1. 课程设计名称

脑电波控制的贪吃蛇游戏系统。

2. 项目描述及要求

设计并实现一个用脑电波控制的贪吃蛇游戏系统。该系统可实现脑电波贪吃蛇游戏系统所需的脑电波信息采集、处理及控制功能，并展示系统工作的基本原理。

基本要求

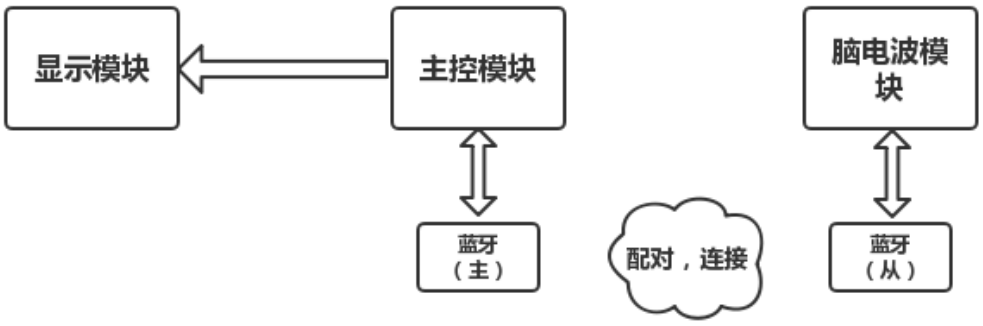
1. 各个电路模块需自行设计完成，机械模块可以购买
2. 能够提取脑电波控制信号
3. 完成贪吃蛇游戏系统

发挥部分

1. 能够用脑电波信号控制单步目标移动
2. 能够用脑电波信号控制多步目标移动
3. 能够用脑电波信号控制游戏整个过程。

3. 系统设计

1） 系统框图与说明



a） 主控模块：

a） 图像绘制：实现游戏图像元素的绘制

b） 蓝牙连接：实现与 TGAM 蓝牙模块的正确连接与配置

c） 数据解析：通过 UART 串口实现蓝牙数据包的接收与参数的解析

d） 控制信号：实现蛇移动控制信号的生成

b） **显示模块：**

通过 VGA 接口实现游戏图像的绘制

c） **脑电波模块：**

模块的正确佩戴与相关参数（如专注度）生成的训练

2） 脑电波模块

a） 关于神念科技 TGAM 脑电模块的说明

TGAM 大约每秒钟发送 513 个包，注意是“大约每秒钟”，意思就是发送包的个数是不会变的，只是发送 513 个包所花费的时间是一秒左右。

发送的包有小包和大包两种：小包的格式是 AA AA 04 80 02 xxHigh xxLow xxCheckSum 前面的 AA AA 04 80 02 是不变的，后三个字节是一只变化的，xxHigh 和 xxLow 组成了原始数据 rawdata，xxCheckSum 就是校验和。所以一个小包里面只包含了一个对开发者来说有用的数据，那就是 rawdata，可以说一个小包就是一个原始数据，大约每秒钟会有 512 个原始数据。

那怎么从小包中解析出原始数据呢？

```
rawdata = (xxHigh << 8) | xxLow;
if(rawdata > 32768){ rawdata -=65536; }
```

现在原始数据就这么算出来了，但是在算原始数据之前，我们先应该检查校验和。

校验和怎么算呢？

```
sum = ((0x80 + 0x02 + xxHigh + xxLow)^ 0xFFFFFFFF) & 0xFF
```

什么意思呢？就是把 04 后面的四个字节加起来，取反，再取低八位。

如果算出来的 sum 和 xxCheckSum 是相等的，那说明这个包是正确的，然后再去计算 rawdata，否则直接忽略这个包。丢包率在 10%以下是不会对最后结果造成影响的。

现在，原始数据出来了，那我们怎么拿信号强度 Signal,专注度 Attention,放松度 Meditation,和 8 个 EEG Power 的值呢？就在第 513 个这个大包里面，这个大包的格式是相当固定的。

b） 脑电波数据解析

TGAM 模块通过工作在从模式的蓝牙模块将数据发送至连接至 N4 DDR 上的 HC-05 蓝牙模块（这里两个模块的配对密码均设置为 0000），串口数据解析模块将接收到的脑电波数据按照数据格式解析出数据：

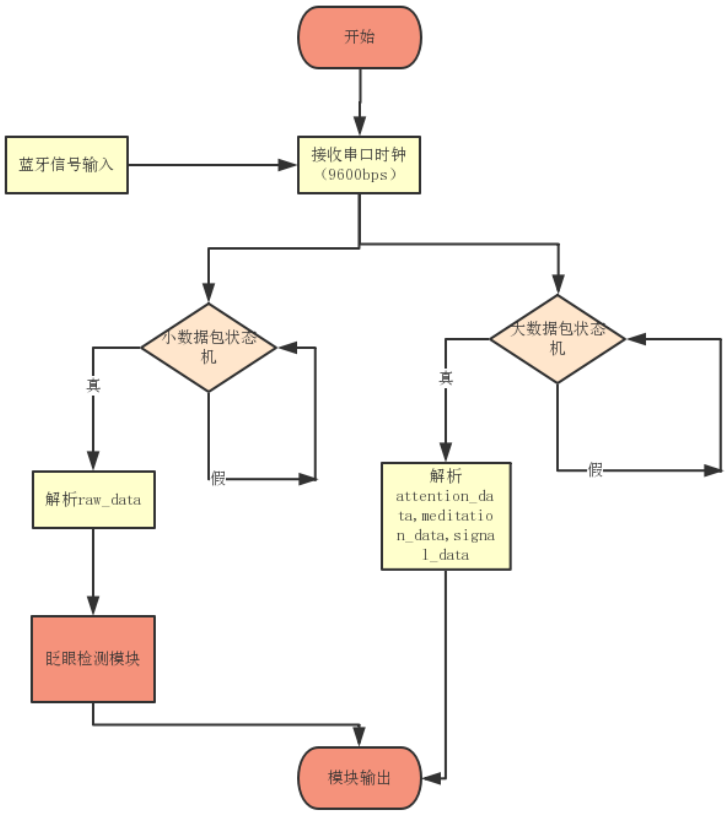
raw_data：原始脑电波形数据；

attention_data：专注度数据；

meditation_data：冥想度数据；

signal_data：信号强度数据；

解析过程中的流程框图为：



模块大致分为以下几个：

1. 接收数据同步时钟产生模块：

处于数据连接模式下的 HC-05 蓝牙模块的波特率为 9600bps，因此接收数据的同步时钟需要对板载时钟 100MHz 进行分频至 9600Hz 来对串口接收到的数据进行同步接收，该分频模块的核心代码如下：

```
always @(posedge clk or negedge rst)
begin
   if (!rst)
    rs_clk<='b0;
    else
      if(count==N2&&rs_ena)      //当且仅当count计数到一半 5207 且通信使能时允许时钟翻转
         rs_clk<='b1;            //
         else
         rs_clk<='b0;            //使得rs_clk是一个小波峰的时钟信号，在有效信号的数据位为高电平。

end
```

2. 字节数据解析模块：

首先关于 RS232 串口通讯协议的数据格式：1 位起始位，8 位数据位，1 位停止位，通过对接收到的电平检测起始位，并将后续 8 位数据寄存输出；核心代码如下：

```
always @(posedge clk or negedge rst)
begin
       if(!rst)
           num<='d0;
        else
        if(rx_int)
         begin
             if(rs_clk)//在接收中断的情况下，到来一次采样时钟进行一次如下运算，
                begin
                   num<=num+1'b1;
                   case(num)
                    'd1:rx_data_buf[0]<=rx_data;
                    'd2:rx_data_buf[1]<=rx_data;
                    'd3:rx_data_buf[2]<=rx_data;
                    'd4:rx_data_buf[3]<=rx_data;//数据依次加入到缓存区

                    'd5:rx_data_buf[4]<=rx_data;
                    'd6:rx_data_buf[5]<=rx_data;
                    'd7:rx_data_buf[6]<=rx_data;
                    'd8:rx_data_buf[7]<=rx_data;
                    default:;
                   endcase
                 end
               else
                 if(num=='d10)
                    begin//数据接收完成，将缓存数据交给寄存器，同时计数器清零
                       num<='d0;
                       data_byte_r<=rx_data_buf;
                    end
          end
end
```

3. 小数据包解析模块：

该模块主要根据接收字节数据解析模块输出的字节类型数据结合小包数据流格式：AA AA 04 80 02 xxHigh xxLow xxCheckSum，通过校对校验和输出 raw_data 数据，核心代码块如下：

```
    if(byte_int)
        begin
        if(num == 2'd3)
           num = 2'd1;
        else
           num = num + 1'b1;
        case(num)
            2'h1: high_data = byte_data_in;
            2'h2: low_data = byte_data_in;
            2'h3:
            begin
            check_sum = byte_data_in;
            if((((8'h80+8'h02+high_data+low_data)^32'hffffffff) & 8'hff) == check_sum)
            begin
              //$display("check right!");
              if((high_data << 8 | low_data) > 32768)
                 out_data = (high_data << 8 | low_data) - 65536;
              else
                 out_data = high_data << 8 | low_data;
            end
/*          else
              out_data = 16'h0;*/
            end
            default: {high_data,low_data,check_sum} = 24'h000000;
        endcase
        end
```

4. 大数据包解析模块：

该模块主要根据接收到的字节数据解析模块输出的字节类型数据，结合大包数据流格式：AA 20 02 signal_data 83 18 ……解析出包含 attention_data，meditation_data，signal_data 等数据，解析方法与 3 中解析小包数据类型数据相似，核心代码块如下：

```
reg [4:0] num = 5'd0;
always @(posedge clk or negedge rst)
begin
  if(!rst)
    begin
      signal_data = 8'h00;
      attention_data = 8'h00;
      meditation_data = 8'h00;
    end
  else
    if(frame_int)
    begin
        if(byte_int)
            begin
            if(num == 5'd31)
               num = 2'd1;
            else
               num = num + 1'b1;
            case(num)
                5'd1:  signal_data = byte_data_in;
                5'd29: attention_data = byte_data_in;
                5'd31:
                begin
                meditation_data = byte_data_in;
                out_data = {signal_data,attention_data,meditation_data};
                end
                default: ;
            endcase
            end
    end
```

5. 眨眼检测模块：

该模块对小包解析模块解析出的 raw_data 数据结合眨眼检测算法进行眨眼信号的判断，输出信号形式为一个矩形脉冲信号，该模块设计需要注意的几点：

a) 小数的计算：

i. 可以采取浮点计算 IP 核进行计算，大致思路：将数据先用定点转浮点 IP 核转换成固定格式的浮点数，然后输入到浮点计算 IP 核计算；

ii. 自定义定点数小数点位置，以定点数的形式进行计算；

这里我们考虑到资源与面积的原因，采用了 b 方式，将小数精度定义为 8 位二进制表示：2^{-8} 满足计算精度要求；

b) 计算周期问题：

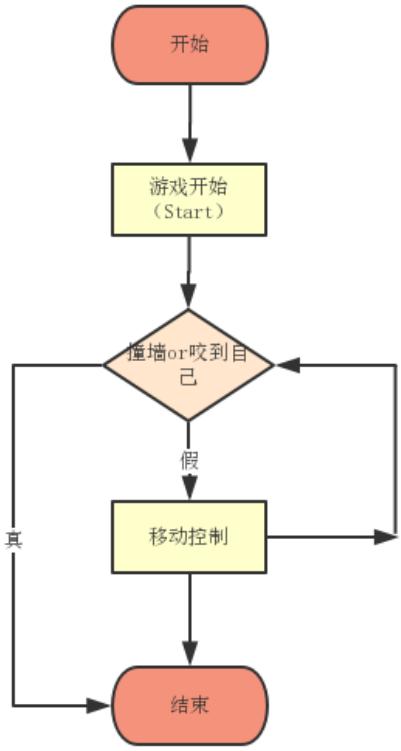
当采用板载时钟 100MHz 进行上升沿触发计算时，由于计算中涉及到的有 32 位乘除法，在综合时出现了时钟约束问题，尝试产生一个持续 100 个 100MHz 周期的时钟，其中的时钟周期足以满足计算要求；

其中核心代码块如下：

```
   always @ (posedge clk_calcu or negedge rst)
   begin
      if(!rst)
          begin
             mean_value = 32'd0;
             mean_square_value = 32'd2560000;
             s_square = 32'd0;
          end
       else
          begin
//==============================计算均值=================================================
          mean_value =  (k1 * mean_value + k2 * exten_data) >> 8;
//==============================计算均方值===============================================
          mean_square_value = (k1 * mean_square_value  + ((k2 * exten_data) >> 8 ) * exten_data) >> 8;
//==============================计算s_square=============================================
//====                    存在数据截断问题，加1寄存器作为中间输出
          if(exten_data  > mean_value)
             begin
                tmp_1 = ((exten_data - mean_value) * (exten_data - mean_value)) >> 8;
                tmp_2 = (mean_value * mean_value) >> 8;
                s_square = (tmp_1 << 8) / (mean_square_value - tmp_2);
             end
          else
             begin
                tmp_1 =  ((mean_value - exten_data) * (mean_value - exten_data)) >> 8;
                tmp_2 = (mean_value * mean_value) >> 8;
                s_square = (tmp_1 << 8) / (mean_square_value - tmp_2);
             end
          end

    end
```

3） 贪吃蛇模块

游戏状态控制的流程：



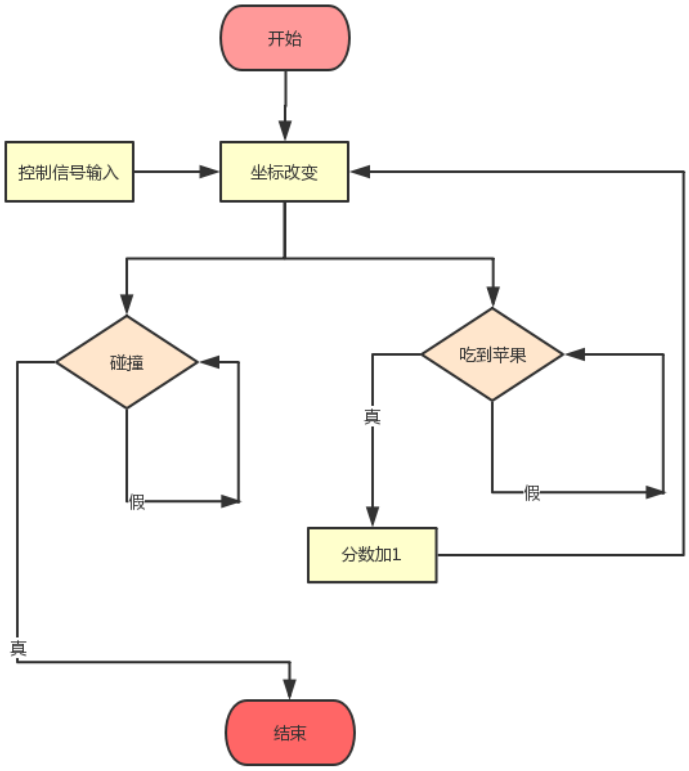
模块结构：

游戏进行状态控制模块：

该模块主要对整个游戏过程的开始结束进行控制，默认游戏开始，当碰到墙壁或者自己咬到自己时即判定游戏结束，核心代码如下：

```
always @ (posedge clk_1s or negedge rst)
begin
  if(!rst)
    game_status <= `Start;
  else
    if(is_crash || is_suicide)
        game_status <= `Over;
end
```

游戏移动控制流程：



模块结构如下：

1. 移动控制模块：

该模块为整个贪吃蛇游戏的核心模块，主要功能为：游戏结束判定，蛇身坐标的保存，按照方向改变蛇身坐标，加分的判定，核心代码如下：

```
begin
    case(direction)
    up: snake_y_pos[0] <= snake_y_pos[0] - 1;
    right: snake_x_pos[0] <= snake_x_pos[0] + 1;
    left: snake_x_pos[0] <= snake_x_pos[0] - 1;
    down: snake_y_pos[0] <= snake_y_pos[0] + 1;
    default: ;
    endcase
    for(i=1;i<=9;i=i+1)
    begin
    snake_x_pos[i] <= snake_x_pos[i-1];
    snake_y_pos[i] <= snake_y_pos[i-1];
    end
end
```

2. 分频模块：

主要是按照 VGA 640x480 显示时序产生 25MHz 的扫描时钟和按照速度等级产生蛇身移动速度时钟，核心代码如下：

```
always @(posedge clk or negedge rst)
begin
  if(!rst)
     clk_25M <= 1'b0;
  else
     if(count == N1)
        clk_25M <= 1'b0;
     else if(count == N2)
        clk_25M <= 1'b1;
 end
```

3. 苹果产生模块：

该模块主要在苹果被吃掉后产生新的苹果坐标信息，主要工作为：生成类随机的苹过坐标信息；避免产生至边界与蛇身上，核心代码如下：

```
always @ (posedge clk_25M or negedge rst)
begin
  if(!rst)
    begin
      apple_x_pos <= 7'd75;
      apple_y_pos <= 7'd30;
    end
  else
    if(apple_gen)         //  当请求产生信号与重合信号出现时刷新苹果坐标
        begin
          apple_x_pos <= apple_x_pos[4:0] + apple_y_pos[4:1] + 25;          // 范围: 5 -- 67
          apple_y_pos <= apple_x_pos[6:2] + apple_y_pos[5:2] + 7;          //  范围: 7 -- 53
        end
end
```

4. 边界与分数显示：

该模块主要显示类数码管的数字信息，主要思路为借用数码管的段显示思想将屏幕坐标固定设计成三组两位七段数码管用来显示数字信息，核心代码如下：

```
assign is_score = (x_pos>4 && x_pos<9 && y_pos==5 && seg_1[0]) ||
                  (x_pos>4 && x_pos<9 && y_pos==10 && seg_1[6]) ||
                  (x_pos>4 && x_pos<9 && y_pos==15 && seg_1[3]) ||
                  (y_pos>5 && y_pos<10 && x_pos==4 && seg_1[5]) ||
                  (y_pos>5 && y_pos<10 && x_pos==9 && seg_1[1]) ||
                  (y_pos>10 && y_pos<15 && x_pos==4 && seg_1[4]) ||
                  (y_pos>10 && y_pos<15 && x_pos==9 && seg_1[2]) ||

                  (x_pos>12 && x_pos<17 && y_pos==5 && seg_0[0]) ||
                  (x_pos>12 && x_pos<17 && y_pos==10 && seg_0[6]) ||
                  (x_pos>12 && x_pos<17 && y_pos==15 && seg_0[3]) ||
                  (y_pos>5 && y_pos<10 && x_pos==12 && seg_0[5]) ||
                  (y_pos>5 && y_pos<10 && x_pos==17 && seg_0[1]) ||
                  (y_pos>10 && y_pos<15 && x_pos==12 && seg_0[4]) ||
                  (y_pos>10 && y_pos<15 && x_pos==17 && seg_0[2]);
```

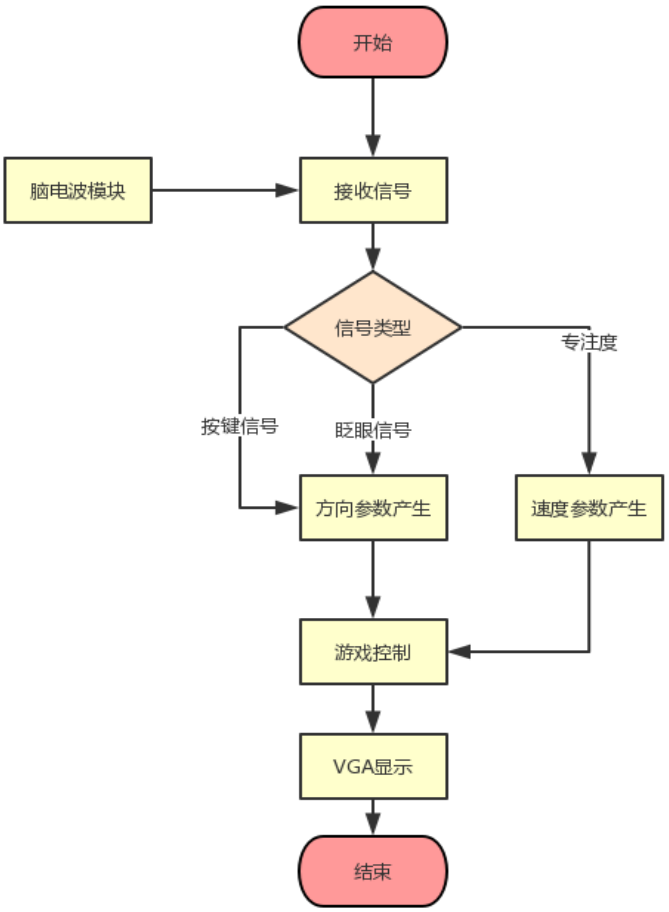
5. VGA 显示：

该模块并没有使用 RAM 用来作为显示存储，而是实时根据扫描到的坐标信息，来根据处于不同位置的确认信号来实现不同形式的显示，核心代码如下：

```
always @ (posedge clk_25M or negedge rst)
begin
   if(!rst)
      {red,green,blue} <= 12'b0000_0000_0000;
   else
      if(game_status == `Start)
      begin
      if(is_snake)                                    // 红色蛇
         {red,green,blue} <= 12'b1111_0000_0000;
      else if(is_apple)                               // 绿色苹果
         {red,green,blue} <= 12'b0000_1111_0000;
      else if(is_score)                               // 蓝色分数
         {red,green,blue} <= 12'b0000_0000_1111;
      else if(is_border)                              // 黄色边界
         {red,green,blue} <= 12'b1111_1111_0000;
      else if(is_attention)
         {red,green,blue} <= 12'b0000_0000_1111;
      else if(is_meditation)
         {red,green,blue} <= 12'b0000_0000_1111;
      else if(is_signal)
         {red,green,blue} <= 12'b0000_0000_1111;
      else
      begin
         {red,green,blue} <= 12'b0000_0000_0000;
         //   固定显示文字信息：分数
```

4） 中间连接模块

脑电波与贪吃蛇中间连接流程图：



模块结构如下：

游戏控制参数产生模块：

该模块主要根据脑电波模块输出的专注度，眨眼等数据产生游戏控制的速度等级，控制方向等信息：专注度用来控制蛇的移动速度，根据专注度的高位来确定速度等级信息，核心代码如下：

```
begin
   snake_speed <= attention_data[7:4];
end
```

5） 一些数据结构和设计思想的说明

a） 蛇身坐标的保存

关于蛇身坐标的保存，我们采用将 640 x 480 像素分为 8x8 像素的单位方格，进而将整个屏幕划分成 80 x 60 的二维坐标，因为蛇在移动时的特点为蛇头的行进决定了前进的方向，后续蛇身的坐标均赋值前一个蛇身位置的坐标，因此采用三组寄存器来保存蛇的坐标，snake_x_pos 保存蛇头及蛇身的 x 轴坐标，snake_y_pos 保存蛇头及蛇身的 y 轴坐标，snake_valid 代表此节蛇身是否得到使能即：是否为增加之后的长度，通过操作蛇头坐标来更改蛇的位置。

b） VGA 的显示控制

VGA 的显示通常需要存储当前像素点的颜色值的 RAM，我们并没有采取该方法，而是对每次扫描到的方格坐标与蛇的坐标，苹果坐标，边界坐标进行比较，若结果为真，代表当前扫描的位置为对应的目标，即按照预设颜色信息进行输出即可，这种方法节约了 RAM 资源，但操作略微繁琐；

c） 苹果坐标的产生

苹果坐标的产生应该为类随机的，因此采用来蛇当前坐标的一些位来作为苹果坐标进行输出，因为当蛇吃到苹果的坐标一般有上下左右四种不同的方位，因此可以产生多种苹果坐标数据。

d） 按键的防抖

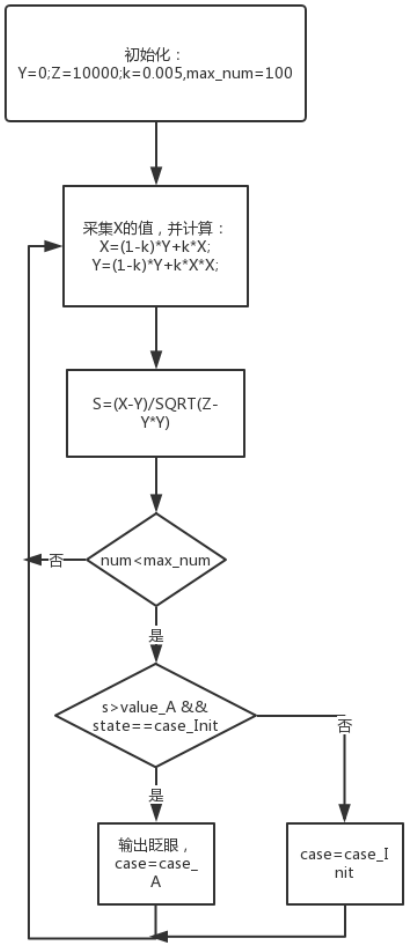
这块最开始并没有注意到，当设计手动控制贪吃蛇进行移动时，按键控制出现了蛇头与蛇身分离的状况，并且蛇的移动方向也出现了非预想结果，考虑应该是按键抖动问题，因此加入按键去抖模块，大致思路就是：通过加入延时来检测按键电平的跳变，这里预设的按键去抖延时为 20ms，核心代码如下：

```
always @ (posedge clk or negedge rst)
begin
  if(!rst)
  begin
    count_button <= 11'd0;
    reg_left_bt_1 <= 1'b0;
    reg_left_bt_2 <= 1'b0;
    reg_right_bt_1 <= 1'b0;
    reg_right_bt_2 <= 1'b0;
  end
  else
    begin
      if(count_button == 11'h7ff)
      begin
        reg_left_bt_1 <= left_bt;
        reg_right_bt_1 <= right_bt;
        count_button <= 11'h0;
     reg_left_bt_2 <= reg_left_bt_1;
     reg_right_bt_2 <= reg_right_bt_1;
      end
      else
        count_button <= count_button + 1;
    end
end

assign left_button = ~reg_left_bt_2 & reg_left_bt_1;
assign right_button = ~reg_right_bt_2 & reg_right_bt_1;
```

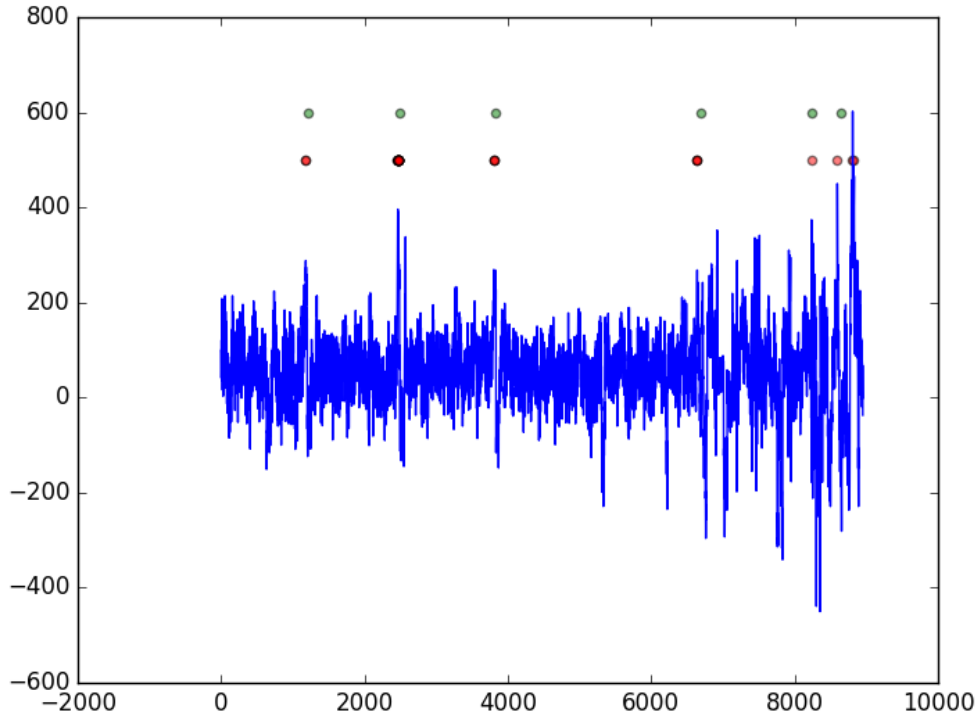
6） 关于眨眼检测算法说明

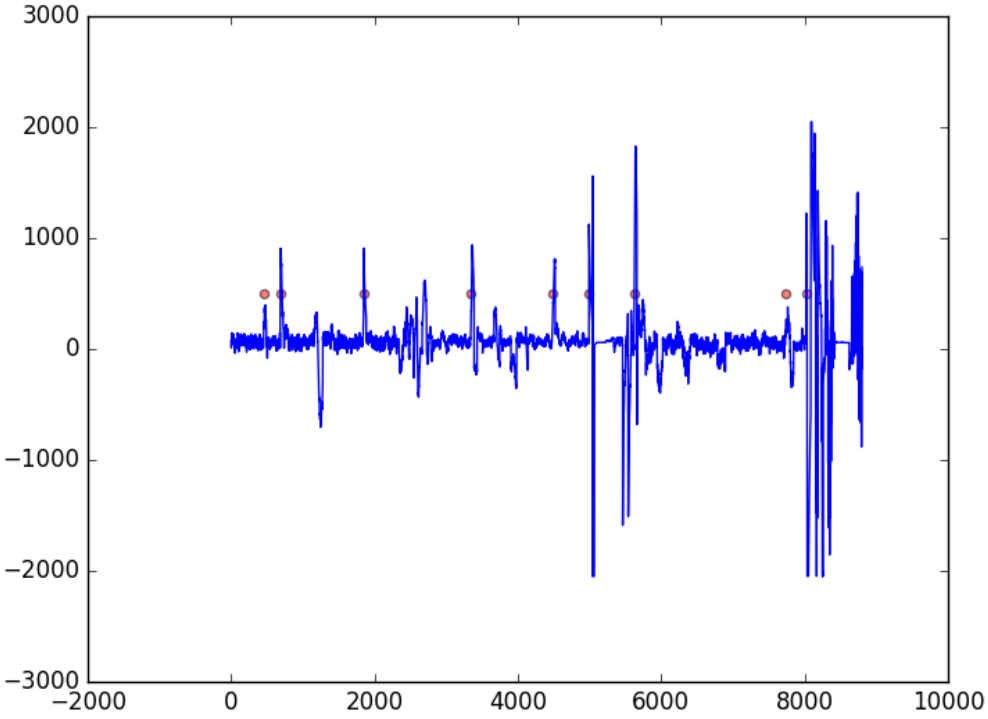
我们眨眼检测模块改进了一种通过脑电波识别眨眼力度的方法[1]，该方法具有自适应的特点，即：对眨眼的电波信号表现出的波峰，能够实现自适应的检测，检验结果并不会随着不同环境下波形峰值的不同而不同，计算流程大致为对波形数据 raw_data 求均值，方差，标准差，对偏差进行标准化，求出每次标准化的偏差值与设置的固定阈值进行比较判断是否为眨眼；计算流程如下：

1


初始化：
Y=0;Z=10000;k=0.005,max_num=100
采集X的值，并计算：
X=(1-k)*Y+k*X;
Y=(1-k)*Y+k*X*X;
S=(X-Y)/SQRT(Z-Y*Y)
num<max_num
否
是
s>value_A &&
state==case_Init
否
是
输出眨眼，
case=case_A
case=case_Init

改进后的算法仿真结果：





该算法对于眨眼信号表现出的峰值检测还是相对准确，verilog 实现的该算法见上文。

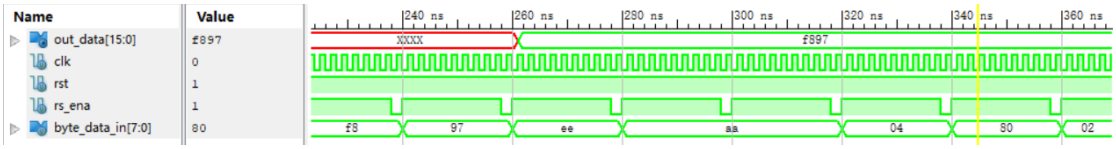
6. 系统仿真

a） 核心模块 Testbench 及仿真波形

小包数据解析模块仿真：

Testbench：（部分）

```
        rs_ena = 1;
        byte_data_in = 8'hAA;
        #18;
        rs_ena = 0;
        #2;

        rs_ena = 1;
        byte_data_in = 8'hAA;
        #18;
        rs_ena = 0;
        #2;

        rs_ena = 1;
        byte_data_in = 8'hAA;
        #18;
        rs_ena = 0;
        #2;

        rs_ena = 1;
        byte_data_in = 8'h04;
        #18;
        rs_ena = 0;
        #2;

        rs_ena = 1;
        byte_data_in = 8'h80;
        #18;
        rs_ena = 0;
        #2;

        rs_ena = 1;
        byte_data_in = 8'h02;
        #18;
        rs_ena = 0;
        #2;

        rs_ena = 1;
        byte_data_in = 8'hF8;
        #18;
        rs_ena = 0;
        #2;

        rs_ena = 1;
        byte_data_in = 8'h97;
        #18;
        rs_ena = 0;
        #2;
```

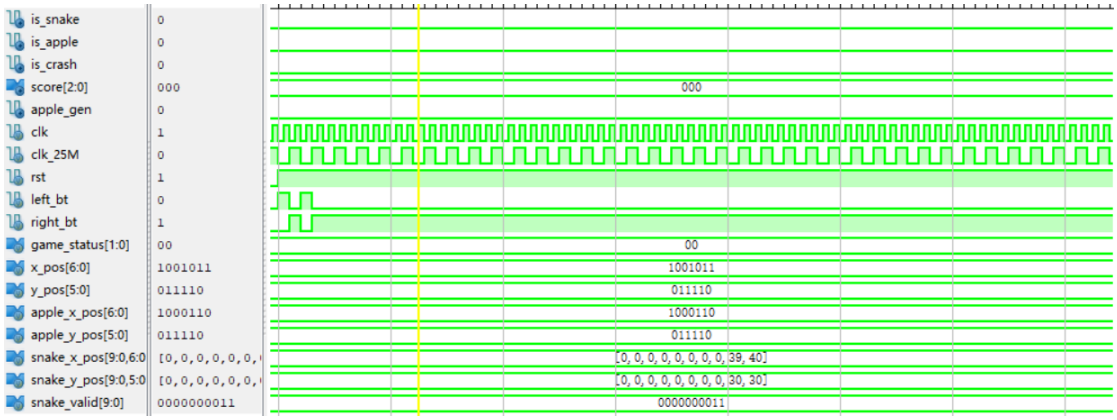


仿真波形显示：当按照小包数据流格式输入数据时，该模块能准确将其中包含的 raw_data 数据解析出来，且在实测过程中通过蓝牙调试助手 APP 发送该数据流格式的数据包，raw_data 数据能准确显示在数码管上，因此该模块调试正确。

贪吃蛇蛇身控制模块仿真：

Testbench:（部分）

```
     always begin
     #1;
     clk = 1'b1;
     #1;
     clk = 1'b0;
     end
// clk_25M
     always begin
     #2;
     clk_25M = 1'b1;
     #2;
     clk_25M = 1'b0;
     end

     initial begin
          // Initialize Inputs
          clk = 0;
          clk_25M = 0;
          rst = 0;
          left_bt = 0;
          right_bt = 0;
          game_status = 0;
          x_pos = 75;
          y_pos = 30;
          apple_x_pos = 70;
          apple_y_pos = 30;

          // Wait 100 ns for global reset to finish
          #100;
          rst = 1'b1;
          left_bt = 1'b1;
          #2;
          left_bt = 1'b0;
          right_bt = 1'b1;
          #2;
          left_bt = 1'b1;
          right_bt = 1'b0;
          #2;
          left_bt = 1'b0;
          right_bt = 1'b1;
```



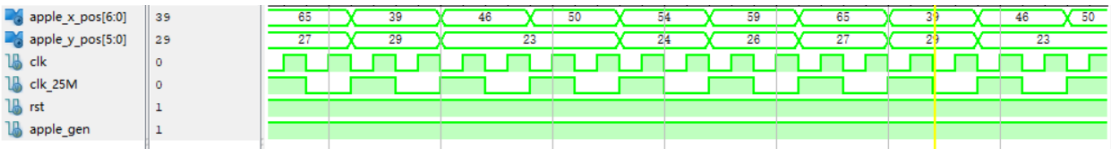
由仿真波形可知：按照按键的改变，蛇身坐标在移动时钟的上升沿按照预设方向进行改变，该模块工作正常。

苹果坐标产生模块仿真：

TestBench：

```
     always begin
     #1;
     clk = 1'b1;
     #1;
     clk = 1'b0;
     end
```

```
// clk_25M
     always begin
     #2;
     clk_25M = 1'b1;
     #2;
     clk_25M = 1'b0;
     end

     initial begin
          // Initialize Inputs
          clk = 0;
          clk_25M = 0;
          rst = 0;
          apple_gen = 0;

          // Wait 100 ns for global reset to finish
          #100;
          rst = 1;
          apple_gen = 1;
```



由仿真波形可以看出：在苹果产生信号的持续作用下，苹果坐标不断改变，且坐标的改变并没有固定规律可循，达到设计目的。

7. 实验总结

a） 待改进的问题

i. 眨眼检测模块由于计算精度与数据速率较快没有进行实测仿真，而导致并不能准确识别出眨眼信息；可通过蓝牙串口调试助手编辑数据包并通过数码管将运算的中间结果显示进行调试；

ii. 蛇身移动由于对单步按键移动均能记录，没有加入方向与此时运动方向判断模块可能出现后退问题，该问题可通过对移动之后的方向与此前移动方向逻辑进行判断决定下一步移动方向。

b） 心得体会

在本次课程设计中，我主要负责脑电波数据的接收与调试，首先，脑电波 TGAM 模块在脑电数据检验中表现出较强的准确性，且 TGAM 模块利用自身的算法运算出关于专注度，冥想度及其它脑电信号的指标值给脑机接口的扩展带来了很大的空间。

其次，在脑电波数据的接收中由于是使用蓝牙串口进行数据的收发，因此应编写串口数据的接收模块，这一块对数据的同步和接收的时序要求很高，否则会出现误码；同时对脑电波数据的处理特别是眨眼信号的检测并没有做到较高的准确性，反映出在算法的实现与调试方面还存在很多的弱点，有待于加强。

在与同组成员关于不同模块之间数据结构设计问题讨论中也让我受益很多，

在贪吃蛇的设计中，在资源与面积及实现复杂度方面进行了详细的讨论，查阅了一些相关的资料借鉴了好的实现思想，确实达到了事半功倍的效果，同时关于不同模块输入与输出数据的接口的同一方面进行了详尽的讨论为后续的联合调试节省了不少时间并且减少了很多低级错误的出现。

最后，感谢老师的详尽指导，感谢同组成员的合作与支持，感谢其他给予我们帮助与指导的人。

参考文档：

[1] 刘厚康 陈法圣《一种通过处理脑电波识别眨眼力度的方法》.

[2] EDA 先锋工作室《轻松成为设计高手---Verilog HDL 使用精解》北京航空航天大学出版社