同济大学计算机系

数字逻辑课程综合实验报告

学　　号 **1651047**

姓　　名 杨文翔

专　　业 工科试验班（计算机类）

授课老师 郭玉臣

一、 实验内容

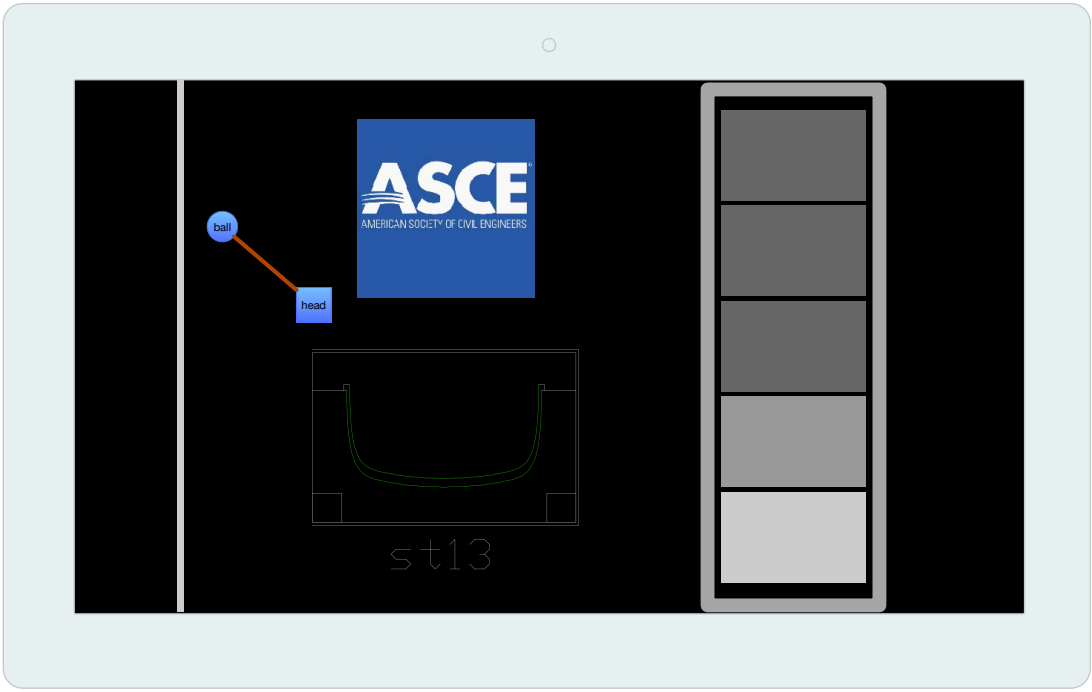
本实验要求学生在前面部件实验的基础上，依照数字系统设计自顶向下的原则完成一个应用数字系统设计。要求系统至少包含 5 个基本部件模块实现的子系统，且系统中必须有输入模块(如开关、按钮或键盘等)及输出显示模块(如 LED 灯、数码管或显示器等)。实验中分频器可以使用 IP 核，VGA 显存也可以使用板上存储器的 IP 核来实现，除此之外，所有模块要求用数字逻辑设计方法实现。

CNC 模拟器

受土木工程学院混凝土轻舟队船模制造过程的切模步骤启发，这个 CNC 模拟项目希望利用充分数字系统自动控制的特点，对原本手动操控电热丝的切模过程进行优化，用 FPGA 开发板控制切模路线、操控电机移动，从而提升船模准备效率，展现自动化的优势。

受时间和资源限制，综合试验只实现了完整项目的部分功能。以手动方式切模步骤主要分为路线规划、模具准备和用电热丝对泡沫进行热切割几个步骤。对此过程自动化主要需要解决如下几个问题：将轻舟截面的 CAD 模型转换为切割路径；使用电机代替人手控制移动方向和速度；用户界面设计。

Figure 2 of 9: The EPS mold was divided into 19 cross-sections. Each cross-section was placed between 2 wooden molds and cut by a hotwire system



界面图

用户界面包括左边正方形区域的控制区和右半边及左方的距离指示。控制区内有控制点，用(ball_x, ball_y)表示坐标；电机头，用(head_x, head_y)表示坐标；ball 和 head 之间使用橙色线连接，表示目的移动方向。背景分别是上方的图标和下方的切割路线。右侧 dashboard 区域的五个色块表示距离，由下至上亮起的由浅至深的色块依次表示距离的递增。所有边界的颜色也会随着距离改变而变化。

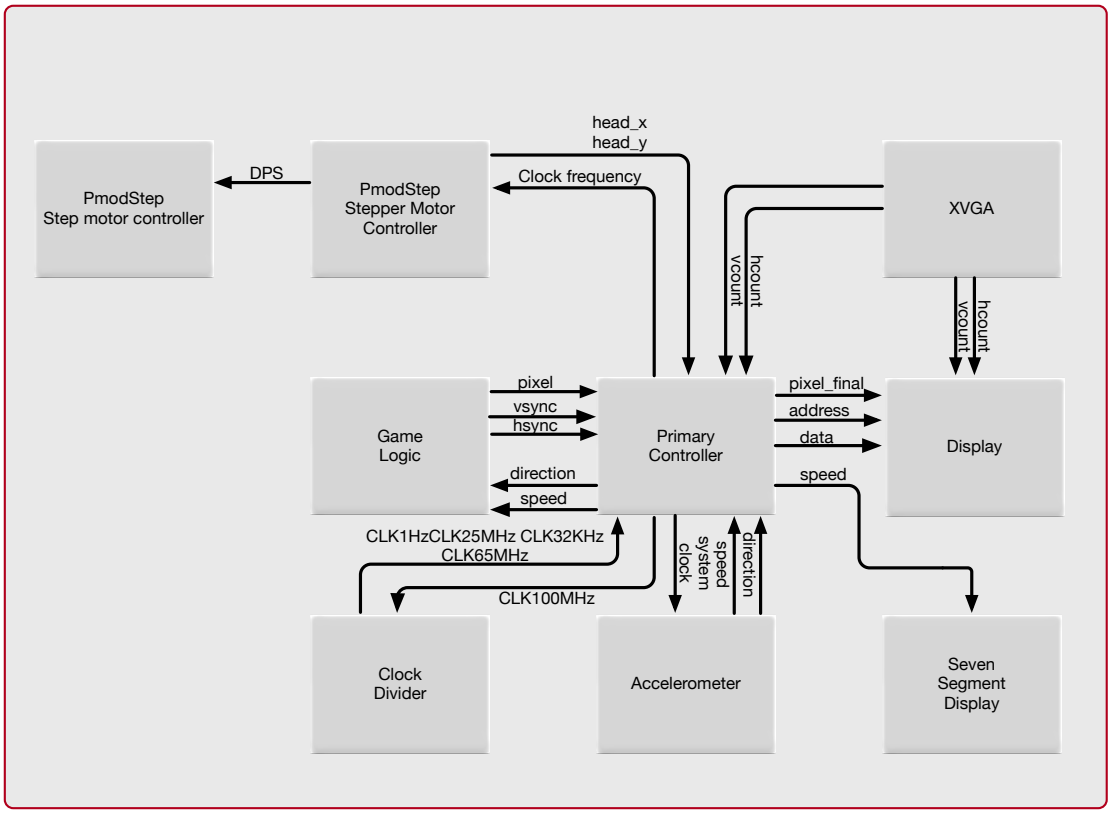
操作说明：比特流下板后，顶层模块的状态将在约三秒内从 transition1 过渡到 transition6，表示启动初始化完成。初始化过程中，画面渐变出现（用 SW[14]再现），两张背景图片以及 head 会从初始坐标(0,0)移动到 game_board 中心，同时屏幕右侧的 dashboard 区域色块指示 head 与位于屏幕中心的控制点之间的距离从最高红色的远端色块逐渐熄灭，直至仅剩最低绿色的重合色块点亮，七位数码管上现实的距离平方的十六进制数目也逐渐下降到零。在 head 和 ball 完全重合后，它们的颜色由绿、蓝变为红色，所有边框从草绿色变成紫色。球和方块间有橙色线条连接，代表目标方向。SW[15]的开关依次控制模式选择加速度传感器或方向按钮，SW[2:0]是显示器 DEBUG 开关，按下表顺序依次显示白色边框、色条、圆和方块、背景图片、dashbord、除背景图片外全部内容和最终效果。使用时 SW[2:0]应置为 3'b111。

```
case (switch[2:0])
  3'b000: rgb <= { 12{border} };
  3'b001: rgb <= { vcount[4:1] + hcount[9:6], hcount[8:5], hcount[3:0] };
  3'b010: rgb <= pixel_gameboard;
  3'b011: rgb <= picture;
  3'b100: rgb <= pixel_dashboard;
  3'b110: rgb <= pixel;
  3'b111: rgb <= pixel_final;
endcase
```

SW[5:3]控制 gravity。gravity 越大，加速度传感器灵敏度越高。
SW[5:3]的值取 1 时进入精细控制模式，球的最大速度和加速度传感器灵敏度都受到额外限制。
方块表示的电机追寻控制点的过程中 JB Pmod 接口会输出对应的电机控制信号，连接上 Pmod Step 和 4 口步进电机后，电机在 X-Y 平面内的位置和方块保持同步。

二、 CNC 数字系统总框图

（按由顶向下方法进行子系统的划分，给出包含各子系统相互关系及控制信号的总框图，并对各子系统功能及实现进行概述。具体可参考教材 183 页的相关描述方法。）



子系统：

1. Game Logic

第一个子系统对目标点位进行控制。

主要输入参数为目标点控制信号，分别为来自加速度传感器的 X 和 Y 方向大小和板上方向按钮输入的移动方向。加速度数据范围是由原始加速度数据缩放得到的(-4, 4)，用于速度的积累；板上方向按钮（`manual[3:0]`）选择移动方向，在由 `gravity` 确定的每个更新周期内移动 `ball` 坐标 `localparam STEP` 个像素。这两种更新方式的选取取决于 `accel_en` 的电平高低。

在使用加速度传感器作为控制参数时，若 `gravity` 为 0，`ball` 不移动；若 `gravity > 1`，每 `1/gravity` 秒速度以 `gravity` 大小为增量进行积累，速度范围被限制在 `(6'sb10_0001, 6'sb01_1111)`；若 `gravity = 1`，每 1 秒速度以 `gravity` 大小为增量进行积累，速度范围被限制在 `(6'sb00_0010, 6'sb11_1110)`，所以设置 `gravity = 1` 可以进入降低更新频率、限制速度的精细重力控制模式。

2. Stepper Motor Controller

这个子模块的功能是接收控制点的坐标 `(ball_x, ball_y)`，让（`head_x, head_y`）连续以限制的最大速度追寻控制点，并向主控制器输出 `(head_x, head_y)` 以显示在显示器上。这个模块还输出 `(signal_x, signal_y)`，是连接至 `Pmod Step` 电机控制模块的接口。电机运行速度由计算中间变量 `dps` 确定。

3. MOTOR

电机控制信号生成器。输入系统时钟和目标带符号速度，使用内部分频器将系统时钟转换成对应的驱动频率，利用步进电机有限状态机的状态转换决定驱动的线圈的依次转换，驱动电机以正确方向和速度运转。

4. XVGA

符合 VGA 协议的 VGA 基本信号生成器。输入对应此项目使用的 1024x768x60Hz 显示器使用的 65MHz 像素时钟，进行逐行扫描，输出水平扫描像素数 hcount 和竖直扫描像素数 vcount。hsync 和 vsync 分别对应每行扫描结束和每帧扫描结束的标记。

5. DISPLAY

VGA 信号输出选择。全部需要显示的物体各自像素颜色由主控制器依照程序状态决定是否显示，然后通过 blend 模块对所有物体的像素进行 Alpha Blending 叠加，最后合成为 12 位的和 vcount、hcount 对应的当前像素的 rgb 值通过 VGA 接口传给显示器。

6. DRAWING

画一个可变大小、坐标和颜色的方块或指定半径的圆。

7. DASHBOARD

左右边框和距离色块显示。输出当前 hcount、vcount 对应像素颜色值。

8. BACKGROUND

输入 ball 和 head 坐标，画一条橙色的线连接 ball 和 head，作为背景像素输出。

9. CLOCK DIVIDER

时钟分频使用了自定义时钟分频器转换低频时钟和 Clock wizard IP 核转换高频时钟。

自定义时钟分频器实现了动态分频，传入系统时钟和目标频率，输出目标频率的时钟。

10. ACCELEROMETER

通过 SPI 协议与板载加速度传感器通信，传输系统时钟，读出 X、Y 轴加速度大小，读出数据是 32 位补码，对应(-2g, 2g)加速度信号，使用时需要进行额外转换。

11. SEVEN SEGMENT DISPLAY

七段数码管显示模块。输入系统时钟和 32bit 数据，数码管输出对应 16 进制数据。

12. SD CARD/AUDIO

未完成，代码未删除。

13. DEBOUNCE

连接开关，防止开关金属片在 6.5ms 内的跳动造成的多次触发。

14. SYNCHRONIZE

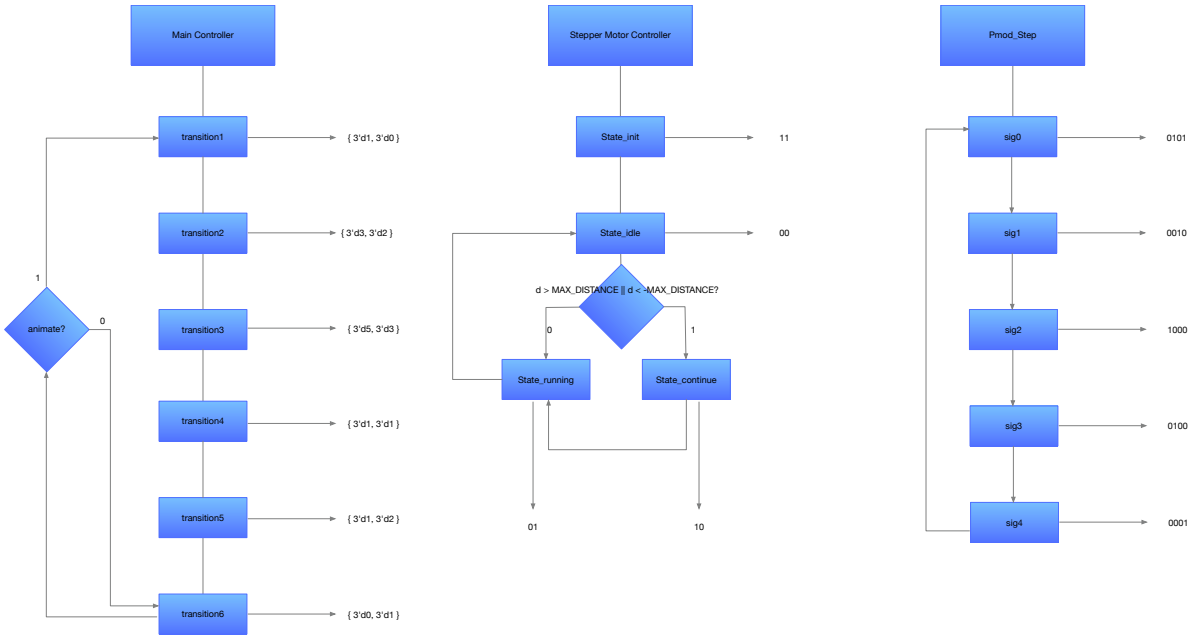
通过 `pipeline` 稳定信号。连接系统时钟加速度传感器以 `100MHz` 频率读取加速度值，这个信号传入运行在 `65Mhz` 的像素时钟上的 `game module` 时可能造成 `65Mhz` 时钟上升沿读取加速度信号时加速度信号正好进行 `0-1` 转换过程中，造成读入无效值引起加速度信号无效。解决方案是在加速度传感器的信号输出后增加两个以上 `100MHz` 时钟周期的触发器，同步信号。

15. 主模块

连接子模块。主模块使用一个有限状态机控制开场音效（未完成）和开场动画（完成）。

三、 系统控制器设计

（要求画出所设计数字系统的 ASM 流程图，列出状态转移真值表。由状态转移真值表，求出系统控制器的次态激励函数表达式和控制命令逻辑表达式，并用 Logisim 画出系统控制器逻辑方案图。）



四、 子系统模块建模

（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模的 verilog 代码）

1. GAME LOGIC

```
input                       reset,
input     signed  [4:0]  ACCEL_X_RAW,
input     signed  [4:0]  ACCEL_Y_RAW,
input             [2:0]   gravity,
input                     vsync,
input                     accel_en,
input             [3:0]   manual,
output reg signed [10:0] ball_x,
output reg signed  [10:0] ball_y
```

对目标点位进行控制。由顶层模块传入的来自加速度传感器的 X、Y（反向）方向的加速度值(ACCEL_X_RAW、ACCEL_Y_RAW)和方向按钮控制的方向(manual[3:0])在每个由 gravity 控制的周期内对当前目标点的移动速度和位置进行更新。若 accel_en 为高电位，即加速度模式，控制点的位置是由用加速度积累得到的速度进行更新的。在这个模式下，输入范围是(-8, 8)的加速度值在每个 vsync 上升沿累加在内部变量 speed 上，每次累加都会确认速度是否在设好的范围内，确认方法是对累加结果进行上溢和下溢检测：若上溢，将速度设为最大值；若下溢出，设为最小值。

```
{extra_x, speed_x_accu} =

$signed({speed_x_accu[5], speed_x_accu}) + ACCEL_X;
```

若 accel_en 为低电位，控制点位置由板载方向按钮控制进行以局部变量 STEP 为步长的移动。
两种模式下，都会持续在每次位置更新前进行边界检测，若更新后的位置将超出显示区域，加速度模式将速度反向，方向键模式将球的位置限制在边界上。球的坐标通过 ball_x, ball_y 传回顶层模块。

2. STEPPER MOTOR CONTROLLER

```
input                    CLK100MHZ;
input                    reset;
input       signed  [10:0]  ball_x;
input       signed  [10:0]  ball_y;
input                    vsync;
output  reg  signed  [10:0]  head_x;
output  reg  signed  [10:0]  head_y;
output             [3:0]   signal_x;
output             [3:0]   signal_y;
```

这个模块输入圆的坐标，输出方块坐标和电机驱动信号。圆的坐标代表控制点位置，方块代表电机当前位置。控制目的是让电机从(head_x,head_y)在规定时间内以不超过最大速度移动到(ball_x, ball_y)。因为电机速度不能突变，且速度越快扭矩越小，所以必须限制电机运行最大速度，用局部变量MAX_DISTANCE/count_freq表示，其中MAX_DISTANCE为每一个移动周期内最大移动距离，count_freq为周期长度。

程序每(count_wait+count_freq)/60秒对目标点位置进行采样，将方块（电机）位置和圆形位置坐标差值的绝对值保存在(d_x, d_y)中，接下来的一个周期开始时算法先计算是否满足最大速度限制的要求，然后按照距离大小将移动状态分为三类，设置有限状态机至对应运行状态。这个周期内，每种运行状态对应着的运行模式会接管方块坐标的更新，并维持一些中间变量的更新。

状态分为四类，已在第三节中介绍。

```
localparam          [1:0]   State_idle    = 2'b00;
localparam          [1:0]   State_running  = 2'b01;
localparam          [1:0]   State_continue = 2'b10;
localparam          [1:0]   State_init    = 2'b11;
reg                [1:0]   STATE;
```

3. MOTOR

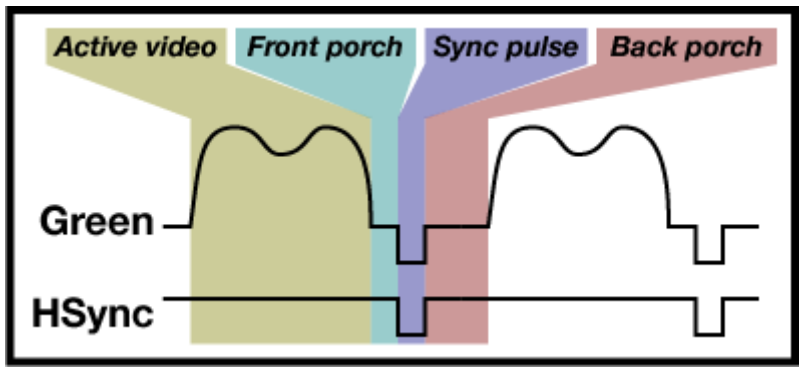
```
input             CLK100MHZ;
input             reset;
input signed [10:0] step_x; // pixels per 1/60 seconds
input signed [10:0] step_y;

output [3:0] signal_x_auto, signal_y_auto;
```

Motor 输入 Stepper Motor Controller 中的方块带符号的运行状态，在内部计算相应的方向、角速度和步进电机时钟频率，利用五状态有限状态机确定激活的步进电机线圈，传出。

4. XVGA

```
input vclock,
output reg [10:0] hcount,    // pixel number on current line
output reg [10:0] vcount,   // line number
output reg vsync,hsync
```



基础 VGA 信号生成器。每行有效像素扫描完成后都有三段 porch，时长查表得到图中对应四段时间点分别在 hcount：1047，1183，1343，vcount：776，782，805。hsync 和 vsync 照图输出。

5. DISPLAY

```
assign vga_out_red   = rgb[11:8];
assign vga_out_green = rgb[7:4];
assign vga_out_blue  = rgb[3:0];
assign vga_out_hsync = hs;
assign vga_out_vsync = vs;
```

SW[2:0]是显示器 DEBUG 开关，按下表顺序依次显示白色边框、色条、圆和方块、背景图片、dashbord、除背景图片外全部内容和最终效果。使用时 SW[2:0]应置为 3'b111。

```
  initial begin
    ball_x = planWidth;
    ball_y = planHeight;
    #360
    ball_x = ball_x - 200; ball_y = ball_y - 100;
    #360
    ball_x = 100; ball_y = planHeight;
  end
```

各像素 来源定义如下，箭头表示进行 Alpha blending，即以预定的参数对两个来源的像素以α和$(1-\alpha)$的比例混合，代码如下页。

```
  // pixel_bal      + pixel_head      --> pixel_out
  // pixel_out      + background_pixel --> pixel_gameboard
  // pixel_gameboard + pixel_dashboard  --> pixel
  // pixel          + piture          --> pixel_final
```

使用了多重 reg 做 pipeline，因为多位乘法器和移位器在 65MHz 的像素时钟下使用阻塞复制可能不能满足时间要求。

```
module blend
  #(parameter m = 1,
            n = 1)
  (input     [11:0] pixel_bal, pixel_obj,
   output reg [11:0] pixel_blend
  );

  reg [11:0] tmp;
  reg [3:0] sft11, sft12;
  reg [6:0] sft111, sft121, sft211, sft221, sft311, sft321;
  reg [3:0] sft21, sft22;
  reg [3:0] sft31, sft32;
  reg [3:0] cst = ((2**n)-m);

  always @ * begin //perform alpha blending
    tmp <= pixel_bal & pixel_obj;
    if (tmp != 0) //both on --> overlap
      begin
```

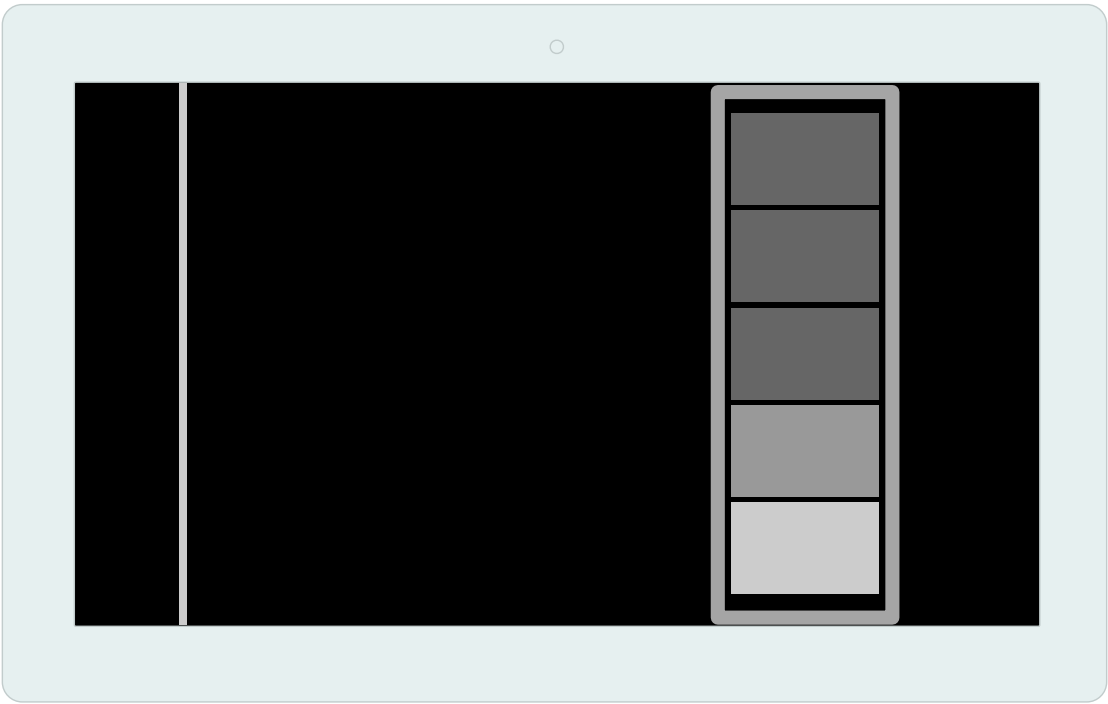
```
        sft111 <= pixel_bal[11:8] * m;
        sft11  <= sft111 >> n;
        sft121 <= cst * pixel_obj[11:8];
        sft12  <= sft121 >> n;
        sft211 <= pixel_bal[7:4] * m;
        sft21  <= sft211 >> n;
        sft221 <= cst * pixel_obj[7:4];
        sft22  <= sft221 >> n;
        sft311 <= pixel_bal[3:0] * m;
        sft31  <= sft311 >> n;
        sft321 <= cst * pixel_obj[3:0];
        sft32  <= sft321 >> n;

        pixel_blend[11:8] <= sft11 + sft12;
        pixel_blend[7:4]  <= sft21 + sft22;
        pixel_blend[3:0]  <= sft31 + sft32;
      end
    else //not overlapping
      begin
        pixel_blend <= pixel_bal | pixel_obj;
      end
  end

endmodule
```

6. DASHBOARD

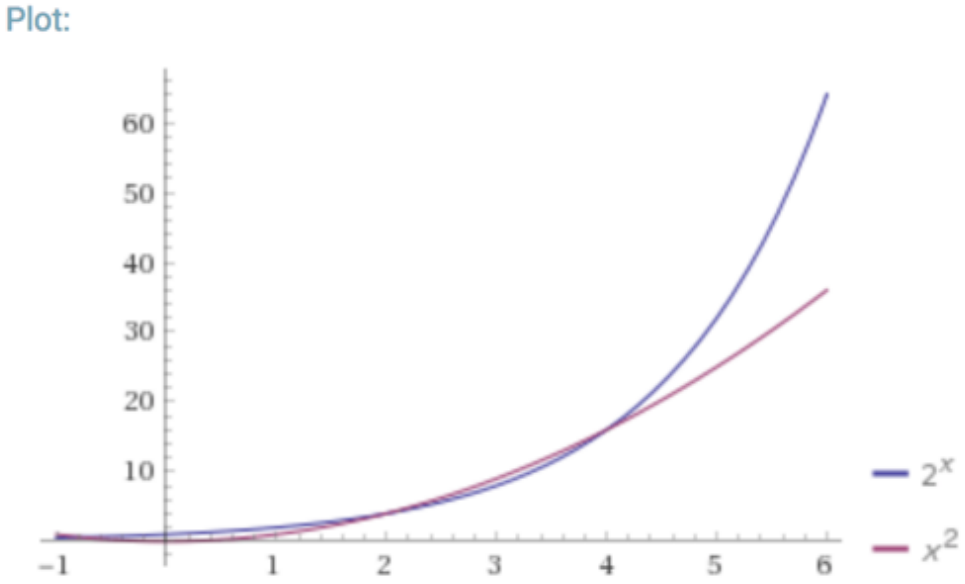
```
                input reached;
                input [20:0] distance;
                input [10:0] hcount;
                input [10:0] vcount;
                output [11:0] pixel_dashboard;
```



依照输入参数

```
distance（distance = (head_x - ball_x)**2 + (head_y - ball_y)**2;）
```

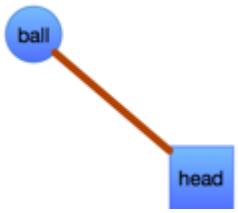
点亮右侧指示色块和确定边界颜色。因为传入参数是距离平方，而使用硬件计算平方根效率不高，所以选择使用以 2 为底的指数函数与距离的二次函数做比较，于是越靠上的色块越难被点亮，靠下的色块在距离较近时有更高灵敏度，取得更精确的结果。



7. Background

```
input                CLK65MHZ;
input  signed [10:0]   hcount;
input         [10:0]   vcount;
input         [10:0]   ball_x;
input         [10:0]   ball_y;
input         [10:0]   head_x;
input         [10:0]   head_y;
output        [11:0]   background_pixel;
```

画出 ball 和 head 之间的橙色连接线，表示目标方向。



考虑到 VGA 逐行扫描的特性，需要计算的参数主要是在每个(hcount, vcount)下的纵向线粗

```
t_y <= thickness * d_y / d_x;
```

和线段中心坐标

```
y_c = (d_y * (hcount - x_s)) / d_x;
if (!((ball_x >= head_x && ball_y >= head_y) ||
   (ball_x)<= head_x && ball_y <= head_y)))
                    y_c = d_y - y_c;
```

若(hcount, vcount)落在

```
$signed(vcount) - $signed(y_s) - $signed(y_c) <= t_y &&
$signed(vcount) - $signed(y_s) - $signed(y_c) >= 0
```

范围内，可以输出橙色像素。

8. Clock Divider

```
module clock_div(
    input clk,
    input [8:0] dps, // Desired degrees per second
    output reg new_clk
     );
```

用于计算步进电机以 dps 表示的每秒旋转度数时需要的时钟周期。步进电机每个整步旋转 1.8°，经计算在系统时钟积累到

```
27'b101_0101_1101_0100_1010_1000_0000 / dps; //
                Calculated magic number
```

时反转输出时钟可以正确输出电机时钟。调用 Clock Divider 模块时只需传入正确的角速度，就可以获得可以直接接入电机的时钟。

9. ACCELEROMETER

```
ADXL362Ctrl accel
(.SYSCLK(CLK100MHZ),.RESET(reset),.ACCEL_X(ACCEL_X),.ACCEL_Y(ACCEL_Y)
,// .Data_Ready(Data_Ready), .MISO(MISO),.SCLK(SCLK),.MOSI(MOSI),.SS(
SS));
```

使用 SPI 协议与板载加速度传感器沟通，传出三轴传感器 32 位范围在(-2g, 2g)的补码。依照说明书，Y 轴需要翻转。

10. SEVEN SEGMENT DISPLAY

```
module display_8hex(
    input clk,              // system clock
    input [31:0] data,        // 8 hex numbers, msb first
    output reg [6:0] seg,     // seven segment display output
    output reg [7:0] strobe    // digit strobe
    );
```

每段数码管用 seg 电量对应灯管，用 strobe 选择点亮第 i 位灯泡。

11. DEBOUNCE

```
module debounce (input reset, clock, noisy,
                output reg clean);
   reg [19:0] count;
   reg new;
   always @(posedge clock)
     if (reset) begin new <= noisy; clean <= noisy; count <= 0; end
     else if (noisy != new) begin new <= noisy; count <= 0; end
     else if (count == 650000) clean <= new;
     else count <= count+1;
endmodule
```

连接开关，防止开关金属片在 6.5ms 内的跳动造成的多次触发。使用触发器输出保持住超过 6.5ms 的按钮信号。

12. SYNCHRONIZE

通过 pipeline 稳定信号。连接系统时钟加速度传感器以 100MHz 频率读取加速度值，这个信号传入运行在 65Mhz 的像素时钟上的 game module 时可能造成 65Mhz 时钟上升沿读取加速度信号时加速度信号正好进行 0-1 转换过程中，造成读入无效值引起加速度信号无效。解决方案是在加速度传感器的信号输出后增加两个以上 100MHz 时钟周期的触发器，同步信号。

```
module synchronize #(parameter NSYNC = 2)
                (input clk,in,
```

```
                output reg out);
  reg [NSYNC-2:0] sync;
  always @ (posedge clk)
  begin
    {out,sync} <= {sync[NSYNC-2:0],in};
  end
endmodule
```

13. 主模块

```
 module top(
  animate,
  test_led,
  CLK100MHZ,
  vga_out_red, vga_out_green, vga_out_blue, vga_out_hsync,
vga_out_vsync,
  switch, reset_n, reset_game,
  MISO,SCLK,MOSI,SS,
  SEG, AN, state,
  // JA, // Gyroscope
  signal_x,
  gravity,
  accel_en,
  manual,

  // audio
  AUD_PWM,
  AUD_SD,
  route,

  // SD card
  SD_CD,
  SD_RESET,
  SD_SCK,
  SD_CMD,
  SD_DAT,
  sd_state
  );
```

连接所有子模块。使用了 6 状态有限状态机，在第三节中已描述。超 500 行，参见 GitHub：https://github.com/Twofyw/fpga_final_project。

五、 测试模块建模

（要求列写各建模模块的 test bench 模块代码）
拍多个 VGA 输出画面和 LED 灯；再跑模拟，复制测试输出。

1. STEPPER MOTOR CONTROLLER

```
 initial begin
   ball_x = planWidth;
   ball_y = planHeight;
   #360
   ball_x = ball_x - 200; ball_y = ball_y - 100;
   #360
   ball_x = 100; ball_y = planHeight;
 end
```

```
$display("running d_x:%d head_x:%d counter:%d count:%d
speed_x:%d",d_x, head_x, counter, count, speed_x);

$display("running d_y:%d head_y:%d counter:%d count:%d
speed_y:%d\n",d_y, head_y, counter, count, speed_y);
```

输出结果是每次电机移动时的状态。

```
# run 1000ns
Initiated
continue d_x: 1021 head_x:   2 counter:        1 mids:  8 speed_x:   2
continue d_y:  766 head_y:   1 counter:        1 mids:  8 speed_y:   1

continue d_x: 1019 head_x:   4 counter:        2 mids:  8 speed_x:   2
continue d_y:  765 head_y:   2 counter:        2 mids:  8 speed_y:   1

.  .  .
.  .  .
.  .  .

continue d_x:  163 head_x:  860 counter:       17 mids:  1 speed_x:   2
continue d_y:  337 head_y:  430 counter:       17 mids:  1 speed_y:   1

continue d_x:  161 head_x:  862 counter:       18 mids:  1 speed_x:   2
continue d_y:  336 head_y:  431 counter:       18 mids:  1 speed_y:   1
```

2. VGA：

```
case (switch[2:0])
     3'b000: rgb <= { 12{border} };
     3'b001: rgb <= { vcount[4:1] + hcount[9:6], hcount[8:5],
hcount[3:0] };
     3'b010: rgb <= pixel_gameboard;
     3'b011: rgb <= picture;
     3'b100: rgb <= pixel_dashboard;
     3'b110: rgb <= pixel;
     3'b111: rgb <= pixel_final;
   endcase
 end
```

用板载 SW 控制输出进行 Alpha Blending 之前的每个像素分量。

3. STEPPER MOTOR CONTROLLER

```
                        assign state = STATE;
```

将运行状态输出至 LED[15:14]，

```
 localparam         [1:0]   State_idle    = 2'b00;
 localparam         [1:0]   State_running  = 2'b01;
 localparam         [1:0]   State_continue = 2'b10;
 localparam         [1:0]   State_init    = 2'b11;
```

指示运行状态。

4. BACKGROUND

```
 initial begin
   hcount = 0;
   vcount = 0;
   // Perpendicular
   // ball_x = 20;
   // ball_y = 20;
   // head_x = 20;
   // head_y = 40;
   // 45 degrees
   ball_x = 20;
   ball_y = 20;
   head_x = 40;
   head_y = 40;
 end
```

对多种 head 和 ball 的相对位置进行测试，通过波形检查着色情况。

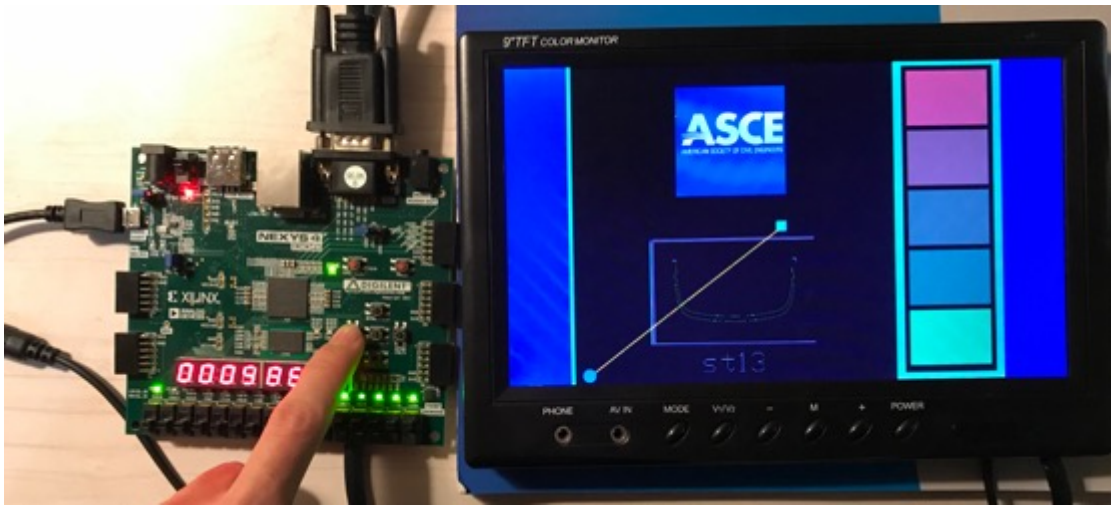
六、 实验结果

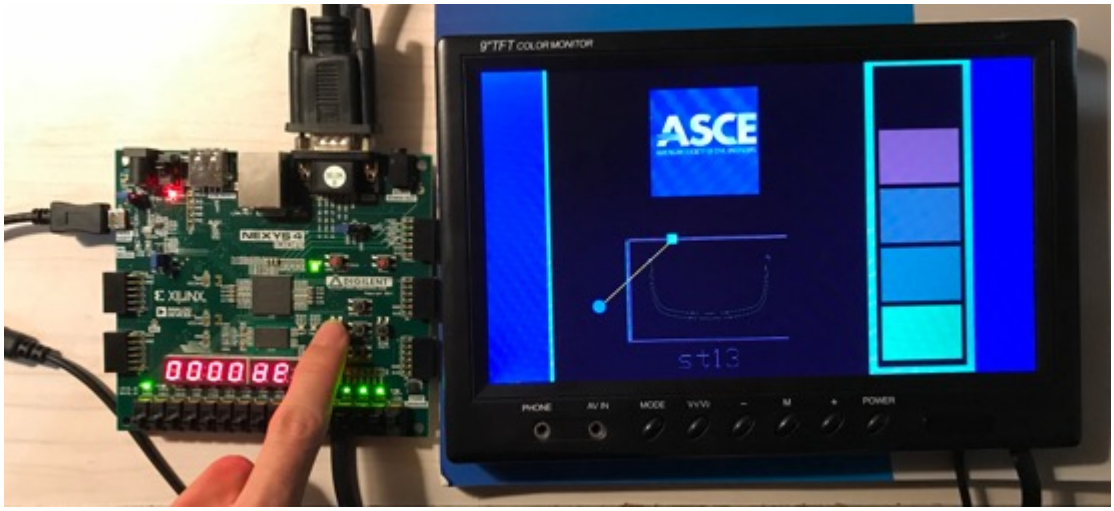
a) 用户界面

启动时画面渐变出现（用 SW[14]复位）。启动完成后电机复位

b) 操作

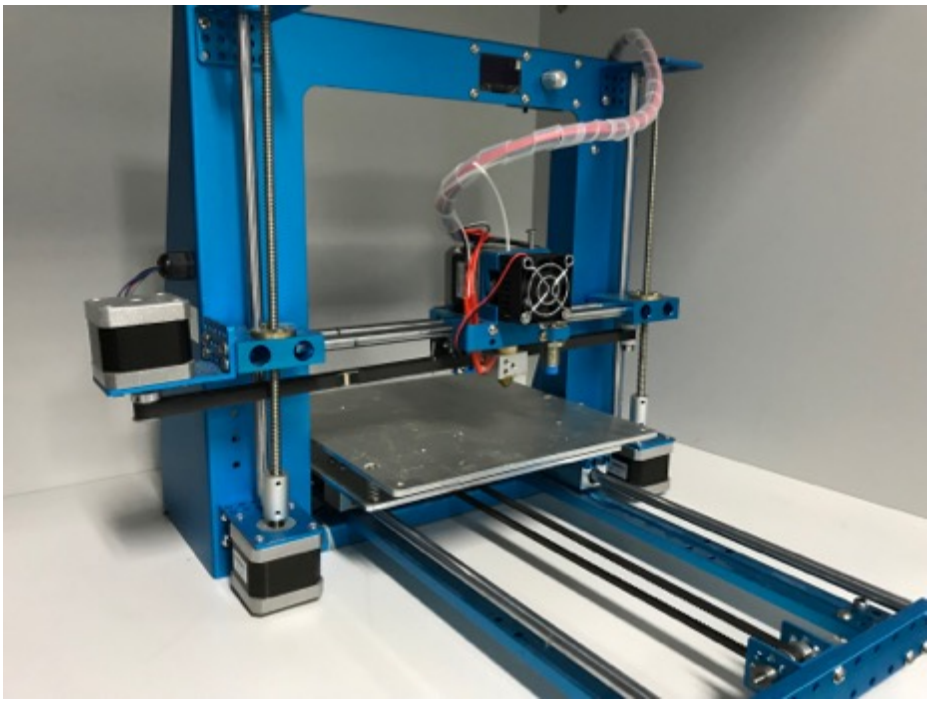
用 SW[5:3]设置 gravity 后按下方向键，球移动，电机追踪球跑。用 SW[15]设置加速度传感器模式后，球在重力作用下弹跳。

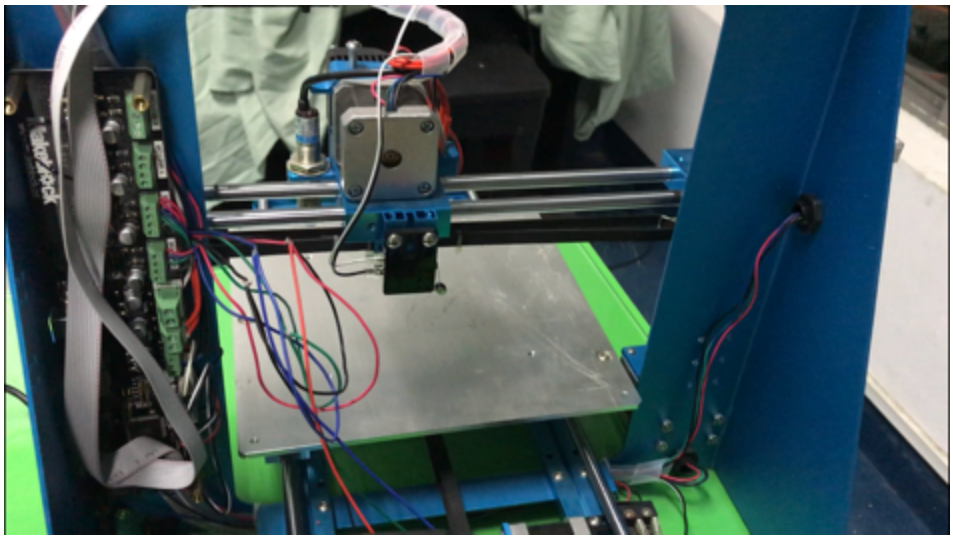




c) 电机

把四根步进电机导线连接上 Pmod Step 后接上 Pmod JB 口，随意移动 ball，平台 X 轴以符合 ball_x 移动速度运行；球停止移动后电机也停止移动。





七、 结论

综合试验依照数字系统设计自顶向下的原则完成了一个应用数字系统设计，以混凝土轻舟队对切模过程自动化的控制需求为动机，尝试了用户界面设计、通过有限状态机的应用设计了 VGA 信号输出、使用 SPI 协议的传感器沟通、电机控制等顶层数字系统功能设计；在电路底层，通过对时序控制的设计比较对阻塞与非阻塞赋值在高频率时钟下的延迟要求有更深的理解；为解决多个计算时不满足 Timing Constrain 要求的设计，对计算的中间结果进行 pipeline 以后显著提升性能，特别是在背景里需要像素时钟周期内传播过除法器的橙色连线和 Alpha Blending 的过程，经过 pipeline 处理流畅度提升显著。

使用 Verilog 硬件描述语言进行数字系统设计，自顶向下的原则极大地加强了模块化设计的调试优势。在设计过程中为每个模块写 Testbench，既防止难以察觉的边界条件在最终成品里过多显现，因为 Testbench 对大部分代码少走过的分支都进行了详尽的测试，而且避免了生成比特流过程大量计算消耗的时间和能源，同时提高了设计过程的可靠性和时间效率。

这次作业未完成的完整自动化 CNC 控制仍待完善。

大号切割框架

八、 心得体会及建议

以切模自动化的实际需求为背景，实现了从用户界面到电机控制的数字系统设计，这次综合试验提供了从了解需求、设计、查找资料、调试修改循环、验证逻辑到总结的全过程，收获颇丰。工程设计让我对硬件描述语言、时钟、pipeline、硬件计算特点等方面都从实践中积累经验，给概念补齐了直觉上的感知。FPGA 作为具有可编程、并行计算的特性，不仅在如这次的小型电路系统设计中体现价值，将来在人工智能领域可发挥编程后并行能力强、专业水平高的特点涉足 TPU 进行神经网络计算等应用，或成为高频交易、图像处理时的一项候选，初步体验 FPGA 设计仅触碰了冰山一角，但这为将来的广泛探寻过程立下了基础。