**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：lab2\_寄存器堆与队列

学生姓名：胡毅翔

学生学号：PB18000290

完成日期：2020年5月5日

计算机实验教学中心制

2019年09月

# **实验目的**

1. 掌握寄存器堆（Register File）和存储器（Memory）的功能、时序及其应用
2. 熟练掌握数据通路和控制器的设计和描述方法

# **实验环境**

1. PC一台
2. Windows或Linux操作系统
3. Vivado
4. vlab.ustc.edu.cn
5. Logisim
6. Nexys4DDR开发板

# **实验设计**

## **实验1 寄存器堆**

### **设计目标**

设计参数化的寄存器堆，其逻辑符号如图1所示。该寄存器堆含有32 个寄存器（r0 ~ r31，其中r0的内容恒定为零），寄存器的位宽由参数WIDTH指定，具有2个异步读端口和1个同步写端口。数据通路如图2。

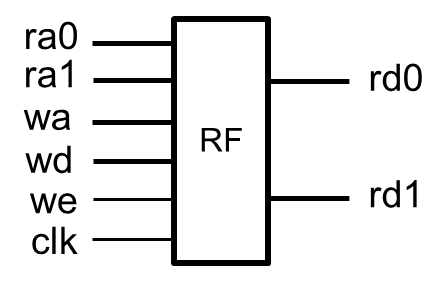


图 1

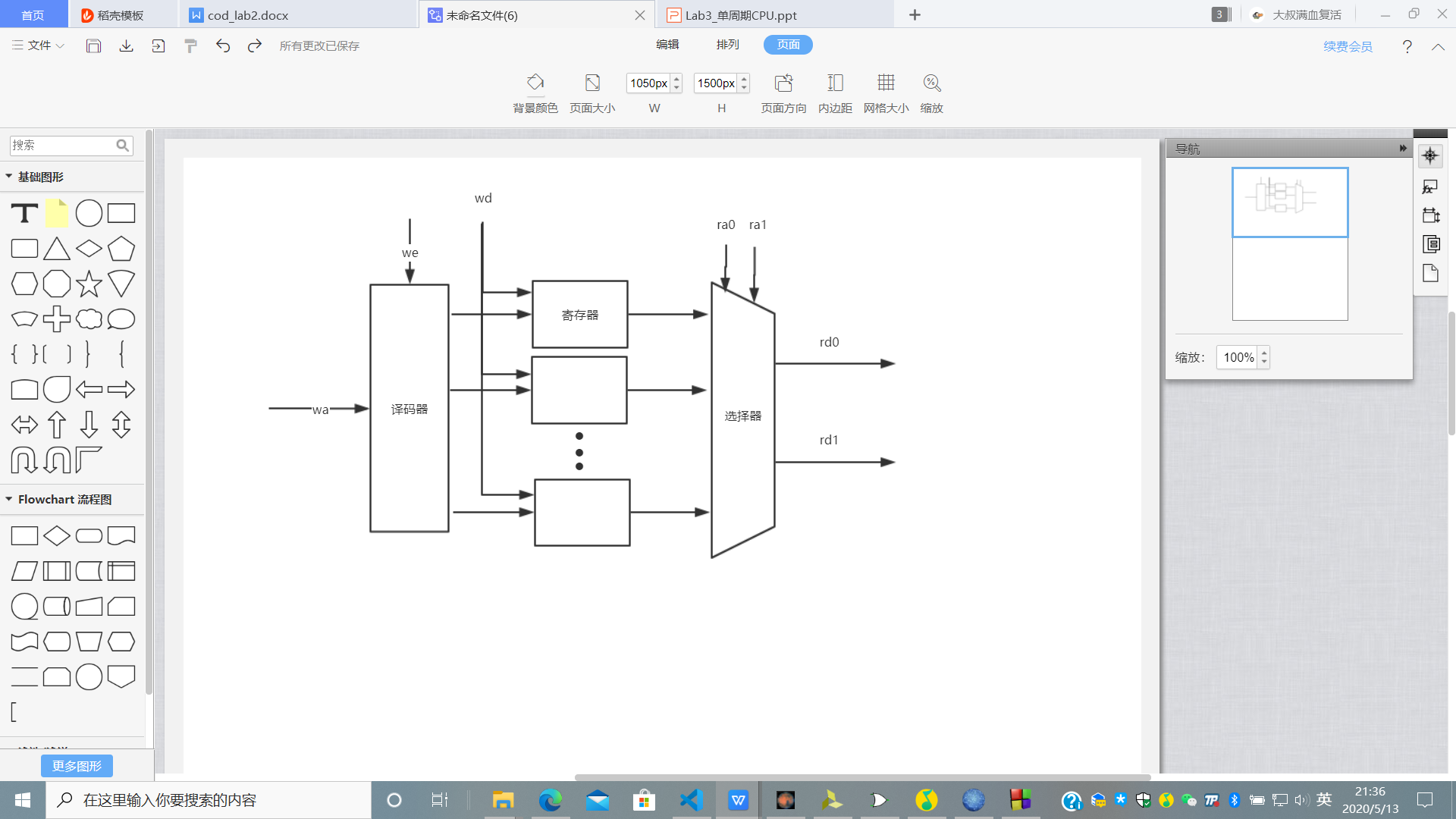


图 2

### **端口声明**

端口声明如下：

module register\_file  //32 x WIDTH register file

       #( parameter WIDTH = 32 ) (  //data width

         input clk,  //clock

         input [ 4: 0 ] ra0,  //read address 0

         output [ WIDTH - 1: 0 ] rd0, //read data 0

         input [ 4: 0 ] ra1,  //read address 1

         output [ WIDTH - 1: 0 ] rd1, //read data 1

         input [ 4: 0 ] wa,  //write address

         input we,  //write enable

         input [ WIDTH - 1: 0 ] wd  //write data

       );

### **设计实现**

将对应地址的数据送至对应端口，同时根据写信号是否有效，写入数据。

### **核心代码**

核心代码如下：

assign rd0 = mem[ ra0 ];

assign rd1 = mem[ ra1 ];

always @( posedge clk )

  begin

    if ( we )

      begin

        mem[ wa ] <= wd;

      end

  end

### **仿真结果**

仿真结果如下(图2)：

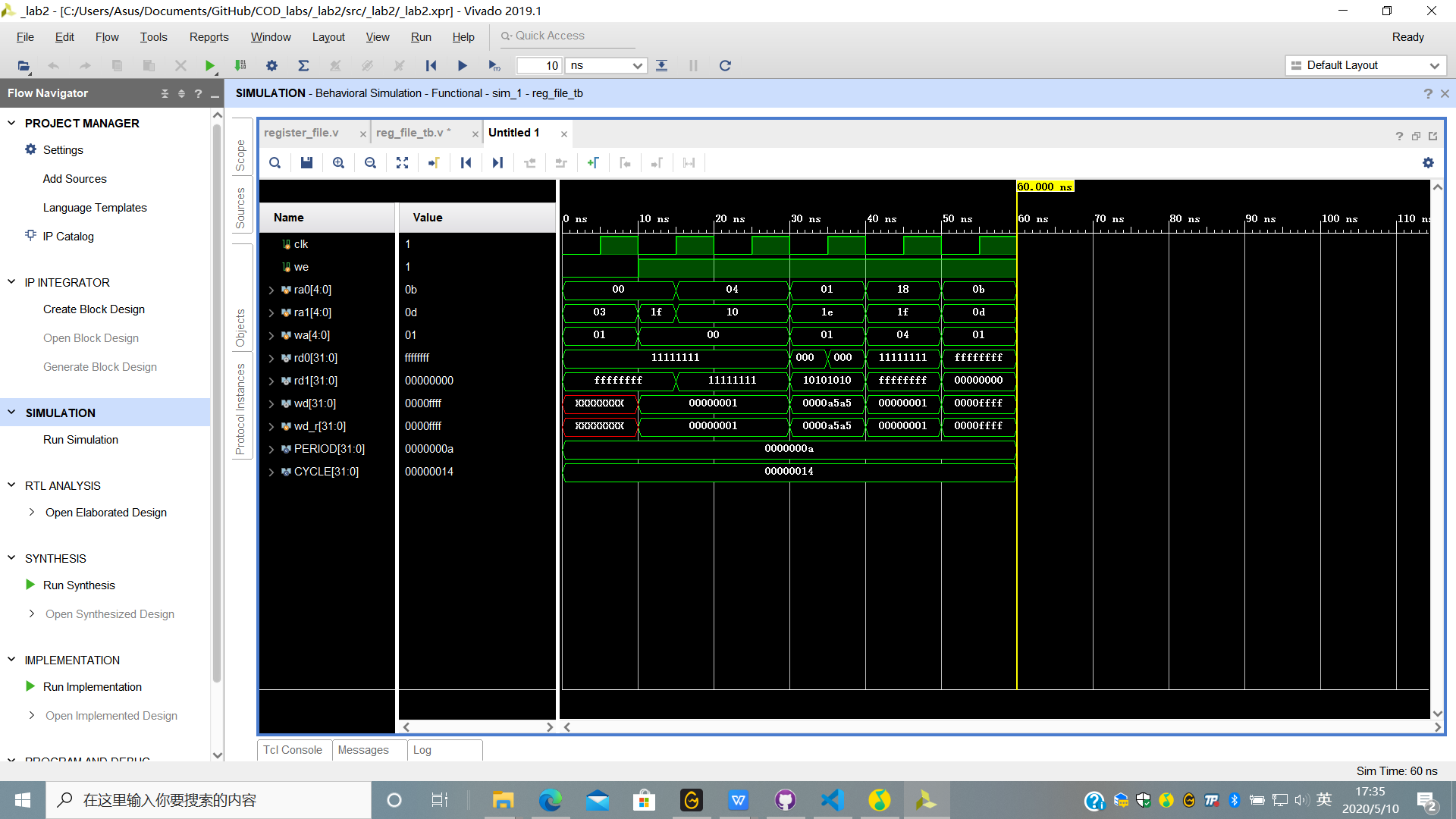


图 3

### **结果分析**

仿真结果表明设计，运行效果与设计目标相同。

## **实验2 存储器**

### **设计目标**

存储器与寄存器堆的功能类似，都是用于存储信息，只是存储器的容量更大，配置方式更多，例如ROM/RAM、单端口/简单双端口/真正双端口、分布式/块式等方式。设计存储器可以通过行为方式描述，也可以通过IP例化方式实现。

设计一容量为16 x 8位（即深度DEPTH：16，宽度WIDTH：8）的单端口RAM，其逻辑符号如图3所示。

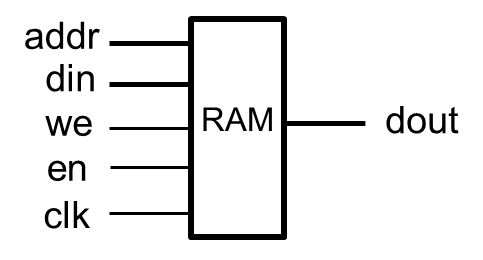


图 4

### **端口声明**

端口声明如下：

module  ram\_16x8       //16x8bit single portRAM

      (input  clk,       //clk (posegde enable)

        input en, we,        //enable,write enable

        input [ 3: 0 ]  addr, //address

        input [ 7: 0 ]  din,  //input data

        output [ 7: 0 ]  dout //output data

      );

### **设计实现**

数据通路如下：利用IP核中的Block Memory Generator以及Distributed Memory Generator，分别生成16×8位块式存储器及16×8位分布式存储器

### **仿真结果**

#### **块式存储器**

仿真结果如下：

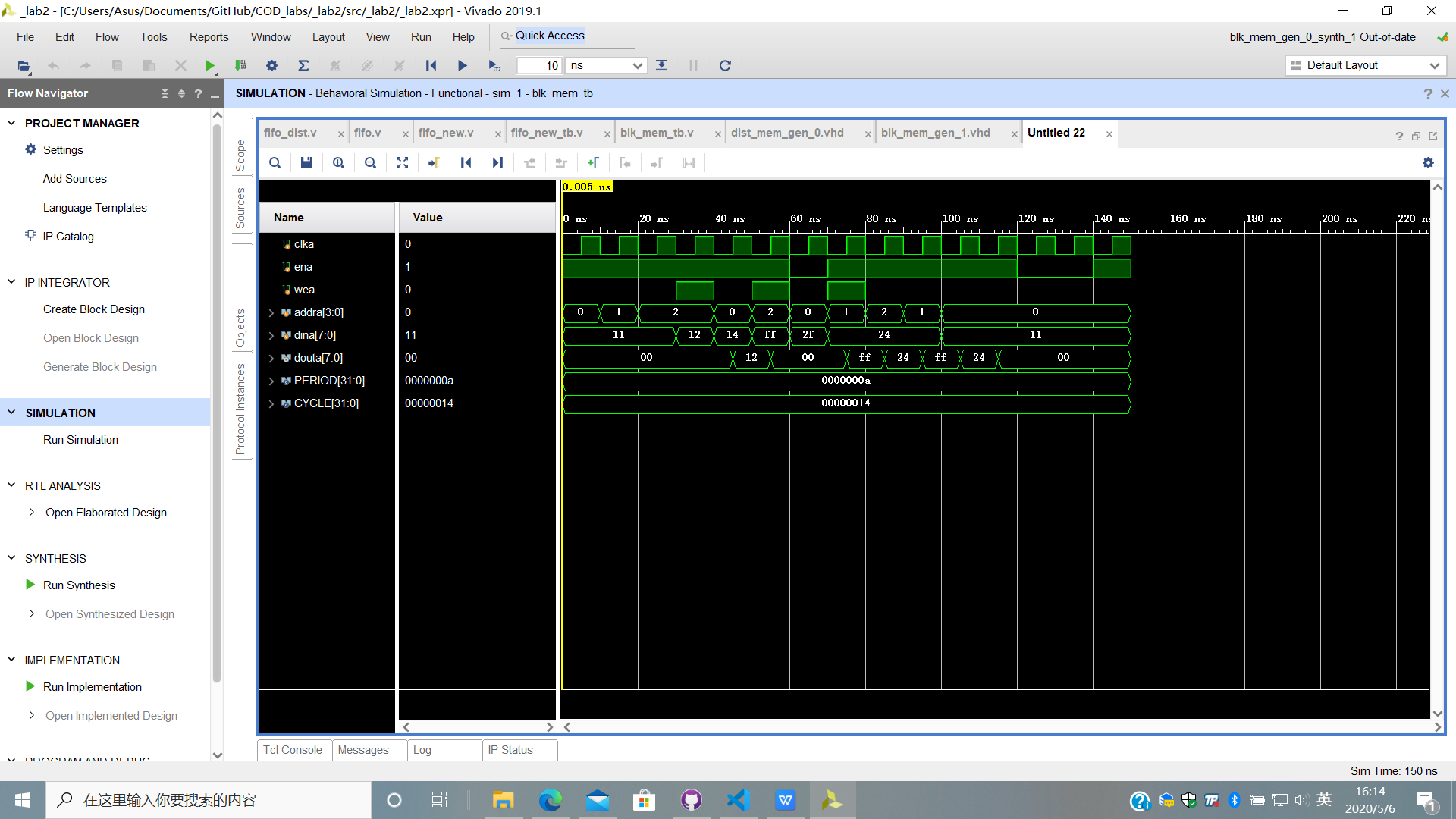


图 5

#### **分布式存储器**

仿真结果如下：

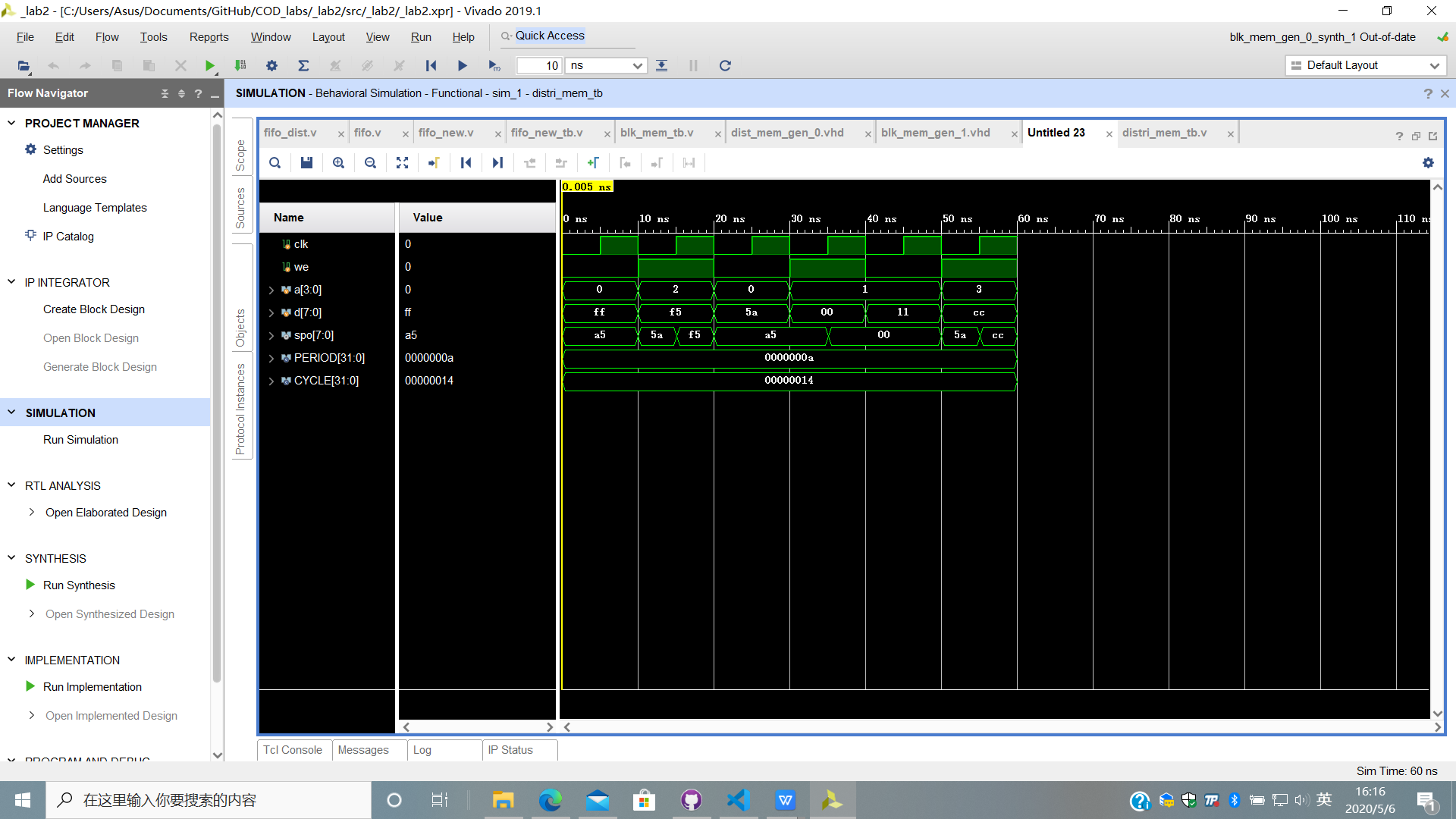


图 6

## **实验3 FIFO队列**

### **设计目标**

利用例化的存储器IP（16 x 8位块式的单端口RAM）和适当的逻辑电路，设计实现数据宽度为8位、最大长度为16的FIFO队列，其逻辑符号如图-9所示。入队列使能（en\_in）有效时，将输入数据（din）加入队尾；出队列使能（en\_out）有效时，将队列头数据输出（dout）。队列数据计数（count）指示队列中有效数据个数。当队列满（count = 16）时不能执行入队操作，队列空（count = 0）时不能进行出队操作。在入对使能信号的一次有效持续期间，仅允许最多入队一个数据，出队操作类似。FIFO队列逻辑符号及数据通路如下：

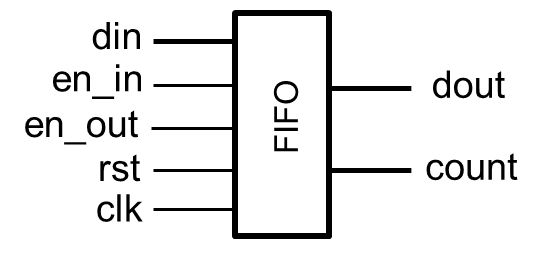


图 7

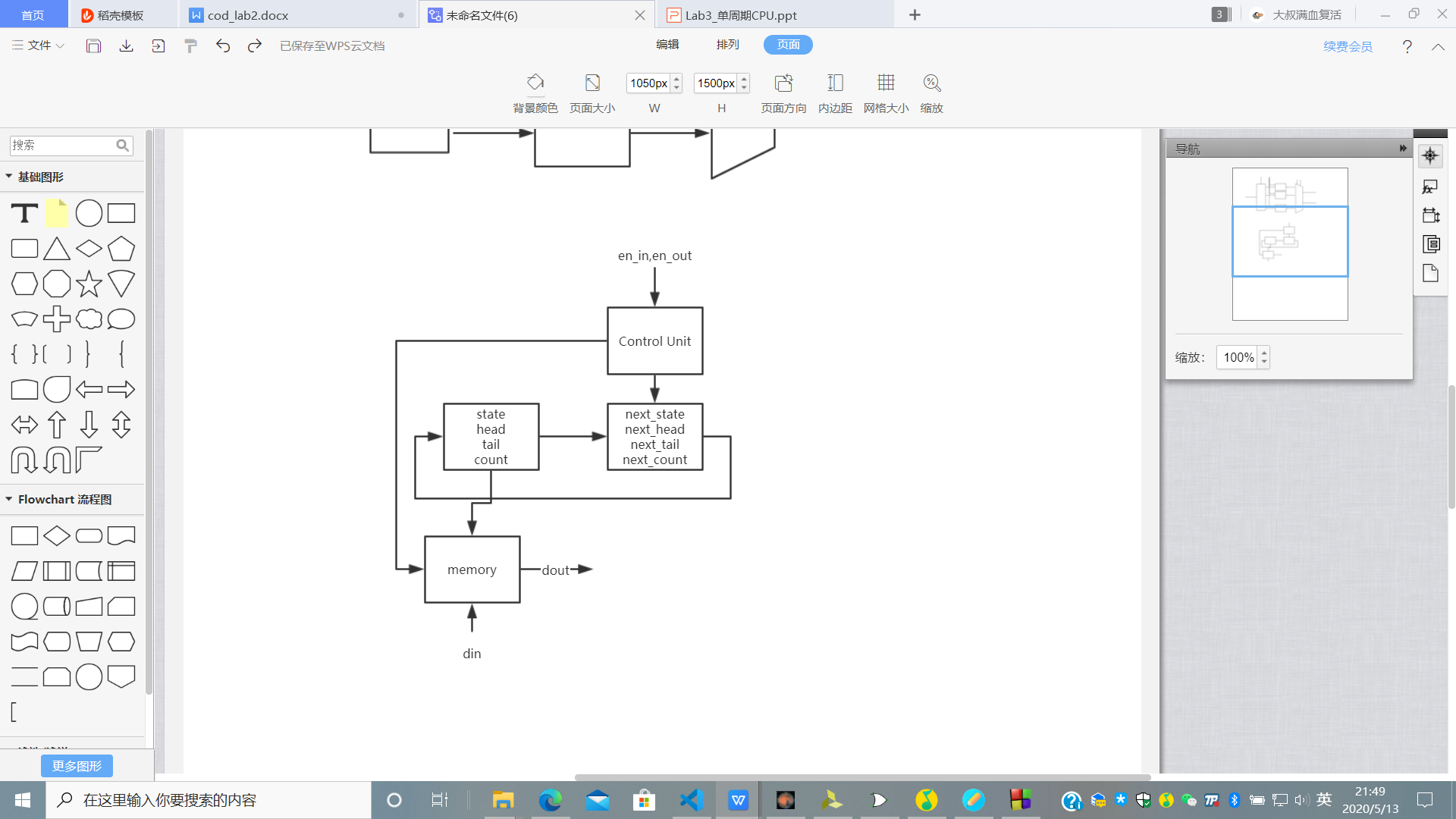


图 8

由上述目标知，须完成取信号边沿的模块，取信号边沿后的使能信号仅维持一个周期。

### **端口声明**

#### **取信号边沿**

module edg(

         input clk, rst,//clock,reset

         input y,       //input signal

         output p       //outout signal

       );

#### **FIFO队列**

module fifo\_dist

       (

         input clk, rst,  //clock,reset

         input [ 7: 0 ] din,  //enqueue data

         input en\_in,  //enqueue enable

         input en\_out,  //dequeue enable

         output [ 7: 0 ] dout,   //dequeue data

         output reg [ 4: 0 ] count //length of queue

       );

### **设计实现**

#### **取信号边沿**

通过有限状态机实现，状态图如下：

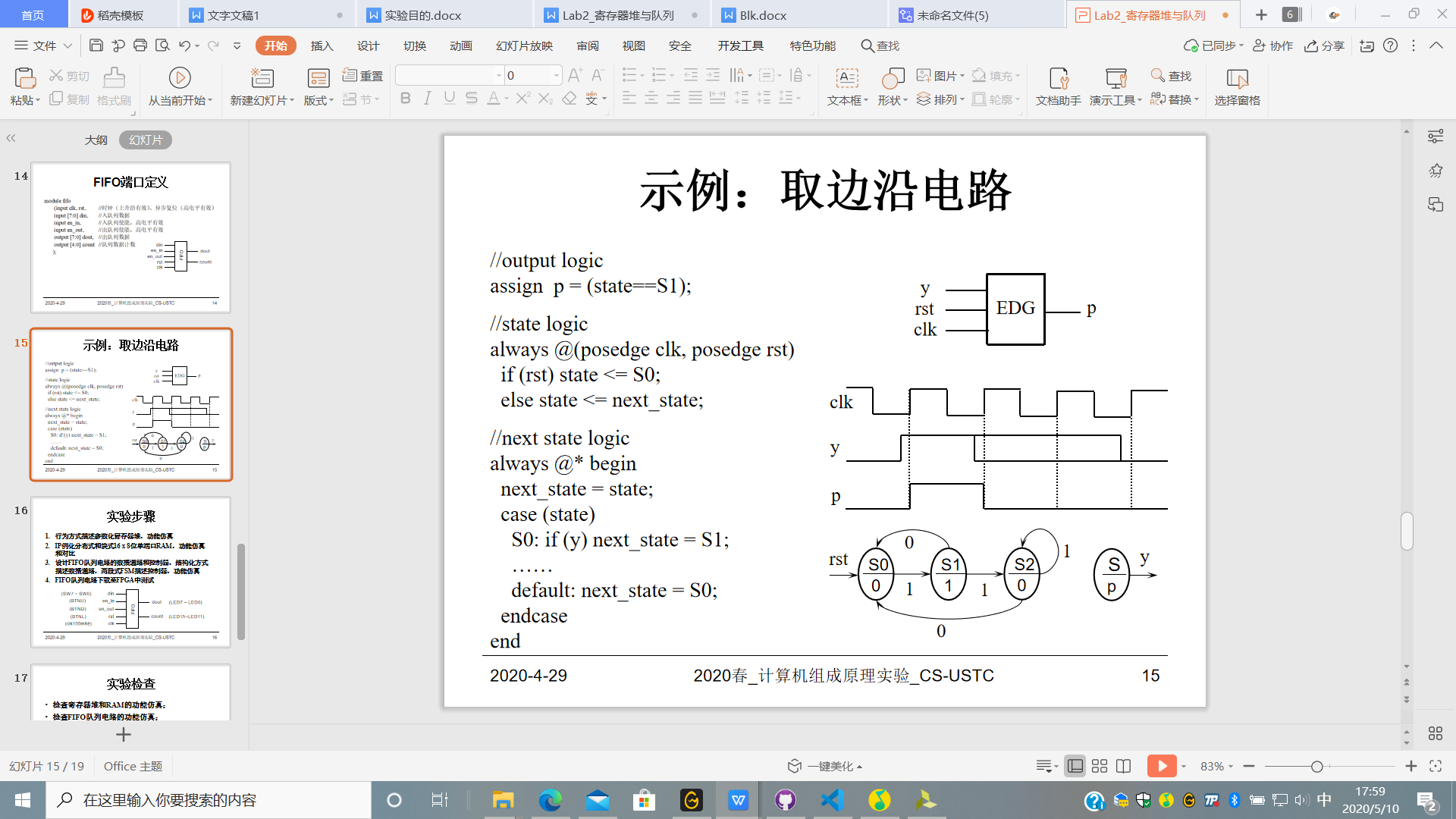


图 9

#### **FIFO队列**

通过有限状态机实现，分为队空，队满，其他三个状态。

状态图如下：

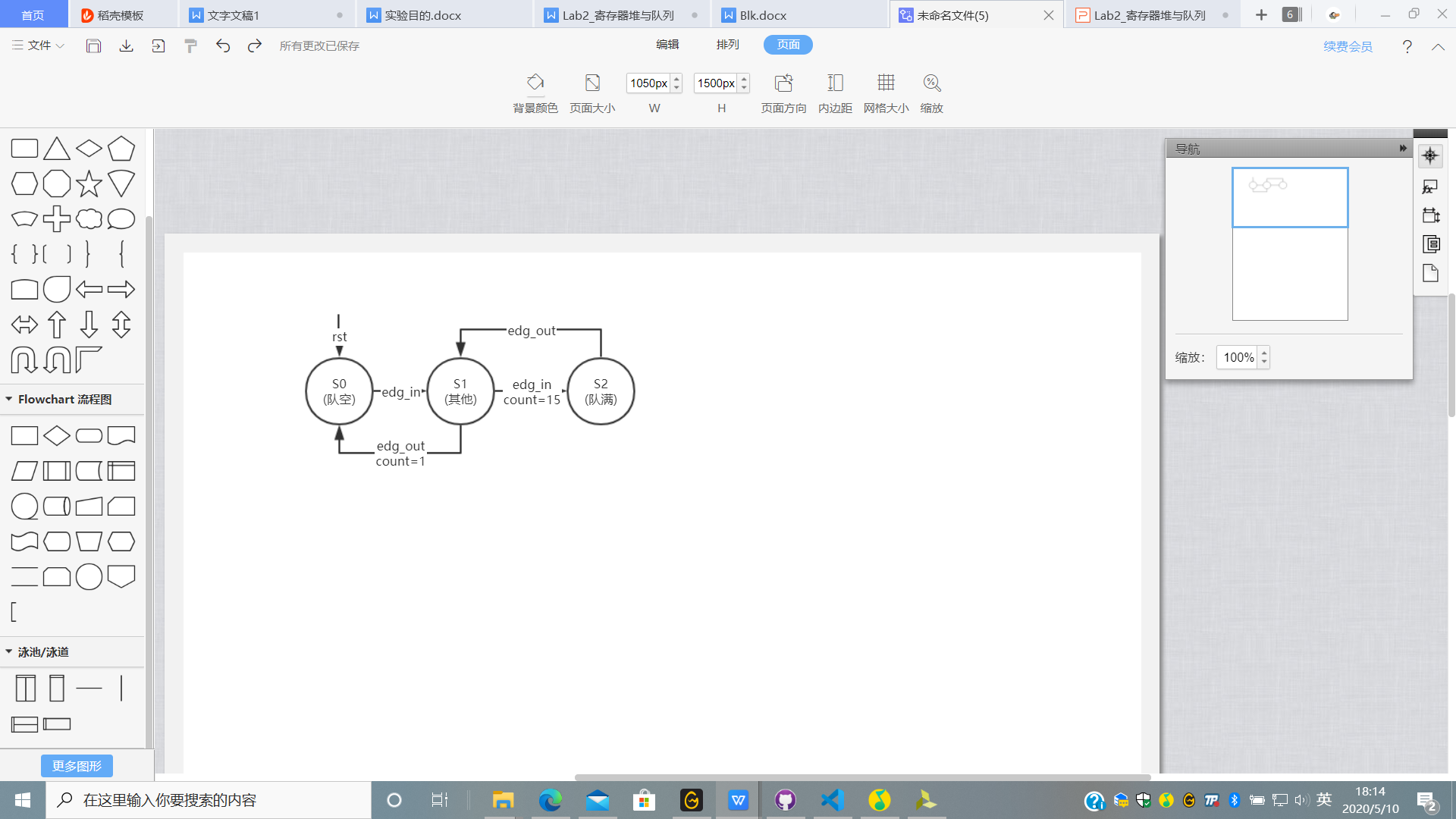


图 10

同时通过两段式状态机，修改head，tail，count等数据，实现对队列的控制。

### **核心代码**

#### **取信号边沿**

//output logic

assign p = ( state == S1 );

//next state logic

always @ \*

  begin

    next\_state = state;

    case ( state )

      S0:

        if ( y )

          next\_state = S1;

      S1:

        if ( y )

          next\_state = S2;

        else

          next\_state = S0;

      S2:

        if ( !y )

          next\_state = S0;

      default:

        next\_state = S0;

    endcase

  end

#### **FIFO队列**

//next\_state logic

always @( posedge clk, posedge rst )

  begin

    if ( rst )

         next\_state = 2'b0;

    else

      begin

        case ( { edg\_in, edg\_out, state } )

        4'b1000:

            next\_state <= S1;

        4'b1001:

            if ( count == 5'd15 )

              next\_state <= S2;

        4'b0110:

          next\_state <= S1;

        4'b0101:

            if ( count == 5'd1 )

               next\_state <= S0;

        default:

          next\_state <= next\_state;

      endcase

  end

状态的控制如上代码所示，其他状态值，如head，tail，count作类似处理即可。

### **仿真结果**

#### **取信号边沿**

仿真结果如下：

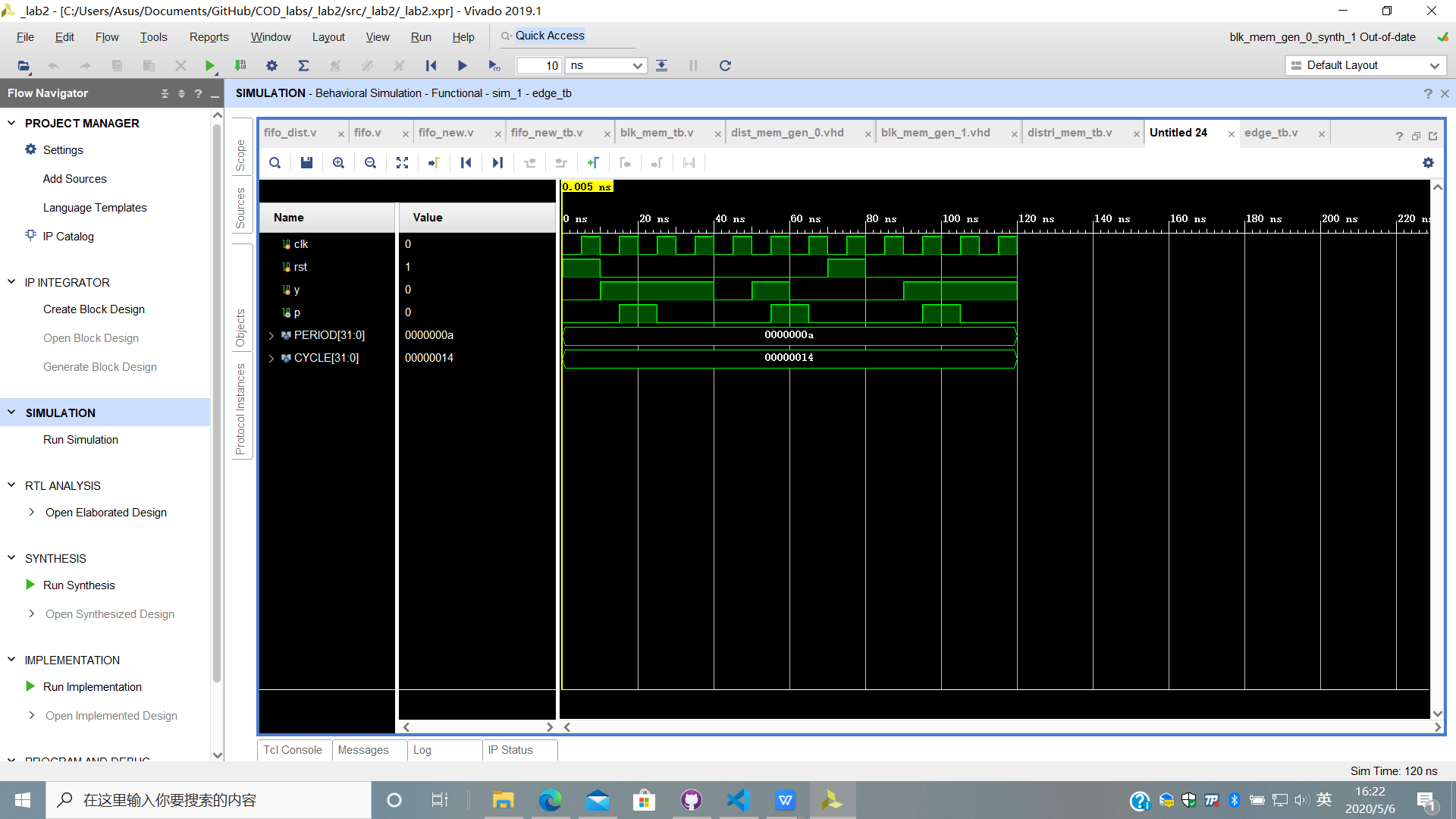


图 11

#### **FIFO队列**

仿真结果如下：

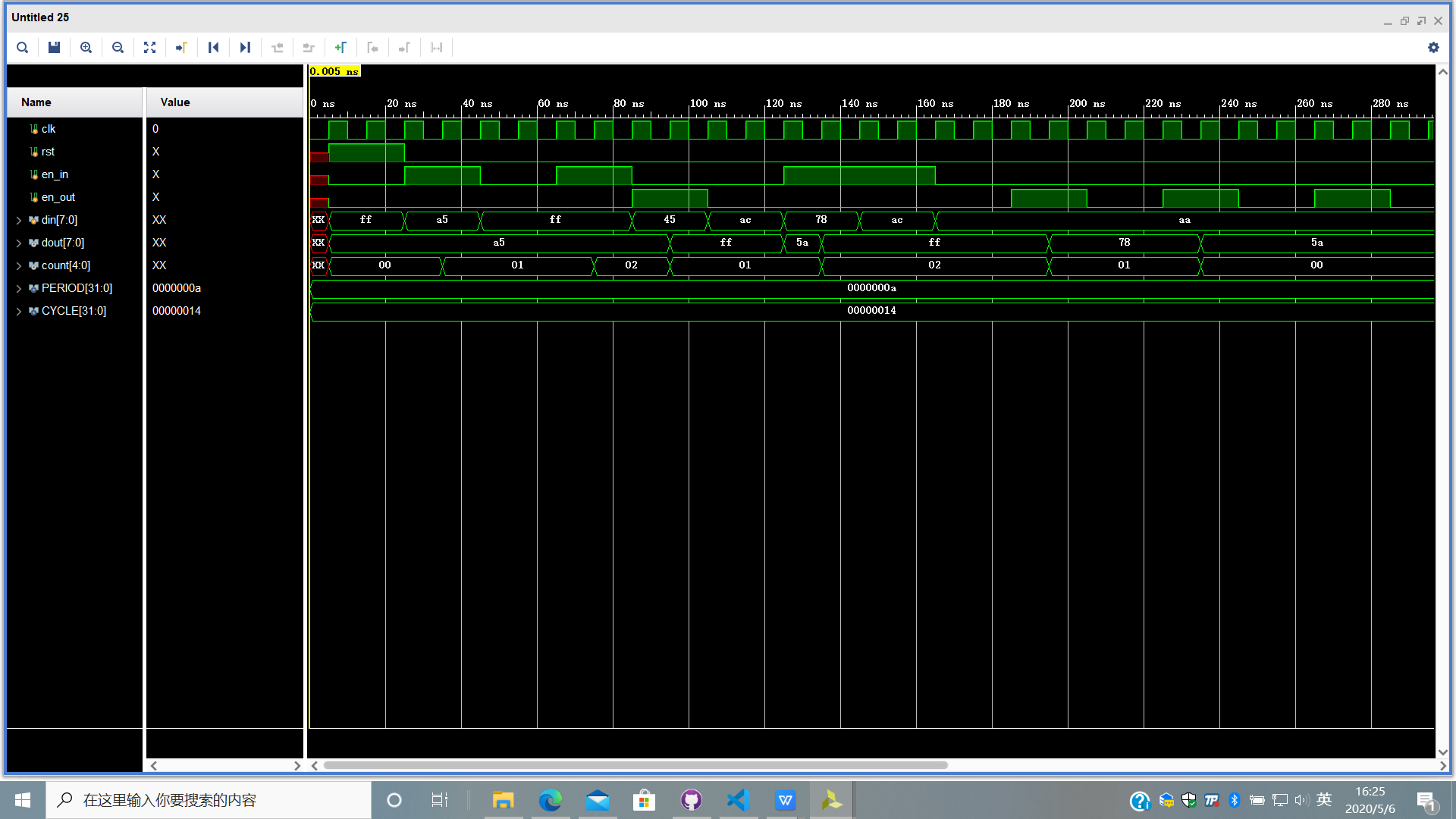


图 12

## **结果分析**

队列的仿真结果符合设计目标。

## **实验总结**

本次实验复习了上学期所学的Verilog的基本知识，并通过寄存器堆、存储器及队列，实现了对组合逻辑电路和时序逻辑电路的巩固，为下一个实验单周期CPU设计打下基础。

## **意见建议**

希望能提供testbench文件，以统一检查效果。

## **思考题**

### **题目**

如何利用寄存器堆和适当电路设计实现可变个数的数据排序电路。

### **解答**

已在实验报告中的进阶目标中体现 对每个数据增加顺序域(4位)和有效位(1位)，共5位。每次有数据入队时，将该数据的有效位置1，同时与队列中的数据比较。根据比较结果(若入队数据比某队内数据大，则返回1)，根据返回的1的个数，对入队数据的顺序域赋值。若入队数据比某队内数据小，则该队内数据的顺序域加一。出队时，同理，将出队数据与其他队内数据比较，修改对应顺序域，同时将出队数据的顺序域及有效位置0。