数字逻辑电路实验报告

实验题目：计数器设计

实验日期：2025 年 11 月 01 日

实验者姓名：丁毅　2023K8009908031

1　实验目的

1、掌握简单时序逻辑电路的设计方法；

2、理解同步计数器和异步计数器的原理；

3、了解任意进制计数器的设计方法。

2　实验原理

在数字电路系统中，计数器 (Counter) 作为一种基础且重要的时序电路 (Sequential Circuit) ，其功能是驱动内部触发器 (Flip-Flop) 按照特定规律进行状态转换，从而对输入时钟脉冲 (Clock Pulse) 的数量进行累计。根据计数顺序 (Counting Sequence) 的差异，计数器主要分为加法计数器 (Up Counter) 、减法计数器 (Down Counter) 、可逆计数器 (Up/Down Counter) 以及拥有不同模值 (Modulus) 的计数器；若按其工作模式 (Operating Mode) 划分，则可分为异步计数器 (Asynchronous Counter) 与同步计数器 (Synchronous Counter) 。

以二进制加法计数器为例，其计数值随着每一个计数脉冲的输入而递增。具体表现为：最低位触发器 QA 在每一个时钟脉冲到来时都会改变状态 (Toggle) ，而更高位的触发器则是在其相邻低位的状态从 1 跳变到 0 时才会发生翻转。这种机制可以通过将低一位触发器的反相输出 (Inverted Output) 连接到高一位的时钟输入端 (Clock Input) 来实现，从而构建出异步加法计数器的结构。与之相反，二进制减法计数器的计数值随脉冲输入而递减，其关键区别在于，它是利用低一位触发器的同相输出 (Direct Output) ，而非反相输出，来作为高一位的时钟信号。

上述基于脉冲逐级传递的计数器均属于异步计数器。在这种设计中，计数控制信号 (Control Signal) 需要串行地通过每一级触发器，这导致从时钟有效沿 (Clock Edge) 到达直至最后一级触发器翻转并进入稳定状态 (Stable State) ，需要经历较长的传播延迟 (Propagation Delay) 。该延迟时间通常会随着计数器位数的增加而累加，从而限制了电

路的工作速度。为了提升性能，可以采用同步计数器 (Synchronous Counter) 方案。在同步计数器中，所有触发器共享同一个时钟信号，并且每一位触发器是否在时钟上升沿翻转，取决于其所有低位触发器的输出是否同时为 1 (即进位条件) 。由于所有触发器的状态更新都是在同一时钟沿同步 (Synchronously) 完成的，其输出达到稳定所需的时间仅由单个触发器的翻转延迟决定，而与计数器的总位数无关，从而实现了更高速度的计数操作。

3 实验仪器和设备

EGO1 开发板 (xc7a35ticsg324-1L)、EDA 软件 Vivado 2019.1 等。

4 实验方法和步骤

1、设计具有异步复位控制的 4-bit synchronous up counter (四位同步加法计数器)

2、设计具有异步复位控制的 4-bit asynchronous up counter (四位异步加法计数器)

3、在以上基础上设计一个减法计数器，同步异步均可 (我们选择同步)

4、编写 testbench 测试代码，完成三种计数器的行为级仿真

5、设置 constraints 文件，导出 bitstream 文件烧录至板上进行验证

注：数码管文件和按键防抖模块可以参考老师提供的现有文件，直接调用。

5 实验内容和结果

1、4-bit synchronous up counter (四位同步加法计数器)

按老师要求，异步计数器的每一位都应该被一个单独的 always 逻辑控制，因为每一位的触发信号不完全相同。但同步计数器无需这样，直接在一个 always 块中统一赋值即可。当前位是否翻转取决于比他低的所有位是否全为 1，当然，我们还加入了异步复位。

```
module md_counter_syn_4bit(  // 4-bit synchronous up counter (同步加法计数器)
    input wire clk,
    input wire rst,
    output reg [3:0] count
);

always@(posedge clk or posedge rst)begin
    if(rst) begin
        count <= 4'b0;
    end else begin
```

```
        count <= count + 1'b1;
    end
end

    endmodule
```

2、4-bit asynchronous up counter (四位异步加法计数器)

较高位以较低位的反相为时钟信号，控制信号逐级传递，每一级随本级控制时钟翻转，由此实现了异步加法计数器。

```
module md_counter_asyn_4bit(    // 4-bit asynchronous up counter (异步加法计数器)
    input wire clk,
    input wire rst,
    output reg [3:0] count
);

    // 异步计数器 "必须" 使用多个 always 块，因为每级时钟不同
    always@(posedge clk or posedge rst)begin
        if(rst)
            count[0]<=0;
        else
            count[0]<=~count[0];
    end
    always@(posedge ~count[0] or posedge rst)begin
        if(rst)
            count[1]<=0;
        else
            count[1]<=~count[1];
    end
    always@(posedge ~count[1] or posedge rst)begin
        if(rst)
            count[2]<=0;
        else
            count[2]<=~count[2];
    end
    always@(posedge ~count[2] or posedge rst)begin
        if(rst)
            count[3]<=0;
        else
            count[3]<=~count[3];
    end
```

```
endmodule
```

3、4-bit asynchronous down counter (四位同步减法计数器)

只需要将 4-bit synchronous up counter 的计数逻辑反过来即可：

```
module md_counter_down_syn_4bit(    // 4-bit synchronous down counter (同步减法计数器)
    input wire clk,
    input wire rst,
    output reg [3:0] count
);

    // 同步计数器使用单个 always 块，代码简洁清晰
    always@(posedge clk or posedge rst) begin
        if(rst)
            count <= 4'b1111;          // 复位到最大值(15)
        else
            count <= count - 4'b0001;  // 每个时钟周期减 1
    end

endmodule
```

4、编写 testbench 测试代码，完成三种计数器的行为级仿真

不妨同时测试三种计数器以方便对比，testbench 代码如下：

```
`timescale 1ns/1ps

`include "md_counter_syn_4bit.v"
`include "md_counter_asyn_4bit.v"
`include "md_counter_down_syn_4bit.v"

module tb_counter;
    // 时钟和复位信号
    reg clk;
    reg rst;

    // 三个计数器的输出
    wire [3:0] count_syn_up;
    wire [3:0] count_asyn_up;
    wire [3:0] count_syn_down;

    // 实例化三个计数器模块
```

```
    md_counter_syn_4bit u_syn_up(
        .clk(clk),
        .rst(rst),
        .count(count_syn_up)
    );

    md_counter_asyn_4bit u_asyn_up(
        .clk(clk),
        .rst(rst),
        .count(count_asyn_up)
    );

    md_counter_down_syn_4bit u_syn_down(
        .clk(clk),
        .rst(rst),
        .count(count_syn_down)
    );

    // 时钟生成：100MHz
    initial begin
        clk = 0;
        forever #5 clk = ~clk;  // 10ns 周期
    end

    // 测试过程
    initial begin
        // 初始化
        rst = 1;

        // 生成 VCD 文件用于波形查看
        $dumpfile("counter_wave.vcd");
        $dumpvars(0, tb_counter);

        // 打印表头
        $display("================================================================
======================");
        $display("Time(ns) | Sync_Up_Dec | Sync_Up_Bits | Async_Up_Dec |
Async_Up_Bits | Sync_Down_Dec | Sync_Down_Bits");
        $display("================================================================
======================");

        // 复位
        #20;
        rst = 0;

        // 运行足够多的时钟周期来观察行为
```

```
        #200;  // 20 个时钟周期

        // 再次复位测试
        rst = 1;
        #30;
        rst = 0;
        #100;

        $display("================================================================
====================");
        $finish;
    end

    // 实时显示计数器值
    integer last_syn_up = -1;
    integer last_asyn_up = -1;
    integer last_syn_down = -1;

    always @(posedge clk) begin
        // 只在值变化时显示，避免重复信息
        if (last_syn_up !== count_syn_up || last_asyn_up !== count_asyn_up ||
last_syn_down !== count_syn_down) begin
            $display("%8t | %11d |    %4b     | %12d |     %4b      | %13d
|      %4b",
                     $time,
                     count_syn_up, count_syn_up,
                     count_asyn_up, count_asyn_up,
                     count_syn_down, count_syn_down);

            last_syn_up = count_syn_up;
            last_asyn_up = count_asyn_up;
            last_syn_down = count_syn_down;
        end
    end

    // 监控异步计数器的纹波效应
    always @(count_asyn_up[0]) begin
        $display("          [Async Bit0 changed at %t ns]", $time);
    end

    always @(count_asyn_up[1]) begin
        $display("          [Async Bit1 changed at %t ns]", $time);
    end

    always @(count_asyn_up[2]) begin
        $display("          [Async Bit2 changed at %t ns]", $time);
```

```
    end

    always @(count_asyn_up[3]) begin
        $display("          [Async Bit3 changed at %t ns]", $time);
    end

    endmodule
```

行为级仿真结果如下：

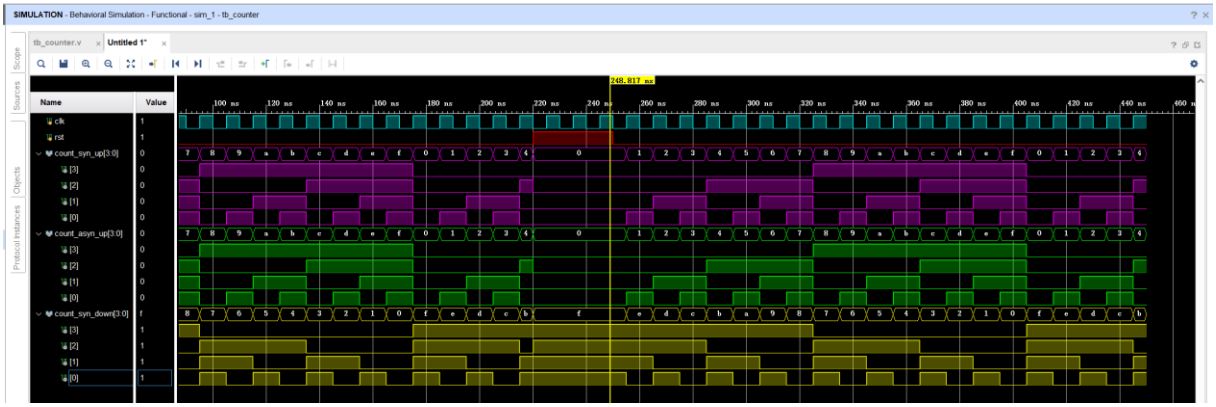

图 1. 三种计数器模块的行为级仿真结果

图中可以看出，三种计数器的计数和复位均正常工作。具体而言，随着时钟上升沿出现，计数器数值按照预期增大/减小，任意时刻打开复位 RST，计数器数值被复位到 0（加法计数器）或 F（减法计数器）。

5、设置好 constraints 文件，导出 bitstream 文件烧录至板上进行验证

基于三种计数器，分别生成三个 birstream 文件，下载至 FPGA，效果如下：

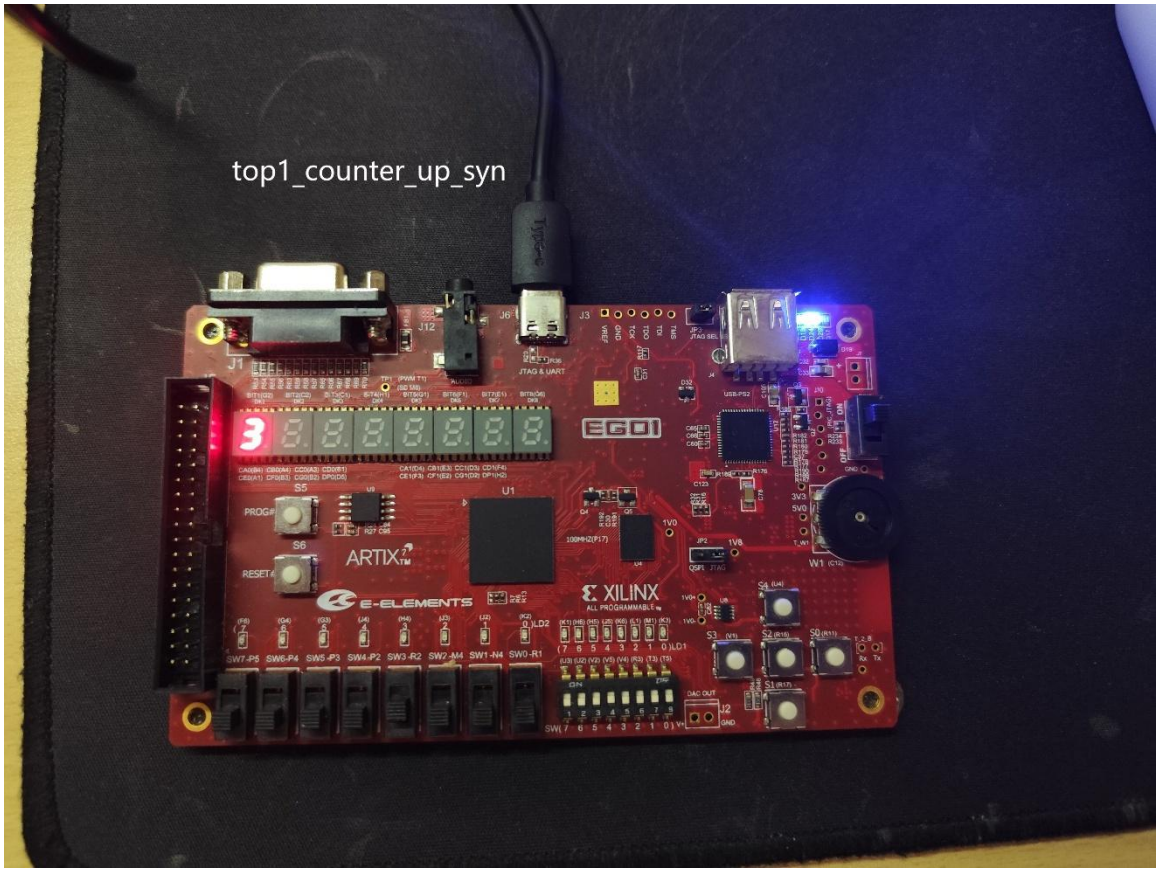

图 2. (1) 4-bit synchronous up counter (四位同步加法计数器) 实际测试结果

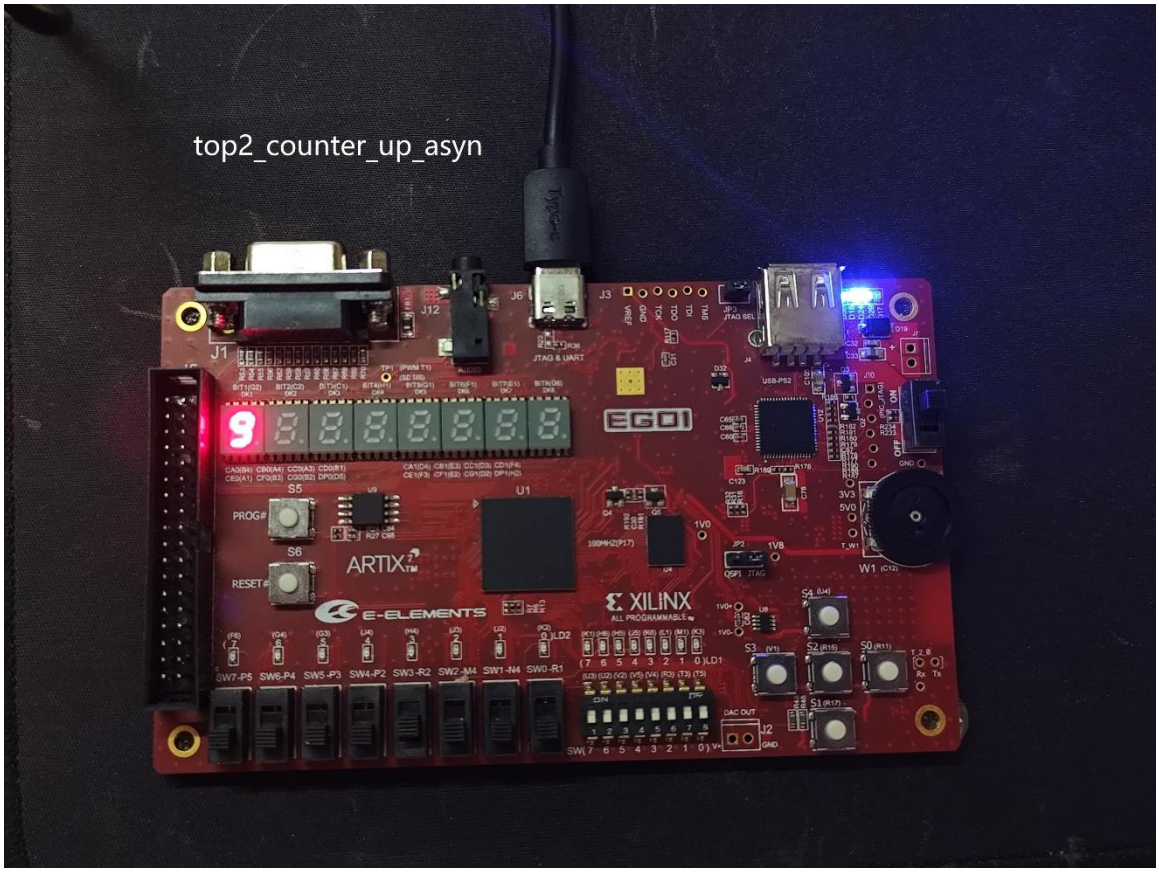

图 3. (2) 4-bit asynchronous up counter (四位异步加法计数器) 实际测试结果

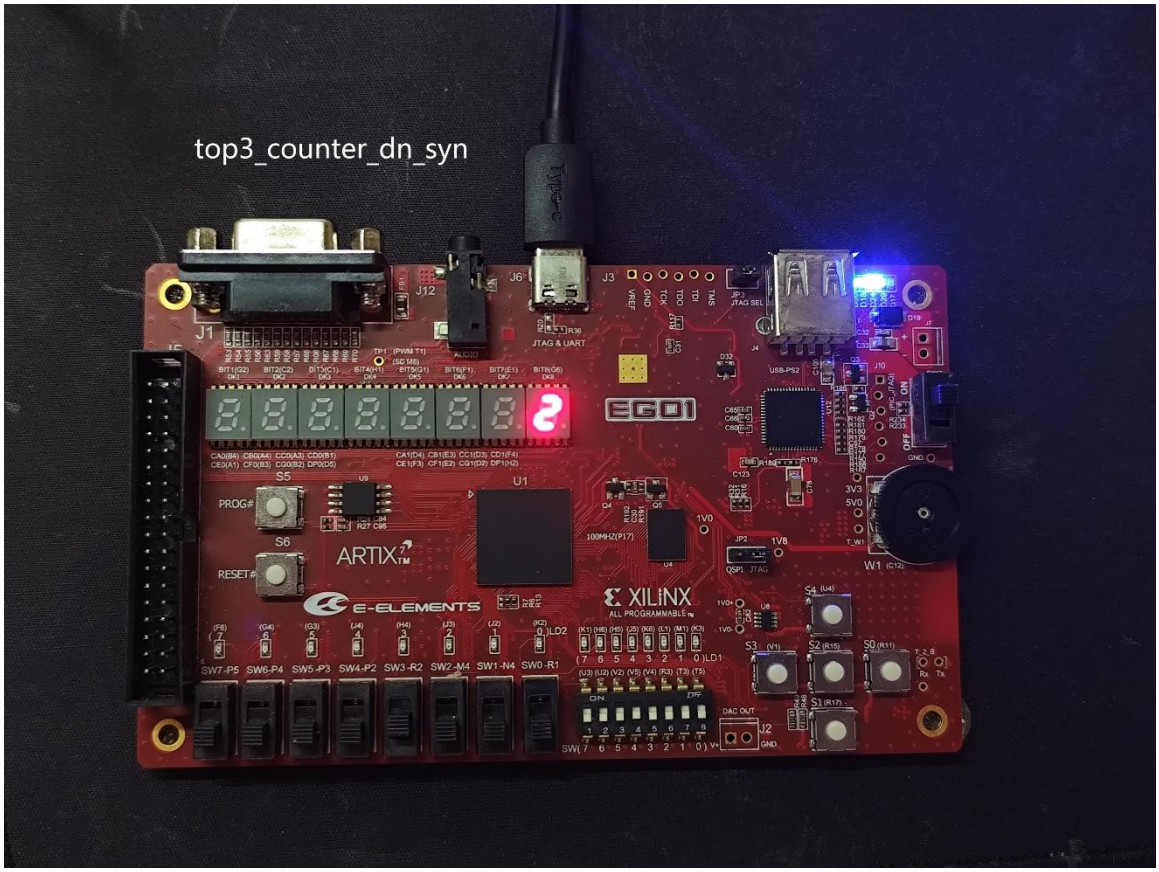

图 4. (3) 4-bit asynchronous down counter (四位同步减法计数器) 实际测试结果

经过测试，三种计数器的计数、复位功能均正常。具体现象为：按下 S2 (R15) 按键后，计数器数值按照预期增大/减小，任意时刻打开 SW0-R1 复位开关，计数器数值被复位到 0 (加法计数器) 或 F (减法计数器)。

6 思考题及结论

(1) 同为加法计数器，同步与异步的区别是什么？

同步计数器与异步计数器的核心区别在于其时钟信号的分配方式与由此产生的工作时序特性。异步计数器采用了一种串行或行波式的时钟机制，即只有最低位的触发器直接接收外部的计数脉冲，而高位触发器的时钟则来源于其相邻低位的输出，这导致各级触发器的状态翻转并非在同一时刻发生，而是像多米诺骨牌一样从低位到高位依次传递。这种工作方式使得异步计数器的总传播延迟是各级触发器延迟的累积，其计数速度会随着位数的增加而显著下降，并且在暂态过程中可能产生短暂的、非预期的毛刺状态。

相比之下，同步计数器则采用了并行时钟结构，所有触发器的时钟输入端都连接到同一个外部计数脉冲上。这意味着，当有效的时钟边沿到来时，所有触发器都会同时检查其数据输入端的当前值（该值由前级的组合逻辑决定），并据此决定是否翻转。由于所有位的更新是同步进行的，其最终输出状态几乎在同一时刻建立，因此计数器的总延迟仅等于单个触发器的翻转延迟加上一层组合逻辑的延迟，与计数器的位数无关，从而实现了更高的工作速度和可靠的输出，避免了异步计数器中常见的毛刺现象。

但是，在实际实验中我们并没有观察到明显的由异步传输控制信号导致的明显延迟，可能是因为计数器只有四位，产生的控制延迟过小，现象不明显。

(2) 防抖模块的作用：

实验中，我们还尝试了将防抖模块取消然后烧录进行测试，发现常常出现按一次按钮，数字却跳动两次甚至更多次的情况。相比之下，使用防抖模块来计数时，其计数精准度和稳定性大大提高。