数字电路实验

实验 1：基础实验
实验报告

选课编号：K0166210.02

姓名 周子涵 选课号 53

电子科技大学
通信与信息工程学院

2019 年 11 月

一、实验目的

1. 掌握使用硬件描述语言 Verilog 设计数字逻辑电路、模块和系统的方法 2. 掌握设计编写测试验证数字逻辑电路、模块和系统的激励文件的方法 3. 熟悉 Quartus II 软件工具的各项操作，熟练进行设计输入、综合、仿真 4. 熟悉 DE1-SOC 开发板的硬件电路结构以及下板验证的方法

二、 实验内容

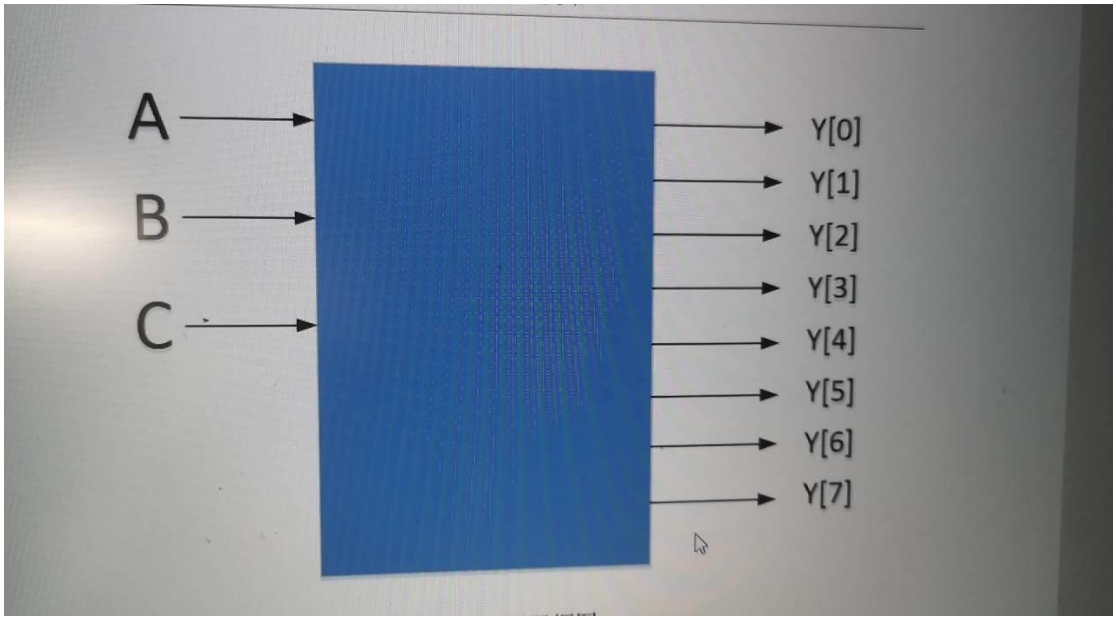
1.组合逻辑电路的实现 编写一个 2 输入的简化 ALU，其中利用 2bit 的操作码实现 4 种运算功能的选择， 完成相应的 Test Bench 编写，实现在 ModelSim 上的软件仿真，软件仿真无误后完 成下板验证操作。 2.时序电路的实现 根据状态转移图，实现一个流水灯的时序电路，完成相应的 Test Bench 编写， 实现在 ModelSim 上的软件仿真，软件仿真无误后完成下板验证操作。

三、实验设备及工具

软件：PC 机操作系统 WinXP 或 Win7、Quartus II 13.1、ModelSim-Altera 10.1d 硬件：DE1-SOC 实验板，计算机一台 仪器：示波器，逻辑分析仪

四、实验原理

组合逻辑电路：任意时刻的输出仅仅取决于该时刻的输入，与电路原来的状态 无关。图 3-1 是一个具有运算功能的简单译码器。该译码器实现的功能如下：



由功能描述语句可看到，组合逻辑电路的输出仅仅取决于该时刻的输入。为了 验证所建模的组合逻辑电路的功能是否与预期相符，可利用实验平台的软件仿真功 能。具体可使用 Quartus II 调用 modelsim 无缝仿真，通过编写激励文件设定各引脚 的信号输入，然后检查仿真后的输出是否与预期相符。通过软件仿真后的模块并不 等同于最后的实际电路，只有

当 HDL 语言建立的工程被下载到 FPGA 芯片中，该 模块才具有实际的功能意义。我们的 DE1 开发板不仅有 FPGA 核心芯片，还有丰富 的板载资源，通过绑定引脚的方式调用一些外设工作，可以帮助我们验证自己设计 的逻辑电路功能是否正确。 时序电路：具有记忆功能，输出不仅取决于当时的输入值，而且还与电路过去 的状态有关。图 3-2 是一个流水灯的状态转移图。

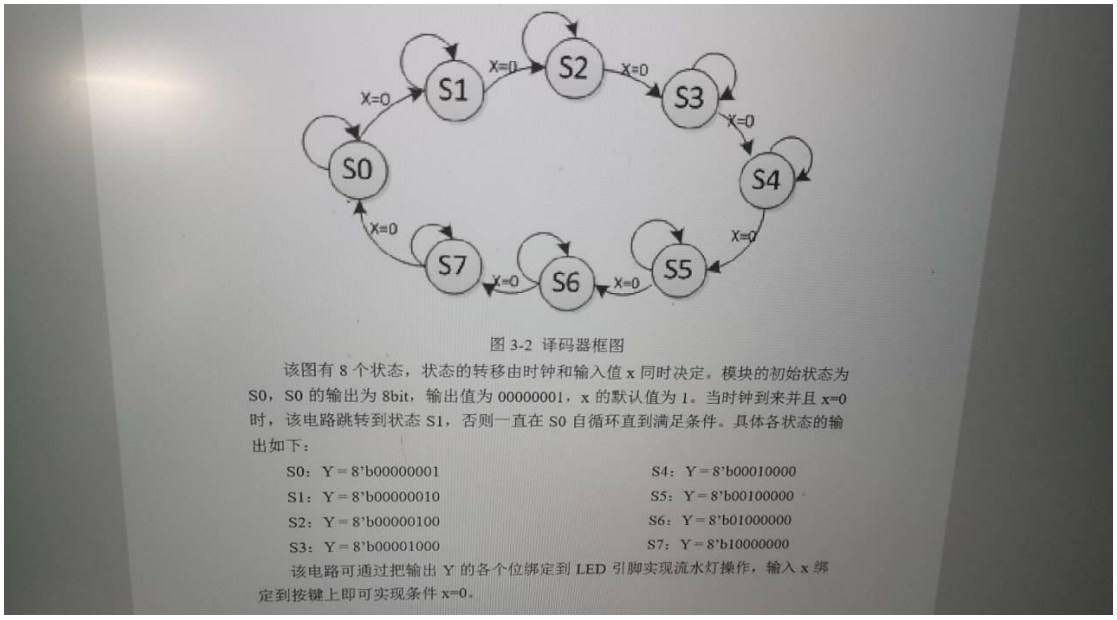


图 3-2 译码器框图

该图有 8 个状态，状态的转移由时钟和输入值 x 同时决定。模块的初始状态为 S0，S0 的输出为 8bit，输出值为 00000001，x 的默认值为 1。当时钟到来并且 x=0 时，该电路跳转到状态 S1，否则一直在 S0 自循环直到满足条件。具体各状态的输出如下：

S0：Y＝8'b00000001　　S4：Y＝8'b00010000
S1：Y＝8'b00000010　　S5：Y＝8'b00100000
S2：Y＝8'b00000100　　S6：Y＝8'b01000000
S3：Y＝8'b00001000　　S7：Y＝8'b10000000

该电路可通过把输出 Y 的各个位绑定到 LED 引脚实现流水灯操作，输入 x 绑定到按键上即可实现条件 x=0。

五、实验步骤

（该部分写清楚实验详细步骤，并且在模块设计部分附上 verilog 实现代码，在软件仿真部分附 testbench 的编写代码以及在下板实现部分附引脚绑定图截图）

双击打开 quartus 软件，选择 File>New Project Wizard。

点击 next。

选择工程存储目录（注意路径为英文路径），填写工程名称和顶层文件名（一 般情况下，工程名称和顶层文件名相同），点击 finish 完成工程创建。

点击选择 Verilog HDL File，然后点击 OK，这样就完成了一个 verilog 文件的 添加

添加 verilog 代码（注意工程名与 module 名须保持一致），点击保存

双击 Compile Design 下的 Analysis&Synthesis 按钮，综合 verilog 代码（若综 合失败，则根据软件下方的错误提示信息修改代码，重新综合，直到成功为止）

查看生成的 RTL 电路。综合完成以后，可以通过 RTL Viewer 来观察 v 代码 文件生成的对应的寄存器传输级电路，以辅助检查电路功能是否达到设计要求

点击 Assignments>Settings，进入设置界面。

点击 EDA Tool Settings>Simulation,这里可以设置仿真工具、仿真时间、仿真 文件等信息。在右边文件输出格式选项下选中 Verilog HDL，点击 ok 点击 Processing>Start>Start TestBench Template Writer,系统根据之前的工程自 动生成 testbench 文件的模板 。

根据软件提示的生成信息找到生成的 testbench 文件（在工程所保存目录下的 simulation/modelsim 里面），将其打开文件。

观察仿真结果。

首先需要选择下板验证需要的板子型号。右键工程名 decoder 选择 setting，单击右上角的 device，进入器件选择界面。

板子是 DE1，器件型号选择 5CSEMA5FC6，选好以后单击 OK。 选好器件以后，需要对

输入输出信号配置开发板上对应的管脚，管脚信息详见 开发板手册。在 quartus 主界面依次点击 Assignments>Pin Planer 进入管脚配置界面。
decoder 中，有 3 个输入 A、B、C 和 1 个八位的输出 Y[7:0]。分别对它们配置 成相应的管脚，如上图所示。 管脚配置成功以后，对工程进行综合（完整的过程）。综合的方式与仿真过程 一致。这里，我们双击 compile design，然后等待 quartus 自动综合。
编译成功以后，进入下板验证环节。首先给开发板连上电源线，然后用 USB 下 载线连接 PC 机与开发板，检查连好以后给开发板上电，轻按红色按钮。 单击 quartus 主界面的 programmer 按钮，进入程序的下载界面。此时会看见系 统资源管理器中显示插入一个未知设备，因此需要安装安装 USB-Blaster II 驱动。 选择这个未知的装置并更新它的驱动。用户可以在 \<Quartus II 安装路 径 >\drivers\ usb-blaster-ii 文件夹下找到这个装置的驱动 (ex :C:\Altera\13.1\quartus\drivers\usb-blaster-ii\)。 。 安装好后，会在下图红框处显示 DE-SOC 首先设置硬件信息为所用开发板型号， 模式选择 JTAG。
确认以后单击 Auto Detect，选择芯片型号 5CSEMA5
然后右键单击 file 窗口下的空白文件，选择 change file，在弹出的路径窗口选择 output_files 文件夹里面对应工程的.sof 文件。选择完毕以后勾选 sof 文件后面的 Program/Configure 选项。

配置成功以后，单击 start 按钮，开始将程序下载到开发板。下载成功以后会在 Progress 栏显示 100%（Successful）。

模块设计：3—8 译码器：

```
module decoder(A,B,C,Y);
    input A,B,C;
    output reg [7:0] Y;
    always@(*)
        begin
            case({A,B,C})
            3'b000:Y=8'b00000001;
            3'b001:Y=8'b00000010;
            3'b010:Y=8'b00000100;
            3'b011:Y=8'b00001000;
            3'b100:Y=8'b00010000;
            3'b101:Y=8'b00100000;
            3'b110:Y=8'b01000000;
            3'b111:Y=8'b10000000;
            default:Y=8'b00000000;
            endcase
        end
endmodule
```

流水灯：

```
module ledwater(
input clk,
```

```
input rst_n,
input x,
output    reg [7:0] Y
);


reg [2 : 0] state;
reg [31 : 0]cnt;
reg clk_div;

parameter
                S0=0,
                S1=1,
                S2=2,
                S3=3,
                S4=4,
                S5=5,
                S6=6,
                S7=7;



always@(posedge clk_div or negedge rst_n)
begin
    if(!rst_n)
    begin
    Y[7:0]<=8'b0000_0000;
    state<=S0;
    end
    else
    case(state)
    S0:
      begin
        Y[7:0]<=8'b0000_0001;
         if(x==1)
            state<=S0;
         else
            state<=S1;
      end
    S1:
      begin
        Y[7:0]<=8'b0000_0010;
         if(x==1)
            state<=S1;
```

```
        else
            state<=S2;
    end
S2:
    begin
        Y[7:0]<=8'b0000_0100;
        if(x==1)
            state<=S2;
        else
            state<=S3;
    end
S3:
    begin
        Y[7:0]<=8'b0000_1000;
        if(x==1)
            state<=S3;
        else
            state<=S4;
    end
S4:
    begin
        Y[7:0]<=8'b0001_0000;
        if(x==1)
            state<=S4;
        else
            state<=S5;
    end
S5:
    begin
        Y[7:0]<=8'b0010_0000;
        if(x==1)
            state<=S5;
        else
            state<=S6;
    end
S6:
    begin
        Y[7:0]<=8'b0100_0000;
        if(x==1)
            state<=S6;
        else
            state<=S7;
    end
S7:
```

```
        begin
           Y[7:0]<=8'b1000_0000;
            if(x==1)
               state<=S7;
            else
               state<=S0;
        end
     default: begin Y[7:0]<=8'b0000_0000;state<=S0; end
     endcase
end


Endmodule
```

软件仿真：3—8 译码器：

```
            // Copyright (C) 1991-2013 Altera Corporation
// Your use of Altera Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Altera Program License
// Subscription Agreement, Altera MegaCore Function License
// Agreement, or other applicable license agreement, including,
// without limitation, that your use is for the sole purpose of
// programming logic devices manufactured by Altera and sold by
// Altera or its authorized distributors.  Please refer to the
// applicable agreement for further details.

// ********************************************************************************
// This file contains a Verilog test bench template that is freely editable to
// suit user's needs .Comments are provided in each section to help the user
// fill out necessary details.
// ********************************************************************************
// Generated on "02/06/2016 08:28:02"

// Verilog Test Bench template for design : decoder
//
// Simulation tool : ModelSim-Altera (Verilog)
//

`timescale 1 ps/ 1 ps
module decoder_vlg_tst();
// constants
// general purpose registers
```

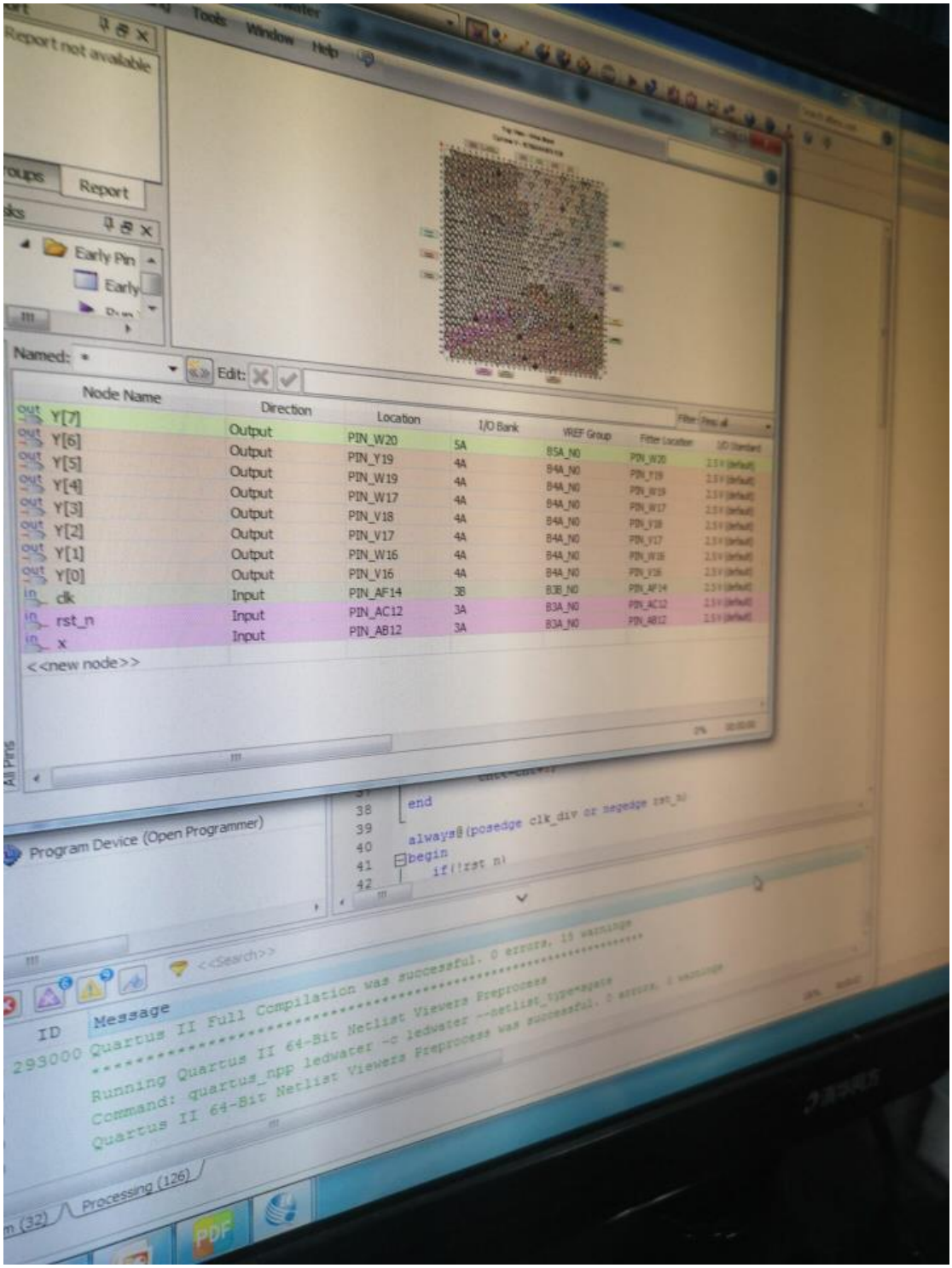
```
reg eachvec;
// test vector input registers
reg A;
reg B;
reg C;
// wires
wire [7:0]    Y;

// assign statements (if any)
decoder i1 (
// port map - connection between master ports and signals/registers
      .A(A),
      .B(B),
      .C(C),
      .Y(Y)
);
initial
begin
A=0;B=0;C=0;

#10 A=0;B=0;C=1;
#10 A=0;B=1;C=0;
#10 A=0;B=1;C=1;
#10 A=1;B=0;C=0;
#10 A=1;B=0;C=1;
#10 A=1;B=1;C=0;
#10 A=1;B=1;C=1;

#30 $stop;
end

endmodule
```

流水灯：

```
// applicable agreement for further details.

// ********************************************************************************
// This file contains a Verilog test bench template that is freely editable to
// suit user's needs .Comments are provided in each section to help the user
// fill out necessary details.
// ********************************************************************************
// Generated on "11/11/2019 15:14:39"

// Verilog Test Bench template for design : FlowLED
//
// Simulation tool : ModelSim-Altera (Verilog)
//

`timescale 1 ps/ 1 ps
module ledwater_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg clk;
reg reset;
// wires
wire [3:0]    led;

// assign statements (if any)
ledwater i1 (
// port map - connection between master ports and signals/registers
      .clk(clk),
      .led(led),
      .reset(reset)
);
initial
begin
clk=0;
x=0;
end
always
begin
#5 clk=~clk;
end
endmodule
```

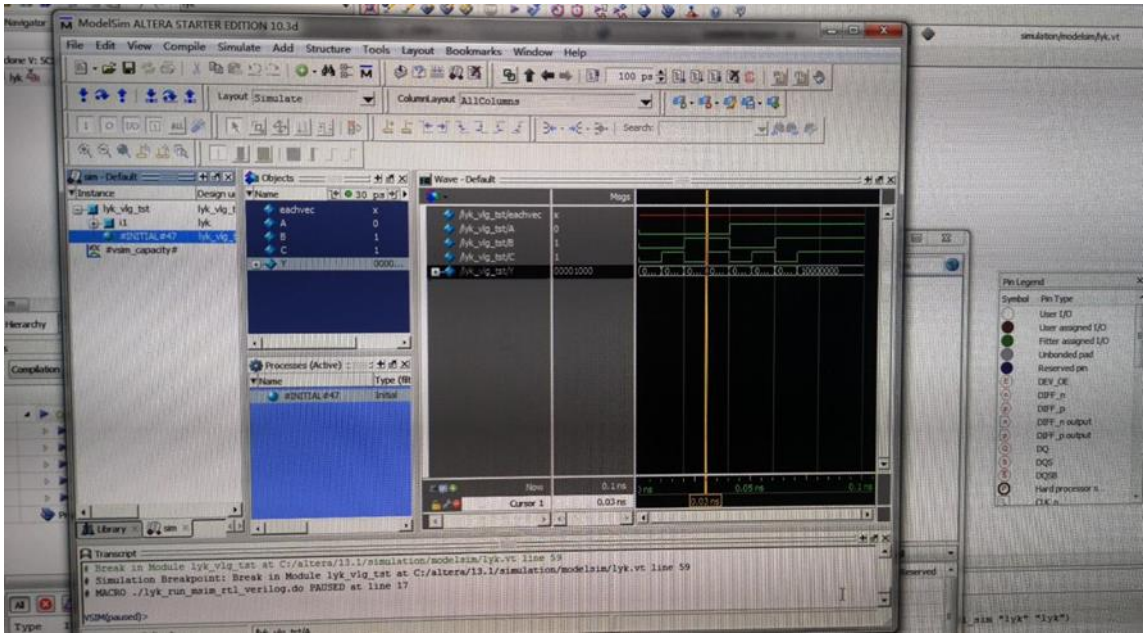
下板实现：3-8 译码器

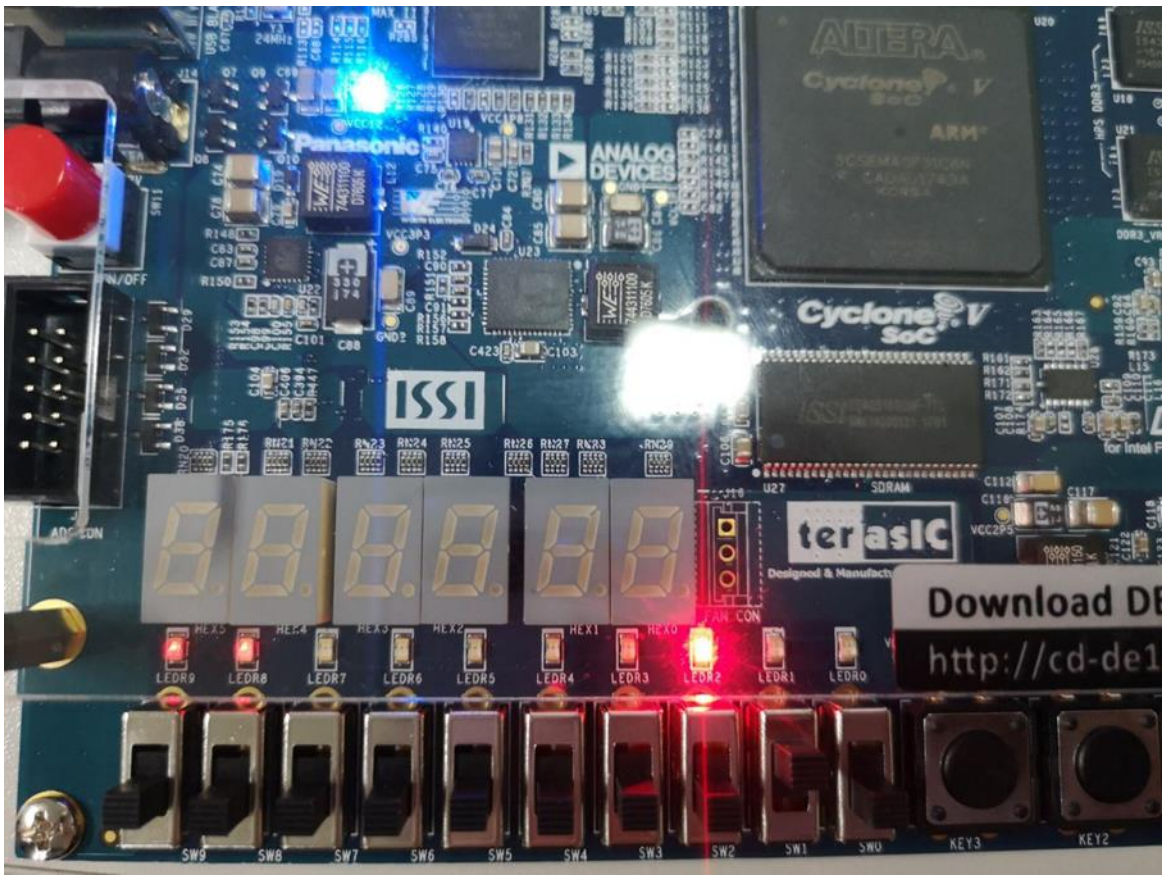
流水灯



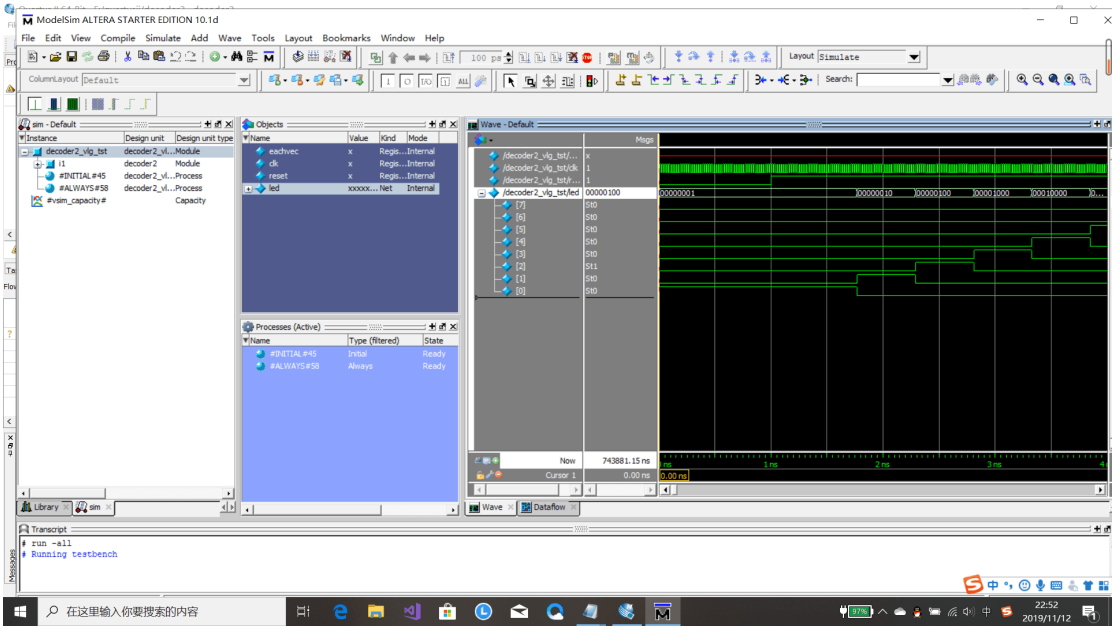
六、实验结果及分析

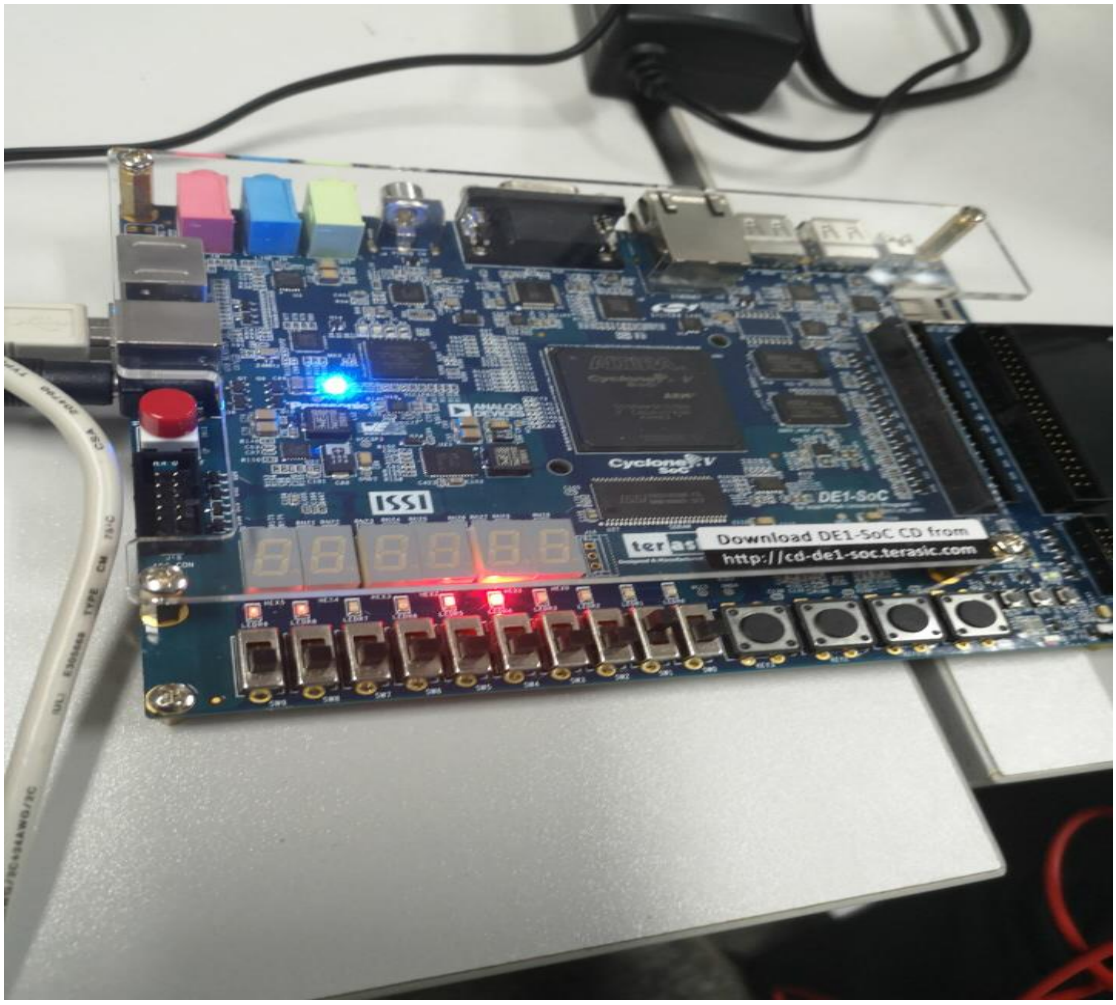
（该部分记录软件仿真结果的截图并附上对时序图的解释，最后附下板的实验现象照片）





二进制的高位到低位用 ABC 分别表示，当 A=0，B=1，C=0 时，表示输出为 2，第三个灯亮，如二图所示；当 A=0,B=1,C=1 时，表示输出为 3，对应 00001000，如图





clk 是时钟信号，时钟信号在不停地做周期变化。rst_n 是复位信号，所以当经过复位信号有效后，由于 x 一直为 0，因此开始计数。输出的八位二进制中，1 的位置对应的就是 LED 灯亮的位置。如：若输出为 00000100，则第三个灯亮。