_1111;
    count_1hz = 2;
```

```
end

assign rst = btn;   // Use button for RESET

always @(posedge clk) begin
    if (rst)
        count_1hz <= 0;
    else if (count_1hz >= TIME_CNT)
        count_1hz <= 0;
    else
        count_1hz <= count_1hz + 1;
end

always @(posedge clk) begin
    if (rst)
        led <= 8'b0000_0000;
    else if (count_1hz == 1) begin
        led <= ~led;
    end
end
endmodule
```

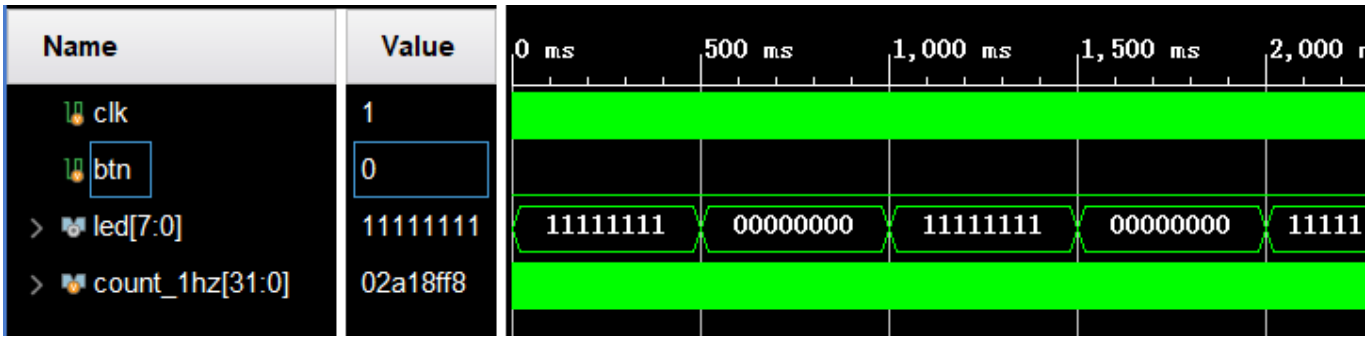
仿真结果与分析

> 根据所写的代码，编写相应的仿真文件验证其正确性。 时钟周期定为10ns，这样可以使时钟频率达到100MHz。运用模块例化分别调用端口，实现仿真。

仿真文件代码如下：

```
module Flash_tb();
reg clk,btn;
wire [7:0] led;

initial begin
    clk = 0; btn=0;
end

always #5 clk = ~clk;//T=10ns

Flash flash(
    .clk(clk),
    .btn(btn),
    .led(led)
);

endmodule
```

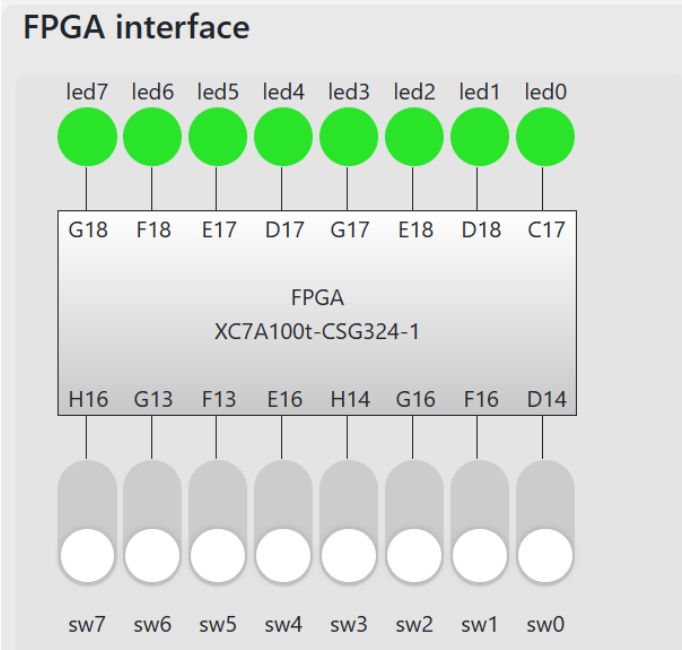
> 仿真得出的波形图如下：



> 对比分析得知程序正确。

测试结果与分析

> 对约束文件，开放相应的led端口，并烧写成bit流文件，上板烧写结果如下。



> 上板烧写后观察到，LED灯每0.5s亮一次,每0.5s灭一次，即可得知程序正确。

3. 必做内容：七段数码管

> **要求：**
>
> 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。为实现此功能，需要完成
>
> 1. 计时切换当前被点亮的数码管编号。
> 2. 根据编号确定数码管显示的内容。
>
> **逻辑实现：**
>
> 首先根据分时复用原理，每0.0025s就需要计数器计数一次，并且计数器每计数一次，需要数码管编号增加一位，以此实现分时复用。 这里直接将计数器的上线=限改为25_0000，让counter在0和最大值之间增长。当counter为1时，则数码管编号增加一位。 **这里需要注意数码管编号的范围是0~7，且if-else**

> **语句的完整性，防止出现锁存器或者程序错误。** 之后需要根据编号确定数码管显示的内容。这里用了case语句，将相应的out_data的内容赋值给seg_data，实现数码管的显示。

代码如下：

> 本题代码包括两个部分：计时器、Top模块

1. 计时器部分

```
module Segment(
    input                       clk,
    input                       rst,
    input       [31:0]          output_data,

    output reg  [3:0]          seg_data,
    output reg  [2:0]          seg_an
);
// 计时器部分
reg [31:0] counter;
parameter TIME_CNT = 25_0000;//计数器的上限为2.5*10^5
always @(posedge clk) begin
    if (rst) begin
        counter <= 0;
    end else begin
        if (counter >= TIME_CNT) begin
            counter <= 0;
        end else begin
            counter <= counter + 1;
        end
    end
end

// 数码管编号部分
reg [2:0] seg_id;
always @(posedge clk) begin
    if (rst) begin
        seg_id <= 0;
    end else begin
        if(counter == 1) begin
            if (seg_id < 3'b111) begin
                seg_id <= seg_id + 1;
            end else seg_id <= 0;
        end else seg_id <= seg_id;
    end
end


// 数码管显示内容部分
always @(*) begin
    seg_data = 0;
    if(rst) seg_an = 3'b000;
    else begin
```

```
        seg_an = seg_id;    // <- 对于所有情况都相同
        case (seg_an)
            0: seg_data = output_data[3:0];
            1: seg_data = output_data[7:4];
            2: seg_data = output_data[11:8];
            3: seg_data = output_data[15:12];
            4: seg_data = output_data[19:16];
            5: seg_data = output_data[23:20];
            6: seg_data = output_data[27:24];
            7: seg_data = output_data[31:28];
            default: ;
            // 可以根据需要继续添加其他编号的情况
        endcase
    end
end
endmodule
```

2. Top模块

```
module Top(
    input                   clk,
    input                   btn,
    output [2:0]            seg_an,
    output [3:0]            seg_data
);
    Segment segment(
        .clk(clk),
        .rst(btn),
        .output_data(32'h22111711),     // <- 改为你学号中的 8 位数字
        .seg_data(seg_data),
        .seg_an(seg_an)
    );
endmodule
```

仿真结果与分析

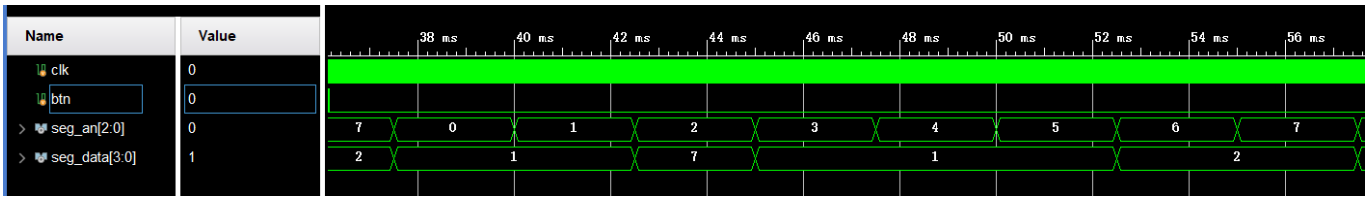
> 根据所写的代码，编写相应的仿真文件验证其正确性。 时钟周期定为10ns，这样可以使时钟频率达到100MHz。运用模块例化分别调用端口，实现仿真。

仿真文件代码如下：

```
module Top_tb();
reg clk,btn;
wire [2:0] seg_an;
wire [3:0] seg_data;

initial begin
    clk = 0;
```

```
    btn = 1;
    #10 btn = 0;
end
always #5 clk = ~clk;//T=10ns
Top top(
    .clk(clk),
    .btn(btn),
    .seg_an(seg_an),
    .seg_data(seg_data)
);

endmodule
```

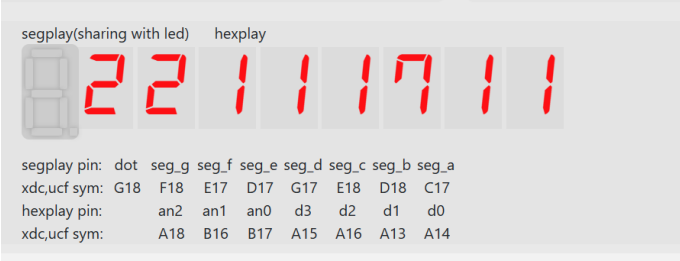
> 仿真得出的波形图如下：



> 我的学号是PB22111711，观察到波形中每位对应的数字正确，即可得知程序正确。

测试结果与分析

> 对约束文件，开放相应的seg_an、seg_data端口，并烧写成bit流文件，上板烧写结果如下。



> 观察比对得知程序正确。

4. 选择性必做内容：带有掩码的数码管

> **要求：**
>
> 请修改 Segment 模块，为前 7 个数码管添加掩码控制功能。
>
> **逻辑实现：**
>
> 对于计数器和数码管编号部分，不需要更改。 只需要对seg_an等于除0之外的值的情况，在case语句中加入if-else语句，首先判断output_valid的值是否为1，即检查该位的掩码，如果可以显示，则将相应位的output_data的值赋给seg_data，否则，让编号0位的七段数码管再次亮起。 这里同样运用了分时复用的思想，利用人眼识别的漏洞，实现带有掩码的数码管的显示功能。 对于Top模块，则需要增加sw输入量来实现掩码功能。

代码如下：

> 本题代码包括两个部分：计时器、Top模块

1. 计时器部分

```
module Segment(
    input                       clk,
    input                       rst,
    input       [31:0]          output_data,
    input       [ 7:0]          output_valid,

    output reg  [ 3:0]          seg_data,
    output reg  [ 2:0]          seg_an
);
// 计时器部分
reg [31:0] counter;
parameter TIME_CNT = 25_0000;//计数器的上限为2.5*10^5
always @(posedge clk) begin
    if (rst) begin
        counter <= 0;
    end else begin
        if (counter >= TIME_CNT) begin
            counter <= 0;
        end else begin
            counter <= counter + 1;
        end
    end
end

// 数码管编号部分
reg [2:0] seg_id;
always @(posedge clk) begin
    if (rst) begin
        seg_id <= 0;
    end else begin
        if(counter == 1) begin
            if (seg_id < 3'b111) begin
                seg_id <= seg_id + 1;
            end else seg_id <= 0;
        end else seg_id <= seg_id;
    end
end

// 数码管显示内容部分
always @(*) begin
    seg_data = 0;
    if(rst) seg_an = 3'b000;
    else begin
        seg_an = seg_id;    // <- 对于所有情况都相同
        case (seg_an)
            0: seg_data = output_data[3:0];
```

```
            1: if(output_valid[1] == 1) begin
                seg_data = output_data[7:4];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            2:if(output_valid[2] == 1) begin
                seg_data = output_data[11:8];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            3: if(output_valid[3] == 1) begin
                seg_data = output_data[15:12];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            4: if(output_valid[4] == 1) begin
                seg_data = output_data[19:16];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            5: if(output_valid[5] == 1) begin
                seg_data = output_data[23:20];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            6: if(output_valid[6] == 1) begin
                seg_data = output_data[27:24];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            7: if(output_valid[7] == 1) begin
                seg_data = output_data[31:28];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            default: ;
            // 可以根据需要继续添加其他编号的情况
        endcase
    end
end
endmodule
```

2. Top模块

```
module Top(
    input                   clk,
    input                   btn,
    input  [7:0]            sw,
    output [2:0]            seg_an,
    output [3:0]            seg_data
);
    Segment segment(
        .clk(clk),
        .rst(btn),
        .output_data(32'h22111711),// <- 改为你学号中的 8 位数字
        .output_valid(sw),
        .seg_data(seg_data),
        .seg_an(seg_an)
    );
endmodule
```

仿真结果与分析

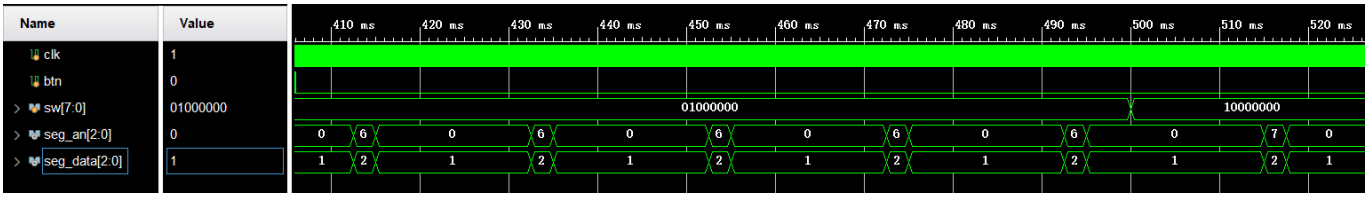
> 根据所写的计数器代码，编写相应的仿真文件验证其正确性。 分别对sw赋不同的值来实现上板时拨动开关的操作并观察波形。 其中考虑了全为0，有多个1和多个0的情况。

仿真文件代码如下：

```
module Top_tb();
reg clk,btn;
reg [7:0] sw;
wire [2:0] seg_an;
wire [2:0] seg_data;

initial begin
    clk = 0;
    btn = 1;
    sw = 8'b0000_0000;
    #10 btn = 0;
    #250_000_000;//250ms,1/4s
    sw = 8'b100_0000;
    #250_000_000;
    sw = 8'b1000_0000;
    #250_000_000;
    sw = 8'b0010_0000;
    #250_000_000;
    sw = 8'b1000_0011;
end
always #5 clk = ~clk;//T=10ns



Top top(
    .clk(clk),
```

```
    .btn(btn),
    .sw(sw),
    .seg_an(seg_an),
    .seg_data(seg_data)
);

endmodule
```

> 仿真得出的波形图如下：

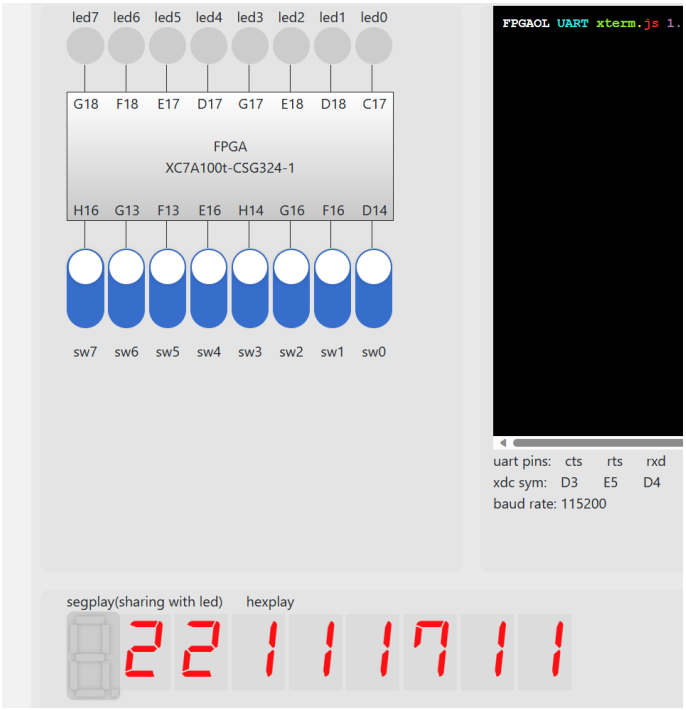


> 分析得知，对于不同的sw，都会在逐个判断应该显示哪个数码管的什么内容，即数码管编号及数码管显示内容。每0.0025s判断一次，不断重复，直到sw的值改变。分析可知程序正确。

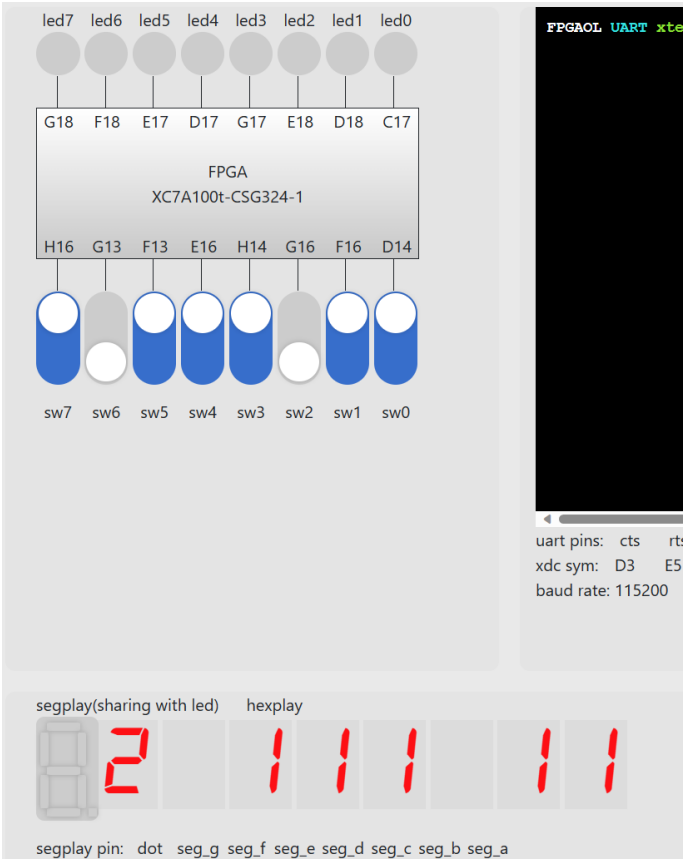
测试结果与分析

> 对约束文件，开放相应的seg_an、seg_data端口，并烧写成bit流文件，上板烧写结果如下。 分别测试了，开关全部打开，开关打开几个，和测试编号为0的数码管是否常亮三种情况。

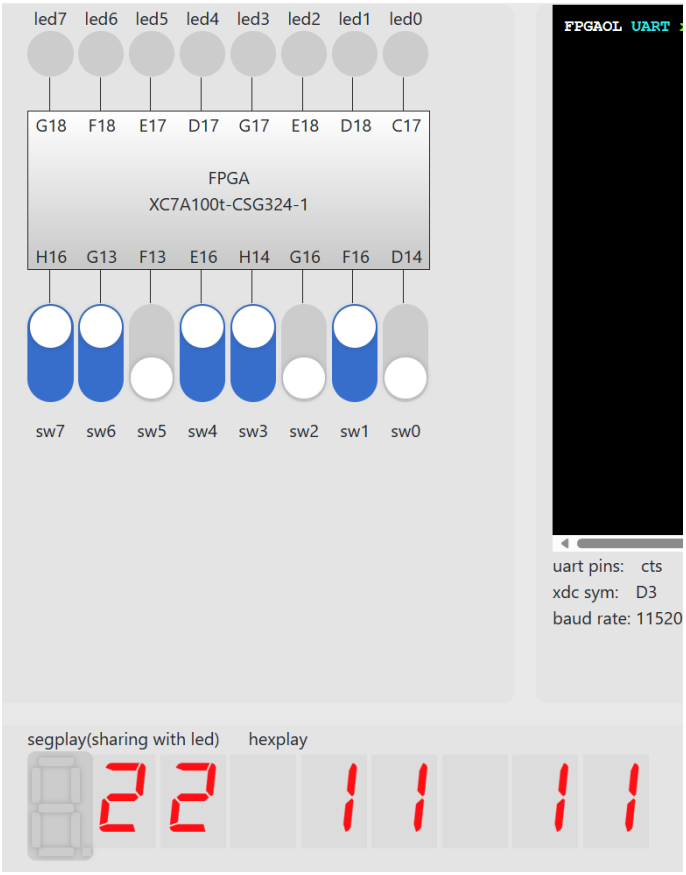
1. 开关全部打开：



1. 开关打开几个：



1. 测试编号为0的数码管：



>观察比对得知程序正确。

总结

> 1. 本次实验，对于Verilog语言有了更深入的了解，同时学会了上板分析和烧写比特流文件。

> 2. 学会使用Vivado创建项目并在FPGAOL上运行的完整过程。可以根据仿真和上板运行结果判断程序的正确与否。
> 3. 初步了解上板运行的过程，为今后的学习和实验打下基础。