静态数码管显示与动态数码管显示实验报告

石禹

2023K8009926030

中国科学院大学

§ 1 小组成员与分工

- 成员: 石禹、许书闻、潘瀚霖
- 分工: 为保证每个小组成员均能够完整地对整个实验内容进行学习和理解，本小组的所有成员均分别独立地完成了所有的实验内容，包括了代码编写、调试和实验报告的撰写。

§ 2 实验目的

- 使用 FPGA 开发板上的 6 位数码管以静态方式依次显示 000000、111111、222222 至 FFFFFF，结束后继续从 000000 开始计数，每 0.5s 变化一次。
- 使用 FPGA 开发板上的 6 位数码管以动态方式从 0 开始计数，每 100ms 计数值增加一，当计数值从 0 增加到 999999 后重新从 0 开始计数。
- 完成拓展部分实验

§ 3 静态数码管显示实验

3.1 硬件设计

新起点 FPGA 开发板上有 6 位共阳数码管，需要注意的是，为了增加 FPGA 输出信号的驱动能力，使用 PNP 型三极管驱动数码管的位选段，所以给三极管基极提供低电平时，位选信号为高电平。

实验中各端口信号的管脚分配如下表所示:

信号名	方向	管脚	端口说明
sys clk	input	M2	系统时钟，50M
sys rst n	input	M1	系统复位，低有效
seg sel[0]	output	N16	数码管位选 0
seg sel[1]	output	N15	数码管位选 1
seg sel[2]	output	P16	数码管位选 2
seg sel[3]	output	P15	数码管位选 3

seg sel[4]	output	R16	数码管位选 4
seg sel[5]	output	T15	数码管位选 5
seg led[0]	output	M11	数码管段选 a
seg led[1]	output	N12	数码管段选 b
seg led[2]	output	C9	数码管段选 c
seg led[3]	output	N13	数码管段选 d
seg led[4]	output	M10	数码管段选 e
seg led[5]	output	N11	数码管段选 f
seg led[6]	output	P11	数码管段选 g
seg led[7]	output	D9	数码管段选 h

为方便引脚分配，编写 TCL 脚本进行自动化引脚分配，脚本部分代码如下:

```
set_location_assignment PIN_M2 -to sys_clk
set_location_assignment PIN_M1 -to sys_rst_n
set_location_assignment PIN_N16 -to sel[0]
set_location_assignment PIN_N15 -to sel[1]
set_location_assignment PIN_P16 -to sel[2]
set_location_assignment PIN_P15 -to sel[3]
set_location_assignment PIN_R16 -to sel[4]
set_location_assignment PIN_T15 -to sel[5]
set_location_assignment PIN_M11 -to seg_led[0]
set_location_assignment PIN_N12 -to seg_led[1]
set_location_assignment PIN_C9 -to seg_led[2]
set_location_assignment PIN_N13 -to seg_led[3]
set_location_assignment PIN_M10 -to seg_led[4]
set_location_assignment PIN_N11 -to seg_led[5]
set_location_assignment PIN_P11 -to seg_led[6]
set_location_assignment PIN_D9 -to seg_led[7]
```

3.2 程序设计

根据实验任务，首先需要一个静态数码管显示模块在数码管上显示数据，其次需要一个计时模块每当计时到 0.5s 时改变数码管显示的数值。为方便代码的调试与编辑，选择编写一个顶层模块与两个子模块，顶层模块负责连接子模块与 FPGA 开发板的 IO 口，子模块负责分别实现系统时钟计数与控制数码管显示的功能。

顶层模块代码如下:

```
module seg_led_static_top (
    input         sys_clk  ,
    input         sys_rst_n,

    output [5:0] seg_sel  ,
    output [7:0] seg_led
);

localparam MAX_25M = 25_000_000;

reg [24:0] max_num;

always @ (posedge sys_clk or negedge sys_rst_n) begin
    if (!sys_rst_n)
        max_num <= MAX_25M;
    else
        max_num <= max_num;
end

wire add_flag;

time_count u_time_count (
    .clk         (sys_clk ),
    .rst_n       (sys_rst_n),
    .max_num     (max_num),
```

```
    .add_flag   (add_flag )
);

seg_led_static u_seg_led_static (
    .clk        (sys_clk ),
    .rst_n      (sys_rst_n),
    .add_flag   (add_flag ),
    .seg_sel    (seg_sel  ),
    .seg_led    (seg_led  )
);

endmodule
```

在 FPGA 顶层 (seg led static top) 例化了以下两个模块: 计时模块 (time count) 和数码管静态显示模块 (seg led static)，实现各模块之间数据的交互。计时模块将计时到 0.5s 时的标志信号 flag 传递给数码管静态显示模块，数码管静态显示模块接收到此信号时显示的数值增加 1。

计时模块代码如下:

```
module time_count (
    input           clk       ,
    input           rst_n     ,
    input  [24:0]   max_num   ,
    output reg      add_flag
);

reg [24:0] cnt;

always @ (posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        add_flag <= 1'b0;
        cnt      <= 25'd0;
    end
    else if (cnt < max_num - 1'b1) begin
        cnt      <= cnt + 1'b1;
```

```
        add_flag <= 1'b0;
    end
    else begin
        cnt      <= 25'd0;
        add_flag <= 1'b1;
    end
end

endmodule
```

数码管静态显示模块的代码如下:

```
module seg_led_static (
    input       clk     ,
    input       rst_n   ,

    input       add_flag,
    output reg [5:0] seg_sel,
    output reg [7:0] seg_led
);

reg [3:0] num;

always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)
        seg_sel <= 6'b111111;
    else
        seg_sel <= 6'b000000;
end

always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)
        num <= 4'h0;
    else if (add_flag)
        if (num < 4'hf)
```

```
            num <= num + 1'b1;
        else
            num <= 4'h0;
    else
        num <= num;
end

always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)
        seg_led <= 8'b0;
    else begin
        case (num)
            4'h0: seg_led <= 8'b11000000;
            4'h1: seg_led <= 8'b11111001;
            4'h2: seg_led <= 8'b10100100;
            4'h3: seg_led <= 8'b10110000;
            4'h4: seg_led <= 8'b10011001;
            4'h5: seg_led <= 8'b10010010;
            4'h6: seg_led <= 8'b10000010;
            4'h7: seg_led <= 8'b11111000;
            4'h8: seg_led <= 8'b10000000;
            4'h9: seg_led <= 8'b10010000;
            4'ha: seg_led <= 8'b10001000;
            4'hb: seg_led <= 8'b10000011;
            4'hc: seg_led <= 8'b11000110;
            4'hd: seg_led <= 8'b10100001;
            4'he: seg_led <= 8'b10000110;
            4'hf: seg_led <= 8'b10001110;
            default: seg_led <= 8'b00000000;
        endcase
    end
end
```

endmodule

至此，静态数码管显示模块的设计完成。该模块在 FPGA 开发板上运行时，数码管将依次显示 000000、111111、222222 至 FFFFFF，结束后继续从 000000 开始计数，每 0.5s 变化一次。

3.3 实验结果

实验结果展示视频链接如下: `https://www.bilibili.com/video/BV1vYExzaEDu`

§ 4 静态数码管显示实验 (拓展部分)

4.1 拓展实验要求

1. 将数字切换的时间间隔从 0.5s 改为 0.25s，并固化程序到开发板中，实现断开下载器、电源后重启仍可实现静态数码管显示。
2. 实现按键 1 控制时间间隔为 0.5s，按键 2 控制时间间隔为 0.25s。

4.2 拓展实验 1

要实现时间间隔的改变，只需要修改顶层模块中的 MAX COUNT 的值即可。

修改后的顶层模块代码如下:

```
localparam MAX_12_5M = 12_500_000;

reg [24:0] max_num;

always @ (posedge sys_clk or negedge sys_rst_n) begin
    if (!sys_rst_n)
        max_num <= MAX_12_5M;
    else
        max_num <= max_num;
end
```

该模块在 FPGA 开发板上运行时，数码管将依次显示 000000、111111、222222 至 FFFFFF，结束后继续从 000000 开始计数，每 0.25s 变化一次。

实验结果展示视频链接如下: https://www.bilibili.com/video/BV1vYExzhEUr

4.3 拓展实验 2

要实现利用按键控制时间间隔的功能，需要在顶层模块中增加按键输入信号，并在计时模块中增加对按键信号的判断。由于加入的新的按键信号，需要对引脚进行重新分配，修改后的引脚分配如下表所示:

信号名	方向	管脚	端口说明
sys clk	input	M2	系统时钟，50M
sys rst n	input	M1	系统复位，低有效
seg sel[0]	output	N16	数码管位选 0
seg sel[1]	output	N15	数码管位选 1
seg sel[2]	output	P16	数码管位选 2
seg sel[3]	output	P15	数码管位选 3
seg sel[4]	output	R16	数码管位选 4
seg sel[5]	output	T15	数码管位选 5
seg led[0]	output	M11	数码管段选 a
seg led[1]	output	N12	数码管段选 b
seg led[2]	output	C9	数码管段选 c
seg led[3]	output	N13	数码管段选 d
seg led[4]	output	M10	数码管段选 e
seg led[5]	output	N11	数码管段选 f
seg led[6]	output	P11	数码管段选 g
seg led[7]	output	D9	数码管段选 h
key1	input	E16	按键 1
key2	input	E15	按键 2

只需在 TCL 脚本中添加如下两行代码:

```
set_location_assignment PIN_E16 -to key1
set_location_assignment PIN_E15 -to key2
```

至此硬件设计部分的修改完成，运行 TCL 脚本后，FPGA 开发板的引脚分配完成。

要实现两个按键能够切换两种不同的时间间隔，需要预先设置两个时间间隔的值，并在计时模块中增加对按键信号的判断。修改后的顶层模块代码如下:

```
module seg_led_static_top (
    input       sys_clk  ,
    input       sys_rst_n,
    input       key1     ,
    input       key2     ,

    output [5:0] seg_sel ,
    output [7:0] seg_led
);

localparam MAX_25M = 25_000_000;
localparam MAX_12_5M = 12_500_000;

reg [24:0] max_num;

always @ (posedge sys_clk or negedge sys_rst_n) begin
    if (!sys_rst_n)
        max_num <= MAX_25M;
    else if (~key1)
        max_num <= MAX_25M;
    else if (~key2)
        max_num <= MAX_12_5M;
    else
        max_num <= max_num;
end
```

该模块在 FPGA 开发板上运行时，数码管将依次显示 000000、111111、222222 至 FFFFFF，结束后继续从 000000 开始计数，且按下按键 1 后，数码管每 0.5s 变化一次；按下按键 2 后，数码管每 0.25s 变化一次。

实验结果展示视频链接如下: https://www.bilibili.com/video/BV19qExzZEPw

§ 5 动态数码管显示实验

本部分实验的硬件设计与静态数码管显示实验相同，引脚分配也相同，故不再赘述。

5.1 程序设计

若要让 6 个数码管轮流显示对应的数字，首先需要一个数码管动态显示模块，能够依次点亮 6 个数码管，并将对应的数据输出至数码管，也就是需要分别控制段选和位选信号; 同时还需要一个计数模块，能够将 0—999999 依次输出至数码管动态显示模块。

与静态数码管显示实验一样，选择编写一个顶层模块与两个子模块，顶层模块负责连接子模块与 FPGA 开发板的 IO 口，子模块负责分别实现系统时钟计数与控制数码管显示的功能。

顶层模块代码如下:

```
module top_seg_led(
        input sys_clk,
        input sys_rst_n,
        output [5:0] seg_sel,
        output [7:0] seg_led
);

wire [19:0] data;
wire [5:0] point;
wire en;
wire sign;

count u_count(
        .clk(sys_clk),
        .rst_n(sys_rst_n),
        .data(data),
        .point(point),
        .en(en),
        .sign(sign)
```

```
);

seg_led u_seg_led(
        .clk(sys_clk),
        .rst_n(sys_rst_n),
        .data(data),
        .point(point),
        .en(en),
        .sign(sign),
        .seg_sel(seg_sel),
        .seg_led(seg_led)
);

endmodule
```

计数模块代码如下:

```
module count(
        input clk,
        input rst_n,
        input key0,
        input key1,
        output reg [19:0] data,
        output reg [5:0] point,
        output reg en,
        output reg sign
);

parameter MAX_NUM = 23'd5000_000;

reg [22:0] cnt;
reg flag;

always @ (posedge clk or negedge rst_n) begin
        if (!rst_n)begin
```

```
                cnt <= 23'b0;
                flag <= 1'b0;
        end
        else if (cnt < MAX_NUM - 1'b1) begin
                cnt <= cnt + 1'b1;
                flag <= 1'b0;
        end
        else begin
                cnt <= 23'b0;
                flag <= 1'b1;
        end
end

always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)begin
        data  <= 20'b0;
        point <=6'b000000;
        en    <= 1'b0;
        sign  <= 1'b0;
    end
    else begin
        point <= 6'b000000;
        en    <= 1'b1;
        sign  <= 1'b0;
        if (flag) begin
            if(data < 20'd999999)
                data <= data +1'b1;
            else
                data <= 20'b0;
        end
    end
end
```

endmodule

数码管动态显示模块的代码如下:

```
module seg_led(
    input                   clk     ,
    input                   rst_n   ,

    input           [19:0]  data    ,
    input           [5:0]   point   ,
    input                   en      ,
    input                   sign    ,

    output  reg     [5:0]   seg_sel ,
    output  reg     [7:0]   seg_led
    );

//parameter define
localparam  CLK_DIVIDE = 4'd10      ;
localparam  MAX_NUM    = 13'd5000   ;

//reg define
reg     [ 3:0]              clk_cnt  ;
reg                         dri_clk  ;
reg     [23:0]              num      ;
reg     [12:0]              cnt0     ;
reg                         flag     ;
reg     [2:0]               cnt_sel  ;
reg     [3:0]               num_disp ;
reg                         dot_disp ;

//wire define
wire    [3:0]               data0    ;
wire    [3:0]               data1    ;
wire    [3:0]               data2    ;
```

```
wire    [3:0]               data3    ;
wire    [3:0]               data4    ;
wire    [3:0]               data5    ;

assign  data0 = data % 4'd10;
assign  data1 = data / 4'd10 % 4'd10   ;
assign  data2 = data / 7'd100 % 4'd10  ;
assign  data3 = data / 10'd1000 % 4'd10 ;
assign  data4 = data / 14'd10000 % 4'd10;
assign  data5 = data / 17'd100000;

always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        clk_cnt <= 4'd0;
        dri_clk <= 1'b1;
    end
    else if(clk_cnt == CLK_DIVIDE/2 - 1'd1) begin
        clk_cnt <= 4'd0;
        dri_clk <= ~dri_clk;
    end
    else begin
        clk_cnt <= clk_cnt + 1'b1;
        dri_clk <= dri_clk;
    end
end

always @ (posedge dri_clk or negedge rst_n) begin
    if (!rst_n)
        num <= 24'b0;
    else begin
        if (data5 || point[5]) begin
            num[23:20] <= data5;
            num[19:16] <= data4;
```

```
        num[15:12] <= data3;
        num[11:8]  <= data2;
        num[ 7:4]  <= data1;
        num[ 3:0]  <= data0;
    end
    else begin
        if (data4 || point[4]) begin
            num[19:0] <= {data4,data3,data2,data1,data0};
            if(sign)
                num[23:20] <= 4'd11;
            else
                num[23:20] <= 4'd10;
        end
        else begin
            if (data3 || point[3]) begin
                num[15: 0] <= {data3,data2,data1,data0};
                num[23:20] <= 4'd10;
                if(sign)
                    num[19:16] <= 4'd11;
                else
                    num[19:16] <= 4'd10;
            end
            else begin
                if (data2 || point[2]) begin
                    num[11: 0] <= {data2,data1,data0};
                    num[23:16] <= {2{4'd10}};
                    if(sign)
                        num[15:12] <= 4'd11;
                    else
                        num[15:12] <= 4'd10;
                end
                else begin
                    if (data1 || point[1]) begin
```

```
                                num[ 7: 0] <= {data1,data0};
                                num[23:12] <= {3{4'd10}};
                                if(sign)
                                    num[11:8]  <= 4'd11;
                                else
                                    num[11:8] <=  4'd10;
                            end
                            else begin
                                num[3:0] <= data0;
                                num[23:8] <= {4{4'd10}};
                                if(sign)
                                    num[7:4] <= 4'd11;
                                else
                                    num[7:4] <= 4'd10;
                            end
                        end
                    end
                end
            end
        end
    end

always @ (posedge dri_clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
        cnt0 <= 13'b0;
        flag <= 1'b0;
     end
    else if (cnt0 < MAX_NUM - 1'b1) begin
        cnt0 <= cnt0 + 1'b1;
        flag <= 1'b0;
     end
    else begin
        cnt0 <= 13'b0;
```

```
            flag <= 1'b1;
        end
    end

    always @ (posedge dri_clk or negedge rst_n) begin
        if (rst_n == 1'b0)
            cnt_sel <= 3'b0;
        else if(flag) begin
            if(cnt_sel < 3'd5)
                cnt_sel <= cnt_sel + 1'b1;
            else
                cnt_sel <= 3'b0;
        end
        else
            cnt_sel <= cnt_sel;
    end

    always @ (posedge dri_clk or negedge rst_n) begin
        if(!rst_n) begin
            seg_sel  <= 6'b111111;
            num_disp <= 4'b0;
            dot_disp <= 1'b1;
        end
        else begin
            if(en) begin
                case (cnt_sel)
                    3'd0 :begin
                        seg_sel  <= 6'b111110;
                        num_disp <= num[3:0] ;
                        dot_disp <= ~point[0];
                    end
                    3'd1 :begin
                        seg_sel  <= 6'b111101;
```

```
                num_disp <= num[7:4] ;
                dot_disp <= ~point[1];
            end
            3'd2 :begin
                seg_sel  <= 6'b111011;
                num_disp <= num[11:8];
                dot_disp <= ~point[2];
            end
            3'd3 :begin
                seg_sel  <= 6'b110111;
                num_disp <= num[15:12];
                dot_disp <= ~point[3];
            end
            3'd4 :begin
                seg_sel  <= 6'b101111;
                num_disp <= num[19:16];
                dot_disp <= ~point[4];
            end
            3'd5 :begin
                seg_sel  <= 6'b011111;
                num_disp <= num[23:20];
                dot_disp <= ~point[5];
            end
            default :begin
                seg_sel  <= 6'b111111;
                num_disp <= 4'b0;
                dot_disp <= 1'b1;
            end
        endcase
    end
    else begin
        seg_sel  <= 6'b111111;
        num_disp <= 4'b0;
```

```
            dot_disp <= 1'b1;
        end
    end
end

always @ (posedge dri_clk or negedge rst_n) begin
    if (!rst_n)
        seg_led <= 8'hc0;
    else begin
        case (num_disp)
            4'd0 : seg_led <= {dot_disp,7'b1000000};
            4'd1 : seg_led <= {dot_disp,7'b1111001};
            4'd2 : seg_led <= {dot_disp,7'b0100100};
            4'd3 : seg_led <= {dot_disp,7'b0110000};
            4'd4 : seg_led <= {dot_disp,7'b0011001};
            4'd5 : seg_led <= {dot_disp,7'b0010010};
            4'd6 : seg_led <= {dot_disp,7'b0000010};
            4'd7 : seg_led <= {dot_disp,7'b1111000};
            4'd8 : seg_led <= {dot_disp,7'b0000000};
            4'd9 : seg_led <= {dot_disp,7'b0010000};
            4'd10: seg_led <= 8'b11111111;
            4'd11: seg_led <= 8'b10111111;
            default:
                   seg_led <= {dot_disp,7'b1000000};
        endcase
    end
end

endmodule
```

5.2 实验结果

将程序下载到开发板上后，6 位数码管将以动态方式从 0 开始计数，每 100ms 计数值增加一，当计数值从 0 增加到 999999 后重新从 0 开始

计数。实验结果展示视频链接如下: https://www.bilibili.com/video/BV19qExzZET3

§ 6 动态数码管显示实验 (拓展部分)

6.1 拓展实验要求

修改原实验的按键逻辑，使其实现如下功能:Key0 可以暂停计数，Key1 可以改变计数加减模式 (到 0 时不再减小)，Key2 可以清零计数。

6.2 拓展实验

要实现拓展实验中的要求，首先需要新增按键输入信号，并且对引脚进行重新分配，修改后的 TCL 脚本如下:

```
# 时钟与复位引脚
set_location_assignment PIN_M2 -to sys_clk
set_location_assignment PIN_M1 -to sys_rst_n

# 数码管位选信号 seg_sel[5:0]
set_location_assignment PIN_N16 -to seg_sel[0]
set_location_assignment PIN_N15 -to seg_sel[1]
set_location_assignment PIN_P16 -to seg_sel[2]
set_location_assignment PIN_P15 -to seg_sel[3]
set_location_assignment PIN_R16 -to seg_sel[4]
set_location_assignment PIN_T15 -to seg_sel[5]

# 数码管段选信号 seg_led[7:0]
set_location_assignment PIN_M11 -to seg_led[0]
set_location_assignment PIN_N12 -to seg_led[1]
set_location_assignment PIN_C9  -to seg_led[2]
set_location_assignment PIN_N13 -to seg_led[3]
set_location_assignment PIN_M10 -to seg_led[4]
set_location_assignment PIN_N11 -to seg_led[5]
set_location_assignment PIN_P11 -to seg_led[6]
```

```
set_location_assignment PIN_D9  -to seg_led[7]

# 按键信号
set_location_assignment PIN_E16 -to key[0]
set_location_assignment PIN_E15 -to key[1]
set_location_assignment PIN_M15 -to key[2]
```

除此之外，还需要定义新的变量用于存储计数的暂停状态和加减模式状态。并且新增一个计数清零的功能。由于引入了新的按键信号，需要同时在顶层模块和计数模块中对信号进行申明。

修改后的顶层模块代码如下:

```
module top_seg_led(
        input sys_clk,
        input sys_rst_n,
        input [2:0] key,
        output [5:0] seg_sel,
        output [7:0] seg_led
);

wire [19:0] data;
wire [5:0] point;
wire en;
wire sign;

count u_count(
        .clk(sys_clk),
        .rst_n(sys_rst_n),
        .data(data),
        .point(point),
        .en(en),
        .sign(sign),
        .key (key),
);
```

```
seg_led u_seg_led(
        .clk(sys_clk),
        .rst_n(sys_rst_n),
        .data(data),
        .point(point),
        .en(en),
        .sign(sign),
        .seg_sel(seg_sel),
        .seg_led(seg_led)
);

endmodule
```

修改后的计数模块代码如下:

```
module count (
    input         clk,
    input         rst_n,
    input  [2:0]  key,
    output reg [19:0] data,
    output reg [5:0]  point,
    output reg        en,
    output reg        sign
);

parameter MAX_NUM = 23'd5_000_000;

reg  [22:0] cnt;
reg         flag;
reg  [2:0]  key_last;
reg         pause;
reg         add;

wire [2:0] key_pressed;
assign key_pressed = (~key) & key_last;
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        key_last <= 3'b111;
    else
        key_last <= key;
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        pause <= 1'b1;
        add   <= 1'b1;
    end
    else begin
        if (key_pressed[0]) begin
            pause <= ~pause;
        end
        else if (key_pressed[1]) begin
            add <= ~add;
        end
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        cnt  <= 23'b0;
        flag <= 1'b0;
    end
    else if (key_pressed[2]) begin
        cnt  <= 23'b0;
        flag <= 1'b0;
    end
    else begin
```

```
        if (!pause) begin
            cnt  <= cnt;
            flag <= 1'b0;
        end
        else begin
            if (cnt < MAX_NUM - 1'b1) begin
                cnt  <= cnt + 1'b1;
                flag <= 1'b0;
            end
            else begin
                cnt  <= 23'b0;
                flag <= 1'b1;
            end
        end
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        data  <= 20'b0;
        point <= 6'b000000;
        en    <= 1'b0;
        sign  <= 1'b0;
    end
    else if (key_pressed[2]) begin
        data <= 20'b0;
    end
    else begin
        point <= 6'b000000;
        en    <= 1'b1;
        sign  <= 1'b0;
        if (flag) begin
            if (add) begin
```

```
                if (data < 20'd999999)
                    data <= data + 1'b1;
                else
                    data <= 20'b0;
            end
            else begin
                if (data > 20'b0)
                    data <= data - 1'b1;
                else
                    data <= 20'b0;
            end
        end
    end
end

endmodule
```

将程序固化到开发板上后，按下按键 0 可以暂停计数，按下按键 1 可以改变计数加减模式 (到 0 时不再减小)，按下按键 2 可以清零计数。

实验结果展示视频链接如下: https://www.bilibili.com/video/BV1YiExzbEQ6

§ 7 实验总结与感想

本次实验通过对静态数码管显示实验和动态数码管显示实验的设计与实现，深入理解了数码管的工作原理和 FPGA 的编程方法。在实验中，使用 Verilog HDL 语言编写了数码管的控制模块和计数模块，并通过 TCL 脚本实现了引脚的自动化分配。本次实验给我带来最大的收获是学会了如何将不同的模块分别在不同的文件中编写，并在顶层模块中进行连接与调用。这一方法大大提高了代码的可读性和可维护性。

在静态数码管实验部分，由于代码较为简单，且拓展内容修改较为容易，因此在实验过程中并没有遇到太大的问题。但是，由于动态数码管实验部分需要涉及数字的各位数值的分离与组合，在编写代码时需要特别注意数据的位宽和赋值方式，避免出现数据溢出或赋值错误的问题。并且在这一部分的拓展实验中需要增加新的按键输入信号，同时对 TCL 脚本进行修改，给

编程带来一定的挑战性。由于对 Verilog 语言的不够熟练，导致我在开始的实验过程中，仅仅在 count 模块中对按键信号进行了申明，而没有在顶层模块中进行申明，导致程序下载到开发板后，数码管的示数始终为 0，后来通过对代码的详细阅读与理解，我及时发现了这个问题，并在顶层模块中对按键信号进行了申明，最终成功实现了拓展实验的要求。