Lab2 时序逻辑电路实验

一、实验目的

1. 掌握锁存器、触发器和寄存器的设计方法和应用。
2. 掌握寄存器堆、计数器、移位寄存器的设计方法和应用。
3. 掌握数字时钟的设计方法。

二、实验环境

软件：Vivado 2020.2

硬件：Nexys A7-100T 开发板

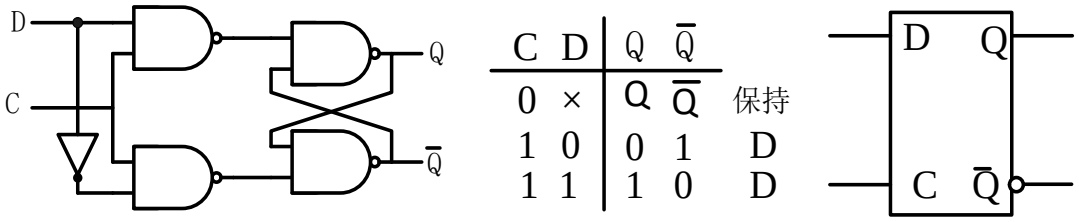
三、实验原理

1、双稳态元件

锁存器和触发器都是双稳态元件，由独立的逻辑门电路和反馈电路构成的。锁存器采用电平控制方式，在其控制信号的有效电平期间，外部输入信号的变化一直能触发其状态发生改变；而触发器状态的改变则采用时钟边沿触发控制方式。使用锁存器和触发器可以构建的时序逻辑电路模块，如数据暂存器（寄存器）、计数器、移位寄存器等。寄存器是用来暂存信息的逻辑部件，根据功能和实现方式的不同，有各种不同类型的寄存器。最简单的寄存器直接由若干个触发器组成。在 CPU 中有一个专门的寄存器堆（register file），也称为通用寄存器组（General Purpose Register set，GPRs），用于暂存指令执行过程中用到的中间数据。它由许多寄存器组成，每个寄存器有一个编号，CPU 可以对指定编号的寄存器进行读写。

1）触发器

带控制端 C 的 D 锁存器电路原理图、真值表和逻辑符号如图 2.1 所示。当控制信号 C 为低电平时，输出保持不变，当控制信号 C 为高电平时，Q 输出 D 的值，即 Q 随着 D 值的改变而改变。



(a)电路原理图　　(b)功能表　　(c)逻辑符号

图 2.1 D 锁存器（a）原理图（b）功能表（c）逻辑符号

Verilog 语言设计时序电路一般采用行为建模方式，使用 always 语句来实现，常用的语法为：

always @ (事件信号列表) 过程语句

其含义是，当事件信号列表中的任意一个信号发生变化时，过程语句中的信号将按照所描述的行为进行更新。事件信号列表中可以只有单个信号，也可以有多个信号。有多个信号时，信号之间需用关键字 or 或者逗号来连接。事件信号前添加关键字 posedge 或 negedge 来修饰，则表示只检测信号上升沿或只检测信号下降沿，一般用于触发器电路中。

过程赋值语句是在过程语句中进行赋值的语句。Verilog 中有两种过程赋值语句，一种是**阻塞赋值**语句，赋值号是“=”，其语法为：寄存器 = 表达式;

另一种是**非阻塞赋值语句**，赋值号是“<=”，其语法为：寄存器 <= 表达式;

对于这两种过程赋值语句，赋值号右边可以是任意有效的表达式，数据类型可以是 wire 类型，也可以是 reg 类型；赋值号左边的寄存器必须是 reg 类型（标量或向量皆可）。Verilog 允许在一个 always 语句中对同一个 reg 类型变量进行多次赋值，此时所描述的电路功能以最后一次赋值为准，但不能在多个 always 语句中对同一个 reg 类型变量进行赋值，因为这样的代码功能上相当于多个电路逻辑同时驱动同一个组合信号或同时更新同一个触发器，这会导致综合器无法综合或者综合出错误的电路。

在仿真的概念中，上述两种过程赋值语句的行为都是将赋值号右边的表达式所描述的电路的输出更新到赋值号左边的 reg 类型变量中，但是，两者的更新时机有所不同。

阻塞赋值是**立即更新**，因而后续语句和其他代码对该 reg 类型变量的引用都按新值来计算。也就是说，一组阻塞赋值语句将以其在代码中出现的顺序来计算，这与高级编程语言中的语句类似。

非阻塞赋值则是**滞后更新**，新值将会在其他所有操作都结束后才会进行更新，因此后续语句和其他代码对该 reg 类型变量的引用都按旧值来计算。也就是说，一组非阻塞赋值语句是并行计算的。

由于阻塞赋值语句具有立即更新的特点，结合隐式事件列表的 always 语句的使用，符合组合逻辑电路持续驱动的特点，因此可用于描述组合逻辑电路。非阻塞赋值语句具有滞后更新的特点，在以边沿事件时

钟变量作为事件信号列表的 always 语句中使用非阻塞赋值语句，相当于只有在时钟边沿到来时才进行更新，符合时序逻辑电路存储数据的特点，因此可用于描述时序逻辑电路。

采用行为建模方式设计带控制端的 D 锁存器的模块代码如下：

```
module Dlatch(
   output reg Q,
   input D, C
   );
 always @ (D or C) begin
      if (C==1) Q <= D;            //当 C 为高电平时，D 赋值给 Q
   end
endmodule
```

采用行为建模方式实现带同步复位 rst 和使能端 en 的 D 触发器，模块的输入端口有时钟信号 clk、复位信号 rst、使能信号 en 和数据输入信号 d，都默认为 wire 类型，输出端口 q 为 reg 类型。

```
module Dff (
   output reg q,
 input clk, rst, en, d
);
   always @(posedge clk) begin       //时钟信号上升沿触发
      if (rst) q <= 1'b0;
      else if (en) q <= d;
   end
endmodule
```

在上述代码中，寄存器变量 q 只会在时钟上升沿到来时才进行更新。当时钟上升沿到来时，若复位信号 rst 有效，则将 q 的值复位为 0；否则，若使能信号 en 有效，则将输入信号 d 更新到 q；否则，q 的值将保持不变。由于复位操作只在时钟上升沿到来时才会进行，因此这种复位方式称为同步复位。

如果要实现异步复位方式的 D 触发器，其模块代码如下：

```
module AsyncDff (
     output reg q,
     input clk, rst, en, d
);
   always @(posedge clk or posedge rst) begin
      if (rst) q <= 1'b0;
      else if (en) q <= d;
   end
endmodule
```

和同步复位方式相比，上述代码在事件信号列表中增加了 posedge rst，表示当复位信号 rst 从 0 跳变到 1 时，就可以马上进行复位，复位操作不需要和时钟上升沿同步进行。

上述两种复位方式各有优缺点，其中同步复位方式由于和时钟信号同步进行，因此不会造成亚稳态，但需要复位信号的维持时间大于一个时钟周期才能被触发器正确采集到；而异步复位方式则相反，只要异步复位信号有效，就可以被触发器正确采集到，但若异步复位信号的撤销时机和时钟上升沿的到来非常接近，则会导致触发器进入亚稳态，造成数字电路无法正确工作。

实验验证：设计仿真测试文件并执行，验证同步和异步清零功能，观测仿真波形图和输出信息，综合实现，并上实验板可以验证。

提示：当把非时钟引脚作为时钟输入信号时，需要在约束文件中添加一行“set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]； ”，用来解除系统设置的缺省时钟引脚约束。

2）寄存器

寄存器是 CPU 用来暂存信息的逻辑部件，根据功能和实现方式的不同，有各种不同类型的寄存器。寄存器可以直接由若干个 D 触发器组成。例如，由 *n* 个 D 触发器可构成一个 *n* 位寄存器。

8 位寄存器实验

利用 D 触发器，实现 8 位数据读写寄存器。当时钟信号 CLK 上升沿时，如果写使能 WE 为高电平时，则输入端 D 的值赋值输出端 Q；当置位端 PRE_L 为低电平时，输出端 Q 为 1；当清零端 CLR_L 为低电平时输出端 Q 为 0；QN 的值和 Q 值相反。

采用行为建模方式实现该功能的模块代码如下：

```
//8 位寄存器
module reg8 (
   output reg [7:0] Q,
   output    [7:0] QN,
   input [7:0] D,
   input CLK, PRE_L, CLR_L, WE
);
   always @ (posedge CLK or negedge CLR_L or negedge PRE_L)
      if (CLR_L==0) Q <= 0;
      else if (PRE_L==0) Q<=255;
            else if (WE==1) Q <= D;
      assign QN=~Q;
endmodule
```

实验验证：设计仿真测试文件并执行，验证功能，观测仿真波形图和输出信息，综合实现，并上实验板可以验证。

2、计数器

计数器是一种对外部激励信号进行总数统计的时序逻辑元件，一般从 0 开始计数，在达到最大计数值时输出一次计数完成（满值）信号，并重新开始计数，最大计数值为计数器的模。计数器在数字系统中主要用于脉冲个数的计数，以实现测量、计数和控制的功能。分频器是将高频脉冲降频为中低脉冲。

一种常用带同步清零和同步置数功能的 4 位二进制加法计数器的功能表如表 2-1 所示。

表 2-1 4 位二进制加法计数器功能表

CLK	CLR	LD	ENP ENT	输出状态 Q3 Q2 Q1 Q0
↑	1	0	x x	0 0 0 0
↑	0	1	x x	D3 D2 D1 D0
↑	0	0	0 x	保持
↑	0	0	x 0	保持，RCO=0
↑	0	0	1 1	加法计数

使用 4 个 D 触发器，可以实现该 4 位二进制加法计数器，其电路原理图如图 2.2 所示。

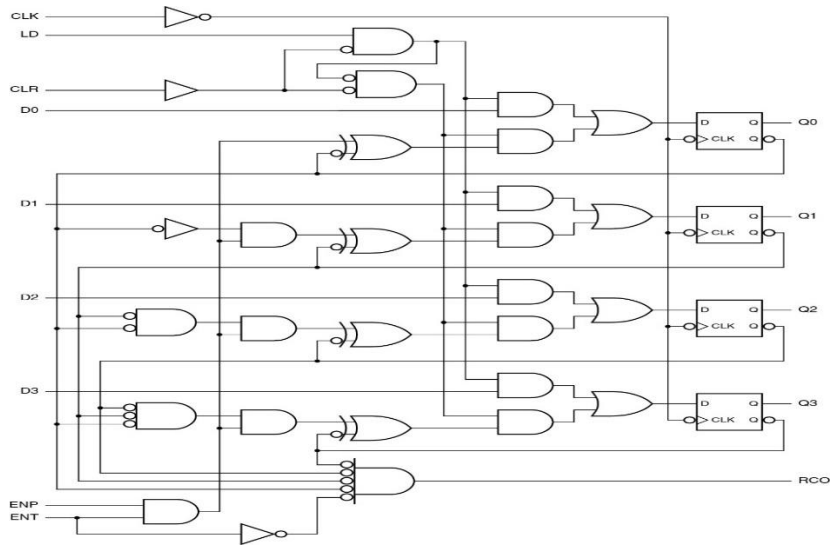


图 2.2 4 位二进制加法计数器原理图

当时钟信号 CLK 上升沿时，如果清零端 CLR 为高电平，则输出为 0000；如果置数端 LD 为高电平，则把输入 D 传送到输出端；如果清零端和置数端都为低电平，使能端 ENP 和 ENT 都为高电平，输出端 Q 为加法计数；当输出 Q 为 1111 且 ENT 不等于 0 时，RCO 输出为 1，否则 RCO 输出为 0。由于该计数器

的下一状态逻辑比较简单和清晰，通过把下一状态逻辑和边沿触发的触发器行为特性放在同一个 always 程序段中，使用另一个 always 程序段来定义 RCO 组合逻辑特性。采用行为建模方式实现 4 位计数器的模块代码如下：

```
module cntr4u(
  output reg [3:0] Q,
  output reg RCO,
  input CLK, CLR, LD, ENP, ENT,
  input [3:0] D
  );
  always @ (posedge CLK)                  //  计数器触发行为
    if (CLR)       Q <= 4'd0;
    else if (LD)    Q <= D;
          else if (ENT && ENP)    Q <= Q + 1;
                else    Q <= Q;
  always @ (Q or ENT)                        // RCO 组合输出逻辑
    if (ENT && (Q == 4'd15)) RCO = 1;
    else      RCO = 0;
endmodule
```

修改上述计数器程序可以实现 4 位十进制计数器、可逆计数器或余 3 码计数器等。在 always 敏感信号列表中加入“posedge CLR”就可以实现异步清零的功能。

上述计数器的仿真测试文件如下：

```
‘timescale 1ns/1ps
module cntr4u_tst(      );
  reg CLK, CLR, LD, ENP, ENT;
  reg [3:0] D;
  wire [3:0] cntr4uQ;
  wire cntr4uRCO;
  always begin                  // 10 ns  时钟周期
    #5.5 CLK = 0;               // 5.5 ns  高电平
    #4.0 CLK = 1;               // 4.0 ns  低电平
    #0.5 ;                      // Plus 0.5 ns  转换时间
  end
  cntr4u      U1 (    .Q(cntr4uQ), .RCO(cntr4uRCO),.CLK(CLK),
                .CLR(CLR), .LD(LD), .ENP(ENP), .ENT(ENT), .D(D) );
  initial begin
    CLR = 0; LD = 0; ENP = 0; ENT =0; D = 0;    //  输入信号初始化
    #10 ;                                          //  等待复位结束
    CLR = 1; D = 4'b1111; #10                     //  计数器清零
    #10 ;
```

```
    CLR = 0; LD = 1; #10                    // 置数 1111
    LD = 0; ENP = 1; #10                    // 保持 (ENT=0)
    ENT = 1;   #320                          // 加法计数 32 个时钟周期
    ENT = 0; #20                             // 保持，RCO=0
    $stop(1);
  end
endmodule
```

实验验证：执行逻辑仿真测试文件验证功能，观测仿真波形图和输出信息，综合实现，并上实验板可以验证。

3、移位寄存器

移位寄存器能够实现暂存信息的左移或右移等功能。数字系统设计中，经常需要使用移位操作来实现特定的功能，例如，浮点数加减运算电路中，需要通过移位操作实现指数对齐的功能；乘法运算电路中，需要具有将部分积右移的功能；除法运算电路中，需要具有将中间余数左移的功能。

移位寄存器通常通过把低位 D 触发器输出接入到高位 D 触发器输入端来实现。通用移位寄存器除了具有数据左移、数据右移功能外，还具有数据保持和数据载入功能。4 位通用移位寄存器的功能如表 2.2 所示。

表 2.2 4 位通用移位寄存器功能表

功能选择	输入			输出状态			
	CLR	S1	S0	QA^*	QB^*	QC^*	QD^*
清零	1	x	x	0	0	0	0
保持	0	0	0	QA	QB	QC	QD
右移	0	0	1	RIN	QA	QB	QC
左移	0	1	0	QB	QC	QD	LIN
载入	0	0	1	A	B	C	D

使用 4 个 D 触发器，可以实现该 4 位通用移位寄存器，其电路原理图和逻辑符号如图 2.3 所示。

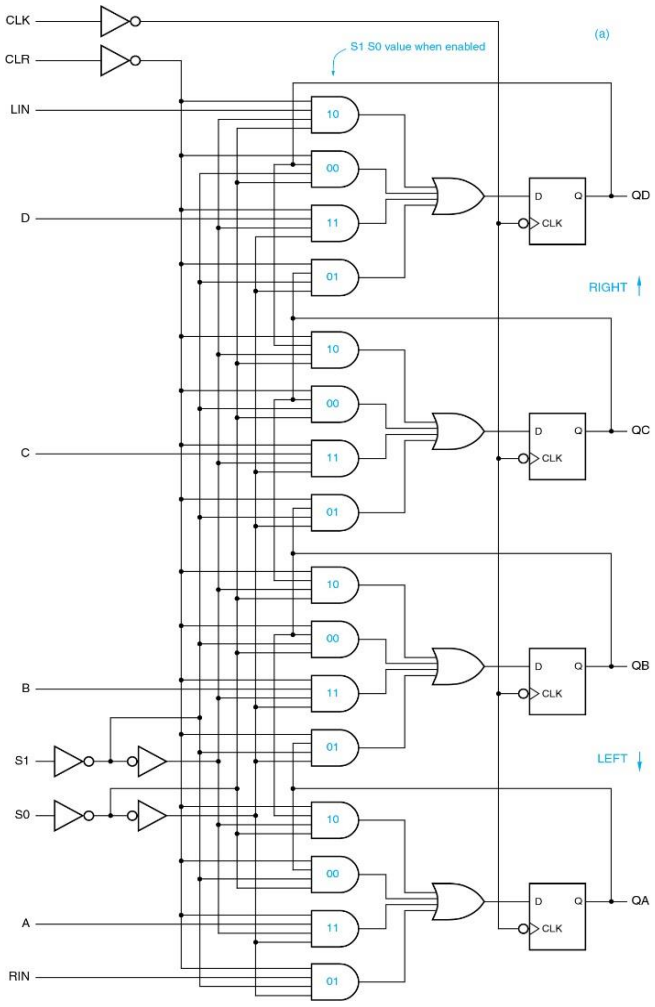


图 2.3 4 位通用移位寄存器原理图和逻辑符号

移位寄存器的功能设置如下：当时钟上升沿到来时，如果 CLR=1，则输出为 0000,；如果 S1S0=00，则输出保持不变；如果 S1S0=01，则实现右移，输出{QA,QB,QC,QD}={RIN,QA,QB,QC}；如果 S1S0=10，则实现左移，输出{QA,QB,QC,QD}={QB,QC,QD,LIN}；如果 S1S0=11，则实现载入，输出{QA,QB,QC,QD}={A,B,C,D}。

使用 Verilog 实现该移位寄存器的代码如下：

```
module shrg4u(
   output reg QA, QB, QC, QD,
   input CLK, CLR, S0, S1, RIN, LIN, A, B, C, D
 );
   always @ (posedge CLK)
      if (CLR == 1'b1) {QA,QB,QC,QD} <= 4'b0;
      else case ({S1,S0})
         2'b00: ;                                          // Hold
         2'b01: {QA,QB,QC,QD} <= {RIN,QA,QB,QC};     // Shift right
         2'b10: {QA,QB,QC,QD} <= {QB,QC,QD,LIN};     // Shift left
```

```
            2'b11: {QA,QB,QC,QD} <= {A,B,C,D};              // Load
            default: {QA,QB,QC,QD} <= 4'bx;                 // should not occur
        endcase
    endmodule
```

移位寄存器的仿真测试文件如下：

```
    ‘timescale 1ns/1ps
    module shrg4u_tst( );
    reg Tclk, CLR, S0, S1, RIN, LIN;
    reg [3:0] I;        // A-D = I[3:0]
    wire [3:0] Q;      // QA-QD = Q[3:0]
    shrg4u UUT ( .QA(Q[3]), .QB(Q[2]), .QC(Q[1]), .QD(Q[0]),
                        .CLK(Tclk), .CLR(CLR), .RIN(RIN), .LIN(LIN), .S0(S0), .S1(S1),
                         .A(I[3]), .B(I[2]), .C(I[1]), .D(I[0]));
    always begin
      #0.5 ; Tclk = 1'b1; #5 ;     // 定义时钟周期 10ns，上升沿发生在 10.5ns, 20.5ns, ....
      Tclk = 1'b0; #4.5 ;
    end

    initial begin : TB
    integer ii, j;
      #10 ;                                          // 等待实验板系统复位时间
      RIN = 1'b0; LIN = 1'b0;                        // 设置左、右移入位为 0
      for (ii=0; ii<=15; ii=ii+1) begin          // 验证载入、保持和清零功能
        CLR = 1'b0; {S1,S0} = 2'b11; I[3:0] = ii; #10 ;        // 载入数据，并等待时钟信号有效后验证
        if (Q != I[3:0]) $display("S1S0 = 11, ABCD = %4b, QA-QD = %4b, load failed", I, Q);
        {S1,S0} = 2'b00; #10 ;                                 // 保持数据,并等待时钟信号有效后验证
        if (Q != I[3:0]) $display("S1S0 = 00, ABCD = %4b, now QA-QD = %4b, hold failed", I, Q);
        CLR = 1'b1; #10 ;                                      // 清零信号有效，并等待时钟信号有效后验证
        if (Q != 4'b0) $display("CLR = 1, QA-QD = %4b, clear failed", Q);
      end
      $display("Clear, load, and hold test completed");
      CLR = 1'b0; LIN=1'b1;                                    //清零信号无效，进入移位状态
      $display("Starting shift-right test for all states");
      for (ii=0; ii<=31; ii=ii+1) begin                  // 右移测试
        {S1,S0} = 2'b11; { RIN, I[3:0]} = ii[4:0]; #10 ; // 载入初始数据，等待时钟信号
        {S1,S0} = 2'b01;    #10 ;                              // 右移，等待时钟信号有效后验证
        if (Q != {RIN,ii[3:1]})
              $display("S1S0 = 01, old QA-QD = %4b, RIN,LIN = %2b, QA-QD = %4b, shift-right failed",
ii[3:0], {RIN,LIN}, Q);
      end
      $display("All states shift-right test completed");
      $display("Starting shift-left test for all states");
```

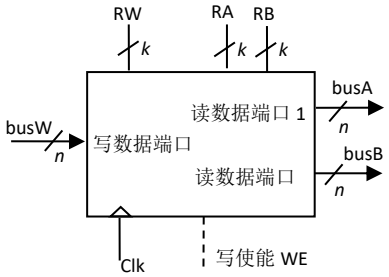
```
        RIN = 1'b0; LIN = 1'b1;                          // 左移移入位置 1
        for (ii=0; ii<=15; ii=ii+1) begin          // 左移测试
           {S1,S0} = 2'b11; I[3:0] = ii; #10 ;   //载入初始数据，等待时钟信号
           {S1,S0} = 2'b10;   #10 ;                    // 左移，等待时钟信号有效后验证
           if (Q != {ii[2:0],LIN})
                $display("S1S0 = 10, old QA-QD = %4b, RIN,LIN = %2b, QA-QD = %4b, shift-left failed", ii[3:0],
{RIN,LIN}, Q);
        end
        $display("All states shift-left test completed");
        $stop;
    end
    endmodule
```

实验验证：执行逻辑仿真测试文件验证功能，观测仿真波形图和输出信息，综合实现，并上实验板可以验证。

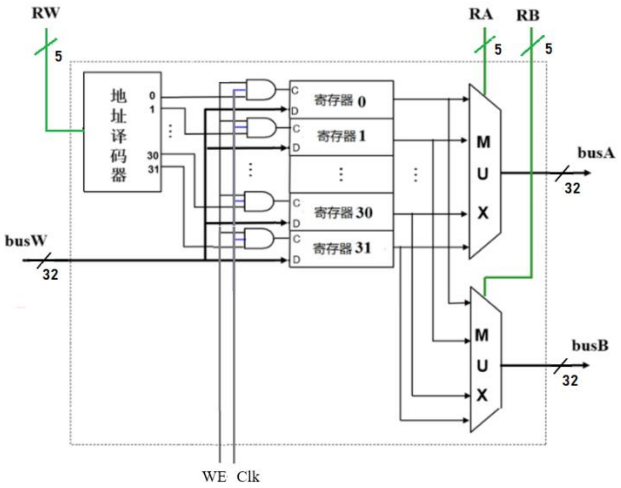
四、实验内容

1、寄存器堆设计

CPU 中有一个专门的寄存器堆，用于暂存指令执行过程中用到的中间数据。寄存器堆也称为通用寄存器组（General Purpose Register set，GPRs），它由许多寄存器组成，每个寄存器有一个编号，CPU 可以对指定编号的寄存器进行读写。图 2.4(a)是一个带时钟控制的双口寄存器堆的示意图，有两个读口和一个写口，每个读口或写口包括一个寄存器编号输入端和一个读数据端或写数据端，此外，还有一个写使能输入端 WE，它用来控制是否在下个时钟触发边沿到来时，开始将 busW 线上的数据写入寄存器堆中。图 2.4(b)所示为寄存器堆的内部结构示意图。



a） 寄存器堆的外部连接



b）寄存器堆的内部结构

图2.4 寄存器堆的外部连接和内部结构示意图

实验要求增加译码器和多路选择器，实现 32 个 32 位寄存器组成的寄存器堆。RA 和 RB 分别是读口 1 和读口 2 的寄存器编号，RW 是写口的寄存器编号。寄存器堆的读操作属于组合逻辑操作，无须时钟控制，即当寄存器地址信号 RA 或 RB 到达后，经过一个“读取时间”的延迟，读出的信息在 busA 或 busB 上开始有效。寄存器堆的写操作属于时序逻辑操作，需要时钟信号的控制；即在写使能信号（WE）有效的情况下，下个时钟触发边沿到来时开始将 busW 上的信息写入 RW 所指定的寄存器中。

假设写使能信号 we 高电平有效，写入时钟 clk 下降沿有效，实现 32 个 32 位寄存器堆的模块端口定义如下：

```
module regfile32 (       //32 个 32 位寄存器堆
    output   [31:0] busa,
    output   [31:0] busb,
    input [31:0] busw,
    input [4:0] ra,
    input [4:0] rb,
    input [4:0] rw,
    input clk, we
);
// Add your code
endmodule
```

为了能够在实验开发板中进行验证，写入数据 busw 使用 4 位拨档开关，然后重复 8 次表示 32 位输入数据；3 个地址端口 ra、rb、rw 分别使用 3 位输入开关表示端口地址的低 3 位，使用 2 位输入开关表示三个寄存器端口地址 ld_hi 的高 2 位；写使能信号 we 占用 1 位开关；写入时钟连接到按钮 BTNC 端，A 口和 B 口读出的 32 位数据中的低 8 位数据 busa8、busb8 分别用 8 个 led 指示灯来显示。

寄存器堆的父模块 regfile_top 端口定义如下：

```
module regfile_top (
```

```
    output    [7:0] busa8,
    output    [7:0] busb8,
    input [3:0] busw,
    input [2:0] ra,
    input [2:0] rb,
    input [2:0] rw,
    input [1:0] rd_hi,
    input clk, we
    );
    wire [31:0] busa32;
    wire [31:0] busb32;
    wire [31:0] busw32;
    wire [4:0] ra32;
    wire [4:0] rb32;
    wire [4:0] rw32;
//  Add your code
   regfile32 regfile32_check
        (.busa(busa32),.busb(busb32),.busw(busw32),.ra(ra32),.rb(rb32),.rw(rw32),.clk(clk),.we(we));
//  Add your code
endmodule
```

请根据上述描述，按照下列步骤完成实验。

1、 使用 Vivado 创建一个新工程。

2、 点击添加设计源码文件，加入 lab2.zip 里的 regfile32.v、regfile_top.v 文件。

3、 点击添仿真测试文件，加入 lab2.zip 里的 regfile32_tb.v 文件

4、 点击添加约束文件，加入 lab2.zip 里的 regfile_top.xdc 文件。

5、 根据实验要求，完成源码文件的设计。

6、 对工程进行仿真测试，分析输入输出时序波形和控制台信息。

7、 仿真通过后，进行综合、实现并生成比特流文件。

8、 生成比特流文件后，加载到实验开发板，进行调试验证，并记录验证过程。

2、 比特流加密实验

本实验将利用线性移位寄存器 LFSR 产生的随机比特流来对字符串进行加密和解密。异或操作可以用于加密和解密。假设消息数据为二进制变量 M，利用随机二进制密钥 K 可以通过 C=M⊕K 来生成加密信息 C。在接收端，如果已知密钥 K，可以用 C⊕K=M⊕K⊕K=M⊕(K⊕K)=M⊕0=M 来恢复原始消息 M。

用 64 位线性移位寄存器 LFSR 生成长度为 $2^{64}-1$ 的随机二进制流，并利用 64 位的 seed 来初始化随机二进制流，然后用该随机二进制流作为密钥 K 对数据流进行加密。假设需加密的数据是以 ASCII 表示的二进制字符串，每个字符以 8 比特表示。为了方便显示，规定输入的字符的二进制应在 0x40 至 0x7f 之间。对于每一个字符，先在 LFSR 生成的比特流中截取 6 比特，然后将该 6 比特与字符的低 6 位进行异或，并将结果输出。这样，如果输入字符的二进制在 0x40 至 0x7f 之间（高 2 比特固定为 01），输出字符的二进制也在 0x40 至 0x7f 之间，并且输出字符会被 LFSR 的随机比特混淆。在解密时，进行同样的操作，输入设置为混淆后的字符再次异或即可恢复出原始信息。

64 比特 LFSR 的反馈方程是 x64=x4⊕x3⊕x1⊕x0。假设 LSFR 的初始化 seed 为 64'ha845fd7183ad75c4。则 6 个周期后，LFSR 的最高 6 位二进制码流为 000010；12 个周期后，LFSR 新移入的高位 6 个比特是 011011，…。

假设需要加密的信息为“Nanjing_University_FPGA_Lab”。将字符串中第一个字母 N 的 ASCII 码 8'h4E=8'b01001110 的低 6 位和 LFSR 第 6 个周期后的高 6 位 000010 进行异或，得到 8'h4C，对应字符 L，因此加密后的第一个字符是 L。将第二个字符 a 的 ASCII 码 8'h61 的低 6 位和 LFSR 第 12 个周期后的高 6 位 011011 进行异或，得到 8'h7a，对应字符 z，因此加密后的第二个字符是 z。以此类推，得到的加密结果为“LzegCo\g~aQsU^?|QyBQZxP@wdr”。其中 ASCII 码 8'h7f 无法正常显示，以“? ”替代（实际输入输出仍然按照 8'h7f，此处只是显示改变而已）。

同样的，输入字符串“LzegCo\g~aQsU^?|QyBQZxP@wdr”，如果将 LSFR 的 seed 同样设置为 64'ha845fd7183ad75c4，则可得到原始数据为“Nanjing_University_FPGA_Lab”。

实验要求：

加密解密以输入时钟为上升沿有效，如果时钟上升沿时，load 信号为高电平，将 LFSR 的状态初始化为 seed 提供的 64 比特数据。同时加密解密过程也初始化，即生成比特计数重置为 0。

在 load 为低电平时，正常工作，每个时钟上升沿进行移位操作，并将生成比特计数加 1。生成比特计数器计数周期为 6，即在计数值从 5 变为 0 时，输出 ready 信号为高电平，持续一个时钟周期，其他情况 ready 信号为低电平。在 ready 为高电平时，将输入数据的低 6 位与 LSFR 输出的高 6 位（[63:58]，即刚刚新移入的 6 个比特）进行异或并输出。如果第六周期 ready 信号不是高电平，测试系统会用 x 替代实际输出。

加密电路模块端口定义如下：

```
module encryption6b(
    output [7:0] dataout,      //输出加密或解密后的 8 比特 ASCII 数据。
    output reg ready,          //输出有效标识，高电平说明输出有效，6 倍数周期输出高电平
    output [5:0] key,          //输出 6 位加密码
    input clk,                 // 时钟信号，上升沿有效
    input load,                //载入 seed 指示，高电平有效
    input [7:0] datain         //输入数据的 8 比特 ASCII 码。
```

```
);
   wire    [63:0] seed=64'ha845fd7183ad75c4;          //初始 64 比特 seed=64'ha845fd7183ad75c4
//add your code here
endmodule
module lfsr(                        //64 位线性移位寄存器
      output reg [63:0] dout,
      input    [63:0]    seed,
      input    clk,
      input    load
      );
         //add your code here
endmodule
```

在实验开发板中，使用拨动开关输入 7 位输入数据，使用 led 指示灯输出密码和加密后的数据，为了方便观察，在实验板验证时，时钟信号 clk 连接到按钮 BTNC。请根据上述描述，按照下列步骤完成实验。

1、 使用 Vivado 创建一个新工程。

2、 点击添加设计源码文件，加入 lab2.zip 里的 encryption6b.v 文件。

3、 点击添仿真测试文件，加入 lab2.zip 里的 encryption6b_tb.v 文件。

4、 点击添加约束文件，加入 lab2.zip 里的 encryption6b.xdc 文件。

5、 根据实验要求，完成源码文件的设计。

6、 对工程进行仿真测试，分析输入输出时序波形和控制台信息。

7、 仿真通过后，进行综合、实现并生成比特流文件。

8、 生成比特流文件后，加载到实验开发板，进行调试验证，并记录验证过程。

3、数字时钟实验

数字时钟是一个基于计数器和时钟发生器的系统，计数器用于计算时钟的时间，通常分为时、分、秒三个计数器。时钟发生器产生固定频率的时钟信号，用于驱动计数器的计数。七段数码管用于显示时钟的时间。数字时钟实验旨在通过设计和实现一个基于 FPGA 的数字时钟系统，加深对数字逻辑电路、时序电路和 FPGA 开发的理解，并提升学生的实践能力。

设计一个数字时钟，具备以下功能：数字时钟、倒计时、计时器和闹钟。数字时钟功能，包括显示当前的时、分、秒，并支持 12 小时制或 24 小时制，支持时制切换，支持复位功能，以设置初始时间；实现整点报时功能，同时闪烁点亮三色 LED 灯。倒计时功能，包括倒计时时间的设置和倒计时过程的显示，并在倒计时结束时触发警报。计时器功能，能够以毫秒为单位精确计时，包括计时器的开始、暂停和复位功能，并显示计时时间。配置闹钟功能，允许用户设置多个闹钟，并在设定时间到达时触发警报。

使用 FPGA 内部的时钟管理资源，确保时钟的准确性和稳定性。使用 Nexys A7-100T 开发板上的七段数码管显示当前时间、倒计时、计时器和闹钟的状态。

实验要求如下：用按钮 BTNC 表示复位信号 RST，当复位信号 RST 为 1 时有效，时钟、计时器、倒计时和闹钟设置清零；用按钮 BTNU 表示计时器开始和暂停功能，按下 1 次时，开始计时，再按 1 次时，暂停计时；用按钮 BTNP 表示读取拨档开关设置的参数。用拨档开关 SW15 表示 12 小时制还是 24 小时制，0 表示 24 小时制；1 表示 12 小时制，当处于 12 小时制的下午时 LED 指示灯 LD0 点亮；计时到 1 天结束时清零重新计时。用拨档开关 SW14~SW13 表示数字时钟功能选择，00 表示数字时钟，01 表示倒计时，10 表示计时器，11 表示设置闹钟。用拨档开关 SW12~SW11 表示参数设置，00 表示没有参数设置，01 表示设置秒参数，10 表示设置分钟参数，11 表示设置小时参数。用拨档开关 SW10~SW09 表示设置闹钟序号，00 表示设置第一个闹钟，01 表示设置第二个闹钟，以此类推，最多设置四个闹钟。用拨档开关 SW7~SW4、SW3~SW0 分别用表示参数的高位和低位（BCD 码）。时钟显示在七段数码管 AN7AN6-AN4AN3-AN1AN0 上。当到整点时，轮流 3 色指示灯 LD16 的灯光，持续 5 秒钟后熄灭；当到设定闹钟时，3 色指示灯 LD17 显示红色灯持续 10 秒后熄。实验开发板的输入输出布局如图 2-5 所示。

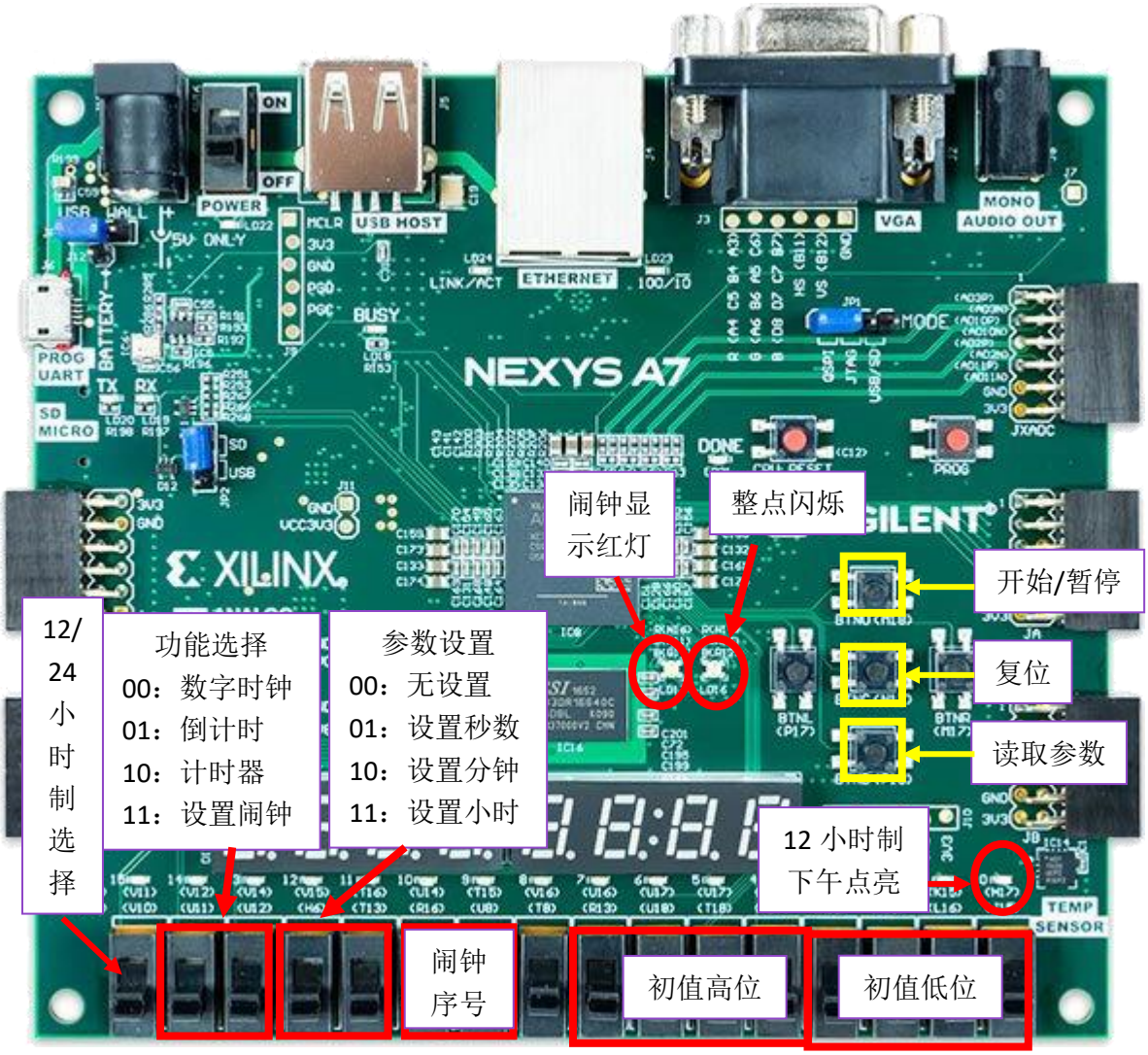


图 2-5 数字时钟显示及功能设置

首先利用开发板上的频率为 100MHz 的时钟，分频到 1Hz 计时。

数字时钟的模块端口定义如下：

```
module DigitalTimer (       //端口声明
  input clk,                //连接到时钟端口 CLK100MHZ，引脚 E3
  input RST,                //复位按钮，单击有效
  input StartOrPause,       //计时器开始或暂停，单击 1 次开始，再按 1 次暂停
  input ReadPara,           //读取参数，当参数设置结束后，单击 1 次，读取数据
  input TimeFormat          //=0 表示 24 小时制，=1 表示 12 小时制
  input [1:0] mode,         //功能选择，00 数字时钟，01 倒计时，10 计时器，11 设置闹钟
  input [1:0] ParaSelect,     // 参数设置，00 无；01 设置秒数；10 设置分钟；11 设置小时
  input [1:0] AlarmNo,        // 闹钟序号，0~3
  input [3:0] data_h,         //设置参数高位，使用 BCD 码表示
  input [3:0] data_l,         //设置参数低位，使用 BCD 码表示
  output   Afternoon，        //12 小时制时，下午时间输出为 1
  output [2:0] TimeKeeper,    //整点输出 3 色指示灯
  output [2:0] AlarmDisplay,  //闹钟输出 3 色指示灯
  output [6:0] segs,          //七段数码管输入值，显示数字
  output [7:0] an             //七段数码管控制位，控制时、分、秒
);
// Add your code
endmodule
```

请根据上述描述，按照下列步骤完成实验。

1、 使用 Vivado 创建一个新工程。

2、 点击添加设计源码文件，加入 lab2.zip 里的 DigitalTimer.v 文件。

3、 点击添加约束文件，加入 lab2.zip 里的 DigitalTimer.xdc 文件并修改。

4、 根据实验要求，完成源码文件的设计。

5、 对工程进行仿真测试，分析输入输出时序波形和控制台信息。

6、 仿真通过后，进行综合、实现并生成比特流文件。

7、 生成比特流文件后，加载到实验开发板，进行调试验证，并记录验证过程。

五、思考题

1、分析 32 个 32 位的寄存器堆占用的逻辑片资源。

2、分析 64 位移位寄存器的时序性能和资源占用情况；并通过资料查找到其他的生成 LFSR 的反馈公式。

3、数字时钟中是如何实现倒计时和毫秒计时器功能。

4、如何实现寄存器堆中 0 号寄存器的值始终为零。