浙江大学

本科实验报告

课程名称：	数字逻辑电路设计
姓　　名：	
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
邮　　箱：	
QQ　号：	
电　　话：	
指导教师：	洪奇军
报告日期：	2020 年 12 月 8 日

浙江大学实验报告

课程名称：数字逻辑设计 实验类型：综合

实验项目名称：实验十一——寄存器堆及寄存器传输设计

学生姓名： 学号： 同组学生姓名：

实验地点：紫金港东四 509 室 实验日期：2020 年 12 月 8 日

一、实验目的

1.1 掌握寄存器结构及设计实现方法；
1.2 掌握寄存器堆的工作原理及设计实现方法；
1.3 掌握寄存器传输控制及设计实现方法；
1.4 掌握基于总线的寄存器传输设计；
1.5 了解计算机中寄存器及寄存器堆的概念；

二、实验内容和原理

2.1 实验内容

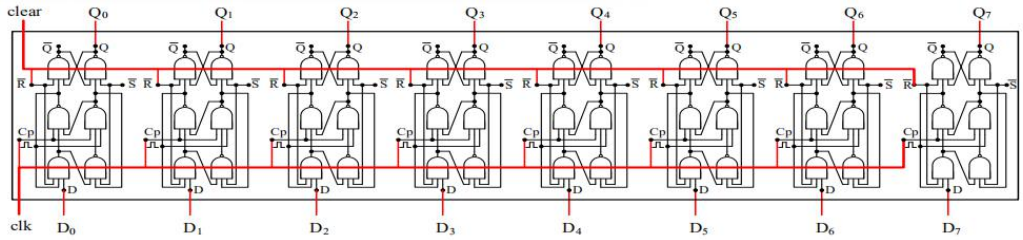
- 设计 32 位时钟写入的寄存器；
- 设计 8×32bits 的寄存器堆(组)；
- 集成实验环境接口，实现寄存器传输控制 ALU 运算；

2.2 实验原理

2.2.1 寄存器

寄存器是计算机常用的基本器件，由同一信号（Clk）控制的一组触发器(锁在器)组成。其包含存储、处理和传输等功能，可以构成寄存器组使用。

其原理图如图表 2.2.1-1 所示。

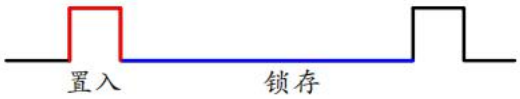


图表 2.2.1-1 八位寄存器

也可以采用 Verilog 代码描述八位寄存器。

2.2.2 锁存器

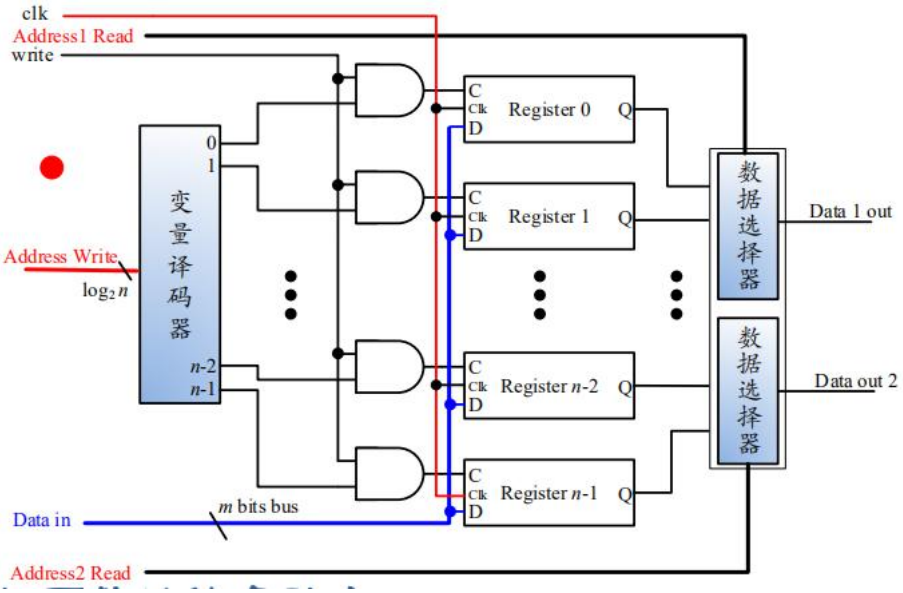
锁存器由多个一位锁存器/触发器并联构成，一般采用电平控制锁存。锁存器在高电平时置入数据，在低电平时锁存数据，为后续逻辑电路提供持续的信号。



图表 2.2.2-1 锁存器功能原理图

2.2.3 寄存器组

寄存器组是计算机的基本部件，是多个寄存器的集合。写入数据时，由寄存器地址控制数据通过变量译码器写入寄存器。读取数据时，由寄存器地址控制数据通过数据选择器进入数据通路。



图表 2.2.3-1 寄存器组功能原理图

三、操作方法与实验步骤

3.1 实验设备与材料

1. 装有 ISE 14.7 的计算机　　1 台
2. SWORD 开发板　　1 套

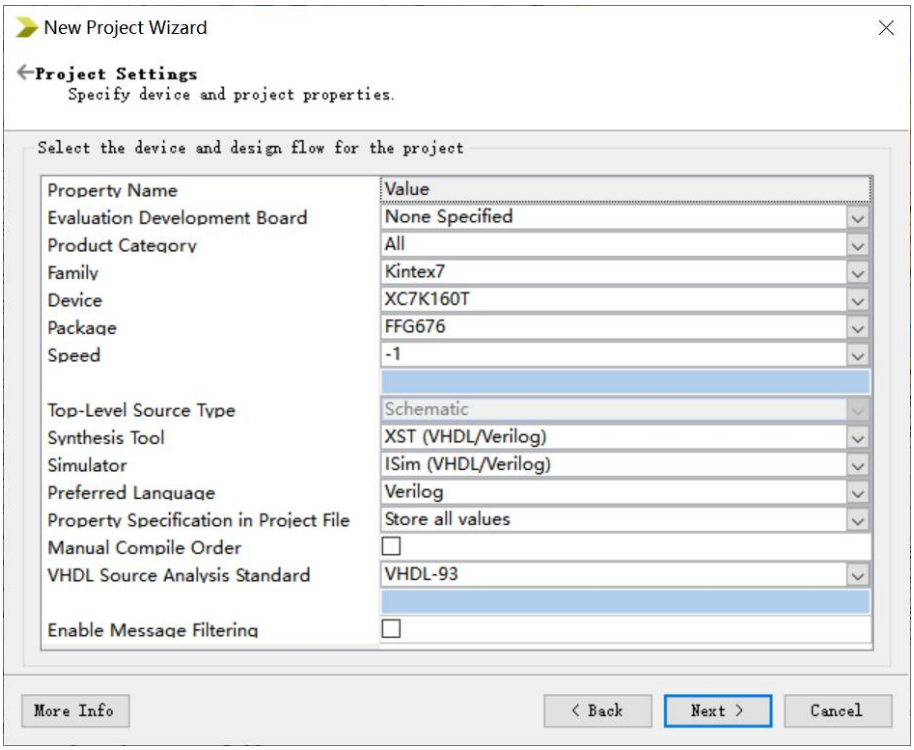
3.2 实验步骤

3.2.1 设计实现 32 位寄存器

1. 建立工程

打开 ISE Design Suite 14.7，在左上角点击 File，再点击 New Project 命名为 Exp11-REGS。将工程命名，选择 Top-level source type 为 HDL 并选择项目保存位置。

点击 Next 后，出现如图表 3.2.1-1 所示的对话框，如图表所示选择对应的 Family、Device、Package、Speed 等属性，确认无误后一直点击 Next 直至创建工程结束。



图表 3.2.1-1 项目配置

2. 设计 8 位寄存器

在左侧 Sources 窗口空白处右键菜单中选择 New Source。

在对话框中，选择源类型为 Verilog Module，输入文件名 Regs_8bit，并且勾选下方 Add to Project。此后，一直点击 Next 直至结束。

此后，双击打开左侧 Sources 窗口中的 Regs_8bit.v 文件，调用之前实验的 MB_DFF，输入代码：

```
`timescale 1ns / 1ps

module Regs_8bit(input clk,
                 input [7:0]D,
                 input clear,
                 output [7:0]Q
    );

    wire [7:0]Qbar;
    assign cr=~clear;

    MB_DFF
T0(.Cp(clk), .D(D[0]), .Rn(cr), .Sn(1'b1), .Q(Q[0]), .Qn(Qbar[0])),

T1(.Cp(clk), .D(D[1]), .Rn(cr), .Sn(1'b1), .Q(Q[1]), .Qn(Qbar[1])),

T2(.Cp(clk), .D(D[2]), .Rn(cr), .Sn(1'b1), .Q(Q[2]), .Qn(Qbar[2])),

T3(.Cp(clk), .D(D[3]), .Rn(cr), .Sn(1'b1), .Q(Q[3]), .Qn(Qbar[3]));
    MB_DFF
T4(.Cp(clk), .D(D[4]), .Rn(cr), .Sn(1'b1), .Q(Q[4]), .Qn(Qbar[4])),

T5(.Cp(clk), .D(D[5]), .Rn(cr), .Sn(1'b1), .Q(Q[5]), .Qn(Qbar[5])),

T6(.Cp(clk), .D(D[6]), .Rn(cr), .Sn(1'b1), .Q(Q[6]), .Qn(Qbar[6])),

T7(.Cp(clk), .D(D[7]), .Rn(cr), .Sn(1'b1), .Q(Q[7]), .Qn(Qbar[7]));


endmodule
```

完成后，在 Source 窗口单击选中 Regs_8bit.v 文件，然后双击下方 Process 窗口的 Synthesize 下的 Check Syntax 检查语法。

3. 设计 32 位寄存器

左侧 Sources 窗口空白处右键菜单中选择 New Source。

在对话框中，选择源类型为 Verilog Module，输入文件名 Reg32，并且勾选下方 Add to Project。此后，一直点击 Next 直至结束。

此后，双击打开左侧 Sources 窗口中的 Reg32.v 文件，输入行为描述代码如下：

```
`timescale 1ns / 1ps

module Reg32(input clk,
               input [31:0]D,
               input clear,
               input Load,
               output [31:0]Q
    );

    wire [31:0]Qbar;
    wire [31:0]Di;

    assign Di=Load?D:Q;
    BUFG cc(clk1, clk);

    Regs_8bit R0(.clk(clk1), .D(Di[7:0]), .clear(clear), .Q(Q[7:0]));
    Regs_8bit R1(.clk(clk1), .D(Di[15:8]), .clear(clear), .Q(Q[15:8]));
    Regs_8bit
R2(.clk(clk1), .D(Di[23:16]), .clear(clear), .Q(Q[23:16]));
    Regs_8bit
```

```
R3(.clk(clk1), .D(Di[31:24]), .clear(clear), .Q(Q[31:24]));


endmodule
```

完成后，进行仿真验证。仿真代码如下：

```
`timescale 1ns / 1ps

module reg32_test;

    // Inputs
    reg clk;
    reg [31:0] D;
    reg clear;
    reg Load;

    // Outputs
    wire [31:0] Q;

    // Instantiate the Unit Under Test (UUT)
    Reg32 uut (
        .clk(clk),
        .D(D),
        .clear(clear),
        .Load(Load),
        .Q(Q)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        D = 0;
        clear = 0;
        Load = 0;

    fork
        forever #20 clk<=~clk;
        #20; clear=0;
        begin
            D=32'hAAAAAAAA;
            #50; Load<=1;
            #40; Load<=0;

            D=32'h55555555;
            #20; Load<=1;
            #40; Load<=0;

            D=32'hA5A5A5A5;
            #70; Load<=1;
            #30; Load<=0;
            #40; clear=1;
            #30; clear=0;
            #25; Load<=1;
            #45; Load<=0;

            D=32'h5A5A5A5A;
            #65; Load<=1;
            #45; Load<=0;
        end
    join
```

```
    end

endmodule
```

3.2.2 设计实现 8×32 位寄存器组

在 Exp11-REGS 工程下，左侧 Sources 窗口空白处右键菜单中选择 New Source。

在对话框中，选择源类型为 Verilog Module，输入文件名 Regs_8_32，并且勾选下方 Add to Project。此后，一直点击 Next 直至结束。

此后，双击打开左侧 Sources 窗口中的 Regs_8_32.v 文件，调用之前实验完成的 HCT138，输入行为描述代码如下：

```
`timescale 1ns / 1ps

module Regs_8_32(input clk,
                 input cr,
                 input WE,
                 input [2:0]Addr_W,
                 input [2:0]Addr_A,
                 input [2:0]Addr_B,
                 input [31:0]Di,
                 output [31:0]QA,
                 output [31:0]QB
    );

    wire [7:0]Yi, Y;
    wire [7:0]CLK_R;
    wire [2:0]SEL;
    wire [31:0]Do0, Do1, Do2, Do3, Do4, Do5, Do6, Do7;

    assign Y=~Yi;
    assign CLK_R={8{clk}};

    Reg32 R0(CLK_R[0], Di, cr, Y[0], Do0);
    Reg32 R1(CLK_R[1], Di, cr, Y[1], Do1);
    Reg32 R2(CLK_R[2], Di, cr, Y[2], Do2);
    Reg32 R3(CLK_R[3], Di, cr, Y[3], Do3);
    Reg32 R4(CLK_R[4], Di, cr, Y[4], Do4);
    Reg32 R5(CLK_R[5], Di, cr, Y[5], Do5);
    Reg32 R6(CLK_R[6], Di, cr, Y[6], Do6);
    Reg32 R7(CLK_R[7], Di, cr, Y[7], Do7);

    HCT138_sch D(.A(Addr_W[0]),
                 .B(Addr_W[1]),
                 .C(Addr_W[2]),
                 .G(WE),
                 .G_2A(1'b0),
                 .G_2B(1'b0),
                 .Y0(Yi[0]),
                 .Y1(Yi[1]),
                 .Y2(Yi[2]),
                 .Y3(Yi[3]),
                 .Y4(Yi[4]),
                 .Y5(Yi[5]),
                 .Y6(Yi[6]),
                 .Y7(Yi[7]));

    MUX8T1_32
MUX_REGA(.I0(Do0), .I1(Do1), .I2(Do2), .I3(Do3), .I4(Do4), .I5(Do5), .I6
```

```
(Do6), .I7(Do7),
                            .s(Addr_A), .o(QA));

    MUX8T1_32
MUX_REGB(.I0(Do0), .I1(Do1), .I2(Do2), .I3(Do3), .I4(Do4), .I5(Do5), .I6
(Do6), .I7(Do7),
                            .s(Addr_B), .o(QB));


endmodule
```

完成后，进行仿真验证。仿真代码如下：

```
`timescale 1ns / 1ps

module reg832_test;

    // Inputs
    reg clk;
    reg cr;
    reg WE;
    reg [2:0] Addr_W;
    reg [2:0] Addr_A;
    reg [2:0] Addr_B;
    reg [31:0] Di;

    // Outputs
    wire [31:0] QA;
    wire [31:0] QB;

    // Instantiate the Unit Under Test (UUT)
    Regs_8_32 uut (
        .clk(clk),
        .cr(cr),
        .WE(WE),
        .Addr_W(Addr_W),
        .Addr_A(Addr_A),
        .Addr_B(Addr_B),
        .Di(Di),
        .QA(QA),
        .QB(QB)
    );

    integer i=0;
    initial begin
        // Initialize Inputs
        clk = 0;
        cr = 1;
        WE = 0;
        Addr_W = 0;
        Addr_A = 0;
        Addr_B = 0;
        Di = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
    fork
        forever #20 clk<=~clk;
        #10; cr=0;
```

```
        begin
            for(i=0; i<8; i=i+2)begin
                Addr_W<=i;
                Addr_A<=i;
                Addr_B<=i;
                Di<=32'hAAAAAAA0+i;
                #10; WE<=1;
                #15; WE<=0;
                #5;

                Addr_W<=i+1;
                Addr_A<=i+1;
                Addr_B<=i+1;
                Di<=32'h55555551+i;
                #20; WE<=1;
                #15; WE<=0;
                #15;
            end

            WE=0;
            for(i=0; i<8; i=i+1)begin
                #30;
                Addr_W<=i;
                Addr_A<=i;
                Addr_B<=i;
            end
        end
    join

    end

endmodule
```

仿真结果确认无误后，创建逻辑符号，方便加入实验十的框架进行物理验证。

3.2.3 寄存器组物理验证

1. 建立工程

打开 ISE Design Suite 14.7，在左上角点击 File，再点击 New Project 命名为 RTL。

将工程命名，选择 Top-level source type 为 Schematic 并选择项目保存位置。

点击 Next 后，选择对应的 Family、Device、Package、Speed 等属性，确认无误后一直点击 Next 直至创建工程结束，同 3.2.1。

2. 设计必要元件

复制实验十中含有 ALU、Counter_4bit 以及 counter_32_rev 的 Framework，加入新设计的 Regs_8_32。还要补充一个 32 位的 2 选 1 选择器 MUX2T1_32，计时器 Timer 和毫秒信号发生器 ms1。

分别创建相应的 Verilog Module 文件，MUX2T1_32.v 代码如下：

```
`timescale 1ns / 1ps

module MUX2T1_32(input s,
                 input [31:0]I0,
                 input [31:0]I1,
                 output [31:0]o
    );

     assign o=s?I1:I0;
```

```
endmodule
```

Timer.v 代码如下：

```
`timescale 1ns / 1ps

module Timer(input clk,
              input Up,
              input Load,
              input Start,
              input [31:0]Timing_const,
              output reg [31:0]cnt,
              output reg Alarm
    );

    reg [1:0]go;

    always@(posedge clk or posedge Start)begin
        if(Start)begin
            go<=2'b01;
            Alarm<=0;
            cnt<=Timing_const;
        end
        else begin
            if(Load)cnt<=Timing_const;
            else begin
                if(go==2'b01)begin
                    Alarm<=0;
                    if(Up)cnt<=cnt+1;
                    else cnt<=cnt-1;
                end
                if((~Up&(~|cnt))|(Up&(&cnt)))begin
                    Alarm<=1;
                    go<=0;
                end
            end
        end
    end

endmodule
```

ms1.v 代码如下：

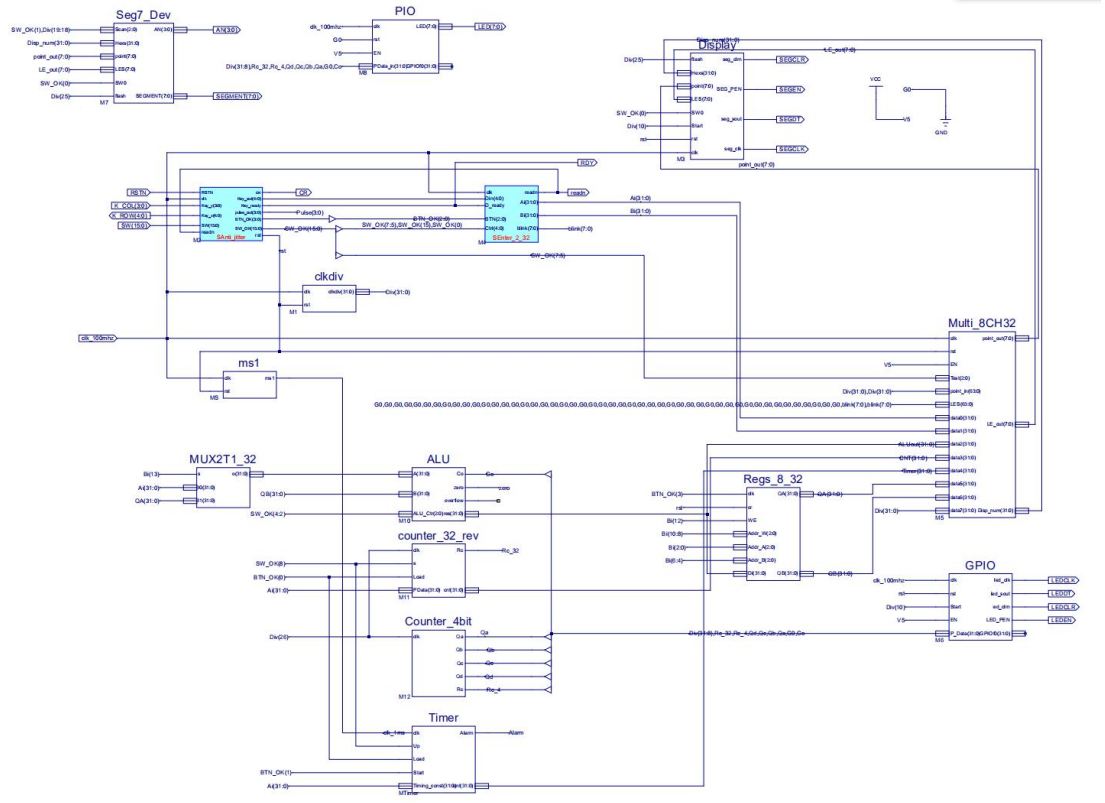
```
`timescale 1ns / 1ps

module ms1(input clk,
            input rst,
            output ms1
    );

    parameter COUNTER=16;
    reg [COUNTER-1:0]count;
    reg second_m;
    initial count<=0;

    always@(posedge clk)begin
        if(rst || (count==16'hC34F))begin
            count<=0;
            second_m<=1;
        end
        else begin
            count<=count+1;
            second_m<=0;
```

```
        end
    end

    assign ms1=second_m;

endmodule
```

确认代码无误后，检查语法并生成逻辑符号。

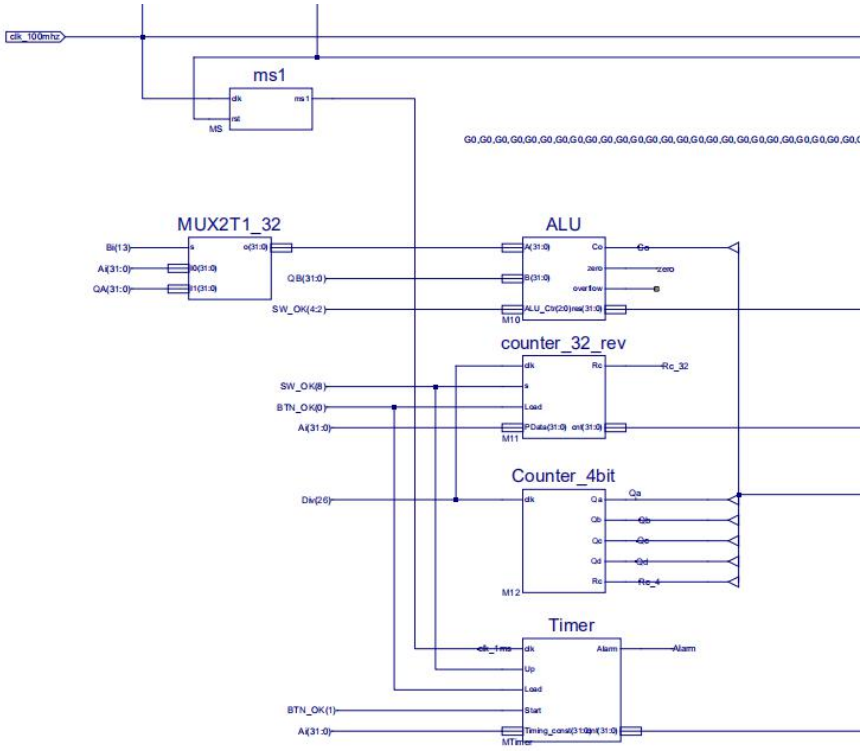
3. 集成到实验环境

完成上述元件后，将新增元件加入实验十的框架。加入后的原理图如图表 3.2.3-1。

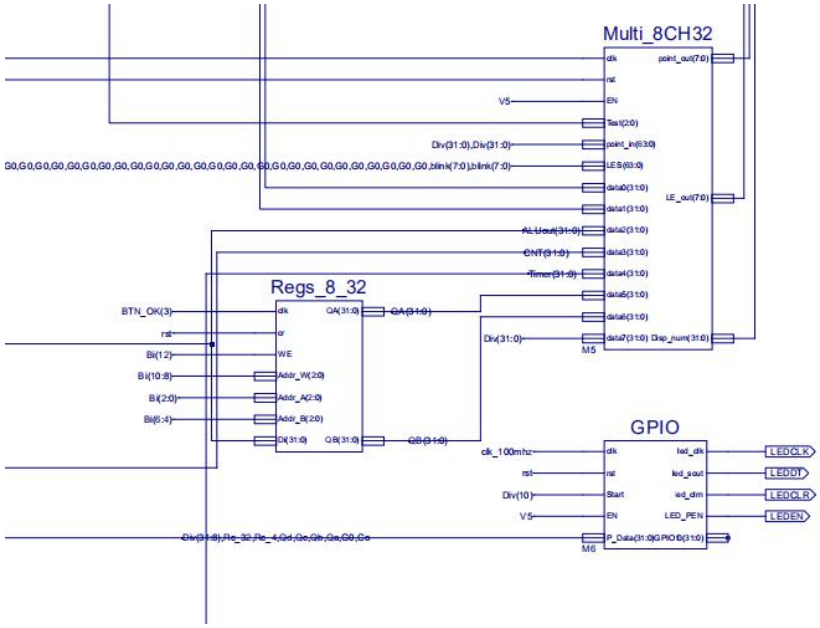


图表 3.2.3-1 RTL

其中相比原框架需要修改的部分如图表 3.2.3-2 和 3.2.3-3。



图表 3.2.3-2 修改部分 1



图表 3.2.3-3 修改部分 2

后续步骤则为检查文件，生成比特流文件，并将文件下载到实验板，在实验板物理运行，不再赘述。

四、实验结果与分析

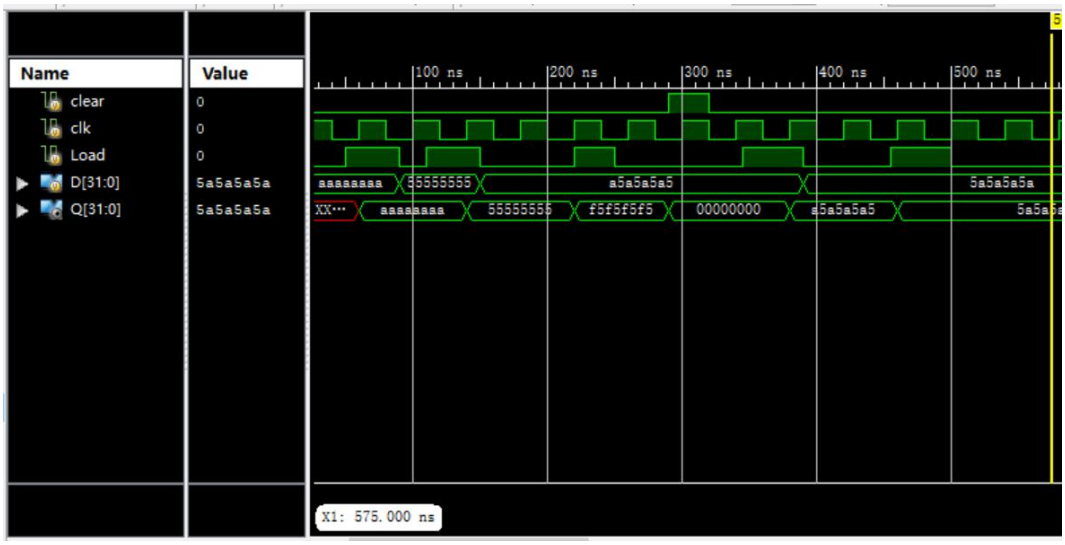
4.1 32 位寄存器仿真验证

图表 4.1-1 可见，根据代码，在 Load 信号的正脉冲内产生了 clk 时钟信号的上升沿时，输出 Q 为输入 D 的值。一旦出现 clear 信号的上升沿，输出 Q 即被清零。

其中，60ns 之前 Q 的信号不确定，这是由于 60ns 之前没有时钟上升沿出现在 Load 信号的正脉冲内，Q 的值处于一个未初始化的不确定的状态。

而 240ns 时，由于仿真代码要求此时 clk 与 Load 信号同时发生变化。clk 产生上升沿时，输入信号未稳定，导致输出信号 Q 为 F5F5F5F5 而不是 D 的 A5A5A5A5。

由仿真结果可知，设计正确。



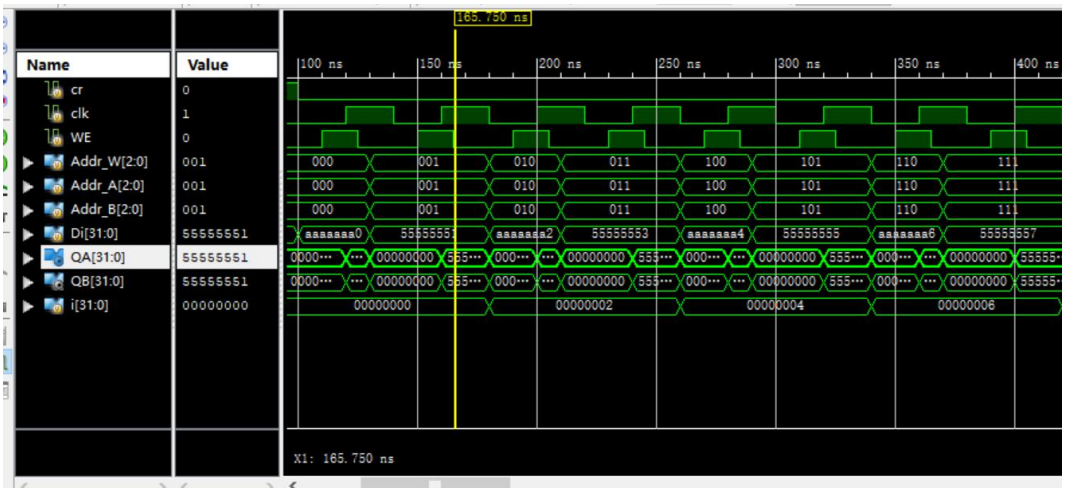
图表 4.1-1 32 位寄存器仿真模拟实验结果

4.2 8×32 位寄存器组仿真验证

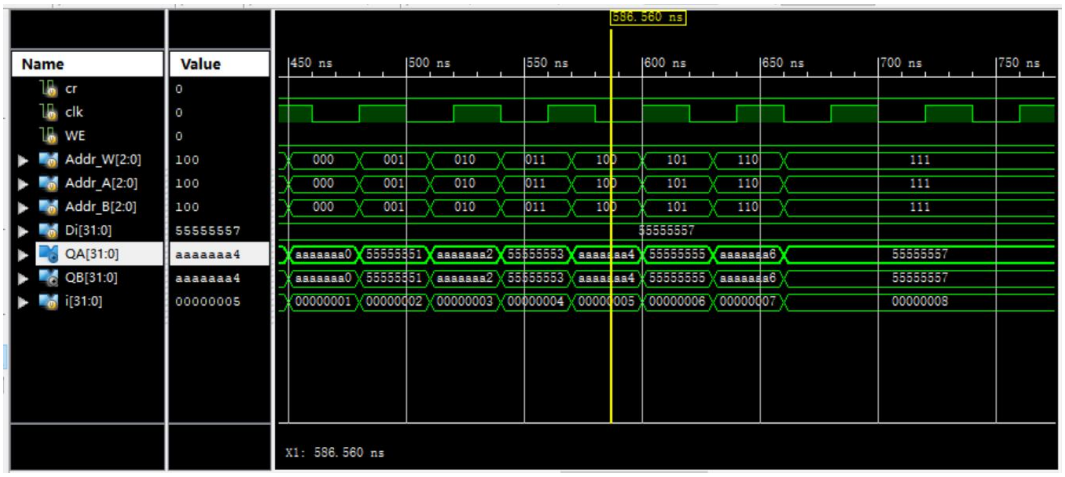
如图表 4.2-1 可见，根据代码，每当时钟上升沿出现在 WE 信号的正脉冲内，Di 的值被写入相应的寄存器内。可见写入之后，通过 QA 与 QB 读出的对应寄存器的值为 Di 的值。由此可知寄存器组可以正常写入。

如图表 4.2-2，每当 Addr_A 与 Addr_B 发生变化，QA 与 QB 立刻发生变化，显示出对应寄存器的值，且读出值与 450ns 之前的写入值一一对应。可知寄存器组可以正常实现寄存功能，且可以正常读取值。

可知设计正确。



图表 4.2-1 8×32 位寄存器组仿真模拟实验结果 1



图表 4.2-2 8×32 位寄存器组仿真模拟实验结果 2

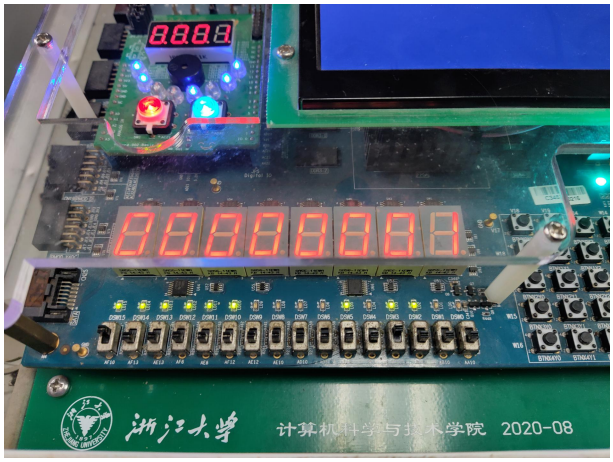
4.3 集成到实验环境并物理测试

先设置 Ai 为 0000 0001 并通过 RST 将所有寄存器置 0。测试结果如图表：

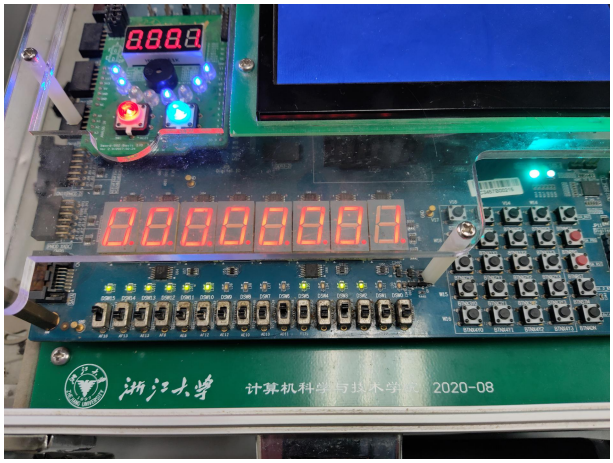
图表 4.3-1 测试结果

Bi	操作	结果
0000 1101	R1 <- Ai + R0	按下 BTN[3]后 R1=1
0000 3311	R3 <- R1 + R1	按下 BTN[3]后 R3=2
0000 0033	查看 R3	R3=2
0000 1333	R3 <- Ai + R3	按下 BTN[3]后 R3=R3+1

其中部分实验结果如下所示：



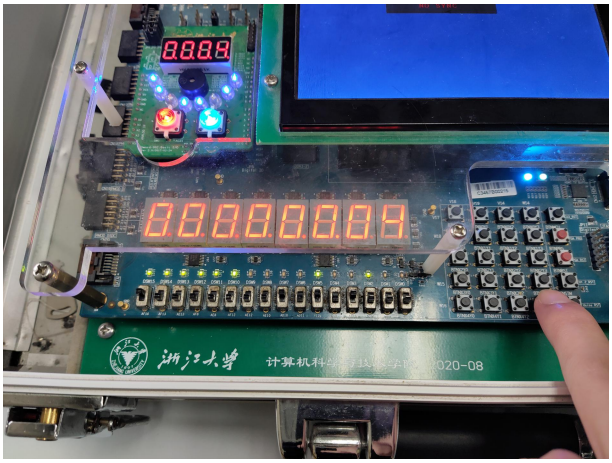
图表 4.3-2 Ai 置 1



图表 4.3-3 按下 BTN[3]前，加法结果 1 未写入寄存器

图表 4.3-4 按下 BTN[3]后，加法结果 1 写入寄存器

图表 4.3-5 Bi=0000 3311 操作结果，R3=2



图表 4.3-6 Bi=0000 1333 时，每次按下 BTN[3]，寄存器值递增

五、讨论、心得

本次实验我遇到的最大困难在于 32 位寄存器的设计，由于对 Verilog 代码了解不够深入，采用 Verilog 代码描述时遇到了麻烦。

同时，本次实验让我对寄存器有了更深刻的认识。实验最后的 ALU 运算编程已经开始出现了汇编语言的味道。