浙江大学

本科实验报告

课程名称：计算机逻辑设计基础

姓　　名：秦嘉俊

学　　院：竺可桢学院

系：所在系

专　　业：计算机科学与技术

学　　号：3210106182

指导教师：董亚波

2022 年 12 月 10 日

浙江大学实验报告

课程名称： 计算机逻辑设计基础 实验类型： 综合

实验项目名称: 寄存器和寄存器传输设计

学生姓名： 秦嘉俊 专业： 计算机科学与技术 学号： 3210106182

同组学生姓名： 钟梓航 指导老师： 董亚波

实验地点： 东 4-509 实验日期： 2022 年 11 月 23 日

一、实验目的和要求

1. 掌握寄存器传输电路的工作原理
2. 掌握寄存器传输电路的设计方法
3. 掌握 ALU 和寄存器传输电路的综合应用

二、实验内容和原理

实验设备

- 装有 Xilinx ISE 14.7 的计算机 1 台
- SWORD 开发板 1 套

内容

1. 任务：基于 ALU 的数据传输应用设计

原理

2.1 寄存器

- 一组二进制存储单元
- 一个寄存器可以用于存储一列二进制值，通常用于进行简单数据存储、移动和处理等操作
- 能存储信息并保存多个时钟周期，能用信号来控制“保存”或“加载”信息

2.1.1 采用门控时钟的寄存器

- 如果 Load 信号为 1，允许时钟信号通过，如果为 0 则阻止时钟信号通过
- 例如：对于上升沿触发的边沿触发器或负向脉冲触发的主从触发器：

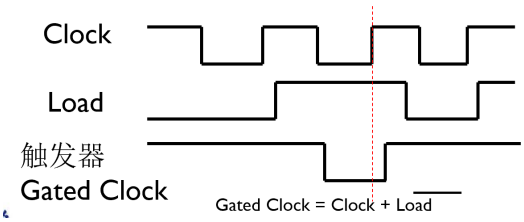

图 1: 采用门控时钟的寄存器

2.1.2 采用 Load 控制反馈的寄存器

进行有选择地加载寄存器的更可靠方法是：保证时钟的连续性；且选择性地使用加载控制来改变寄存器的内容

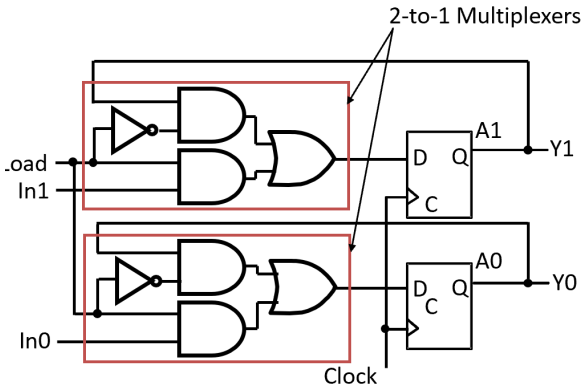

图 2: 采用 Load 控制反馈的寄存器

Verilog 代码如下

```
……
    reg [3:0] OUT;
    ……
    always @ (posedge clk) begin
            if (Load) OUT <= IN;
    end
    ……
```

2.2 寄存器传输

2.2.1 寄存器传输方式

- 寄存器传输：寄存器中数据的传输和处理。
- 三个基本单元：寄存器组、操作、操作控制。
- 基本操作: 加载、计数、移位、加法、按位操作等。

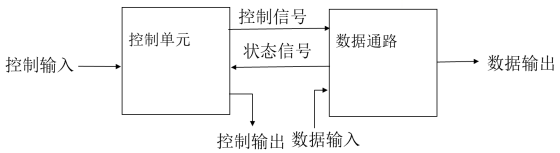

图 3: 寄存器传输示意图

2.2.2 采用寄存器传输原理的寄存器

- 功能：SW[2] 拨动一次，计一次数
- Load 控制模块——在 SW[2] 的上升沿产生 1 个时钟周期宽度的 Load 信号
- 自增/自减器可以用 4 位加减法器实现
- 功能：
  - SW[2]：寄存器加载
  - SW[0]：向上/下计数
  - SW[15]：寄存器清零

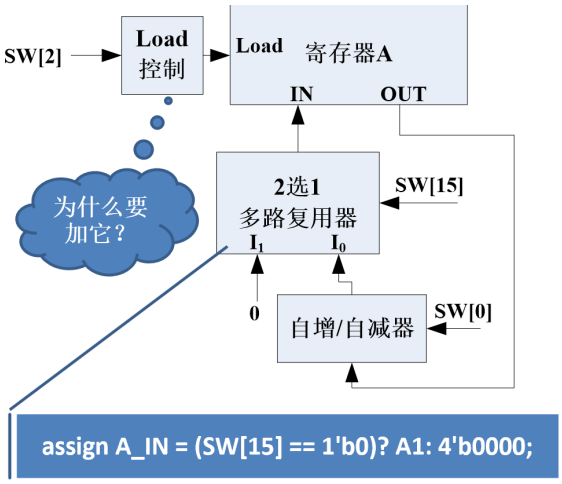

图 4: 采用寄存器传输原理的寄存器

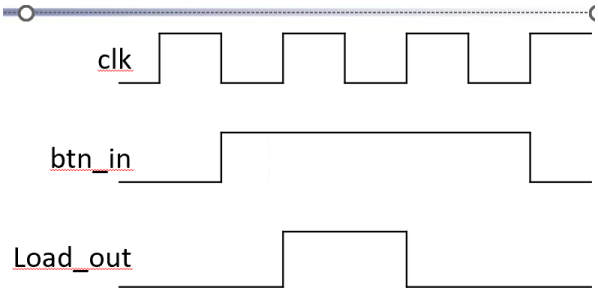

图 5: 波形图

Verilog 代码如下

```
1     module Load_Gen(
2         input wire clk,
3         input wire clk_1ms,
4         input wire btn_in,
5         output reg Load_out
6         );
7         initial Load_out = 0;
8         wire btn_out;
9         reg old_btn;
10        pbdebounce p0(clk_1ms, btn_in, btn_out);//防抖动
11        always@(posedge clk) begin
12        if ((old_btn == 1'b0) && (btn_out == 1'b1)) //btn 出现上升
              沿
13        Load_out <= 1'b1;
```

```
14          else
15          Load_out <= 1'b0;
16          end
17          always@(posedge clk) begin //保存上一个周期 btn 的状态
18          old_btn <= btn_out;
19          end
20          endmodule
```

2.3 基于多路选择器总线的寄存器传输

- 由一个多路选择器驱动的总线可以降低硬件开销。
- 这个结构不能实现多个寄存器相互之间的并行传输操作。

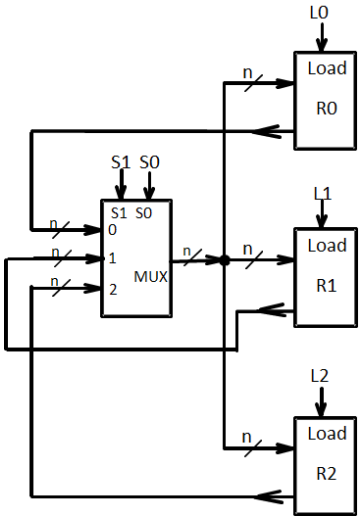

图 6: 基于多路选择器总线的寄存器传输 (1)

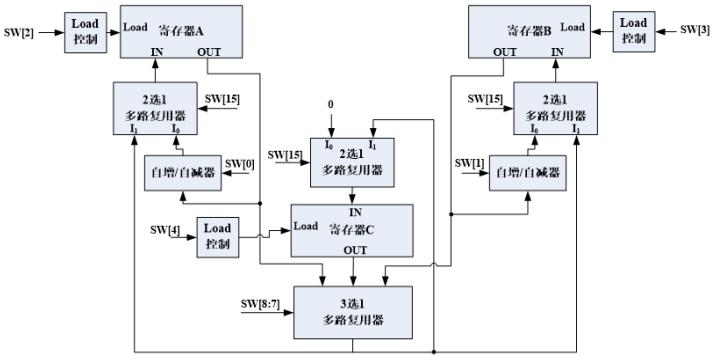

图 7: 基于多路选择器总线的寄存器传输 (2)

- SW[2]、SW[3]、SW[4]：更新 A、B、C 寄存器，更新的值由 SW[15] 和 SW[0]、SW[1] 决定。
- SW[15]=0：初始化各寄存器。
- SW[15]=1：A、B、C 寄存器之间相互传输，源寄存器由 SW[8:7] 确定。

2.4 寄存器传输应用设计

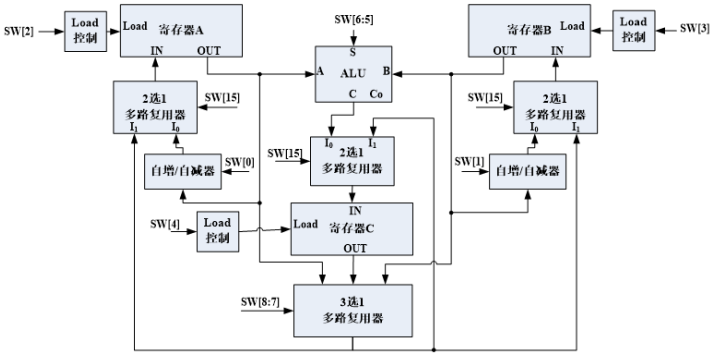

图 8: 寄存器传输应用设计

- 功能：在上一个任务基础上，可以用 A 与 B 的 ALU 计算结果初始化 C
- SW[15]=0：初始化 A、B 寄存器，ALU 运算输出控制
- SW[15]=1：A、B、C 寄存器之间相互传输

三、实验过程和数据记录

3.1 采用寄存器传输原理设计计数器

任务一，我们需要验证寄存器的设置初值功能；验证寄存器自增、自减功能；合理设计 4 位数码管上的显示内容

1. 新建工程文件，命名为 MyRegCounter，Top Level Source Type 为 HDL。
2. 新建类型为 Verilog 的源文件，命名为 MyRegister4b，用 Verilog 代码设计。输入 Verilog 代码如下:

```
module MyRegister4b(
    input wire clk,
    input [3:0] IN,
    input wire Load,
    output reg [3:0] OUT
);
    initial OUT = 0;
    always @ (posedge clk) begin
    if (Load) OUT <= IN;
    end
endmodule
```

3. 新建类型为 Verilog 的源文件，命名为 Load_Gen，用 Verilog 代码设计，参照 2.2.2 中代码。

4. 将需要调用的模块的.v 文件或者.sch 文件，以及必要的.sym 文件复制到工程文件目录下，并添加到工程文件中。

5. 新建类型为 Verilog module 的源文件，命名为 top，并右键 Set as Top Module。用 Verilog 代码设计，具体如下所示。

```
module top(
    input clk,
    input SW[15:0],
    output [3:0] AN,
    output [7:0] Segment
    );

    wire Load_A;
    wire [3:0] A, A_IN, A1;
    wire [31:0] clk_div;

    MyRegister4b RegA(.clk(clk), .IN(A_IN), .Load(
       Load_A), .OUT(A));
    Load_Gen m0(.clk(clk), .clk_1ms(clk_div[17]), .
       btn_in(SW[2]),
    .Load_out(Load_A)); //寄存器 A 的 Load 信号
    clkdiv m3(clk, 1'b0, clk_div);
```

```
16              AddSub4b m4(.A(A), .B(4'b0001), .Ctrl(SW[0]), .S(A1
                    )); //自增/自减逻辑
17              assign A_IN = (SW[15] == 1'b0)? A1: 4'b0000; //2 选
                    1 多路复用器, 复位寄存器初值
18              DispNum m8(.clk(clk), .HEXS({A, A1, A_IN, 4'b0000})
                    , .LES(4'b0),
19              .points(4'b0), .RST(1'b0), .AN(AN), .Segment(
                    Segment));
20          endmodule
```

6. 准备仿真

(a) 修改 Load_Gen 模块代码

- 去掉输入引脚：input wire clk_1ms
- 去掉按键去抖模块：pbdebounce p0(...)

得到代码如下:

```
1               module Load_Gen(
2                   input wire clk,
3                   input wire clk_1ms,
4                   input wire btn_in,
5                   output reg Load_out
6                   );
7                    initial Load_out = 0;
8                    reg old_btn;
9               //   pbdebounce p0(clk_1ms, btn_in, btn_out);
10                   always@(posedge clk) begin
11                      if ((old_btn == 1'b0) && (btn_in == 1'
                            b1))  //btn出现上升沿
12                           Load_out <= 1'b1;
13                      else
14                           Load_out <= 1'b0;
15                   end
16                   always@(posedge clk) begin              //
                        保存上一个周期btn的状态
17                      old_btn <= btn_in;
18                   end
19
```

```
endmodule
```

(b) 修改 top.v 模块代码

- 去掉 DispNum m8(...) 模块，不做数码管输出
- 模块输入端口增加 output wire [15:0] num
- 模块输出端口去掉 AN, SEGMENT
- 模块内增加 assign num={A, A1, A_IN, 4'b0000};
- 把 A 寄存器、A 寄存器自增/自减 1 的结果，寄存器 A 输入这 3 个数据作为仿真结果输出 num

得到代码如下:

```
module top(
    input clk,
    input [15:0]SW,
    output wire [15:0] num
//  output [3:0] AN,
//  output [7:0] Segment

    );

    wire Load_A;
    wire [3:0] A, A_IN, A1;
    wire [31:0] clk_div;

    assign num = {A, A1, A_IN, 4'b0000};

    MyRegister4b RegA(.clk(clk), .IN(A_IN), .Load(Load_A), .OUT(A));
    Load_Gen m0(.clk(clk), .clk_1ms(clk_div[17]), .btn_in(SW[2]),
    .Load_out(Load_A)); //寄存器 A 的 Load 信号
    clkdiv m3(clk, 1'b0, clk_div);
    AddSub4b m4(.A(A), .B(4'b0001), .Ctrl(SW[0]), .S(A1)); //自增/自减逻辑
    assign A_IN = (SW[15] == 1'b0)? A1: 4'b0000; //2 选 1 多路复用器，复位寄存器初值
//  DispNum m8(.clk(clk), .HEXS({A, A1, A_IN, 4'b0000}), .LES(4'b0),
```

```
23              //  .points(4'b0), .RST(1'b0), .AN(AN), .
                   Segment(Segment));
24              endmodule
```

(c) 仿真

仿真代码如下：

```
1               module top_sim;
2
3               // Inputs
4               reg clk;
5               reg [15:0]SW;
6
7               // Outputs
8               wire [15:0]num;
9
10              // Instantiate the Unit Under Test (UUT)
11              top uut (
12                  .clk(clk),
13                  .SW(SW),
14                  .num(num)
15              );
16          //      assign SW[15:0] = 15'b0000_0000_0000_0000;
17
18              initial begin
19                  // Initialize Inputs
20                  SW = 0;
21                  SW[15] = 1;
22                  SW[2] = 0; #50
23                  SW[2] = 1; #40
24                  SW[15] = 0;
25                  SW[0] = 0;
26                  SW[2] = 0;#60
27                  SW[2] = 0;#50
28                  SW[2] = 1;#50
29                  SW[2] = 0;#50
30                  SW[2] = 1;#50
31                  SW[2] = 0;#50
32                  // Wait 100 ns for global reset to finish
```

```
33                  #100;
34
35                  // Add stimulus here
36              end
37
38              always begin
39                  clk = 1;#10
40                  clk = 0;#10;
41
42              end
43
44          endmodule
```

得到波形图如下 SW[15]=1 的时候我们的 2-1Mux 会选择 0，用来对寄存器进行初始化。而我们每次拨动并拨回 SW[2] 就能使寄存器写入新的值 (A_IN) 在这里我们的仿真输出 num 分别表示 A(寄存器里的值)A1(A 自增后的值)A_IN(将要写入 A 的值)，根据原理图可以知道 A1=A+1, 当 SW[15]=1 时 A_IN=0, SW[15]=0 时 A_IN=A1;

3.2 基于多路选择器总线的寄存器传输

任务二，我们需要验证 A、B、C 寄存器的设置初值功能；验证 A、B 寄存器的自增、自减功能；验证 A、B、C 寄存器之间的传输功能；合理设计 4 位数码管上的显示内容

1. 新建工程文件，命名为 RegDataPathTrans，Top Level Source Type 为 HDL。
2. 将需要调用的模块的.v 文件或者.sch 文件，以及必要的.sym 文件复制到工程文件目录下，并添加到工程文件中。
3. 新建类型为 Verilog 的源文件，命名为 top，并右键 Set as Top Module。用 Verilog 代码设计，top.v 代码如下:

```
1           module top(
2               input clk,
3                input [15:0] SW,
4                output [3:0] AN,
5                output [7:0] Segment
6            );
7                wire Load_A, Load_B, Load_C;
```

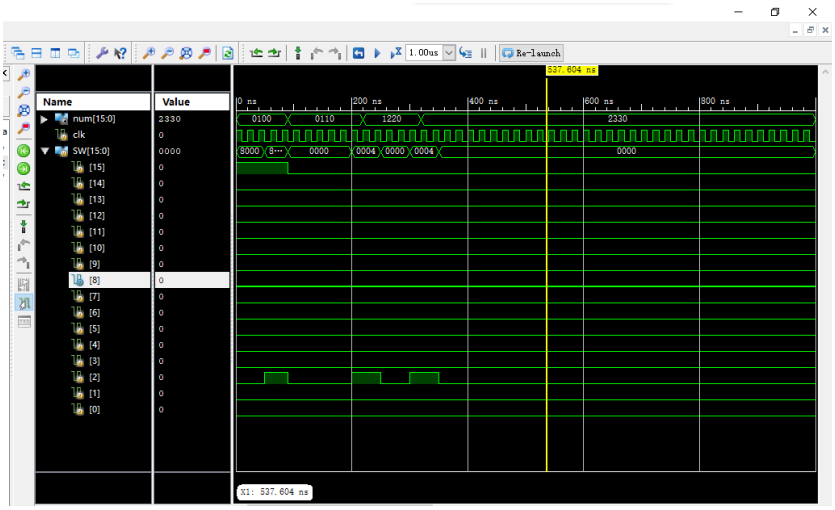

图 9: 仿真波形图

```
8       wire [3:0] A_IN, A_I0, A_OUT;
9       wire [3:0] B_IN, B_I0, B_OUT;
10      wire [3:0] C_IN, C_I0, C_OUT;
11      wire [3:0] I1;
12      wire [31:0] clk_div;
13
14      clkdiv m0(clk, 1'b0, clk_div);
15
16      MyRegister4b m1(.clk(clk), .IN(A_IN), .Load(Load_A
            ), .OUT(A_OUT));
17      Load_Gen m2(.clk(clk), .clk_1ms(clk_div[17]), .
            btn_in(SW[2]),
18      .Load_out(Load_A)); //寄存器 A 的 Load 信号
19      AddSub4b m3(.A(A_OUT), .B(4'b0001), .Ctrl(SW[0]),
            .S(A_I0)); //自增/自减逻辑
20      assign A_IN = (SW[15] == 1'b0)? A_I0: I1; //2 选 1
            多路复用器，复位寄存器初值
21
22      MyRegister4b m4(.clk(clk), .IN(B_IN), .Load(Load_B
            ), .OUT(B_OUT));
23      Load_Gen m5(.clk(clk), .clk_1ms(clk_div[17]), .
            btn_in(SW[3]),
24      .Load_out(Load_B)); //寄存器 B 的 Load 信号
25      AddSub4b m6(.A(B_OUT), .B(4'b0001), .Ctrl(SW[1]),
```

```
                .S(B_I0)); //自增/自减逻辑
26              assign B_IN = (SW[15] == 1'b0)? B_I0: I1; //2 选 1
                    多路复用器，复位寄存器初值
27
28              MyRegister4b m7(.clk(clk), .IN(C_IN), .Load(Load_C
                    ), .OUT(C_OUT));
29              Load_Gen m8(.clk(clk), .clk_1ms(clk_div[17]), .
                    btn_in(SW[4]),
30              .Load_out(Load_C)); //寄存器 C 的 Load 信号
31              assign C_IN = (SW[15] == 1'b1)? I1: 4'b0000; //2
                    选 1 多路复用器，复位寄存器初值
32
33              Mux4to1b4 m9( .I0(A_OUT), .I1(B_OUT), .I2(C_OUT),
                    .I3(4'b0000), .S(SW[8:7]), .o(I1) );
34
35              DispNum m10(.clk(clk), .HEXS({A_OUT, B_OUT, C_OUT,
                     4'b0000}), .LES(4'b0),
36              .points(4'b0), .RST(1'b0), .AN(AN), .Segment(
                    Segment));
37          endmodule
```

3.3 基于 ALU 的数据传输应用设计

任务三，我们需要验证 A、B、C 寄存器的设置初值功能；验证 A、B 寄存器的自增、自减功能；验证 ALU 运算功能；验证寄存器传输功能；合理设计 4 位数码管上的显示内容；

1. 新建工程文件，命名为 MyALUTrans，Top Level Source Type 为 HDL。
2. 将需要调用的模块的.v 文件或者.sch 文件，以及必要的.sym 文件复制到工程文件目录下，并添加到工程文件中。
3. 新建类型为 verilog 的源文件，命名为 top，并右键 Set as Top Module。用 Verilog 代码设计。top.v 代码如下：

```
1           module top(
2               input clk,
3                input [15:0] SW,
4                output [3:0] AN,
```

```
5               output [7:0] Segment
6           );
7               wire Load_A, Load_B, Load_C, carry;
8               wire [3:0] A_IN, A_I0, A_OUT;
9               wire [3:0] B_IN, B_I0, B_OUT;
10              wire [3:0] C_IN, C_I0, C_OUT;
11              wire [3:0] I1, I0;
12              wire [31:0] clk_div;
13
14              clkdiv m0(clk, 1'b0, clk_div);
15
16              MyRegister4b m1(.clk(clk), .IN(A_IN), .Load(Load_A
                    ), .OUT(A_OUT));
17              Load_Gen m2(.clk(clk), .clk_1ms(clk_div[17]), .
                    btn_in(SW[2]),
18              .Load_out(Load_A)); //寄存器 A 的 Load 信号
19              AddSub4b m3(.A(A_OUT), .B(4'b0001), .Ctrl(SW[0]),
                    .S(A_I0)); //自增/自减逻辑
20              assign A_IN = (SW[15] == 1'b0)? A_I0: I1; //2 选 1
                     多路复用器，复位寄存器初值
21
22              MyRegister4b m4(.clk(clk), .IN(B_IN), .Load(Load_B
                    ), .OUT(B_OUT));
23              Load_Gen m5(.clk(clk), .clk_1ms(clk_div[17]), .
                    btn_in(SW[3]),
24              .Load_out(Load_B)); //寄存器 B 的 Load 信号
25              AddSub4b m6(.A(B_OUT), .B(4'b0001), .Ctrl(SW[1]),
                    .S(B_I0)); //自增/自减逻辑
26              assign B_IN = (SW[15] == 1'b0)? B_I0: I1; //2 选 1
                     多路复用器，复位寄存器初值
27
28              MyRegister4b m7(.clk(clk), .IN(C_IN), .Load(Load_C
                    ), .OUT(C_OUT));
29              Load_Gen m8(.clk(clk), .clk_1ms(clk_div[17]), .
                    btn_in(SW[4]),
30              .Load_out(Load_C)); //寄存器 C 的 Load 信号
31              assign C_IN = (SW[15] == 1'b1)? I1: I0; //2 选 1
                    多路复用器，复位寄存器初值
```

```
32
33              myALU m9( .S(SW[6:5]), .A(A_OUT), .B(B_OUT), .C(I0
                    ), .Co(carry) );
34              Mux4to1b4 m10( .I0(A_OUT), .I1(B_OUT), .I2(C_OUT),
                     .I3(4'b0000), .S(SW[8:7]), .o(I1) );
35
36              DispNum m11(.clk(clk), .HEXS({A_OUT, B_OUT, C_OUT,
                     3'b000, carry}), .LES(4'b0),
37              .points(4'b0), .RST(1'b0), .AN(AN), .Segment(
                    Segment));
38          endmodule
```

4. 准备仿真

(a) 修改 `Load_Gen` 模块代码，同上，这里不再重复

(b) 修改 `top.v` 模块代码

- 不输出 `AN, SEGMENT`, 增加 `assign num={A, B, C, Bus}`, 并作为 Top 模块的输出引脚进行观察
- Bus 是 3 选 1 多路复用器的输出结果，即为总线数据

得到代码如下:

```
1               module top(
2                   input clk,
3                    input [15:0] SW,
4                    output wire [15:0] num
5                // output [3:0] AN,
6                // output [7:0] Segment
7                );
8                    wire Load_A, Load_B, Load_C, carry;
9                    wire [3:0] A_IN, A_I0, A_OUT;
10                   wire [3:0] B_IN, B_I0, B_OUT;
11                   wire [3:0] C_IN, C_I0, C_OUT;
12                   wire [3:0] I1, I0;
13                   wire [31:0] clk_div;
14
15                   assign num={A_OUT, B_OUT, C_OUT, I1};
16                   clkdiv m0(clk, 1'b0, clk_div);
17
```

```
18              MyRegister4b m1(.clk(clk), .IN(A_IN), .
                    Load(Load_A), .OUT(A_OUT));
19              Load_Gen m2(.clk(clk), .clk_1ms(clk_div[17
                    ]), .btn_in(SW[2]),
20              .Load_out(Load_A)); //寄存器 A 的 Load 信
                    号
21              AddSub4b m3(.A(A_OUT), .B(4'b0001), .Ctrl(
                    SW[0]), .S(A_I0)); //自增/自减逻辑
22              assign A_IN = (SW[15] == 1'b0)? A_I0: I1;
                    //2 选 1 多路复用器，复位寄存器初值
23
24              MyRegister4b m4(.clk(clk), .IN(B_IN), .
                    Load(Load_B), .OUT(B_OUT));
25              Load_Gen m5(.clk(clk), .clk_1ms(clk_div[17
                    ]), .btn_in(SW[3]),
26              .Load_out(Load_B)); //寄存器 B 的 Load 信
                    号
27              AddSub4b m6(.A(B_OUT), .B(4'b0001), .Ctrl(
                    SW[1]), .S(B_I0)); //自增/自减逻辑
28              assign B_IN = (SW[15] == 1'b0)? B_I0: I1;
                    //2 选 1 多路复用器，复位寄存器初值
29
30              MyRegister4b m7(.clk(clk), .IN(C_IN), .
                    Load(Load_C), .OUT(C_OUT));
31              Load_Gen m8(.clk(clk), .clk_1ms(clk_div[17
                    ]), .btn_in(SW[4]),
32              .Load_out(Load_C)); //寄存器 C 的 Load 信
                    号
33              assign C_IN = (SW[15] == 1'b1)? I1: I0; //
                    2 选 1 多路复用器，复位寄存器初值
34
35              myALU m9( .S(SW[6:5]), .A(A_OUT), .B(B_OUT
                    ), .C(I0), .Co(carry) );
36              Mux4to1b4 m10( .I0(A_OUT), .I1(B_OUT), .I2
                    (C_OUT), .I3(4'b0000), .S(SW[8:7]), .o(
                    I1) );
37
38          // DispNum m11(.clk(clk), .HEXS({A_OUT, B_OUT,
```

```
                    C_OUT, 3'b000, carry}), .LES(4'b0),
39              // .points(4'b0), .RST(1'b0), .AN(AN), .
                   Segment(Segment));
40              endmodule
```

5. 仿真

   仿真代码如下

```
1           module sim;
2
3           // Inputs
4           reg clk;
5           reg [15:0] SW;
6
7           // Outputs
8           wire [15:0] num;
9
10          // Instantiate the Unit Under Test (UUT)
11          top uut (
12              .clk(clk),
13              .SW(SW),
14              .num(num)
15          );
16
17          initial begin
18              // Initialize Inputs
19
20              SW = 0;
21              SW[15] = 0;
22
23              SW[0] = 0;
24              SW[2] = 1;#50
25              SW[2] = 0;#50
26            SW[2] = 1;#50
27              SW[2] = 0;#50
28            SW[2] = 1;#50
29              SW[2] = 0;#50
30              SW[2] = 1;#50
31              SW[2] = 0;#50
```

```
32
33              SW[1] = 1;
34              SW[3] = 1;#50
35              SW[3] = 0;#50
36              SW[3] = 1;#50
37              SW[3] = 0;#50;
38
39              SW[6:5] = 2'b01;
40              SW[4] = 1;#50
41              SW[4] = 0;#50;
42
43              SW[15] = 1;
44              SW[8:7] = 2'b10;
45              SW[2] = 1; #50
46              SW[2] = 0; #50;
47
48              SW[8:7] = 2'b01;
49              SW[4] = 1;#50
50              SW[4] = 0;#50;
51          end
52
53
54          always begin
55              clk = 1;#10
56              clk = 0;#10;
57
58          end
59      endmodule
```

得到波形图如下

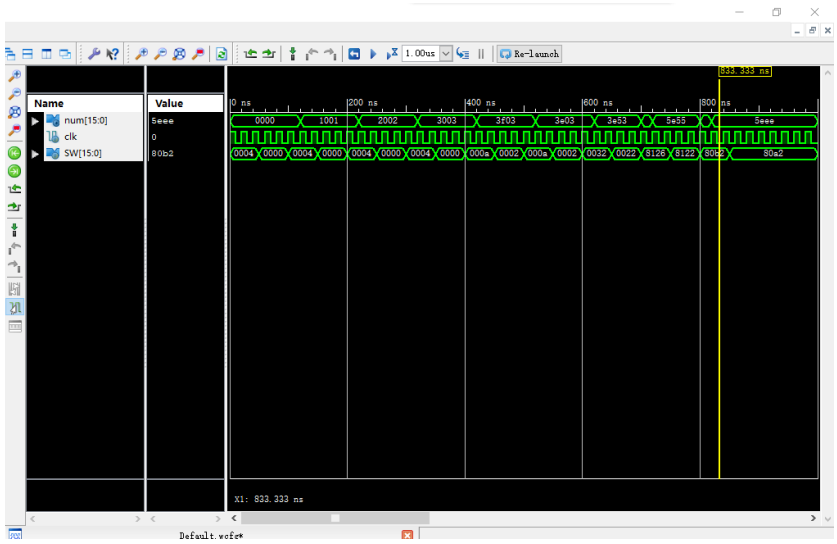

图 10: 仿真波形图

`SW[15]=0` 时可以通过自增自减 ALU 给寄存器赋初值，`SW[15]=1` 时可以利用 bus 实现数据传输。最开始我们将 `SW[15]=0`，`SW[0]=0` 表示对寄存器 A 选择自增的功能和第一部分中一样拨动 `SW[2]`，这样可以使寄存器 A 加到 3，类似地我们将 `SW[1]=1`（自减功能），随后拨动 `SW[3]` 即可使寄存器 B 减到 e. 最后我们初始化寄存器 C 的值，C 的输入是一个 ALU，对 A B 进行算术运算，`SW[6:5]` 表示我们要进行的运算（01 代表减法），这样我们拨动 `SW[4]` 即可将运算后的值赋给寄存器 C，完成初始化。

`SW[15]=1` 我们开始利用 bus 传输数据，其原理是利用 3-1 Mux 选择源寄存器，随后通过拨动对应寄存器的`Load`开关将 bus 上的数据传输进寄存器。因此最开始我们的`SW[8:7]=10`(3-1Mux 的选择信号，10 表示选择寄存器 C) 随后拨动`SW[2]`就将 C 的值存入寄存器 A. 类似将`SW[87]=01`(表示源寄存器 B) 拨动`SW[4]`即可将其存入寄存器 C

四、实验结果分析

相关结果都已经在前文写出。实验结果基本符合要求：仿真激励波形与真值表都相对应；下仿真结果和 Verilog 代码分析在前文已经给出。

值得一提的是，我们发现当我们拨动开关时，如果恰好达到 clk 的上升沿，在下一个时钟上升沿的时候才能将值写入寄存器。我们认为 `Load_Gen` 产生信号需要时间，而我们 clk 周期本身较短，因此可能会导致寄存器能够读取数据时时钟已经到了下降沿，因此需要下一个时钟上升沿才能写入数据。（个人想法）

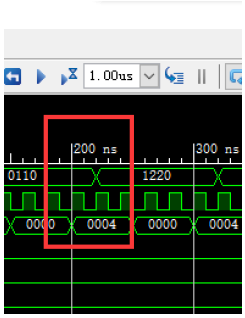

图 11: 波形图（节选）

五、讨论与心得

本次实验全部使用了 Verilog 代码实现，不用再画电路图，也不涉及仿真，这样最好是能在实验板旁开始做实验，有问题能及时找出。可惜的是实验课因为疫情停了，没能及时验收，也不知道什么时候能回归（好在我提前蹭了另一个是一颗已经做好并上板，喜）

反转了，提前写好报告做完实验，但是上板没有拍照片！悲，谁能想到线下课直接取消了！以后都要仿真，这次也是好好熟悉了一下流程，最开始听老师讲的时候有点畏惧，一直不愿开始，直到周末要结束时，慢慢告诉自己开始尝试，发现其实也不过如此！

再接再厉，继续努力！