Alinx AX7035t 开发板实验

——基于埃氏筛法的 6 位素数计算与显示

姜俊彦[1]，刘镇豪[2]，吴尚哲[3]

目录

[1]姜俊彦，中国科学院大学，2022K8009970011，jiangjunyan22@mails.ucas.ac.cn
[2]刘镇豪，中国科学院大学，2022K8009929027，liuzhenhao22@mails.ucas.ac.cn
[3]吴尚哲，中国科学院大学，2022K8009909015，wushangzhe@mails.ucas.ac.cn

1 实验题目

素数循环显示: 利用 Alinx AX7035t 开发板，使用 Verilog 编程实现计算并显示 2-999999 之间的素数。6 个七段数码管显示素数，4 个 LED 显示当前模式，4 个按钮选择循环模式

- 按键 1：递增，每秒变一次（上电之后的默认模式）
- 按键 2：递减，每秒变一次
- 按键 3：递增，最快速度
- 按键 4：递减，最快速度

注意事项：

1. 不可以提前将素数计算好存储在 FPGA 内，必须运行时计算
2. 每个小组独立完成，提交一份源码, 一份说明文档，均使用 PDF 电子版提交
3. 组内成员最终共享大作业成绩，不做区别对待

2 实验过程

2.1 实验平台

- 操作系统：Windows 11 专业版 22H2
- 开发平台：Vivado v2023.1(64-bit)
- 硬件平台：XLINX ARTIX-7 系列 AX7035 FPGA 开发平台
- 协作平台：GitHub

2.2 算法调研及选择

笔者所在小组首先对于素数计算的已有算法进行了理论调研，调研结果有两种主流思路：循环取余（除法）判断法和筛法。

循环取余判断法的思路是遍历小于待判断数 N 的所有数，判断这些数中有无 N 的因子，若无则为质数。其一种 C 代码实现如下所示

```
#include <stdio.h>
void main()
{
for(int i=2;i<=1000000;i++) {
   int flag = 0;
   for(int j=2;j<i;j++)
      if(i % j == 0 )
         flag = 1;
   if(flag==0)
      printf("Prime:%d\n",i)
}
}
```

对于该算法的改进思路有诸如加入退出条件，调整遍历范围为 $[0, \sqrt{N}]$ 等，可以优化算法时间复杂度。

相较于循环取余（除法）判断法，筛法具有更优的时间复杂度。筛法的一种 C 实现如下所示（本次实验使用的算法）

```c
#include <stdio.h>
#define MAXSIZE 100000
int main()
{
   int num[MAXSIZE];
   for (int i = 0; i < MAXSIZE; i++)
      num[i] = 1;
   for (int i = 2; i * i <= MAXSIZE; i++)
      if (num[i])
         for (int j = i + i; j < MAXSIZE; j += i)
            num[j] = 0;
   return 0;
}
```

上述算法为埃氏筛法，其使用一个位图数据结构来标记一个数是否为素数。其算法流程大致为：自最小素数 2 开始，将 2 的所有倍数标记为合数，然后 2 自增为 3，判断 3 是否为标记合数，没有则重复上述过程，依次类推。

对于上述算法的改进可以优化为欧式筛法（线性筛），其时间复杂度为 $O(n)$ 级别。

笔者所在小组针对上述算法的优劣展开了讨论。由于 Verilog 对于除法的实现采用的是循环减法的方式，单次除法需要的运算时间相较于单次加、减、乘法运算要长很多，故通过比较上述两种主流计算思路，我们选择了后者的筛法思路。欧式筛法虽然具有更优秀的时间复杂度表现，但是其在计算过程中需要对存储素数的数组调用两次，且其过程中仍要实现取余操作。综合考虑下欧式筛法的实现难度对于笔者所在小组有些困难（尝试编写过 verilog 程序无果），故我们最终选择实现埃氏筛法。

2.3 组员分工

- 姜俊彦：本次担任统筹工作，负责 Git 库管理、任务分发、代码整理与测试、文档撰写。
- 刘镇豪：本次负责模式选择、LED 控制、按键除抖、脉冲输出、1 秒计时代码的编写。
- 吴尚哲：本次负责 LED 显示、译码器代码及注释的编写。

2.4 开发日志

在项目开发过程中建立了开发日志（见附录 2）

3 实验代码

3.1 LED 灯控制模块-ledcontrol.v

```
module ledcontrol
(
   input           clk,
   input           rstn_signal,
   input  [3:0]    key_pulse,
   output [3:0]    led
);
reg    [3:0]    led_r;
always @(posedge clk or negedge rstn_signal) begin
   if(~rstn_signal) begin
   led_r <= 4'b1110;
end
else if(~key_pulse[0]) begin
   led_r <= 4'b1110;
end
else if(~key_pulse[1]) begin
   led_r <= 4'b1101;
end
else if(~key_pulse[2]) begin
   led_r <= 4'b1011;
end
else if(~key_pulse[3]) begin
   led_r <= 4'b0111;
end
end
assign led = led_r;
endmodule
//LED灯控制模块
//输入：按键脉冲(key_pulse)
//输出：LED灯状态(led)
//功能：按键按下，对应的灯亮起
```

3.2 模式选择-modecontrol.v

```
module modecontrol
(
   input           clk,
   input           rstn_signal,
   input           one_second,
   input [3:0]     led,
   input [3:0]     key_pulse,
   output          reset,
   output          select,
   output          tick
);
reg                tick_r;
reg                select_reg;
reg  [1:0]         reset_r;
always @(posedge clk) begin
   if(~led[0]) begin
      tick_r <= one_second;
      select_reg<=1;
   end
   else if(~led[1])begin
      tick_r <= one_second;
      select_reg<=0;
   end
   else if(~led[2])begin
      tick_r <= 1;
      select_reg<=1;
   end
   else if(~led[3])begin
      tick_r <= 1;
      select_reg<=0;
   end
   else begin
      tick_r<=0;
   end
end
always @(posedge clk) begin
   reset_r<={reset_r[0],(&key_pulse)&rstn_signal};
end//reset信号需要滞后一个周期，以避免同周期复位与传值的冲突
assign tick = tick_r;
assign select=select_reg;
assign reset=reset_r[1];
endmodule
//显示模式选择
//输入：一秒脉冲(one_second)，LED灯状态(LED)，按键脉冲(key_pulse)
//输出：计算复位信号(reset)，增减选择信号(select)，显示间隔信号(tick)
//功能：按照LED灯的状态更改增减选择信号和显示间隔信号
//增减选择信号(select)：1为增，0为减；显示间隔信号(tick)：one_second为1秒间隔，1为最快
```

3.3 按键除抖-Killshake.v

```
module Killshake
(
   input   clk,key,
   output  signal
);
parameter   DEBOUNCE_TIME = 1000000;     // 去抖时间阈值，根据时钟频率调整,1/50s
reg [19:0]  count;                       // 计数器，位宽取决于DEBOUNCE_TIME
reg         key_state;                   // 存储稳定后的按键状态
reg         signal_reg;                  // always块中储存状态
always @(posedge clk) begin
   if (key == key_state) begin
      if (count < DEBOUNCE_TIME) count <= count + 1;
      else signal_reg <= key_state;
   end else begin
      count <= 0;
      key_state <= key;
   end
end
assign signal = signal_reg;
endmodule
//按键除抖
//输入：含噪声的按键输入(key)
//输出：过滤噪声的按键输入(signal)
//功能：将含噪声的按键输入过滤，输出无噪声的按键输入(低有效)[计数器实现]
```

3.4 边沿检测-Edgedetect.v

```
module Edgedetect
(
   input   key,clk
   output  pulse
);
reg     key_prev;
reg     pulse_reg;
always @(posedge clk) begin
   key_prev <= key;
   if (key_prev & ~key) pulse_reg <= 0;
   else pulse_reg <= 1;
end
assign pulse = pulse_reg;
endmodule
//边沿检测
//输入：过滤噪声的按钮信号(key)
//输出：按钮脉冲信号(pulse)
//功能：将过滤噪声的按钮信号转换为按钮脉冲信号
```

3.5 数码管驱动 seg_driver.v

```
module seg_driver #(parameter NPorts=8)
(
   input                      clk,rstn,
   input [NPorts-1:0]         valid_i,
   input [NPorts*8-1:0]       seg_i,
   output reg [NPorts-1:0]    valid_o,
   output [7:0]               seg_o
);

reg [14:0]          cnt;
reg [NPorts-1:0]    sel;

always @(posedge clk or negedge rstn)
   if(~rstn) cnt <= 0;
   else cnt <= cnt + 1;

always @(posedge clk or negedge rstn)
   if(~rstn) sel <= 0;
   else if(cnt == 0) sel <= (sel == NPorts - 1) ? 0 : sel + 1;

always @(sel, valid_i) begin
   valid_o = {NPorts{1'b1}};
   valid_o[sel] = ~valid_i[sel];
end

assign seg_o = ~seg_i[sel*8+:8];
endmodule
//数码管驱动
//输入：使能输入线(valid_i)，数据输入线(seg_i)
//输出：使能输出线(valid_o)，数据输出线(seg_o)
//功能：刷新显示输入的八段数码管编码
```

3.6 十进制八段数码管译码器-led7seg_decode.v

```
module led7seg_decode
(
   input [3:0] digit,
   input valid,
   output reg [7:0] seg
);
always @(digit)
   if(valid)
   case(digit)
      0: seg = 8'b00111111;//0
      1: seg = 8'b00000110;//1
      2: seg = 8'b01011011;//2
      3: seg = 8'b01001111;//3
      4: seg = 8'b01100110;//4
      5: seg = 8'b01101101;//5
      6: seg = 8'b01111101;//6
      7: seg = 8'b00000111;//7
      8: seg = 8'b01111111;//8
      9: seg = 8'b01101111;//9
      default: seg = 0;
   endcase
   else seg = 8'd0;
endmodule
//十进制八段数码管译码器
//输入：三位二进制编码
//输出：十进制八段数码管编码
//功能：将三位二进制编码转换为十进制八段数码管编码[case实现]
```

3.7 埃氏筛法计算模块-isprime.v

```
module isprime #(parameter N=999999)
(
   input clk,rstn,tick,select,
   output [19:0] cnt_20b
);
reg [19:0]      w_addr;          //写入的数据的地址
reg             w_data;          //写入的数据
reg             wea;           //使能端
reg [19:0]      r_addr;          //读取的数据的地址
wire            r_data;          //读取的数据
reg [19:0]      i;               //外层循环变量
reg [19:0]      j;               //内层循环变量
reg             en;
reg             done;
reg [19:0]      cnt_temp_reg;
reg [19:0]      cnt_20b_reg;
reg [2:0]       timer;
reg             hold;
always @(posedge clk or negedge rstn) begin
   if(!rstn) begin
      cnt_temp_reg<=(select)?2:N;
      cnt_20b_reg<=(select)?2:N;
      wea<=0;
      i<=2;
      j<=0;
      en<=0;
      done<=0;
      timer<=0;
      hold<=1;
   end
   else if (i*i<=N) begin
      if(en==0)begin
         r_addr<=i;
         if(timer>2)begin timer<=0; hold<=0; end
         else begin timer<=timer+1; hold<=1; end
         if(!hold) begin
            if (r_data==0) begin en<=1; j<=i+i; end
            else begin i<=i+1; end
         end
      end
      else if(en==1) begin
         if(j<N)begin wea<=1; w_addr<=j; w_data<=1; j<=j+i; end
         else begin wea<=0; en<=0; i<=i+1; end
      end
   end
   else begin
      done<=1;
```

```
48          if(done) begin
49             if (((cnt_temp_reg<N)&(select))|((cnt_temp_reg>=2)&(~select))) begin
50                r_addr<=cnt_temp_reg;
51                if(hold) begin
52                   if(timer>2)begin timer<=0; hold<=0; end
53                   else begin timer<=timer+1; end
54                end
55                else begin
56                   if ((~r_data)&tick) begin
57                      cnt_20b_reg<=cnt_temp_reg;
58                      cnt_temp_reg<=(select)?cnt_temp_reg+1:cnt_temp_reg-1;
59                      hold<=1;
60                   end
61                   else if ((r_data)) begin
62                      cnt_temp_reg<=(select)?cnt_temp_reg+1:cnt_temp_reg-1;
63                      hold<=1;
64                   end
65                   else if((~r_data)&(~tick)) begin
66                      cnt_temp_reg<=cnt_temp_reg;
67                   end
68                end
69             end
70          end
71       end
72    end
73
74    assign cnt_20b=cnt_20b_reg;
75
76    ram_ip ram_ip_inst_1
77    (
78    .clka      (clk          ),     // input clka
79    .wea       (wea          ),     // input [0 : 0] wea
80    .addra     (w_addr       ),     // input [19 : 0] addra
81    .dina      (w_data       ),     // input [0 : 0] dina
82    .clkb      (clk          ),     // input clkb
83    .addrb     (r_addr       ),     // input [19 : 0] addrb
84    .doutb     (r_data       )      // output [0 : 0] doutb
85    );
86    endmodule
87    //埃氏筛法计算模块
88    //输入：复位信号(rstn)，显示间隔信号(tick)，增减选择信号(select)
89    //输出：20位二进制码素数(cnt_20b)
90    //功能：使用埃氏筛法计算素数并输出
91    //构成：其分为两个个主模块，计算模块和输出模块。
92    //步骤：先计算完毕并将计算结果存储到RAM内存中，然后从RAM中读取结果并输出。
93    //循环实现：使用if判断循环变量递增条件
94    //难点1(内存读写延时)：内存读数据传入到值传出有2个时钟周期的延时，使用计数器解决
95    //难点2(多功能代码复用)：通过增加端口(select,tick)和条件判断决定运行逻辑
96    //难点3(复位信号时机)：与外部模式选择模块相适配，实现先传值后复位
```

3.8 20 位二进制到 6 位 BCD 码译码器-binary_20b_to_bcd_6d.v

```
module binary_20b_to_bcd_6d #(parameter N = 20,parameter M = 24)
(
   input  [N-1:0]  input_20b,
   output [M-1:0]  output_6d
);
reg [3:0]       digits [5:0];

integer i;
always @(input_20b) begin
   for (i = 0; i < 6; i = i+1) begin
      digits[i] = 4'd0;
   end
   for(i = N-1; i >= 0; i = i-1) begin  //加3移位法
      if (digits[0] >= 4'b0101) digits[0] = digits[0] + 4'b0011;
      if (digits[1] >= 4'b0101) digits[1] = digits[1] + 4'b0011;
      if (digits[2] >= 4'b0101) digits[2] = digits[2] + 4'b0011;
      if (digits[3] >= 4'b0101) digits[3] = digits[3] + 4'b0011;
      if (digits[4] >= 4'b0101) digits[4] = digits[4] + 4'b0011;
      if (digits[5] >= 4'b0101) digits[5] = digits[5] + 4'b0011;

      digits[5][3:0] = {digits[5][2:0], digits[4][3]};
      digits[4][3:0] = {digits[4][2:0], digits[3][3]};
      digits[3][3:0] = {digits[3][2:0], digits[2][3]};
      digits[2][3:0] = {digits[2][2:0], digits[1][3]};
      digits[1][3:0] = {digits[1][2:0], digits[0][3]};
      digits[0][3:0] = {digits[0][2:0], input_20b[i]};
   end
end
assign output_6d ={digits[5],digits[4],digits[3],digits[2],digits[1],digits[0]};
endmodule
//20位二进制到6位BCD码译码器
//输入：20位二进制码(input_20b)
//输出：6位BCD码(output_6d)
//功能：阻塞赋值实现组合逻辑移位寄存器，将20位二进制码转换为用于输出的6位BCD码
```

该部分代码的具体实现参考了 CSDN 博客二进制转 BCD 码原理及 Verilog 实现[1]

[1] https://blog.csdn.net/eroDuanDian123456/article/details/122551060

3.9 一秒计时器-Count_to_one_second.v

```
module Count_to_one_second #(parameter Count_To = 50_000_000)
(
   input clk,
   output one_second
);

reg [31:0]  counter;
reg         one_second_r;

always @(posedge clk) begin
   if((counter < Count_To) && ~one_second_r) begin
      counter <= counter + 1;
   end
   else if((counter < Count_To) && one_second_r) begin
      one_second_r <= 0;
      counter <= counter + 1;
   end
   else begin
      counter <= 0;
      one_second_r <= 1;
   end
end
assign one_second = one_second_r;
endmodule
//一秒计时器
//输入：时钟(clk)
//输出：一秒脉冲(one_second)
//功能：输入为时钟，输出为周期为1s脉冲
```

3.10 顶层文件与走线-top.v

```
module top
(
input           clk,
input           rstn,
output [3:0]    led,
input  [3:0]    key,
output [5:0]    seg_sel,
output [7:0]    seg_dig
);
wire [3:0]          key_signal;
wire [3:0]          key_pulse;
wire                rstn_signal;
wire                tick;
wire                one_second;
wire                select;
wire                reset;
wire [47:0]         seg;
wire [19:0]         cnt_20b;
wire [23:0]         cnt_24d;
//一秒计时器：产生周期为1s的one_second脉冲信号供其他模块使用
Count_to_one_second timer(clk,one_second);
//按键除抖及脉冲：按键信号输入，输出按键的脉冲信号，保证按键单击单次触发
Killshake Killshake(clk,rstn,rstn_signal);
genvar j;
generate for(j = 0; j < 4; j = j + 1) begin
   Killshake Killshake (clk,key[j],key_signal[j]);
   Edgedetect Edgedetect (key_signal[j],clk,key_pulse[j]);
end
endgenerate
//Led控制及模式选择：除抖按键信号输入，控制LED灯亮灭及运算显示模式
ledcontrol ledcontrol(clk,rstn_signal,key_pulse,led);
modecontrol modecontrol(clk,rstn_signal,one_second,led,key_pulse,reset,select,tick);
//埃氏筛法：算法主模块，输出20位二进制数素数(cnt_20b)
isprime solver(clk,reset,tick,select,cnt_20b);
//显示模块：将20位二进制数输入，先转化为6位BCD码，再导入显示模块显示
binary_20b_to_bcd_6d transformer(cnt_20b,cnt_24d);
genvar i;
generate for(i=0; i<6; i=i+1) begin
led7seg_decode d(cnt_24d[i*4 +: 4], 1'b1, seg[i*8 +: 8]);
end
endgenerate
seg_driver #(6) driver(clk, rstn_signal, 6'b111111, seg, seg_sel, seg_dig);
endmodule
```

4 实验难点

4.1 RAM IP 核的设置与实例化

由于埃氏筛法需要对素数进行标记，故需要调用 FPGA 开发板上的内存，通过前期理论调研与研究，我们最终决定调用 RAM 内存进行运算存储。

这里参考了黑金动力社区的《FPGA 片内 RAM 读写例程》文档。（见源程序目录 misc 下）

表 1: RAM IP 核的设置

参数	参数配置
Memory Type	Simple Dual Port RAM
Port A Width	1
Port A Depth	1000001
Enable Port Type(Port A)	Always Enabled
Port B Width	1
Enable Port Type(Port B)	Always Enabled
Fill Remaining Memory Locations	√
Remaining Memory Locations(Hex)	0

本实验中使用如下的 RAM IP 实例化方式

```
reg [19:0]       w_addr;          //写入的数据的地址
reg              w_data;          //写入的数据
reg              wea;             //使能端
reg [19:0]       r_addr;          //读取的数据的地址
wire             r_data;          //读取的数据
ram_ip ram_ip_inst_1
(
.clka      (clk          ),     // input clka
.wea       (wea          ),     // input [0 : 0] wea
.addra     (w_addr       ),     // input [19 : 0] addra
.dina      (w_data       ),     // input [0 : 0] dina
.clkb      (clk          ),     // input clkb
.addrb     (r_addr       ),     // input [19 : 0] addrb
.doutb     (r_data       )      // output [0 : 0] doutb
);
```

4.2 RAM 读的时序时延

在最初撰写埃氏筛法时，忽略了 RAM 内存的读地址输入到读数据输出之间的两个时钟周期的时延问题，导致计算结果出错。经过笔者所在小组逐一 Debug 确认各模块工作正常后，确认了问题出现在 RAM 读写时延上。

我们采用的解决方案是使用计数器与控制信号来进行时钟周期的延时。具体的一种实现方式如下

```
...
r_addr<=cnt_temp_reg;//地址传入后开始计时
if(hold) begin
   if(timer>2)begin timer<=0; hold<=0; end
   else begin timer<=timer+1; end
   end
else begin
(...)//等待2个周期后执行目标代码
end
...
```

在内存地址传入语句后添加了上述延时模块后，埃氏筛法算法工作正常。

4.3 isprime 模块复位时机选择

为了实现正确的按钮功能及复位时机，笔者所在小组做了很多尝试，在实现的过程中出现了诸如“双击复位”、“模式切换复位但不执行”、“模式切换复位错误”等诸多问题，在经过多次尝试后，我们决定将复位信号的传入滞后一个时钟周期，先将参数更改，再传入复位信号。实现方式如下

```
always @(posedge clk) begin
   reset_r<={reset_r[0],(&key_pulse)&rstn_signal};
end//reset信号需要滞后一个周期，以避免同周期复位与传值的冲突
```

5 性能测试

通过在 isprime 模块中加入以下代码，两次分别输出 a,b 的值，即可统计计算素数总共使用多少个时钟周期。

```
if(!done) begin
   b<=b+1;
   if(b>100) begin
      a<=a+1;
      b<=0;
   end
end
```

输出 a=21588，b=18，则一共经历 2158818 个时钟周期，折合 0.04317636s，已经是相当出色的表现。

6 反思总结

本次实验开始于 2023 年 11 月 23 日，完成于 2023 年 12 月 28 日，共耗时 36 日的时间，也是笔者所在小组第一次使用 Git 协作开发的实验。实验本身具有较高的挑战性。计算素数是一个古老经典的问题，而笔者所在小组通过调研与实验，比较不同算法的时间复杂度，深切体会到了算法所带来的极大时间优势，同时也窥见数学、计算机给人类社会带来的生产力变革。

拥有一块可以实际操作的开发板，并将自己实现的代码付诸于实践。Vivado 综合生成电路过程中的等待，Debug 过程中的探索，算法试验成功的喜悦，再到现在回首看这一个月历程的成就与感动。

在这过程中我们不断编写代码，又不断推倒重来。要说在这个过程中什么是最重要的心得，那就是”Less is More”。往往是添加了很多内容，但是代码并不会按照预期的效果被硬件所实现，所以我们选择删除并重新审视问题需求，然后重新编写。而经过实践检验，这确实能够出奇的解决问题。

再就是在合作开发上，每个人都顺利完成了分发的任务，每个人都关切着任务进展，没有大家的努力，也就没有最后实验结果的出彩。

最后提一个小插曲，在模式选择变量传值的过程中，我们之前错误的将按键脉冲信号作为 if 的条件，导致按键按下正确触发的概率只有 0.000002%，我们一一尝试了是否能够“抽中”，结果显然均以失败告终（乐）。

这就是全部了！感谢老师们的准备与付出！让我们能体验到这样充满挑战但又收获满满的课程与实验！

A 开发日志

- 2023-11-23:
  - 建立 Github 协作库、添加模板文件 top.sv
  - 拿到 AX7035 FPGA 开发板
- 2023-11-24:
  - Github 协作邀请完毕
- 2023-12-07:
  - 总体布局与任务分发
    - Edgedetect.v-刘镇豪
    - KillShake.v-刘镇豪
    - FIFO.v-吴尚哲
    - LED_display.v-吴尚哲
    - binary_20b_to_bcd_6d.v-规划中
  - SRAM 片上内存
    - 完成了.xdc 管脚协议的补充（后证实不需要）
    - 研究了 SRAM 的结构与原理
  - 库文件细化
    - 建立了参考文献集 Reference.txt
    - 建立了重要信息共享文档 ShareLog.md
    - 建立了样本数据集 datadic.txt
- 2023-12-09
  - 关于 AX7035 开发板
    - 找到了一份完备的教程
  - 关于 DDR3
    - 建立了 DDR3 的功能及驱动模块
    - 建立了 mem_burst.v 的读写模块，但是还未来得及分析
  - 关于.xdc 文件
    - 恢复了原.xdc 样式，并对修改做了备份
  - 关于 top.sv
    - 仿照样例撰写了 led7seg_decode.v，本质为 0-9 二进制数到 8 端数码管数据译码器
    - 写了一些注释：其中下面一段代码存疑

```
genvar i;
generate
   for(i=0; i<6; i=i+1) begin
      led7seg_decode d(cnt[i*4 +: 4], 1'b1, seg[i*8 +: 8]);//+是做什么的？
end
endgenerate
```

  - 关于组员
    - FIFO.v 已完成
    - LED_display.v 已完成
    - KillShake.v 已完成
    - Edgedetect.v 已完成
- 2023-12-17
  - 关于 top.sv
    - 实现了防抖电路和脉冲输出的测试
    - 撰写了指示灯显示与状态切换代码
- 2023-12-18
  - 关于 top.sv
    - 实现了按键与 'LED'灯对应的代码与测试-刘镇豪
    - 探索了筛法的可能性并决定算法为埃氏筛法，初步完成了埃氏筛法的代码实现，未测试
  - 总体任务分发
    - binary_20b_to_bcd_6d.v-吴尚哲
    - Count_to_one_second.v-刘镇豪
  - 关于组员完成情况
    - Count_to_one_second.v 已完成，未测试
- 2023-12-23
  - 关于 top.sv
    - 实现了八段数码管显示的代码编写及测试（10 进制）
    - 实现了一秒计时器的整合与编写
    - 实现了埃氏筛法（算法层面），但是其对于内存地址的调用目前仍然存在问题
  - 关于组员
    - binary_20b_to_bcd_6d.v 已完成，已测试
    - Count_to_one_second.v 已测试
- 2023-12-24
  - 关于 top.sv
    - 实现了埃氏筛法，最快输出达到 1s 之内完成
- 2023-12-25
  - 关于 top.sv
    - 实现了最快输出的递增和递减功能按钮对应，但是对于 1s 输出的复位目前仍存在问题
  - 关于组员
    - 布置了实验报告撰写的相关任务
- 2023-12-26
  - 关于 top.sv

    - 实现了实验要求的所有功能
    - 美化了整体代码布局
  - 关于库文件
    - 将所有模块分装为.v 文件存储在 src 文件夹下
    - 将未用到的代码及内容存储在 misc/Unused 文件夹下
- 2023-12-28
  - 关于实验报告
    - 完成了实验报告撰写
  - 关于算法性能
    - 通过统计时钟周期数估计了算法性能