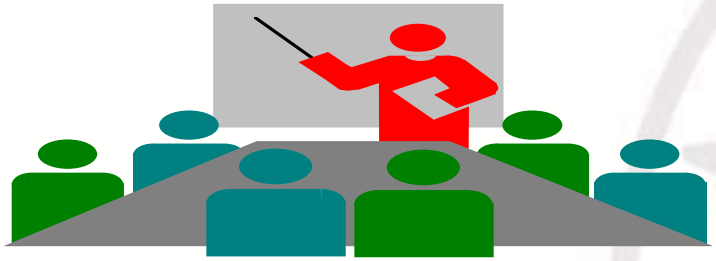


逻辑与计算机设计基础实验与课程设计

实验十三

计数器/定时器设计与应用

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验目的

1. 掌握二进制计数器/定时器的工作原理与设计方法；

2. 掌握用计数器进行分频的概念和方法；

3. 了解计算机中程序计数器（PC）的概念；

4. 了解计算机串行数据传送时波特率的概念。

实验环境

实验设备

1. 计算机(Intel Core i5以上，4GB内存以上)系统

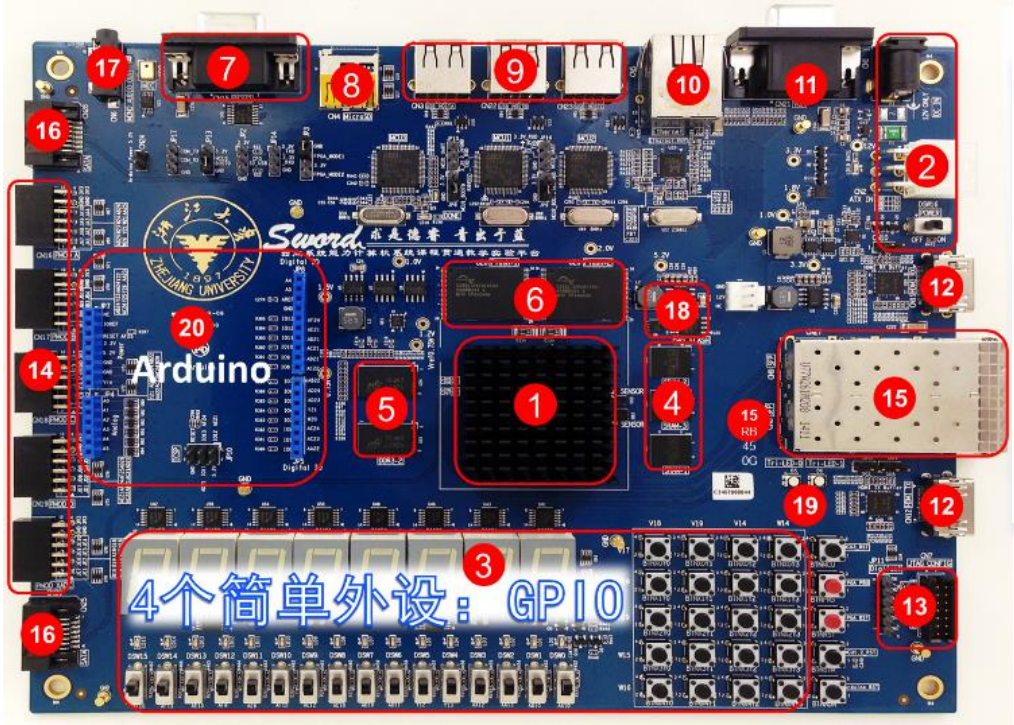
2. 计算机软硬件课程贯通教学实验系统(Sword)

3. Xilinx ISE14.4及以上开发工具

材料

无

计算机软硬件课程贯通教学实验系统



贯通教学实验平台主要参数

▼ **核心芯片**

Xilinx Kintex™-7系列的XC7K160/325资源：

162,240个，Slice：25350，片内存储：11.7Mb

▼ **存储体系** 支持32位存储层次体系结构

6MB SRAM静态存储器：支持32Data，16位TAG

512M BDDR3动态存储：支持32Data

32MB NOR Flash存储 ：支持32位Data

▼ **基本接口** 支持微机原理、SOC或微处理器简单应用

4×5+1矩阵按键、16位滑动开关、16位LED、8位七段数码管

▼ **标准接口** 支持基本计算机系统实现

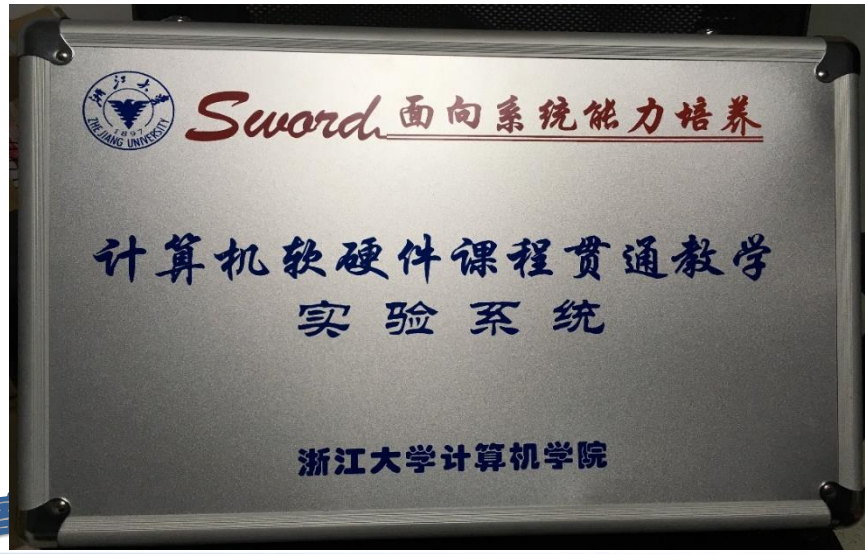
12位VGA接口（RGB656）、USB-HID（键盘）

▼ **通讯接口** 支持数据传输、调试和网络

UART接口、 10M/100M/1000M以太网、 SFP光纤接口

▼ **扩展接口** 支持外存、多媒体和个性化设备

MicroSD(TF）、PMOD、 HDMI、Arduino



Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验任务

1. 设计实现多种进制计数器；

2. 设计定时器带报警功能：Timer；

3. 设计实现24小时挂钟模块：WallClock;

4. 集成WallClock到实验测试环境DSTE测试

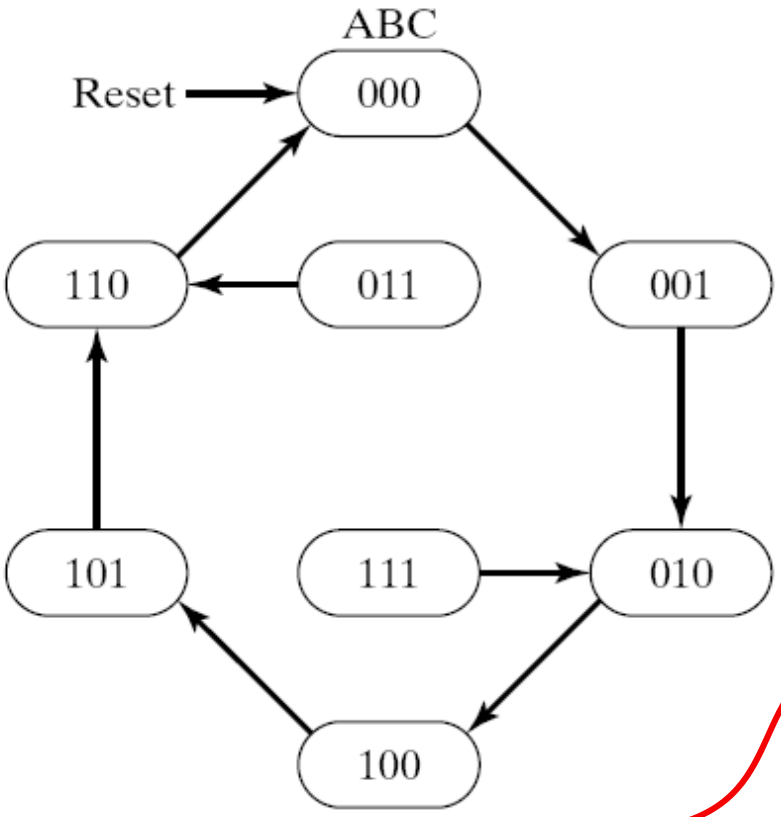
Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

普适计数器定义

- 计数器
  - 对指定状态进行遍历
  - 不一定是满足数制序列
  - 不一定循环
    - 饱和计数器：停止在起始或最后状态
- 模N计数器（Modulo-N counter）
  - 计数器的一般形式
  - 循环遍历N个固定状态的计数器
  - 其N个状态序列可以是任意的
  - 当遍历的序列满足二进制计数序列则为二进制计数器

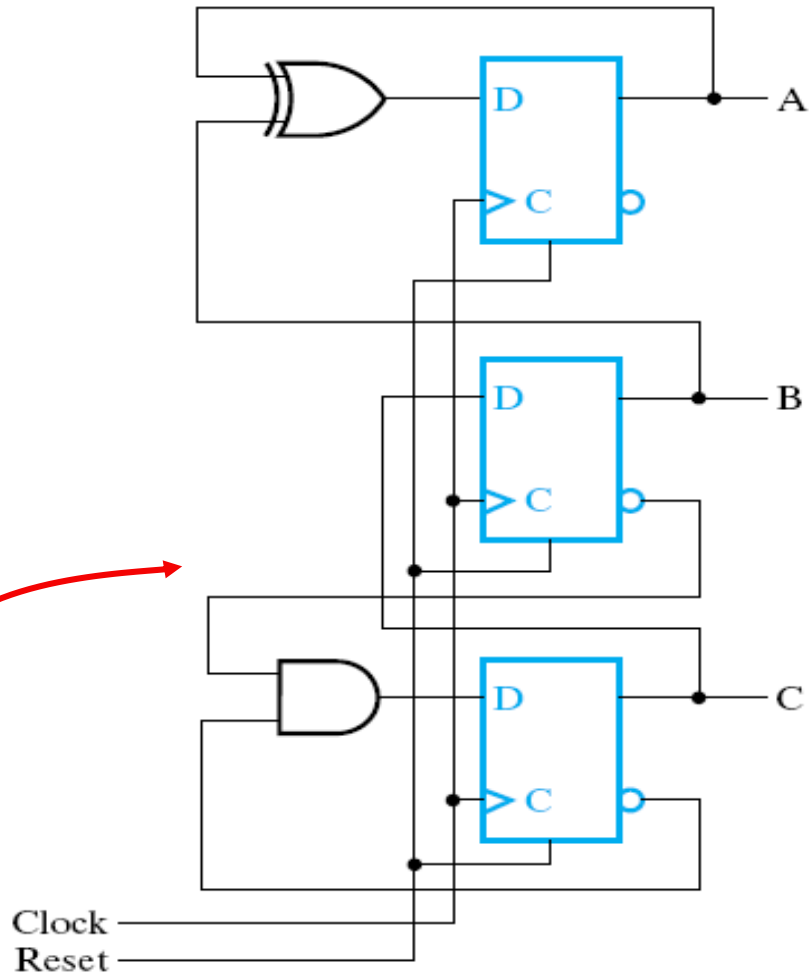
模-6计数器



$Q_A = A \oplus B$ $D_A = A \oplus B$

$Q_B = C$ $D_B = C$

$Q_C = \overline{B}\overline{C}$ $D_C = \overline{B}\overline{C}$



模-6计数器硬件描述

```
module counter_N(clk, reset, A, B, C);
        ……                  //端口及变量定义
parameter   State_0=3'b000, State_1=3'b001, State_2=3'b010,
            State_3=3'b100, State_4=3'b101, State_5=3'b110;
initial
state<=0;
        always@(posedge clk, posedge reset) begin
            if(reset)
                state<=State_0;
            else state<=next_state;
        end
        assign {A,B,C}=state;
        always@(state) begin
              next_state[2] <= state[2]^state[1];
              next_state[1] <= state[0];
              next_state[0] <= (!state[1])&&(~state[0]);
        end

endmodule
```

32位二进制双向计数器

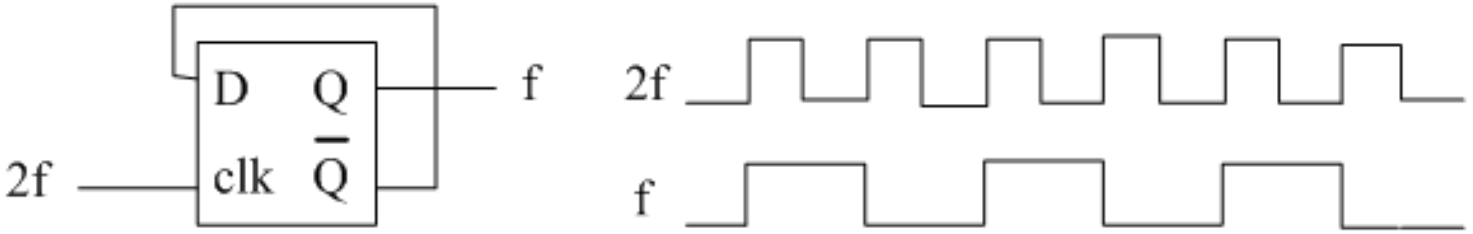
(参考实验十32位同步双向计数器)

- N=2^{32}时,其遍历序一定是满足加一或减一条件
- 32位双向计数器Verilog HDL代码

```
module counter_32bits(clk, reset, dir, count_out);
parameter COUNTER=32;
……                          //端口及变量定义
initial
count<=0;
        always@(posedge clk) begin
          if(!reset)
             count<=0;
          else if(dir)
                    count <= count+1;
                    else    count <= count-1;
        end
         assign count_out = count;
endmodule
```

分频 (参考实验九)

- 分频是将某一信号频率按需要降到另一低频率信号
  - 如二分频是将信号频率降低一倍，而信号的占空比保持不变
- 一般分频系数$N=2^n$
- 一个D触发器构成的二分频

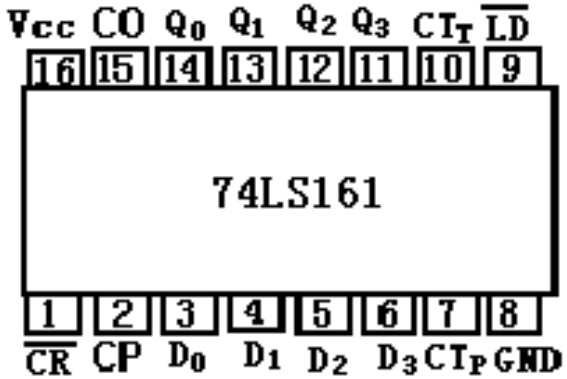


常用芯片：74LS161

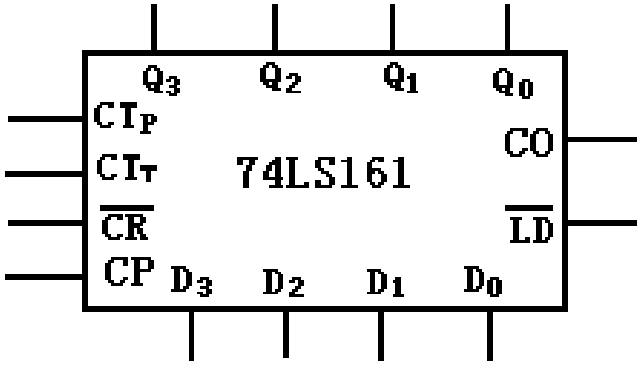
◎ 74LS161

$$Co = Q_3 Q_2 Q_1 Q_0 CT_T$$

- 常用的四位二进制可预置的同步加法计数器

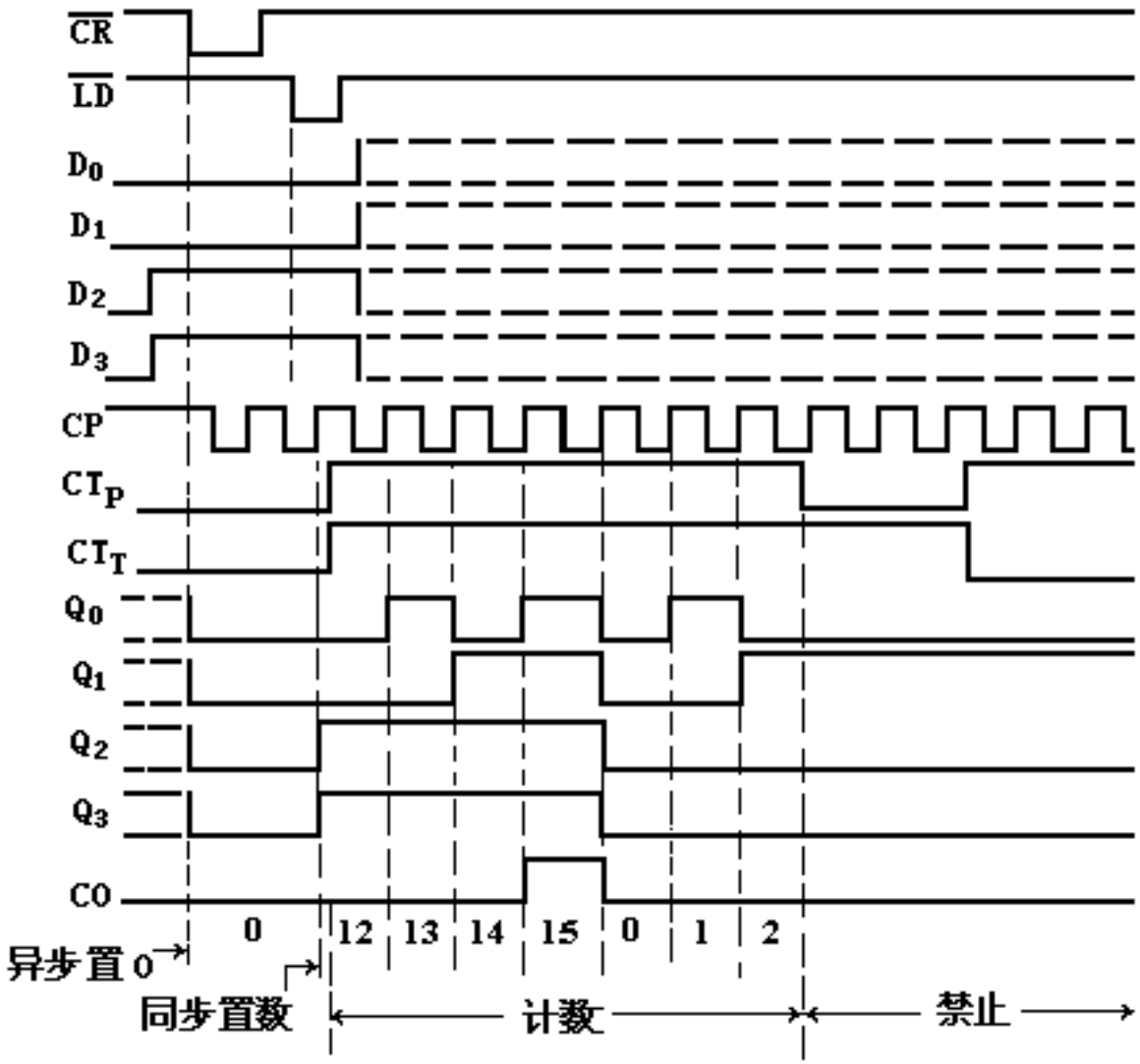


输入						输出
$\overline{CR}$	$\overline{LD}$	CT_P	CT_T	CP	$D_3 D_2 D_1 D_0$	$Q_3 Q_2 Q_1 Q_0$
0	×	×	×	×	××××	0 0 0 0
1	0	×	×	↑	$d_3 d_2 d_1 d_0$	$d_3 d_2 d_1 d_0$
1	1	0	1	×	××××	保 持
1	1	×	0	×	××××	保 持
1	1	1	1	↑	××××	计 数



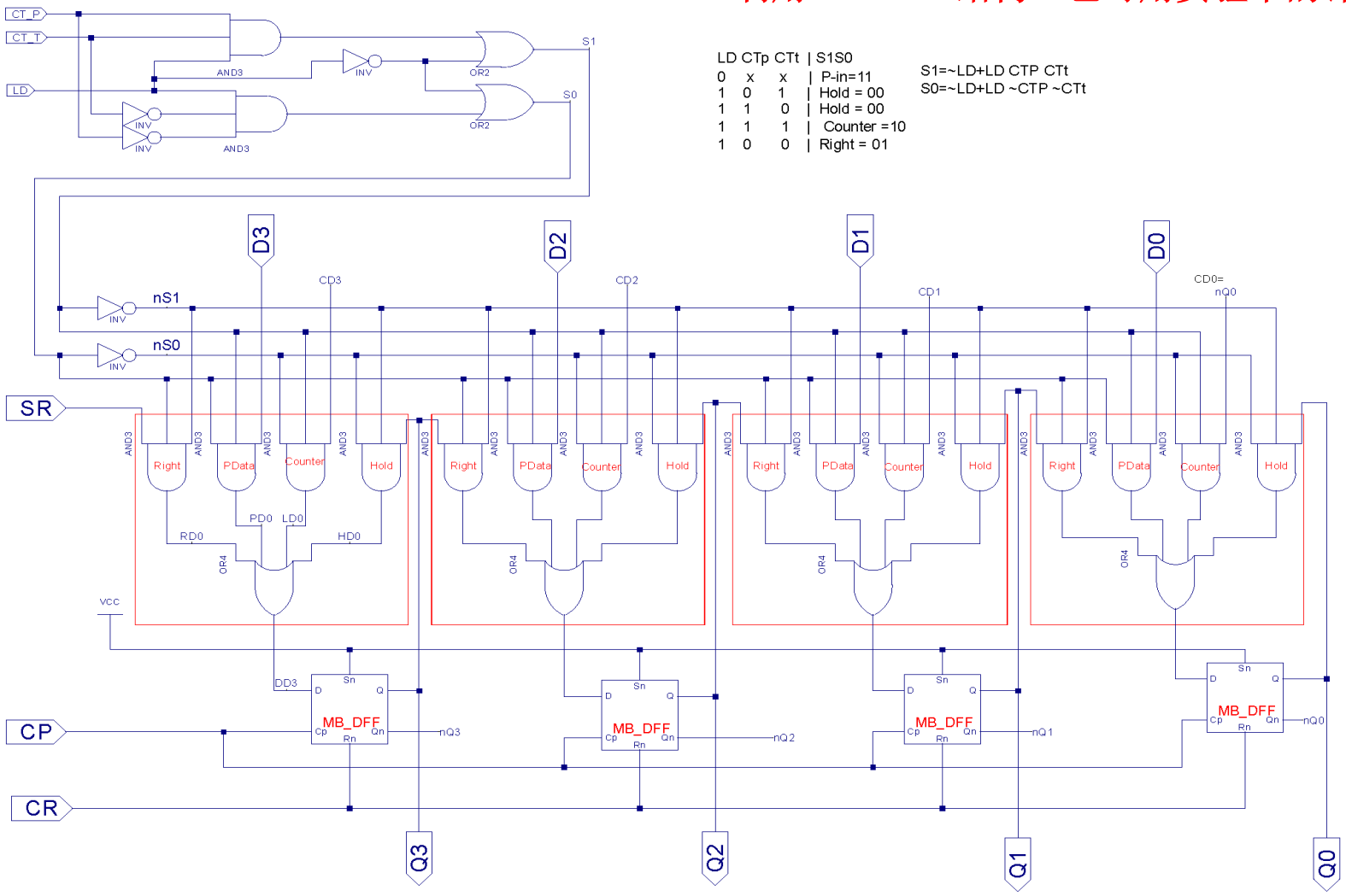
CR：异步清零，低有效
LD： 同步置入，低有效
CT_P，CT_T：使能，高有效
Co：计数进位， $Co = Q_3 Q_2 Q_1 Q_0 CT_T$

74LS161时序图

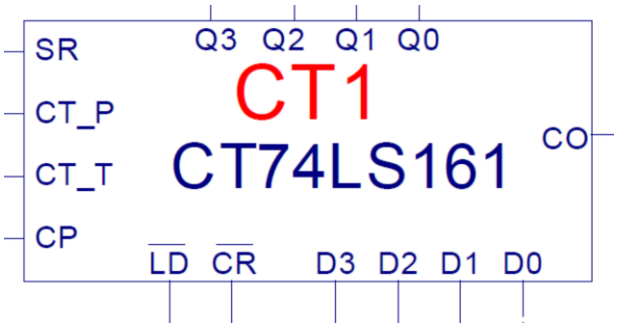

$\overline{CR}$
$\overline{LD}$
D_0
D_1
D_2
D_3
CP
CT_P
CT_T
Q_0
Q_1
Q_2
Q_3
CO
0
12
13
14
15
0
1
2
异步置0
同步置数
计数
禁止

CT74LS161兼容实现

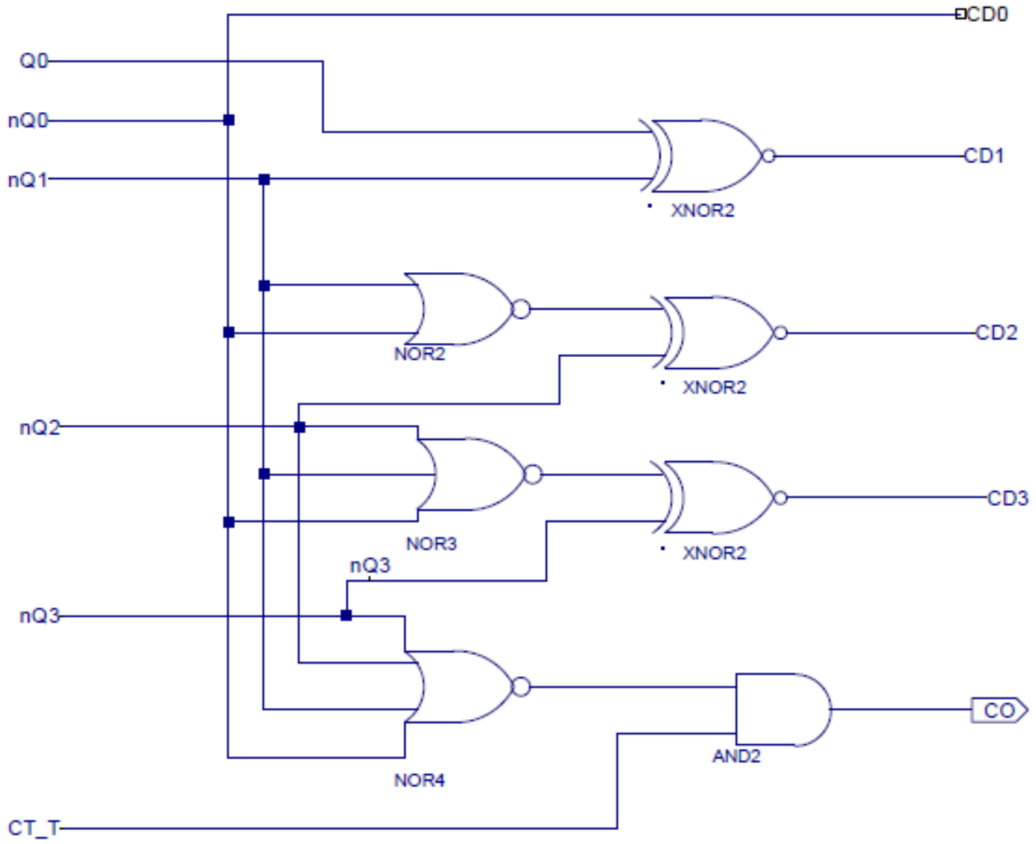
利用74LS164结构。也可用实验十的计数器封装



CT74LS161进位



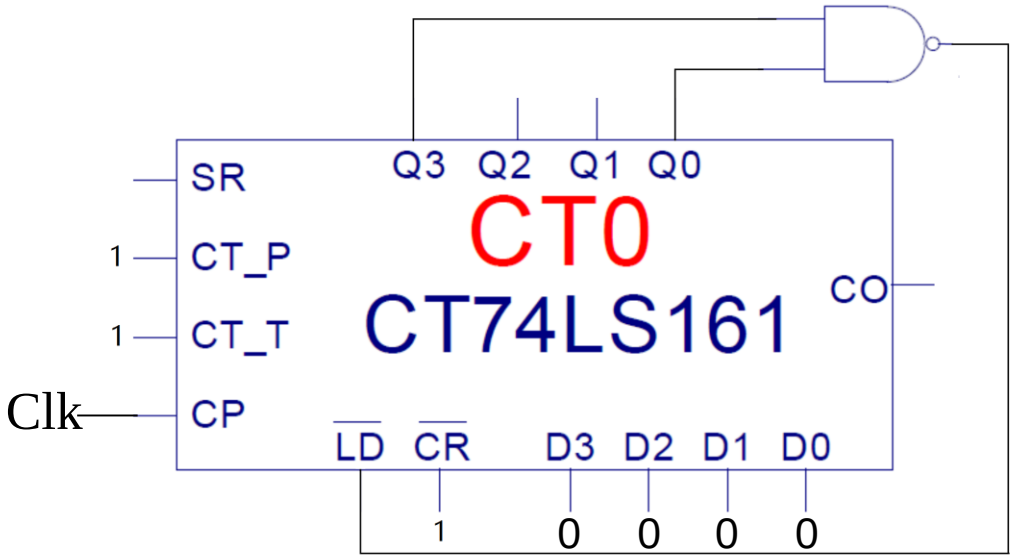
封装后的逻辑符号



CT74LS161实现十进制计数器

◎ 1位十进制计数器

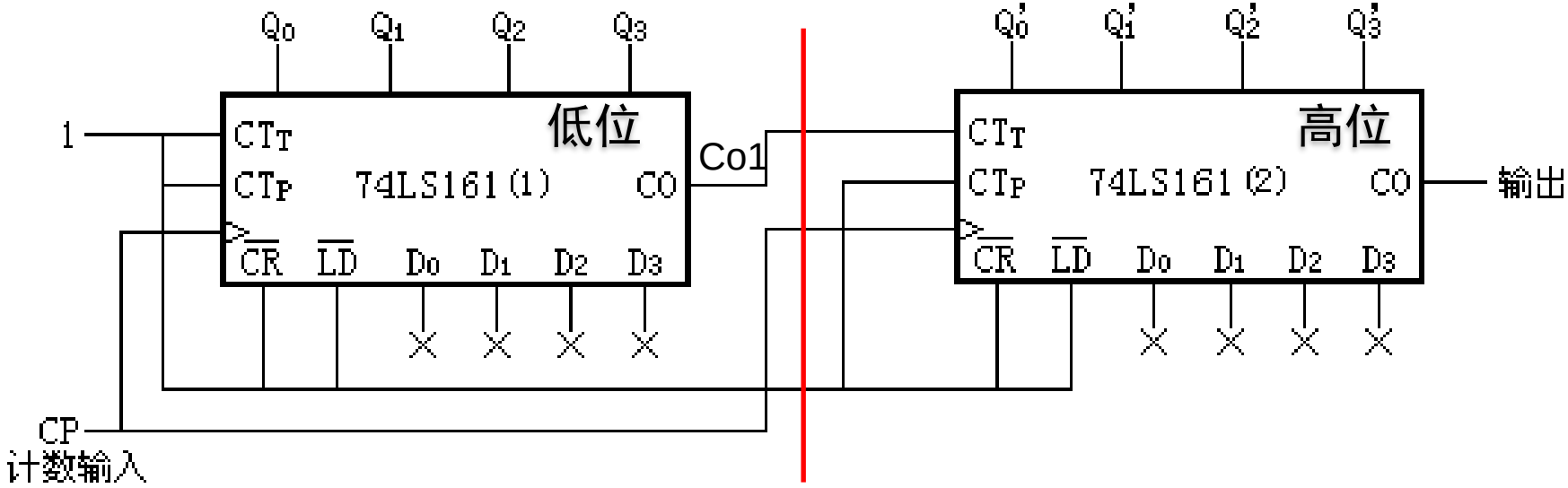
- 0000→1001
- 1001控制同步置入“0”



实现16×16进制计数器

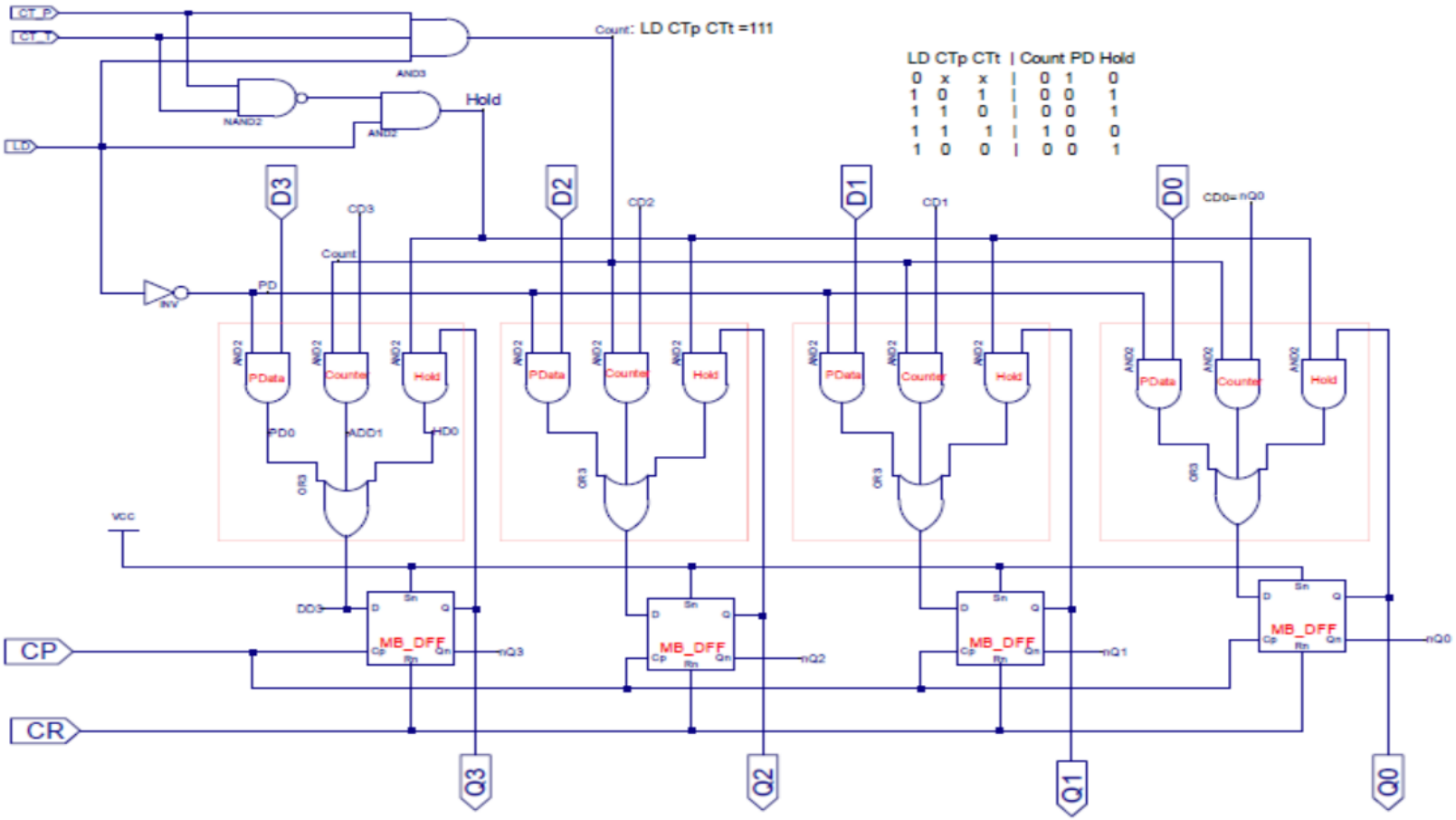
◎ 2位十六进制计数器

- 低位0000→1111→Co进位
  - ⊙ 高位进计数加1
- 同步回零
  - ⊙ 高低位同时1111_1111 →0000_0000



50Mz时钟定时毫秒脉冲：逻辑结构

millisecond



定时：毫秒时钟脉冲

- 分频与定时
  - 非2^n分频需要定时获取分频
  - 定时也是通过计数来获取的
  - 1ms定时：计数1ms产生一个脉冲
    - 从“0”计数到5×10^4-1清“0”，同时输出一个脉冲
    - 预设计数初值为15536=0011110010110000开始计数，计到“0”时重新置入初值，同时输出一个脉冲
  - 也可以用十进制计数器来实现
- 从“0”开始的计时常数（主频50MHz）

$$N = \frac{1\times10^{-3}}{\frac{1}{50\times10^{6}}} = 5\times10^{4} = (1100001101010000)_2$$

毫秒基准时钟 HDL描述

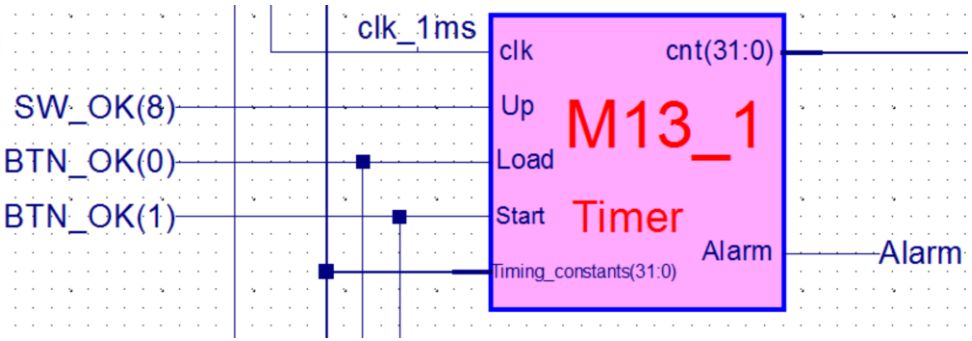
```
module      millisecond(input clk,
                        input rst,
                        output ms1
                        );
parameter COUNTER=16;
reg [COUNTER-1:0] count;
reg second_m;
initial count <= 0;
            always@(posedge clk)begin
                    if(rst || (count[15:0]==16'hC34F))begin        //=49999
                       count       <= 0;
                      second_m <= 1;
                    end
                    else begin
                       count[15:0] <= count+1;
                       second_m   <= 0;
                    end
            end
            assign ms1=second_m;
endmodule
```

定时器与多种进制计数器

- 定时器的核心是计数器
  - 用已知的基准时钟计数构成
- 二种实现方式：
  - 一是采用计数到某一指定值；
  - 二是设定某一时间常数，递减计数至零
  - 常用的方法是采用设时常数递减方法，
    - 如串行通讯的波特率、PC机内部定时等

定时器(带报警)：Timer

```
1  module       Timer(input clk,                          //时钟
2                     input Up,                           //计数方向
3                     input Load,                         //计数初值加载控制
4                     input Start,
5                     input[31:0]Timing_constants,        //定时常数
6                     output reg[31:0]cnt,                //32计数器
7                     output reg Alarm                    //定时结束(溢出)
8                     );
9  reg [1:0]go;
10   always @(posedge clk or posedge Start) begin
11      if(Start)begin
12         go <= 2'b01; Alarm <= 0;
13         cnt <= Timing_constants;
14         end
15      else begin
16         if(Load)cnt <= Timing_constants;
17         else begin
18            if (go==2'b01)begin
19            Alarm<=0;
20            if (Up) cnt <= cnt + 1;          //Up==1时，正向计数
21            else    cnt <= cnt - 1;          //Up==0时，反向计数
22            end
23      //计数溢出也可以如下描述
24            if((|cnt)==0 |                   //cnt[31: 0]=32'h00000000,则|cnt=1
25               (&cnt==1)) begin              //cnt[31: 0]=32'hffffffff,则&cnt=1) Rc<=1;
26               Alarm <= 1;
27               go <= 0;
28            end
29 //          else Alarm<=0;
30         end
31      end
32   end
33 endmodule
```



计数定时应用案例： Anti_jitter

```
module Anti_jitter(input wire clk,                    //时钟
                   input wire RSTN,                   //复位输入，长按复位
                   input wire [3:0] K_COL,            //键盘列信号
                   input wire [15:0] SW,              //滑动开关
                   output reg [3:0]button_out,        //稳定输出按键信号
                   output reg [3:0]button_pulse,      //稳定输出单脉冲
                   output reg [15:0] SW_OK,           //稳定输出滑动开关
                   output [4:0]K_ROW,                 //键盘行信号
                   output reg CR,                     //短按输出独立信号，对应RSTN
                   output reg rst                     //长按输出复位信号，对应RSTN

                  );

reg [31:0] counter, rst_counter;
reg [4:0]  btn_temp;
reg [15:0] sw_temp;
reg pulse;

   wire [4:0] button = {~RSTN,~K_COL[3:0]};
   assign K_ROW = {SW[15:11]};                        //K_ROW 恒等于"0"
```

计数定时应用红色圈内

```
always @(posedge clk) begin
    btn_temp <= button;                                    //采样键值
    sw_temp <= SW;                                         //采样开关
    if(btn_temp != button || sw_temp !=SW) begin           //有键按下或释放，开始计数
        counter <= 32'h00000000;
        rst_counter <= 0;
        pulse <= 0;
        end
    else if(counter<100000)                                //去抖动，计数定时
        counter<=counter+1;
        else begin                                         //定时结束
            button_out <= button[3:0];                     //输出稳定按键
            CR <= ~RSTN;                                   //短按输出稳定CR
            SW_OK <= SW;                                   //输出稳定滑动开关
            pulse <= 1;                                    //输出脉冲
            if(!pulse)button_pulse <= button;              //脉冲结束
            else button_pulse <= 0;
            if(button[4] && rst_counter<200000000)         //长按复位定时
                rst_counter <= rst_counter + 1;
            else rst <= ~RSTN;                             //长按复位输出
        end
    end
endmodule
```

Wall Clock：

- 十进制产生秒时钟 ： 由毫秒作为时钟clk_1ms

```
always@(posedge clk_1ms)begin
    if(!reset)begin
          ms <= 0;
     end
     else begin
          if (ms == 12'b1001_1001_1001)begin
              ms      <= 0;
              clk_1s <= 1;
          end
          else if (ms[7:0] == 8'b1001_1001)begin
                  ms[7:0]   <= 0;
                  ms[11:8] <= ms[11:8]+1;
              end
              else if(ms[3:0]==4'b1001) begin
                      ms[3:0]<=0;
                      ms[7:4]<=ms[7:4]+1;
                  end
                  else begin
                          ms[3:0]<=ms[3:0]+1;
                          clk_1s<=0;
                  end
      end
end
```

挂钟60进制”时”时钟

- 由“分”作为时钟clk_1min
  - 60进制，“分”时钟产生同此相同

```
always@(posedge clk_1min  ) begin
 if (tminute[7:0] == 8'b00000000)begin
            tminute <= 8'b0101_1001;
            clk_1thour<=1;
            end
 else if (tminute[3:0] == 4'b0000) begin
              tminute[3:0] <= 4'b1001;
              tminute[7:4] <= tminute[7:4]-1;
              end
      else begin
            tminute     <= tminute-1;
            clk_1thour<= 0;
            end
end
```

日计时

- 用“时”作为时钟clk_1thour
  - 24进制或12进制

```
always@(posedge clk_1thour )begin
    if (thour == 8'b00000000)begin
                thour <= 8'b0010_0100;
    end
    else if (thour[3:0] == 4'b0000) begin
                thour[3:0] <= 4'b1001;
                thour[7:4] <= thour[7:4]-1;
               end
        else    thour<=thour-1;
end
```

定时报警

- 定时报警参数
  - 启动报警： **timer_start、Atime_start**
  - 定时参数： **0、ahour, aminute**

```
always *                                                    //整点定时报警，一分钟
    if(({thour, tminute} == 0) && (!timer_start))           //定时变量=0
        alert <= 1;
    else alert <= 0;

always *                                                    //可变定时报警，一分钟
    if(({thour, tminute} == ahour, aminute) && (!Aimer_start))
        alert <= 1;                                         //定时变量=ahour, aminute
    else alert <= 0;
```

挂钟集成和I/O接口描述参考

挂钟集成

```
clk_1ms     millisecond (.clk(clk), .clk_1ms(clk_1ms));        //.reset(reset),
ms_1000     m13_ms(.clk(clk_1ms),.reset(reset),.ms(msecond),.clk_1s(time_clk_1s));
                                                              //10进制微秒记数，秒时钟发生器
count_60    m13_sec(time_clk_1s, d_sec , second,clk_1min); //60进制 秒计数，分时钟发生器
count_60    m13_min(clk_1, reset , minute,clk_1hour);         //60进制 分计数，时时钟发生器
count_24    m13_hour(clk_2, reset, hour, clk_1day);           //24进制 时计数，天时钟发生器

assign clk_1 = (d_min  & inc)  | (!d_min & clk_1min);         //分计数时钟，   d_min=1校准
assign clk_2 = (d_hour & inc) | (!d_hour & clk_1hour);        //                    校准

        always@(posedge adj_push[2])
                adjust <= ~adjust;                            //adj_push[2]：时钟与校准切换

        always@(posedge adj_push[0] )                         // adj_push[0]：
            if (!adjust) d_state <= d_state + 2'b01;                  //时钟显示切换
            else     t_state <= t_state + 2'b01;                      //校准位切换
```

Inc=clk_div(21)，计数校准

挂钟集成和I/O接口描述参考

挂钟I/O接口：显示用通道6低16位

```
always@*begin                    //“分”用小数点指示
    case(d_state)
    2'b00: begin Time_out = {minute[7:0],second[7:0]};    //显示 "分.秒" =tminute . tsecond
                 s_point  = {second[0],second[0],2'b00};  end
     2'b01: begin Time_out = {hour[7:0], minute[7:0]};    //显示 "时.分" =thour . tminute
                  s_point  = {2'b00, second[0],second[0]}; end
     2'b10: begin Time_out = {second[3:0],msecond[11:8],msecond[7:4],msecond[3:0]};
                 s_point  = {second};                    end  //显示 "秒.毫秒"= tsecond . msecond
     2'b11: begin Time_out = {second[3:0],msecond[11:8],msecond[7:4],msecond[3:0]};
                  s_point  = {4'b0000}; end
     endcase
     if(!adjust)t_blinke=4'b0000;
      else begin
          case({d_state[0],t_state})
          2'b00: begin t_blinke = 4'b0011; d_sec  = ~adj_push[1]; end     //d_sec秒校准
          2'b01: begin t_blinke = 4'b1100; d_min  =  adj_push[1]; end     //d_min分校准
          2'b10: begin t_blinke = 4'b1100; d_hour =  adj_push[1]; end     //d_hour时分校准
          2'b11: begin t_blinke = 4'b0011; d_min  =  adj_push[1]; end     //d_min分校准
      endcase
end
```

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

设计工程一：Exp13-WallClock

- **设计实现挂钟模块**
  - 功能：
    - 24/12小时内计时
    - 时间显示：毫秒、秒、分和时
    - 报警功能*：整点、任意设定时间
    - 具有秒、分和时的设定
    - 小数点指示秒计数，跟随分或时显示
  - 用行为描述设计实现
- **集成挂钟功能到实验十二的“混合计算器”**
  - 修改实验十二顶层模块为：Top-WallClock
  - 其余功能不变，新增功能：
    - 12/24挂钟
    - 挂钟输出显示用通道6的低16位

设计要点

- 新建工程：Exp13-WallClock
- 设计实现毫秒计数器
  - 命名：clk_1ms.v
- 设计实现秒计数器
  - 命名：ms_1000.v
- 设计实现60进制计数器
  - 命名：count_60.v
- 设计实现24/12进制计数器
  - 命名： count_24.v
- 集成挂钟模块
  - 命名：Wall_CLOCK
- 本实验除顶层模块全部采用行为描述

毫秒、时、分、秒模块接口参考

◎ 实现毫秒时钟定时基准并仿真

```
module ms_1000(input clk,
               input reset,
               output reg[11:0] ms,
               output reg clk_1s
               );
    ..........

endmodule
```

◎ 实现60进制分、秒定时基准并仿真

```
module      count_60(input clk;
                     input reset;
                     output [7:0] six_ten;
                     output count_carry
                     );

..........

endmodule
```

注意：是计时不是定时

◎ 实现24进制 ‘时’ 定时基准并仿真

```
count_two_four(input clk;
               input reset;
               output [7:0] two_four;
              output count_carry
              );
    ..........
endmodule
```

注意：是计时不是定时

◎ 挂钟24小时内模块的调用关系

```
ms_1000      m13_ms(clk_1ms,reset,msecond,time_clk_1s);      //秒
count_60     m13_sec(time_clk_1s, d_sec , second,clk_1min);  //分
count_60     m13_min(clk_1, reset , minute,clk_1hour);       //时
count_24     m13_hour(clk_2, reset, hour, clk_1day);         //日
```

实现24小时内挂钟：Wall_Clock

◎模块I/O功能

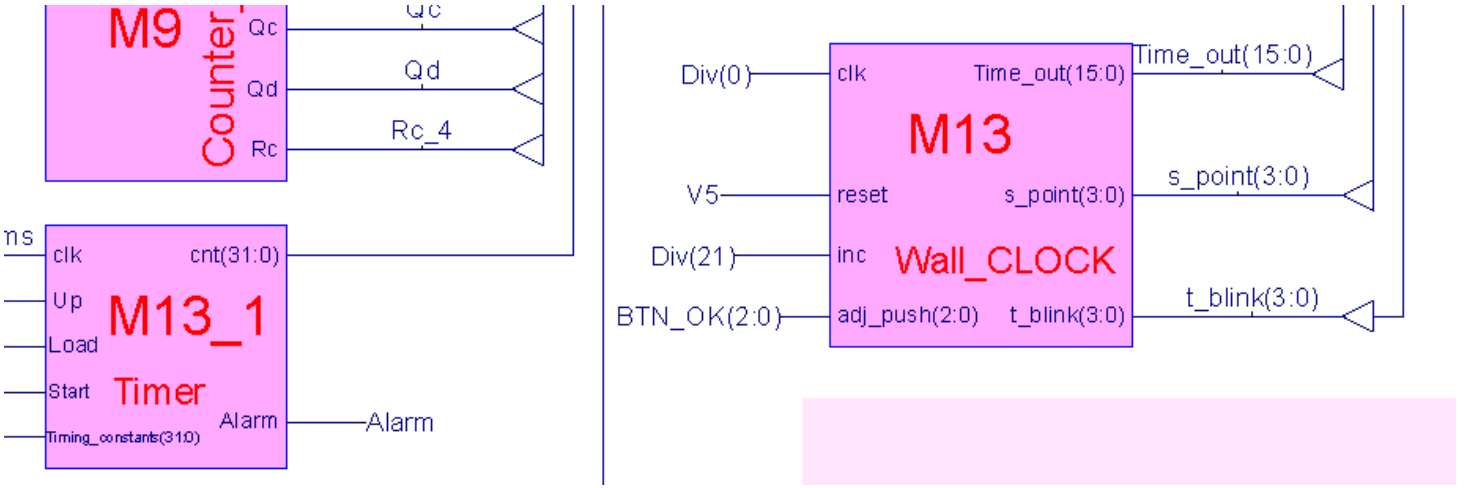
- adj_push[0]： push_out(0)，时、分、秒、毫秒显示切换
- adj_push[1]： push_out(1)，校准/时钟工作切换
- adj_push[2]： push_out(2)，校准，与adj_push[0]对应
- Time_out（15:0）：显示时钟输出：XX XX
- **时钟输出显示：七段码显示通道Disp4，高16位补0**
- Inc：校准用计数时钟（count_out(21)）
- s_point：用于秒跳动指示
- t_blinke：当前校准位指示
  - 注：2位七段码同时闪

◎顶层模块采用Top-WallClock

集成混合计算器：增加挂钟功能

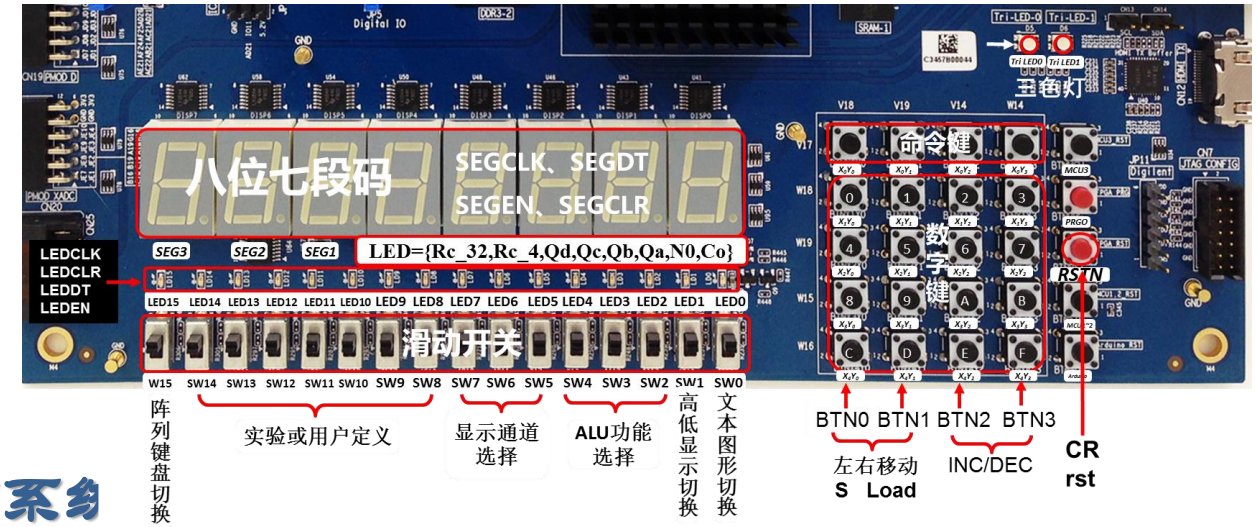
◎ 集成Calculation

- 复制实验十二的顶层模块，并改名为：Top_WallClock.sch
  - ⊙ 集成挂钟模块，命名M13
- 接口分配
  - ⊙ 输入：clk = Div(0)；计数校准时钟inc=Div(21)； reset=V5(注意极性)；校准控制 adj_push(2:0)=BTN(2:0)；
  - ⊙ 输出：Time_out→显示通道6低16位；s_point→对应通道6小数点低4位；t_blink→对应通道6使能控制低4位



物理验证

- 输入
  - SW[7:5]=通道选择
    - =000: 修改被加数、=001: 修改加数、=010: ALU输出、=011:32位计数输出、=100:寄存器堆Q_A输出、=101:32位移位寄存器输出、=110挂钟输出
  - SW[1]= 高低16位数据选择
  - BTN[2]=输入修改、BTN[1]=方向、BTN[0]=移动
  - SW[4:2]=ALU功能控制
- 输出: {a~g，p }= SEGMENT，AN=AN，
  LED={Alarm,Rc_4,Qd,Qc,Qb,Qa,N0,Co}
- 同实验十二

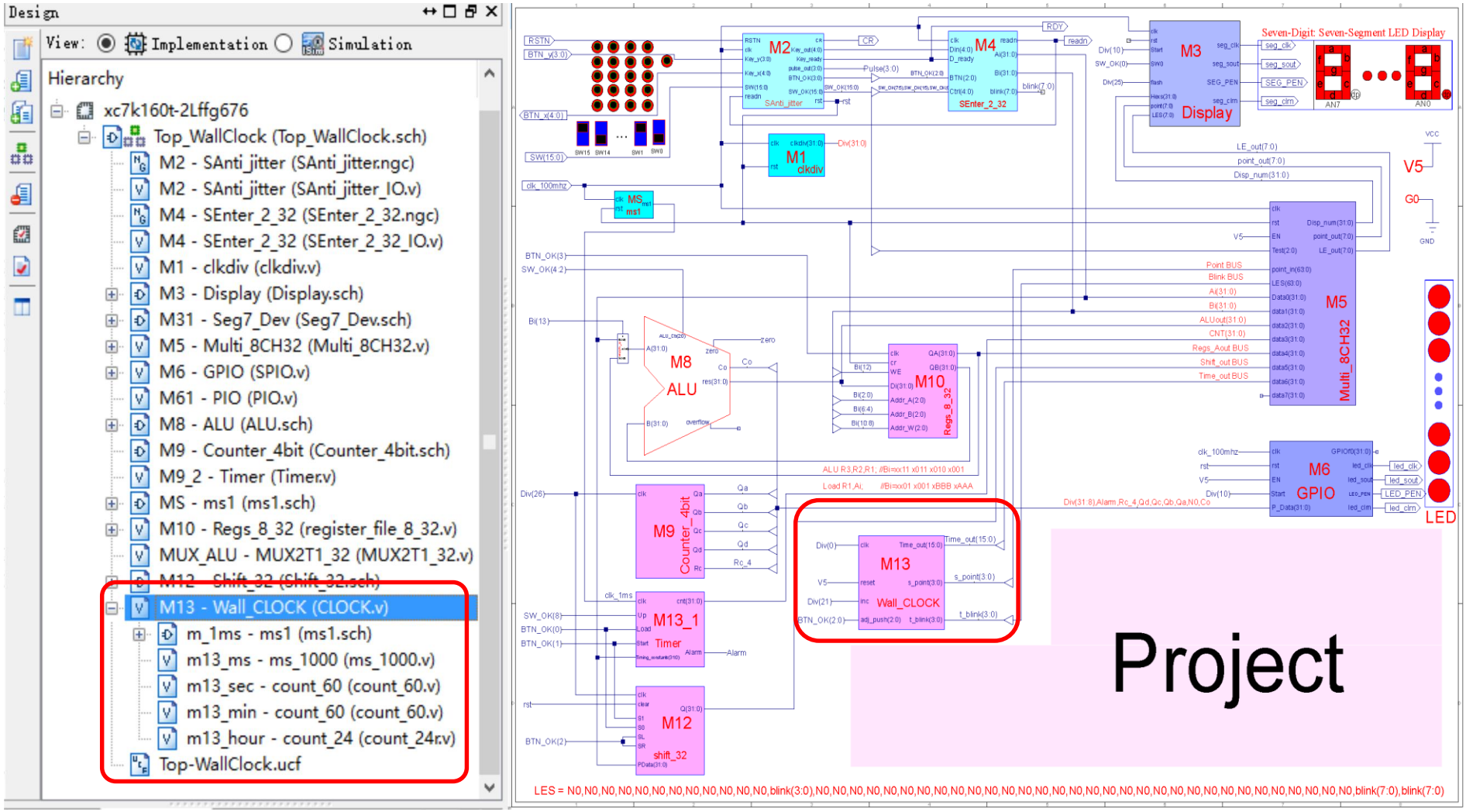


输入设备功能定义

开关定义	=0	=1	备注
SW[0]	未用		
SW[1]	32位二进制高16位	32位二进制低16位	
SW[4:2]	ALU功能选择		参考ALU功能表
SW[7:5]	通道选择 =000 =001 =010 =011 **=100** **=101** =110 =111	 通道0 通道1 通道2 通道3 **通道4** **通道5** 通道6 通道7	 **A*i*** **B*i*** **RES(ALU_Out)** **Cnt** **寄存器A输出QA** **移位寄存器输出Q** **挂钟输出**

按键定义	=0	=1	备注
Button[0]	按键切换显示：分.秒→时.分→秒.毫秒		**循环切换**
Button[1]		**按住不放校准输入**	**递增计数修改**
Button[2]		**显示与校准切换**	
Button[3]			

实验十三的顶层结构


Design
View: Implementation Simulation
Hierarchy
xc7k160t-2Lffg676
Top_WallClock (Top_WallClock.sch)
M2 - SAnti_jitter (SAnti_jitter.ngc)
M2 - SAnti_jitter (SAnti_jitter_IO.v)
M4 - SEnter_2_32 (SEnter_2_32.ngc)
M4 - SEnter_2_32 (SEnter_2_32_IO.v)
M1 - clkdiv (clkdiv.v)
M3 - Display (Display.sch)
M31 - Seg7_Dev (Seg7_Dev.sch)
M5 - Multi_8CH32 (Multi_8CH32.v)
M6 - GPIO (SPIO.v)
M61 - PIO (PIO.v)
M8 - ALU (ALU.sch)
M9 - Counter_4bit (Counter_4bit.sch)
M9_2 - Timer (Timer.v)
MS - ms1 (ms1.sch)
M10 - Regs_8_32 (register_file_8_32.v)
MUX_ALU - MUX2T1_32 (MUX2T1_32.v)
M12 - Shift_32 (Shift_32.sch)
M13 - Wall_CLOCK (CLOCK.v)
m_1ms - ms1 (ms1.sch)
m13_ms - ms_1000 (ms_1000.v)
m13_sec - count_60 (count_60.v)
m13_min - count_60 (count_60.v)
m13_hour - count_24 (count_24r.v)
Top-WallClock.ucf
Seven-Digit: Seven-Segment LED Display
M2
M4
M3
Display
M1
clkdiv
MS
ms1
M8
ALU
M10
Regs_8_32
M5
Multi_8CH32
M6
GPIO
M9
Counter_4bit
M13
Wall_CLOCK
M13_1
Timer
M12
shift_32
LED
Project

同学们：每次做完实验请整理好实验台，放好仪器，理清桌面。


Thank you!