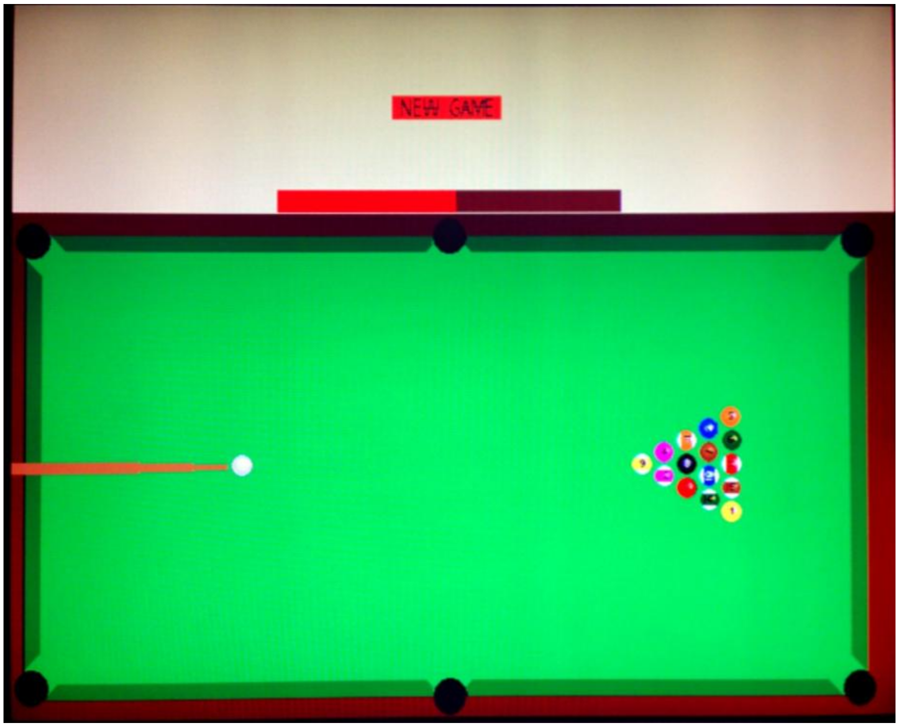
American Pool Video Game

Group Name: Pool-Maniac
CSEE 4840 Embedded System Design



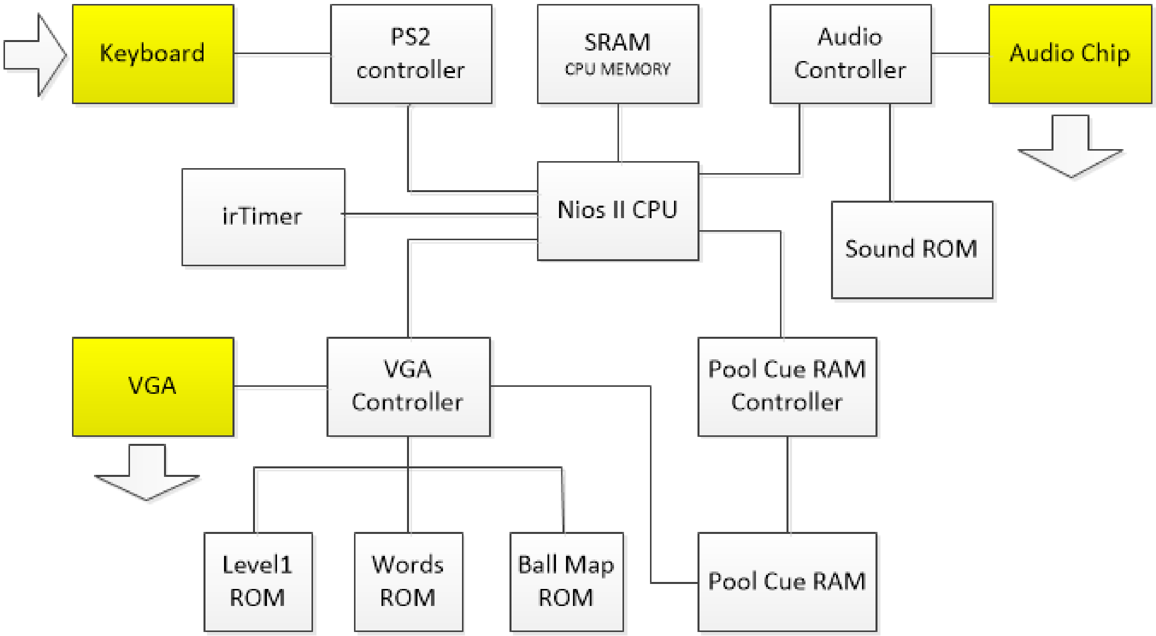
Jiawan Zhang	jz2492
Xunchi Wu	xw2256
Yichen Liu	yl2904
Yuhan Zhang	yz2500
Zeshi Wang	zw2221

1. Project Introduction

In this project, we designed a 2-D American Pool Video Game for two players following the basic American pool rules based on the DE2 board. We have VGA, Keyboard and Audio in our project. We also made the pool game software to realistically simulate the physical movement of the balls on FPGA.

2. Architecture

We wrote the VGA controller, SRAM controller, PS2 controller and Audio Controller to control the peripherals. We also wrote the Pool Cue RAM Controller to use part of the on chip memory as RAM for pool cue display. The irTimer is used to help fix the runtime of the loops.



*The yellow blocks are peripherals.

3. VGA

1. Colors and Sprites

The interface needs only 31 colors, so we made a color table for them. Each color is represented by a 5-bit color code. The color code range from “00001” to “11111” for the 31 colors, and “00000” is saved for transparent. The images are stored with the color codes, so memory could be saved.

The Video display part has 5 sprites (Level 0 to Level 4) concluded as following:

	Contents
Level 0	a. Frames of spool table and serve line b. Strength bar c. Background

Level 1	a. 6 pockets and around areas
Level 2	a. 16 balls
Level 3	a. Pool cue
Level 4	a. Instruction words

Level 4 is the top level and all the way down to Level 0. The level is enabled when the raster scans into the areas of the images on the level and the point is not transparent. If one level is enabled, and none of its upper level is, the RGB outputs of the VGA controller will be given with the color code of this level.

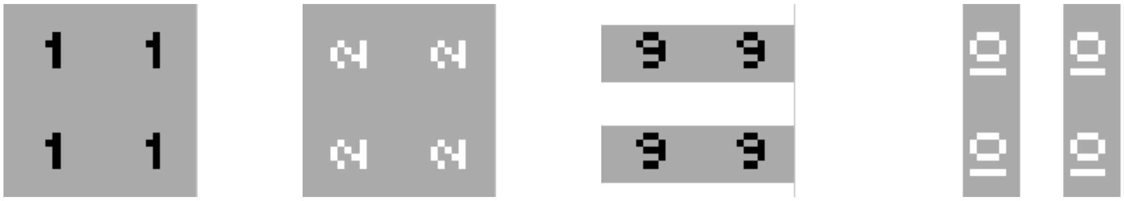
2. Inputs

	Inputs	function
From Nios CPU	x, y positions for 16 balls	Decide the positions of the centers of the balls.
	x, y biases for 16 balls	Decide the positions of the masks related to the ball map for rotation.
	strength	Update the strength bar
	Serve line enable	Show or clear the serve line
	x, y positions for instructions	Decide the positions of the instruction words
	Enable signals for instruction words	Show or hide the instruction words
	Highlight signals for instruction words	Highlight the instruction words
	x, y positions of the instruction words	Give the positions of the instruction words
From Ball_Map_rom	The color codes of ball maps	Give the color code of balls (Level 2) for current pixel
Form Level1_rom	The color codes for pockets and round areas	Give the color code of pockets (Level 1) for current pixel
From Words_rom	The color codes for instruction words	Give the color code of instruction words(Level 4) for current pixel
From PoolCue_ram	The start and end positions for the pool cue image of each line on screen	Give the start and end positions of the pool cue image for the line being scanned (Level 3)

3. Balls

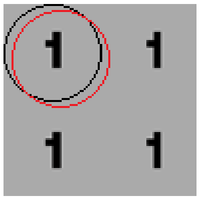
First, in order to make the balls look real, we used three colors for the base image of each ball to give a 3-D visual effect. In the VHDL code, we made a 2-bit 14*14 mask for the balls. "00" represents transparent, "01" represents color 1, "10" represents color 2, and "11" represents color 3. When the raster scans into the area of a ball, we give the color code and level enable signals of this level (Level 2) according to the number in the ball mask and the number of ball.

Second, to make the balls rotate, we made a 2-bit 27*27 map for each ball, which are stored in Ball_Map_rom (on chip memory). Some part of the maps will cover on the base images of the balls. In the map, “00” represents transparent, “10” represents black, and “11” represents white. For example, the map for ball 1, 2, 9 and 10 is shown below.



*Gray represents transparent here.

We also have the ‘biases to map’ of both direction x and y (from 0 to 13) for each ball, which indicate which part of map will be shown on the ball. For example, for ball 1, as shown in the following figure, the black cycle includes the area that will show on the ball, when bias x and y are 0. When the ball moves, we move its bias to the map with an opposite direction. For example, if ball 1 moves up and left by 1 pixel respectively, then the bias increase in both x and y by 1, so the part shown on the ball is now what the red cycle included. This is our trick to make the balls look like rotate.



4. Pool Cue

In order to store enough images for the pool cue’s rotation with limited memory, we made some improvements to reduce the memory needed to store the images. First, we use only 2*355 16-bit data for one image, which gives only the start and end positions of the cue in each line of the image. The data for the images is written in the C code, and we made a PoolCue_ram (using on chip memory) for the VGA controller to use it. The RAM can store 2*480 16-bit data, one start point and one end point for each line on the screen. In the C code, when updating the cue, we first choose an image according the angle of the cue, and then compute the data for the RAM with the image data and the position of the cue, and write it to the RAM. In the VHDL, when scan to certain line, we first read the corresponding data from the RAM, and then enable the level for pool cue (Level 3) from the start position to the end position for each scanning line. The cue is single colored, so a fixed color code is given for this level.

Besides, in order to reduce the number of images needed, we use only 31 images to generate 120 different forms of the cue. The 31 images give all the forms of cue in the 4th quadrant, and generate the forms in other quadrants with mirror image.

5. Others

For all the rectangle things, like table frames and strength bar, we defined the vertexes of them in VHDL, and enable the certain level (Level 0) when the raster scan into the rectangle areas. They are all single color, so their color codes are fixed.

For the pockets (Level 1) and instruction words (Level 4), their images are stored on the Level1_rom (on chip memory) and Words_rom(on chip memory). The data is stored as color codes, and “00000” means transparent. The positions of pockets are written in VHDL as constant .The positions of instruction words are given by the software. When the raster scan into the image area, data will be read from the ROM’s, and generate the enable signals and color codes for VGA controller outputs.

4. Keyboard

The keyboard is the only controller for the pool video game. The players use the PS/2 keyboard to send operations, like starting the game, controlling the position of cue ball and the state of pool cue.

The arrow keys, space key and enter key are used in our game. The arrow keys are used to realize position control, including left, right, up and down. Besides, the arrow key is also used to control the state of pool cue. The up and right key will both realize the clockwise rotate, while down and left key used for anticlockwise. If the player wants to rotate a big angle, he can use the up and down key, and if he just need to tune the pool cue a small angle, the left or right key will be used. The space key is used to adjust the strength of the pool cue, with a red bar shown on the screen to indicate the current strength. After settling the strength and angle, press the enter key and release the pool cue which will hit the cue ball. If a key is hold, the keyboard will continuously send the signal of this key’s code, and this information will be stored in a buffer. When the enter key is pressed and the release code of a key is received, the hold buffer will be cleared and set the angle and strength to its initial value.

The hardware setup for this keyboard controller is to put the PS2 controller based on lab3, into SOPC and connect the PS/2 signal CLK and DATA correspondingly to the top-level board pins.

5. Audio

In our game, the audio block can generate a corresponding sound when the pool cue or the table hits the ball, and the collision happened between different balls. Besides, the volume of the sound is based on the speed of the balls. The collision happened with a fast speed will generate a loud sound while the sound for the slower speed is smaller. Before we use the sound, we utilize the Matlab to quantity the sound. We use 16 bits to represent 1 point of the data. Audio chip clock is 18MHz and sample rate is 48 KHz.

After these preprocess, we got the needy information of the sound and stored them in the on chip ROM with different character bit for different audio data. The total memory used by the sound is 20KB. We successfully solved the interference of the hardware on the Avalon bus and made each part work correctly. When a collision happens, the start audio signal will be generate and enable the corresponding data in the ROM to generate that sound. Once the hardware sending the starting audio signal, the ROM address will start to count and shift out the audio information. When the processes have been done, the address counting buffer will be reset to 0 and the starting audio signal will also set to 0.

In this way, for the software part, we just need to give a start signal to to audio block and generate the corresponding sound according to the writedata signal from the software part. We make the different bit of writedata signal to represent different sound which can generate different volume of the sound. We calculated the ball speed in the software and combined it with the audio writedata interface. When the speed is fast enough, it will send the last bit or the third bit of writedate to be one and enable the correspond louder sound. While the speed is slow, the second or forth bit of writedate signal will be set to one and generate a small sound. If the speed is very small that will generate a sound we cannot heard, there will not be a sound produced.

6. irTimer

The timer is actually a 16-bit down-counter, whose initial data is given by the software. The counter count from the initial data to 0, and stay 0 still a new initial data is sent again. This timer is used to fix the runtime of the ball scan loop. When the software run to the end of the loop, it read the output of the counter, until read a 0. And then write the timer the initial data (3000 for example). We guaranteed that the longest runtime of the loop is small than the time needed for the counter to count from initial data to 0(T_{count}). Therefore, the runtime is fixed to T_{count}, where

$$T_{count} = (1/50MHz) * 3000 = 60us$$

7. Software Design

Software is the control part of the whole project, since all hardware components are functioning according to the commands received from the software program. In our design, software part needs to handle the following situations.

1. The overall game logic control

   Software part controls the stages of the game. stage1: the welcoming at the beginning, stage2: waiting for a cue ball placement, stage3: waiting the player to adjust the angle and strength of the cue ball, stage4: the physical movement of every ball on the table. When all the balls on the table stops, program will go back to stage 3; when the cue ball is pocketed, program will back to stage2; when the black ball 8 is pocketed, program will back to stage1.

2. The collision between balls and the moving parameters afterwards

   Whenever two balls collide with each other, the program will enter the function “bound_balls” and conduct physical calculation on the direction and velocity of every ball involved in this collision incident, and update their corresponding parameters. Also there will be sounds coming out when the collision happens according to their relative vecolity.

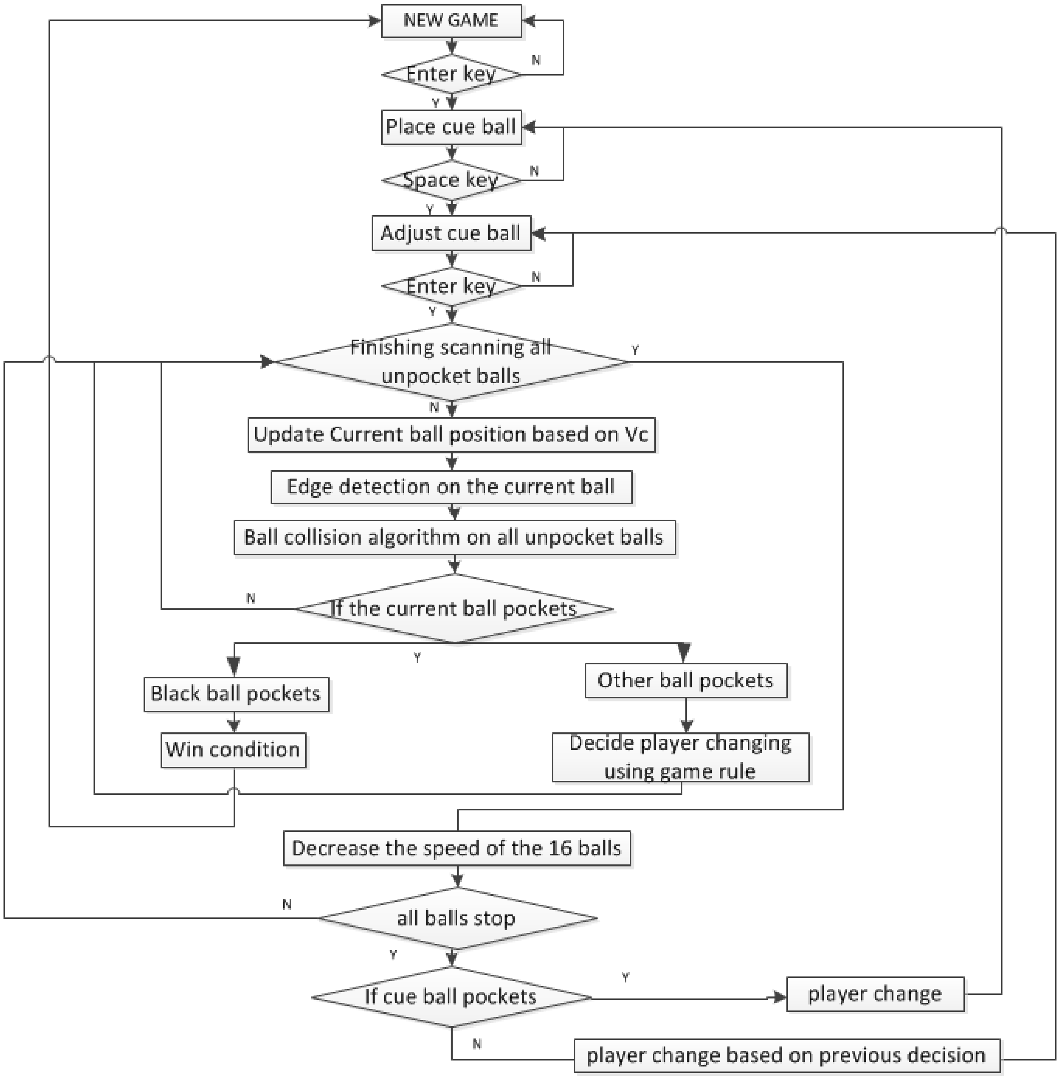
3. The collision between ball and table edges and the moving parameters afterwards.
   Aiming for more realistic performance of the game, we separate the case of the collision with six pockets apart from general collision of the table edge. Because the six pockets have slopes near them, we program a realistic bouncing calculation to this case. The general case of edge collision will follow the mirror reflection.
4. The pocketing of the balls and player changing logic.
   The poolgame rule is also reflected in the software. The change of player will based on whether there's a ball pockets, whether the pocket ball is of the legal type, and whether the cue ball is pocked. And the winning condition will solely base on whether the black ball is pocketed legally.

Challenge Part

After working out the first algorithm for collision and movement between balls (this algorithm is commonly used in computer), we realized the running speed is too slow, and there's serious overlapping of ball pixels. After analyzing the problem, we change a faster CPU platform, and remove and rewrite some commands which cost a long time (etc. pow), the running speed improved significantly, but the overlapping of pixels still exists.

The main reason of the overlapping is: the old "bound_ball" function updates directly the distance that the ball moves for a fixed time, which means the ball will possibly update more than 1 pixel a time (say 5), though it works fine in computer (because the resolution is pretty high in computer, there may be 10 pixels buffer for overlapping, the overlap can be detected and prevented before two balls actually overlap, at least not detected by human eyes.), this method will completely break down in our project, because our VGA resolution is pretty low, and the ball is 14*14 pixels, thus whenever there is a 1 pixel overlap, it will be obvious in human eyes. Therefore, we have to limit the ball movement to only 1 pixel a time. This is the motivation for our own algorithm: "Count Based Speed Control".

The basic idea of our algorithm is: we take in the results from function "bound_ball", and assign the speed (X axis and Y axis) into a variable "Vc" (for actual operation we use "vc" in code). We also make a variable "count", which is given the value of corresponding "Vc" after it count down with step of 1 to 0. The movement operation on this ball (only allow 1 pixel movement a time) will not activate until the variable "count" decreases to 0 inside the ball scan loop (the loop's runtime is fixed with the help of hardware). By this mean we successfully transform the velocity into a countdown-time. The faster the velocity of a ball, the smaller its variable "Vc" will be, meaning it will not wait a long time before it is allowed to make a 1 pixel movement. Then the variable "Vc" is updated once a while with multiplying a parameter (friction in physics), making it possible and realistic to simulate the speed decreasing movement of every ball.


NEW GAME
Enter key
N
Y
Place cue ball
Space key
N
Y
Adjust cue ball
Enter key
N
Y
Finishing scanning all unpocket balls
Y
N
Update Current ball position based on Vc
Edge detection on the current ball
Ball collision algorithm on all unpocket balls
N
If the current ball pockets
Y
Black ball pockets
Other ball pockets
Win condition
Decide player changing using game rule
Decrease the speed of the 16 balls
N
all balls stop
Y
If cue ball pockets
Y
player change
N
player change based on previous decision

Flow Chart of the Software

8. Ethernet & User Interface Exploration

In the project, we also tried to implement the network between the DE2 Board and laptop, with the DE2 Board runs a version with the ball game while the MacBook runs another. The laptop communicates with DE2 board through Ethernet cable. The protocols used for communicating game messages are IP and the unreliable but simple UDP packets.

We first set on testing and allowing communication between the DE2 Board and the MacBook, running the Nios II code on top of the compiled VHDL code provided in lab 2, which includes the Ethernet. We used ARP and designed an appropriate response packet that will assist the MacBook to map our board’s IP address to its physical MAC address. By changing the communication method from port-to-port transportation to UDP broadcasting and allowing the DM9000A some delay time to initialize its configuration before we asked it to send a packet, we can fully receive UDP packets sent from the board. Therefore, the communication between DE2 board and the laptop was successfully set up based on lab2 program.

Additionally, we designed GUI using JAVA programming, and completely finished a separated version of the pool game. But some more work have to be down to successfully setup the Ethernet between the DE2 board and laptop, and this could be solve in the future.

9. Tutorial and rules of this game

Tutorial

(1) Welcoming interface (NEW GAME): press **enter key** to continue.

(2) Main interface:

(a) player can choose the position to place the cue ball along the serve line by pressing **arrow keys.**
Move slowly: left, right arrow
Move quickly: upper arrow, down arrow
Confirm placement: press **space key**

(b) Adjust the pool cue's direction and strength
Left, right arrow: minor adjustment on angle
Upper arrow, down arrow: major adjustment on angle

(c) **Space key:** change the strength (will reverse when it hits the maximum or minimum thresholds).

(d) **Enter key**: release the cue ball.

(3) At the top of the screen the current player will be highlighted, indicating the player's turn. The pocketed ball will be shown accordingly behind the two players, so they can be aware how many balls they still need to pocket.

Rule

Once a player pockets his first ball, he should stick to pocketing the same type of balls (i.e. solid color or stripes), while another player should pocket another group. A player is allowed to continue shooting until he fails to legally pocket a ball of his type. After a player has legally pocketed all his type of balls, he can pocket the 8-ball and win the game. Otherwise, he will lose the game. If the cue ball is pocketed, the player switch, and the next player is allowed to place the cue ball along the serve line and start shooting afterwards. Only when all the balls have stopped moving does it allow player to adjust the pool cue and shoot again.

10. Summary

In this project, we implemented a 2-D American Pool Video Game on DE2 board. The game simulated a physical realistic trajectory of the balls movement, and displayed the game interface on a VGA screen with resolution 640*480. It also followed strictly the traditional American pool game rule.

Moreover, a sound is produced whenever a ball collision happens, and the volume of the sound is linearly dependent on the colliding ball speed. The whole project incorporated the hardware part written in VHDL with the software part written in C language and displayed on the VGA screen. DE2 board FPGA with Nios CPU entity is used as the project platform.

Also, we give a try for an extra function of the project: the network communication between the FPGA and the laptop.

11. **Overview of Personal Responsibilities**

In this project, there were several key components that we need to work out, including VGA driver, Key Board, Audio, Software and Pool game algorithm. We broke down the work according to everyone's interests and skills at the beginning to make clear of each one's responsibility. However, we continuously helped each other as a team.

Jiawan Zhang: She is the group leader of our group and it is she that helped our group work properly and efficiently. She developed the VGA controller and the relative software to use the VGA components. She helped with the main software and came up with the idea of "Count Based Speed Control" method, which is the critical to the game. She also actively joined the debug works of the software. Additionally, she is the problem solver of our group. When there is a problem occurred and she always eager to help and often came up with some creative ideas and successfully solves the problem.

Xunchi Wu: When the problem of balls overlapping happens, he and Jiawan worked together to fix it. When Jiawan brought up the idea of "Count Based Speed Control" method, which turned out to be a turning point for the project, he put this method into practice and implemented in program successfully. He's also in charge of the overall performance of the program. (eg. adding more realistic bouncing movement of balls near the pocket area, main game interface initialization and displaying, etc.) He's hard working and had good comminication.

Yichen Liu: She was in charge of the Ethernet. She used lab 2 and edited the codes to build the communication between DE2 board and laptop (Mac operation). She also made a user interface on the laptop with JAVA. However, she overlooked the importance of hardware and didn't give much attention and time to this project. She failed to inform other members the acture progress of her part, which directly results in our late realization that this part actually can't work.

Yuhan Zhang: She and Xunchi together did the study on the physical movement after collision between two balls, and she successfully worked out a practical algorithm for the project. When realizing the speed of the original version is unacceptably slow, she also improved the algorithm to make it work faster. She also implemented the general rules (including player changing, winning condition, cue ball pocketing, etc.) of the game in the program. She is strict, and we are encouraged by her to make things better.

Zeshi Wang was in charge of the PS2 Keyboard and the audio block in our project. He implemented the hardware for each module with VHDL, and also wrote the C code for these two parts in the software. He successfully debugged several problems both in the hardware and software and made the system work correctly. He is cooperative, friendly and always eager and able to help. After finished his own works in the keyboard and audio, he still worked on the project, and helped others.

VHDL Codes:

-- Top Level

```
--Editor: Zeshi Wang; Jiawan Zhang
-- Data: 2013

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity top_level is
port (
signal CLOCK_50 : in std_logic; --50 MHz
--signal LEDR : out std_logic_vector(17 downto 0); --LEDs
-- PS/2 port
   PS2_DAT,                  -- Data
   PS2_CLK : in std_logic;     -- Clock

         SRAM_DQ : inout std_logic_vector(15 downto 0);
         SRAM_ADDR : out std_logic_vector(17 downto 0);
         SRAM_UB_N, --Highbyte Data Mask
         SRAM_LB_N, --Lowbyte Data Mask
         SRAM_WE_N, --Write Enable
         SRAM_CE_N, --Chip Enable
         SRAM_OE_N : out std_logic; --Output Enable

         --vga
   VGA_CLK,                                -- Clock
   VGA_HS,                                 -- H_SYNC
   VGA_VS,                                 -- V_SYNC
   VGA_BLANK,                                 -- BLANK
   VGA_SYNC : out std_logic;                     -- SYNC
   VGA_R,                                  -- Red[9:0]
   VGA_G,                                  -- Green[9:0]
   VGA_B : out std_logic_vector(9 downto 0);                -- Blue[9:0]

  -- Audio CODEC

   AUD_ADCLRCK : inout std_logic;                   -- ADC LR Clock
```

```
    AUD_ADCDAT : in std_logic;                         -- ADC Data
    AUD_DACLRCK : inout std_logic;                      -- DAC LR Clock
    AUD_DACDAT : out std_logic;                        -- DAC Data
    AUD_BCLK : inout std_logic;                       -- Bit-Stream Clock
    AUD_XCK : out std_logic;                         -- Chip Clock

-- I2C bus

    I2C_SDAT : inout std_logic; -- I2C Data
    I2C_SCLK : out std_logic;   -- I2C Clock

--  Ethernet Interface

    ENET_DATA : inout std_logic_vector(15 downto 0);    -- DATA bus 16Bits
    ENET_CMD,           -- Command/Data Select, 0 = Command, 1 = Data
    ENET_CS_N,                                  -- Chip Select
    ENET_WR_N,                                  -- Write
    ENET_RD_N,                                  -- Read
    ENET_RST_N,                                 -- Reset
    ENET_CLK : out std_logic;                       -- Clock 25 MHz
    ENET_INT : in std_logic;                        -- Interrupt


            LEDG : out std_logic_vector(8 downto 0);       -- Green LEDs
    LEDR : out std_logic_vector(17 downto 0)      -- Red LED
);
end top_level;

architecture rtl of top_level  is

signal counter : unsigned(15 downto 0);
signal reset_n : std_logic;

  signal clk25 : std_logic := '0';
  signal audio_clock : unsigned(1 downto 0) := "00";
  signal network_clock : unsigned(1 downto 0) := "00";
  signal clk_18 : std_logic;

--signals for PoolCue_ram
```

```
signal CueRam_rdaddress_VGA : std_logic_vector(8 downto 0);
signal CueRam_rdclk_VGA : std_logic;
signal CueRam_q_VGA : std_logic_vector(31 downto 0);
signal CueRam_rdaddress_RAM : std_logic_vector(8 downto 0);
signal CueRam_rdclk_RAM : std_logic;
signal CueRam_q_RAM : std_logic_vector(31 downto 0);


  component de2_i2c_av_config is
  port (
    iCLK : in std_logic;
    iRST_N : in std_logic;
    I2C_SCLK : out std_logic;
    I2C_SDAT : inout std_logic
  );
  end component;


begin
        CueRam_rdaddress_RAM <= CueRam_rdaddress_VGA;
        CueRam_rdclk_RAM <= CueRam_rdclk_VGA;
        CueRam_q_VGA <= CueRam_q_RAM;

        process (CLOCK_50)
        begin
                if rising_edge(CLOCK_50) then
                        if counter = x"ffff" then
                                reset_n <= '1';
                        else
                                reset_n <= '0';
                        counter <= counter + 1;
                        end if;
                end if;
        end process;


process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
```

```
              if audio_clock = "11" then
                   audio_clock <= "00";
                                     AUD_XCK <= '1';
              else
      audio_clock <= audio_clock + "1";
                        AUD_XCK <= '0';
                        end if;
    end if;
  end process;

process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
              if network_clock = "01" then
                   network_clock <= "00";
                                     ENET_CLK <= '1';
              else
      network_clock <= network_clock + "1";
                        ENET_CLK <= '0';
                        end if;
    end if;
  end process;

  i2c : de2_i2c_av_config port map (
    iCLK     => CLOCK_50,
    iRST_n   => '1',
    I2C_SCLK => I2C_SCLK,
    I2C_SDAT => I2C_SDAT
  );

nios : entity work.nios_system port map (
            -- the_de2_ps2_inst
            PS2_Clk_to_the_de2_ps2_0  => PS2_CLK,
            PS2_Data_to_the_de2_ps2_0 => PS2_DAT,

            clk_0 => CLOCK_50,
            reset_n => reset_n,
            --leds_from_the_leds => LEDR(15 downto 0),
            SRAM_ADDR_from_the_sram_0 => SRAM_ADDR,
            SRAM_CE_N_from_the_sram_0 => SRAM_CE_N,
            SRAM_DQ_to_and_from_the_sram_0 => SRAM_DQ,
            SRAM_LB_N_from_the_sram_0 => SRAM_LB_N,
            SRAM_OE_N_from_the_sram_0 => SRAM_OE_N,
            SRAM_UB_N_from_the_sram_0 => SRAM_UB_N,
            SRAM_WE_N_from_the_sram_0 => SRAM_WE_N,


            --the audio instruction
             AUD_ADCDAT_to_the_audio_0       => AUD_ADCDAT,
             AUD_ADCLRCK_from_the_audio_0    => AUD_ADCLRCK,
             AUD_BCLK_to_and_from_the_audio_0 => AUD_BCLK,
             AUD_DACDAT_from_the_audio_0     => AUD_DACDAT,
             AUD_DACLRCK_from_the_audio_0    => AUD_DACLRCK,


    --  Ethernet Interface


    ENET_DATA_to_and_from_the_DM9000A_0     => ENET_DATA,
    ENET_CMD_from_the_DM9000A_0             => ENET_CMD,
    ENET_CS_N_from_the_DM9000A_0                        => ENET_CS_N,
    ENET_WR_N_from_the_DM9000A_0                        => ENET_WR_N,
    ENET_RD_N_from_the_DM9000A_0                  => ENET_RD_N,
    ENET_RST_N_from_the_DM9000A_0                         => ENET_RST_N,
    ENET_INT_to_the_DM9000A_0                           => ENET_INT,

            -- the_de2_vga_raster
            VGA_BLANK_from_the_de2_vga_raster_0 => VGA_BLANK,
            VGA_B_from_the_de2_vga_raster_0 => VGA_B,
            VGA_CLK_from_the_de2_vga_raster_0 => VGA_CLK,
            VGA_G_from_the_de2_vga_raster_0 => VGA_G,
            VGA_HS_from_the_de2_vga_raster_0 => VGA_HS,
            VGA_R_from_the_de2_vga_raster_0 => VGA_R,
            VGA_SYNC_from_the_de2_vga_raster_0 => VGA_SYNC,
            VGA_VS_from_the_de2_vga_raster_0 => VGA_VS,
            CUERAM_addr_from_the_de2_vga_raster_0 => CueRam_rdaddress_VGA,
            CUERAM_clk_from_the_de2_vga_raster_0 => CueRam_rdclk_VGA,
            CUERAM_q_to_the_de2_vga_raster_0 => CueRam_q_VGA,
               --LEDD
            LEDG_from_the_de2_vga_raster_0 => LEDG,
     LEDR_from_the_de2_vga_raster_0 => LEDR,

            -- PoolCue_ram
     CUE_q_from_the_PoolCue_ram_controller_0 => CueRam_q_RAM,
            CUE_rdaddress_to_the_PoolCue_ram_controller_0 => CueRam_rdaddress_RAM,
     CUE_rdclock_to_the_PoolCue_ram_controller_0 => CueRam_rdclk_RAM

  );
  end rtl;
```

-- **VGA Controller**

```
-------------------------------------------------------------------------------
--
-- VGA controller for American Pool Game
--
-- Eidtor: Jiawan Zhang
-- Data: 2013
--
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_raster is

  port (
           --****************************************************
    reset : in std_logic;
    clk50   : in std_logic;                    -- Should be 25.125 MHz

           read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  std_logic_vector(6 downto 0); --7 bits 128 addresses
    readdata   : out std_logic_vector(15 downto 0);
    writedata  : in  std_logic_vector(15 downto 0);
```

```
          irq : out std_logic;
          --******************************************************
   VGA_CLK,                     -- Clock
   VGA_HS,                      -- H_SYNC
   VGA_VS,                      -- V_SYNC
   VGA_BLANK,                       -- BLANK
   VGA_SYNC : out std_logic;        -- SYNC
   VGA_R,                       -- Red[9:0]
   VGA_G,                       -- Green[9:0]
   VGA_B : out std_logic_vector(9 downto 0); -- Blue[9:0]
          CUERAM_clk : out std_logic;
          CUERAM_addr : out std_logic_vector(8 downto 0);
          CUERAM_q : in std_logic_vector(31 downto 0);
          LEDR : out std_logic_vector (17 downto 0);
          LEDG : out std_logic_vector (8 downto 0)
   );

end de2_vga_raster;

architecture rtl of de2_vga_raster is

  component Ball_Map_rom port (
          address : in std_logic_vector(13 downto 0);
          clock : in std_logic;
          q : out std_logic_vector(1 downto 0)
  );
          end component;

          component PoolCue1_rom port (
          address : in std_logic_vector(15 downto 0);
          clock : in std_logic;
          q : out std_logic_vector(3 downto 0)
  );
          end component;

          component Level1_rom port (
          address : in std_logic_vector(13 downto 0);
          clock : in std_logic;
          q : out std_logic_vector(4 downto 0)
```

```
 );
        end component;

        component Words_rom port
        (
                address         : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
                clock           : IN STD_LOGIC  := '1';
                q               : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
        );
        end component;

--clock
signal clk : std_logic := '0';
signal clk_count : unsigned(1 downto 0) := "00";

--Interrupt
signal Interrupt : std_logic;

-- Video parameters
--Keep as oringinal
constant HTOTAL       : integer := 800;
constant HSYNC        : integer := 96;
constant HBACK_PORCH  : integer := 48;
constant HACTIVE      : integer := 640;
constant HFRONT_PORCH : integer := 16;

constant VTOTAL       : integer := 525;
constant VSYNC        : integer := 2;
constant VBACK_PORCH  : integer := 33;
constant VACTIVE      : integer := 480;
constant VFRONT_PORCH : integer := 10;
--End: Keep as oringinal

-- Signals for the video controller
--Keep as oringinal
signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)
signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)
signal EndOfLine, EndOfField : std_logic;
signal vga_hblank, vga_hsync, vga_vblank, vga_vsync : std_logic;  -- Sync. signals
--End: Keep as oringinal

--Color table
type colortable_type is array(1 to 31) of std_logic_vector(7 downto 0);
constant COLOR_TABLE_R : colortable_type :=(
"00110010", -- 1. table bed color : limegreen
"11001101", -- 2. pool cue color !!!!
"11111111", -- 3. yellow 1
"11111111", -- 4. yellow 2
"11111010", -- 5. yellow 3
"00000000", -- 6. blue 1
"00000000", -- 7. blue 2
"10100101", -- 8. blue 3
"11111111", -- 9. red 1
"11111111", -- 10. red 2
"11111111", -- 11. red 3
"11011100", -- 12. purple 1
"11101011", -- 13. purple 2
"11101110", -- 14. purple 3
"11111111", -- 15. orange 1
"11111111", -- 16. orange 2
"11111111", -- 17. orange 3
"00000000", -- 18. green 1
"00000000", -- 19. green 2
"11011010", -- 20. green 3
"10100000", -- 21. brown 1
"10110100", -- 22. brown 2
"11110100", -- 23. brown 3
"00000000", -- 24. black 1
"00101000", -- 25. black 2
"01101001", -- 26. black 3
"11100000", -- 27. while 1
"11110101", -- 28. while 2
"11111111", -- 29. while 3
"10000000", -- 30. table edge 1: marron
"01000000" -- 31. table edge 2: oliverdrab
);
```

```
constant COLOR_TABLE_G : colortable_type :=(
"11001101", -- 1. table bed color : limegreen
"10000101", -- 2. pool cue color
"11110000", -- 3. yellow 1
"11111111", -- 4. yellow 2
"11111010", -- 5. yellow 3
"00000000", -- 6. blue 1
"00000000", -- 7. blue 2
"11010000", -- 8. blue 3
"00000000", -- 9. red 1
"01000101", -- 10. red 2
"10100000", -- 11. red 3
"00010101", -- 12. purple 1
"00000000", -- 13. purple 2
"10000010", -- 14. purple 3
"10001100", -- 15. orange 1
"10100101", -- 16. orange 2
"11100100", -- 17. orange 3
"01100100", -- 18. green 1
"01110110", -- 19. green 2
"11001101", -- 20. green 3
"01010010", -- 21. brown 1
"01101001", -- 22. brown 2
"10100100", -- 23. brown 3
"00000000", -- 24. black 1
"00101000", -- 25. black 2
"01101001", -- 26. black 3
"11100000", -- 27. while 1
"11110101", -- 28. while 2
"11111111", -- 29. while 3
"00000000", -- 30. table edge 1: marron
"10001110" -- 31. table edge 2: oliverdrab
);

constant COLOR_TABLE_B : colortable_type :=(
"00110010", -- 1. table bed color : limegreen
"00111111", -- 2. pool cue color
"00000000", -- 3. yellow 1
"00000000", -- 4. yellow 2
```

```
  "10110100", -- 5. yellow 3
  "11001101", -- 6. blue 1
  "11111111", -- 7. blue 2
  "11000110", -- 8. blue 3
  "00000000", -- 9. red 1
  "00000000", -- 10. red 2
  "01111010", -- 11. red 3
  "11001000", -- 12. purple 1
  "11101011", -- 13. purple 2
  "11101110", -- 14. purple 3
  "00000000", -- 15. orange 1
  "00000000", -- 16. orange 2
  "10110101", -- 17. orange 3
  "00000000", -- 18. green 1
  "00000000", -- 19. green 2
  "00110010", -- 20. green 3
  "00101101", -- 21. brown 1
  "00011110", -- 22. brown 2
  "01100000", -- 23. brown 3
  "00000000", -- 24. black 1
  "00101000", -- 25. black 2
  "01101001", -- 26. black 3
  "11100000", -- 27. while 1
  "11110101", -- 28. while 2
  "11111111", -- 29. while 3
  "00000000", -- 30. table edge 1: marron
  "00100011" -- 31. table edge 2: oliverdrab
  );

  --VGA Ram
  type ram_type is array(127 downto 0) of
          std_logic_vector(15 downto 0);
  signal VGA_RAM : ram_type;
  signal vga_ram_address : unsigned(6 downto 0);

  signal x_coord, y_coord : unsigned(9 downto 0);  --scan positions in screen
  signal level0_active, level1_active, level2_active, level3_active : std_logic;  -- = 1 means
certain level has something to print
```

```
--Level 0
--Pool Table parameters
constant TABLE1_HSTART : integer := 0;
constant TABLE1_HEND   : integer := 639;
constant TABLE1_VSTART : integer := 140;
constant TABLE1_VEND   : integer := 479;

constant TABLE2_HSTART : integer := 20;
constant TABLE2_HEND   : integer := 619;
constant TABLE2_VSTART : integer := 155;
constant TABLE2_VEND   : integer := 464;

constant TABLE3_HSTART : integer := 32;
constant TABLE3_HEND   : integer := 607;
constant TABLE3_VSTART : integer := 166;
constant TABLE3_VEND   : integer := 453;

signal table1 : std_logic;  -- table areas
signal table2 : std_logic;  -- table areas
signal table3 : std_logic;  -- table areas
-- strength bar
constant SBAR_HSTART : integer := 197;
constant SBAR_HEND   : integer := 444;
constant SBAR_VSTART : integer := 124;
constant SBAR_VEND   : integer := 138;
signal SBAR_Strength : integer;

signal Strength : unsigned(7 downto 0) := "00011100";  --!!from ram  32 levels use 5 bits

signal sbar_h, sbar_v, sbar : std_logic;  -- strength bar areas
signal sstrengh_h, sstrengh_v, sstrengh : std_logic;  -- strength bar areas

-- Serve line
constant SERVELINE_H : integer := 172;
signal serveline_en : std_logic;
signal serveline : std_logic;

signal colorcode_level0 : unsigned(4 downto 0);
```

```
--Level 1
signal level1_address : unsigned(13 downto 0);
signal level1_data, level1_data_vga : std_logic_vector(4 downto 0);
signal level1_vga : std_logic;
signal pocket_flag1, pocket_flag2 : std_logic_vector(5 downto 0);
type pocket_eachaddrtype is array(0 to 5) of unsigned(10 downto 0);
signal pocket_eachaddr : pocket_eachaddrtype;

--Level 2

type ball_type is array(0 to 195) of unsigned(1 downto 0);
constant BallMask : ball_type :=(
        "00", "00","00","00","00","01","01","01","01","00", "00","00","00","00",  --1
        "00", "00","00","01","01","01","01","01","01","01", "01","00","00","00",  --2
        "00", "00","01","01","01","10","10","10","10","10", "01","01","00","00",  --3
        "00", "01","01","01","10","10","10","10","10","10", "10","01","01","00",  --4
        "00", "01","01","10","10","10","10","10","11","11", "10","10","01","00",  --5
        "01", "01","01","10","10","10","10","11","11","11", "11","10","01","01",  --6
        "01", "01","01","10","10","10","10","11","11","11", "11","10","01","01",  --7
        "01", "01","01","10","10","10","10","10","11","11", "10","10","01","01",  --8
        "01", "01","01","01","10","10","10","10","10","10", "10","01","01","01",  --9
        "00", "01","01","01","01","10","10","10","10","10", "01","01","01","00",  --10
        "00", "01","01","01","01","01","01","01","01","01", "01","01","01","00",  --11
        "00", "00","01","01","01","01","01","01","01","01", "01","01","00","00",  --12
        "00", "00","00","01","01","01","01","01","01","01", "01","00","00","00",  --13
        "00", "00","00","00","00","01","01","01","01","00", "00","00","00","00"  --14
        );

type ball_mask_addrtype is array(0 to 15) of integer;
signal ball_mask_addr : ball_mask_addrtype;
signal ball_mask_addr2 : ball_mask_addrtype;
type ball_map_addrtype is array(0 to 15) of unsigned(9 downto 0);
signal ball_map_addr : ball_map_addrtype;

signal ball_map_address : unsigned(13 downto 0);
--signal ball1_mask_addr : integer;
signal ball_data : std_logic_vector(1 downto 0);
signal ball_data_vga : integer;
--signal ball1_map_addr : unsigned(9 downto 0);
```

```
  signal ball_mask_flag1 : std_logic_vector(15 downto 0);
  signal ball_mask_flag2 : std_logic_vector(15 downto 0);
  signal level2_vga : std_logic;
  signal ball_mask_color : integer;
  type ball_pos is array(0 to 15) of unsigned(15 downto 0);
  signal BALL_X : ball_pos;
  signal BALL_Y : ball_pos;
  type ball_bias is array(0 to 15) of unsigned(3 downto 0);
  signal BALL_BIAS_X : ball_bias;
  signal BALL_BIAS_Y : ball_bias;

  --Level 3 Pool cue
  signal cue_line_begin : unsigned(15 downto 0) := "0000000000001000";
  signal cue_line_end : unsigned(15 downto 0) := "0000000000011000";

  signal level3_vga : std_logic;

  --Level 4 Words
  type word_type is array(0 to 7) of unsigned(15 downto 0);
  signal Word_start_x : word_type;
  signal Word_start_y : word_type;
  type word_eachaddrtype is array(0 to 7) of unsigned(9 downto 0);
  signal Word_eachaddr : word_eachaddrtype;
  signal Word_address : unsigned(12 downto 0);
  signal Word_en : std_logic_vector(7 downto 0);
  signal Word_hl_en : std_logic_vector(7 downto 0);   --hight light
  signal Word_hl : std_logic;
  signal Word_flag1, Word_flag2 : std_logic_vector(7 downto 0);
  signal Word_data : std_logic_vector(0 downto 0);
  signal Word_color : integer;
  signal level4_vga : std_logic;
  signal level4_active : std_logic;

begin
  x_coord <= Hcount - (HSYNC + HBACK_PORCH) + 1;
  y_coord <= Vcount - (VSYNC + VBACK_PORCH);

  CUERAM_clk <= clk;
  irq <= Interrupt;

  --25MHz clock generator
  CLOCK25: process(clk50)
  begin
      if rising_edge(clk50) then
          clk <= not clk;

      end if;
  end process;

  --VGA Ram
  vga_ram_address <= unsigned(address(6 downto 0));
  VGARAM : process(clk50)
  begin
      if rising_edge(clk50) then
          if reset = '1' then
              readdata <= (others => '0');
          else
              if chipselect = '1' then
                  if read = '1' then
--       readdata <= VGA_RAM(to_integer(vga_ram_address));

      elsif write = '1' then
       VGA_RAM(to_integer(vga_ram_address)) <= writedata;
      end if;
              end if;
          end if;

      end if;
  end process VGARAM;

  Updata : process(clk)
  begin
   if rising_edge(clk) then
       if reset = '1' then
              Strength <= "00011100";
              for I in 0 to 15 loop
```

```
                BALL_X(I) <= "0000000000000000";
                BALL_Y(I) <= "0000000000000000";
                BALL_BIAS_X(I) <= "0000";
                BALL_BIAS_Y(I) <= "0000";
            end loop;

            Word_en <= "00000000";
            Word_hl_en <= "00000000";
            for I in 0 to 7 loop
                Word_start_x(I) <= "0000000000000000";
                Word_start_y(I) <= "0000000000000000";
            end loop;

            Interrupt <= '0';

            elsif (Hcount = 0 and Vcount = 0) then
                for I in 0 to 15 loop
                    BALL_X(I) <= unsigned(VGA_RAM(4*I + 0));
                    BALL_Y(I) <= unsigned(VGA_RAM(4*I + 1));
                    if unsigned(VGA_RAM(4*I + 2)) <= 13 then
                        BALL_BIAS_X(I) <= unsigned(VGA_RAM(4*I +
2)(3 downto 0));
                    else
                        BALL_BIAS_X(I) <= "0000";
                    end if;

                    if unsigned(VGA_RAM(4*I + 3)) <= 13 then
                        BALL_BIAS_Y(I) <= unsigned(VGA_RAM(4*I +
3)(3 downto 0));
                    else
                        BALL_BIAS_Y(I) <= "0000";
                    end if;
                end loop;

                Strength <= unsigned(VGA_RAM(64)(7 downto 0));  --Strength:
Addr 1 (7 to 0)
                serveline_en <= VGA_RAM(65)(0);  --Enable show serveline

                Word_en <= VGA_RAM(66)(7 downto 0);
                Word_hl_en <= VGA_RAM(67)(7 downto 0);

                for I in 0 to 7 loop
                    Word_start_x(I) <= unsigned(VGA_RAM(68 + 2*I));
                    Word_start_y(I) <= unsigned(VGA_RAM(68 + 2*I + 1));
                end loop;

                Interrupt <= '1';
            else
                Interrupt <= '0';

        end if;
    end if;
end process Updata;

LEDR(17) <= Interrupt;

-- Horizontal and vertical counters
HCounter : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      Hcount <= (others => '0');
    elsif EndOfLine = '1' then
      Hcount <= (others => '0');
    else
      Hcount <= Hcount + 1;
    end if;
  end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter: process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      Vcount <= (others => '0');
    elsif EndOfLine = '1' then
```

```
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' or EndOfLine = '1' then
      vga_hsync <= '1';
    elsif Hcount = HSYNC - 1 then
      vga_hsync <= '0';
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_hblank <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH then
      vga_hblank <= '0';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
      vga_hblank <= '1';
    end if;
  end if;
end process HBlankGen;

VSyncGen : process (clk)
begin
```

```
  if rising_edge(clk) then
    if reset = '1' then
      vga_vsync <= '1';
    elsif EndOfLine ='1' then
      if EndOfField = '1' then
        vga_vsync <= '1';
      elsif Vcount = VSYNC - 1 then
        vga_vsync <= '0';
      end if;
    end if;
  end if;
end process VSyncGen;

VBlankGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_vblank <= '1';
    elsif EndOfLine = '1' then
      if Vcount = VSYNC + VBACK_PORCH - 1 then
        vga_vblank <= '0';
      elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
        vga_vblank <= '1';
      end if;
    end if;
  end if;
end process VBlankGen;

-- Rectangle generator

-- Generate Table flags
Level0_Gen : process (clk)
variable tempStrength : unsigned(7 downto 0);
begin

  if rising_edge(clk) then
                      --serveline
                      if reset = '1' then
                                  serveline <= '0';
        elsif serveline_en = '1' and x_coord = SERVELINE_H and y_coord >=
TABLE3_VSTART and y_coord <= TABLE3_VEND then
            serveline <= '1';
        else
            serveline <= '0';
        end if;

        --table1
        if reset = '1' then
            table1 <= '0';
        elsif x_coord >= TABLE1_HSTART and x_coord <= TABLE1_HEND and
y_coord >= TABLE1_VSTART and y_coord <= TABLE1_VEND then
            table1 <= '1';
        else
            table1 <= '0';
        end if;

        --table2
        if reset = '1' then
            table2 <= '0';
        elsif x_coord >= TABLE2_HSTART and x_coord <= TABLE2_HEND and
y_coord >= TABLE2_VSTART and y_coord <= TABLE2_VEND then
            table2 <= '1';
        else
            table2 <= '0';
        end if;

        --table3
        if reset = '1' then
            table3 <= '0';
        elsif x_coord >= TABLE3_HSTART and x_coord <= TABLE3_HEND and
y_coord >= TABLE3_VSTART and y_coord <= TABLE3_VEND then
            table3 <= '1';
        else
            table3 <= '0';
        end if;


        -- Strength Bar
```

```
            tempStrength(2 downto 0) := "000";
            tempStrength(7 downto 3) := Strength(4 downto 0);
            SBAR_Strength <= SBAR_HSTART + to_integer(tempStrength);

            if reset = '1' or y_coord = SBAR_VEND then
    sbar_v <= '0';
             sstrengh_v <= '0';
   elsif y_coord = SBAR_VSTART then
    sbar_v <= '1';
             sstrengh_v <= '1';
   end if;

            if reset = '1' then
                  sbar <= '0';
            elsif x_coord >= SBAR_HSTART and x_coord <= SBAR_HEND and
y_coord >= SBAR_VSTART and y_coord <= SBAR_VEND then
                  sbar <= '1';
            else
                  sbar <= '0';
            end if;

            if reset = '1' then
                  sstrengh <= '0';
            elsif x_coord >= SBAR_HSTART and x_coord <= SBAR_Strength and
y_coord >= SBAR_VSTART and y_coord <= SBAR_VEND then
                  sstrengh <= '1';
            else
                  sstrengh <= '0';
            end if;

            level0_active <= table1 or table2 or table3 or sbar or sstrengh;

            if serveline = '1' then
                  colorcode_level0 <= "11101";
            elsif table3 = '1' then
                  colorcode_level0 <= "00001";
            elsif table2 = '1' then
                  colorcode_level0 <= "11111";
            elsif table1 = '1' then
```

```
                    colorcode_level0 <= "11110";
              elsif sstrengh = '1' then
                    colorcode_level0 <= "01001";
              elsif sbar = '1' then
                    colorcode_level0 <= "10101";
              end if;
   end if;
  end process Level0_Gen;
  -----------------------------------------------------------------------------------------
  -----------------------------------------------------------------------------------------
  --Level 1
  Level1_rom_inst : Level1_rom port map(
        address => std_logic_vector(level1_address),
        clock => clk,
        q => level1_data
        );

  Level1_FLAG_Gen : process(clk)
   begin
        if rising_edge(clk) then
              if reset = '1' then
                    for I in 0 to 5 loop
                          pocket_flag1(I) <= '0';
                    end loop;
              else
                    if(x_coord + 2 >= TABLE3_HSTART - 19 and x_coord + 2 <
TABLE3_HSTART + 16 and y_coord >= TABLE3_VSTART - 19 and y_coord <
TABLE3_VSTART + 16) then
                          pocket_flag1(0) <= '1';  --pocket 1
                    else
                          pocket_flag1(0) <= '0';
                    end if;
                    if(x_coord + 2 >= TABLE3_HSTART + 272 and x_coord + 2 <
TABLE3_HSTART + 307 and y_coord >= TABLE3_VSTART - 23 and y_coord <
TABLE3_VSTART + 12) then
                          pocket_flag1(1) <= '1';  --pocket 2
                    else
                          pocket_flag1(1) <= '0';
                    end if;
```

```
                        if(x_coord + 2 >= TABLE3_HEND - 16 and x_coord + 2 <
TABLE3_HEND + 19 and y_coord >= TABLE3_VSTART - 19 and y_coord <
TABLE3_VSTART + 16) then
                                pocket_flag1(2) <= '1';  --pocket 3
                        else
                                pocket_flag1(2) <= '0';
                        end if;
                        if(x_coord + 2 >= TABLE3_HSTART - 19 and x_coord + 2 <
TABLE3_HSTART + 16 and y_coord >= TABLE3_VEND - 19 and y_coord <
TABLE3_VEND + 16) then
                                pocket_flag1(3) <= '1';  --pocket 4
                        else
                                pocket_flag1(3) <= '0';
                        end if;
                        if(x_coord + 2 >= TABLE3_HSTART + 272 and x_coord + 2 <
TABLE3_HSTART + 307 and y_coord >= TABLE3_VEND and y_coord < TABLE3_VEND +
35) then
                                pocket_flag1(4) <= '1';  --pocket 5
                        else
                                pocket_flag1(4) <= '0';
                        end if;
                        if(x_coord + 2 >= TABLE3_HEND - 16 and x_coord + 2 <
TABLE3_HEND + 19 and y_coord >= TABLE3_VEND - 16 and y_coord < TABLE3_VEND +
19) then
                                pocket_flag1(5) <= '1';  --pocket 6
                        else
                                pocket_flag1(5) <= '0';
                        end if;

                end if;

        end if;
  end process Level1_FLAG_Gen;

  Level1_AddrGen1 : process(clk)
  begin
        if rising_edge(clk) then
                if reset = '1' then
                        for I in 0 to 5 loop
```

```
            pocket_eachaddr(I) <= "00000000000";  --11 bits
        end loop;
    else
        -- pocket 1
        if(x_coord = TABLE3_HSTART + 16 and y_coord = TABLE3_VSTART
+ 16) then
            pocket_eachaddr(0) <= "00000000000";  --11 bits
        elsif(x_coord + 1 >= TABLE3_HSTART - 19 and x_coord + 1 <
TABLE3_HSTART + 16 and y_coord >= TABLE3_VSTART - 19 and y_coord <
TABLE3_VSTART + 16) then
            pocket_eachaddr(0) <= pocket_eachaddr(0) + 1;
        end if;
        -- pocket 2
        if(x_coord = TABLE3_HSTART + 307 and y_coord =
TABLE3_VSTART + 12) then
            pocket_eachaddr(1) <= "00000000000";  --11 bits
        elsif(x_coord + 1 >= TABLE3_HSTART + 272 and x_coord + 1 <
TABLE3_HSTART + 307 and y_coord >= TABLE3_VSTART - 23 and y_coord <
TABLE3_VSTART + 12) then
            pocket_eachaddr(1) <= pocket_eachaddr(1) + 1;
        end if;
        -- pocket 3
        if(x_coord = TABLE3_HEND + 19 and y_coord = TABLE3_VSTART +
16) then
            pocket_eachaddr(2) <= "00000000000";  --11 bits
        elsif(x_coord + 1 >= TABLE3_HEND - 16 and x_coord + 1 <
TABLE3_HEND + 19 and y_coord >= TABLE3_VSTART - 19 and y_coord <
TABLE3_VSTART + 16) then
            pocket_eachaddr(2) <= pocket_eachaddr(2) + 1;
        end if;
        -- pocket 4
        if(x_coord = TABLE3_HSTART + 16 and y_coord = TABLE3_VSTART
+ 16) then
            pocket_eachaddr(3) <= "00000000000";  --11 bits
        elsif(x_coord + 1 >= TABLE3_HSTART - 19 and x_coord + 1 <
TABLE3_HSTART + 16 and y_coord >= TABLE3_VEND - 19 and y_coord <
TABLE3_VEND + 16) then
            pocket_eachaddr(3) <= pocket_eachaddr(3) + 1;
        end if;
```

```
            -- pocket 5
            if(x_coord = TABLE3_HSTART + 307 and y_coord = TABLE3_VEND
+ 35) then
                pocket_eachaddr(4) <= "00000000000";  --11 bits
            elsif(x_coord + 1 >= TABLE3_HSTART + 272 and x_coord + 1 <
TABLE3_HSTART + 307 and y_coord >= TABLE3_VEND and y_coord < TABLE3_VEND +
35) then
                pocket_eachaddr(4) <= pocket_eachaddr(4) + 1;
            end if;
            -- pocket 6
           if(x_coord = TABLE3_HEND + 19 and y_coord = TABLE3_VEND + 19)
then
                pocket_eachaddr(5) <= "00000000000";  --11 bits
            elsif(x_coord + 1 >= TABLE3_HEND - 16 and x_coord + 1 <
TABLE3_HEND + 19 and y_coord >= TABLE3_VEND - 16 and y_coord < TABLE3_VEND +
19) then
                pocket_eachaddr(5) <= pocket_eachaddr(5) + 1;
            end if;
        end if;
    end if;
  end process Level1_AddrGen1;

  Level1_AddrGen2 : process(clk)
  begin
    if rising_edge(clk) then
        --      Generate ball address
        if reset = '1' then
            level1_address <= "00000000000000";  -- 14 bits
        elsif pocket_flag1(0) = '1' then
            level1_address(13 downto 3) <= pocket_eachaddr(0);
            level1_address(2 downto 0) <= "000";
        elsif pocket_flag1(1) = '1' then
            level1_address(13 downto 3) <= pocket_eachaddr(1);
            level1_address(2 downto 0) <= "001";
        elsif pocket_flag1(2) = '1' then
            level1_address(13 downto 3) <= pocket_eachaddr(2);
            level1_address(2 downto 0) <= "010";
        elsif pocket_flag1(3) = '1' then
            level1_address(13 downto 3) <= pocket_eachaddr(3);
```

```
                          level1_address(2 downto 0) <= "011";
                  elsif pocket_flag1(4) = '1' then
                          level1_address(13 downto 3) <= pocket_eachaddr(4);
                          level1_address(2 downto 0) <= "100";
                  elsif pocket_flag1(5) = '1' then
                          level1_address(13 downto 3) <= pocket_eachaddr(5);
                          level1_address(2 downto 0) <= "101";
                  else
                          level1_address <= "00000000000000";  -- 14 bits
                  end if;

                  pocket_flag2 <= pocket_flag1;
          end if;
  end process Level1_AddrGen2;

  Level1_ActiveFGen : process(clk)
  begin
          if rising_edge(clk) then
                  if reset = '1' then
                          level1_active <= '0';
                  else
                     level1_active <= pocket_flag2(0) or pocket_flag2(1) or pocket_flag2(2) or
                                                                          pocket_flag2(3) or
pocket_flag2(4) or pocket_flag2(5);

                  end if;
          end if;
  end process Level1_ActiveFGen;

  Level1_Gen : process(clk)
          begin
          if rising_edge(clk) then
                  if reset = '1' then
                          level1_vga <= '0';
                  elsif (level1_active = '1' and (not (level1_data = "00000"))) then
                     level1_vga <= '1';
                  else
                          level1_vga <= '0';
                  end if;
```

```
            level1_data_vga <= level1_data;
        end if;
 end process Level1_Gen;



----------------------------------------------------------------------------------------
 ----------------------------------------------------------------------------------------
 --Level 2

 BALLMaskFLAG_Gen : process(clk)
  begin
        if rising_edge(clk) then
            if reset = '1' then
                ball_mask_flag1 <= "0000000000000000";
            else
                for I in 0 to 15 loop
                    if (x_coord + 2>= BALL_X(I) - 7 and x_coord + 2< BALL_X(I) +
7 and y_coord>= BALL_Y(I) - 7 and y_coord< BALL_Y(I) + 7) then
                        ball_mask_flag1(I) <= '1';
                    else
                        ball_mask_flag1(I) <= '0';
                    end if;
                end loop;

            end if;
        end if;
 end process BALLMaskFLAG_Gen;

        Ball_mask_AddrGen : process(clk)
  begin
        if rising_edge(clk) then
            if reset = '1' then
                for I in 0 to 15 loop
                    ball_mask_addr(I) <= -1;
                    ball_mask_addr2(I) <= -1;
                end loop;
            else
                for I in 0 to 15 loop
                    if (x_coord = BALL_X(I) + 7 and y_coord = BALL_Y(I) + 7) then
```

```
                      ball_mask_addr(I) <= -1;
                  elsif (x_coord + 1+ 7 >= BALL_X(I)  and x_coord + 1 <
BALL_X(I) + 7 and y_coord + 7 >= BALL_Y(I) and y_coord < BALL_Y(I) + 7) then
                      ball_mask_addr(I) <= ball_mask_addr(I) + 1;
                  end if;

                  if (x_coord = BALL_X(I) + 7 and y_coord = BALL_Y(I) + 7) then
                      ball_mask_addr2(I) <= -1;
                  elsif (x_coord + 2+ 7 >= BALL_X(I)  and x_coord + 2 <
BALL_X(I) + 7 and y_coord + 7 >= BALL_Y(I) and y_coord < BALL_Y(I) + 7) then
                      ball_mask_addr2(I) <= ball_mask_addr2(I) + 1;
                  end if;

              end loop;
          end if;
      end if;
  end process Ball_mask_AddrGen;

   Ball_map_AddrGen1 : process(clk)
   begin
      if rising_edge(clk) then
          if reset = '1' then
              for I in 0 to 15 loop
                  ball_map_addr(I) <= "0000000000";   --10 bits
              end loop;
          else
              for I in 0 to 15 loop
                  if (x_coord + BALL_BIAS_X(I) = BALL_X(I) + 20  and y_coord
+ BALL_BIAS_Y(I) = BALL_Y(I) + 20) then
                      ball_map_addr(I) <= "0000000000";
                  elsif (x_coord + 1 + BALL_BIAS_X(I) + 7>= BALL_X(I)  and
x_coord + 1 + BALL_BIAS_X(I) < BALL_X(I) + 20 and y_coord + BALL_BIAS_Y(I) + 7>=
BALL_Y(I) and y_coord + BALL_BIAS_Y(I)< BALL_Y(I) + 20) then
                      ball_map_addr(I) <= ball_map_addr(I) + 1;
                  end if;
              end loop;

          end if;
      end if;
  end process Ball_map_AddrGen1;

  Ball_map_AddrGen2 : process(clk)
  begin
        if rising_edge(clk) then
                --        Generate ball address
                if reset = '1' then
                        ball_map_address <= "00000000000000";  --14 bits
                --elsif ball_mask_flag1(15) = '1' and (not(BallMask(ball_mask_addr(15) + 1) =
"00")) then    --ball number 16
                elsif ball_mask_flag1(15) = '1' and (not(BallMask(ball_mask_addr2(15)) = "00"))
then    --ball number 16
                        ball_map_address(13 downto 4) <= ball_map_addr(15);
                        ball_map_address(3 downto 0) <= "1111";
                --elsif ball_mask_flag1(0) = '1' and (not(BallMask(ball_mask_addr(0) + 1) =
"00")) then    --ball number 1
                elsif ball_mask_flag1(0) = '1' and (not(BallMask(ball_mask_addr2(0)) = "00"))
then    --ball number 1
                        ball_map_address(13 downto 4) <= ball_map_addr(0);
                        ball_map_address(3 downto 0) <= "0000";
                --elsif ball_mask_flag1(1) = '1' and (not(BallMask(ball_mask_addr(1) + 1) =
"00")) then    --ball number 2
                elsif ball_mask_flag1(1) = '1' and (not(BallMask(ball_mask_addr2(1)) = "00"))
then    --ball number 2
                        ball_map_address(13 downto 4) <= ball_map_addr(1);
                        ball_map_address(3 downto 0) <= "0001";
                --elsif ball_mask_flag1(2) = '1' and (not(BallMask(ball_mask_addr(2) + 1) =
"00")) then    --ball number 3
                elsif ball_mask_flag1(2) = '1' and (not(BallMask(ball_mask_addr2(2)) = "00"))
then    --ball number 3
                        ball_map_address(13 downto 4) <= ball_map_addr(2);
                        ball_map_address(3 downto 0) <= "0010";
                --elsif ball_mask_flag1(3) = '1' and (not(BallMask(ball_mask_addr(3) + 1) =
"00")) then    --ball number 4
                elsif ball_mask_flag1(3) = '1' and (not(BallMask(ball_mask_addr2(3)) = "00"))
then    --ball number 4
                        ball_map_address(13 downto 4) <= ball_map_addr(3);
                        ball_map_address(3 downto 0) <= "0011";
                --elsif ball_mask_flag1(4) = '1' and (not(BallMask(ball_mask_addr(4) + 1) =
"00")) then    --ball number 5
                elsif ball_mask_flag1(4) = '1' and (not(BallMask(ball_mask_addr2(4)) = "00"))
then    --ball number 5
                        ball_map_address(13 downto 4) <= ball_map_addr(4);
                        ball_map_address(3 downto 0) <= "0100";
                --elsif ball_mask_flag1(5) = '1' and (not(BallMask(ball_mask_addr(5) + 1) =
"00")) then    --ball number 6
                elsif ball_mask_flag1(5) = '1' and (not(BallMask(ball_mask_addr2(5)) = "00"))
then    --ball number 6
                        ball_map_address(13 downto 4) <= ball_map_addr(5);
                        ball_map_address(3 downto 0) <= "0101";
                --elsif ball_mask_flag1(6) = '1' and (not(BallMask(ball_mask_addr(6) + 1) =
"00")) then    --ball number 7
                elsif ball_mask_flag1(6) = '1' and (not(BallMask(ball_mask_addr2(6)) = "00"))
then    --ball number 7
                        ball_map_address(13 downto 4) <= ball_map_addr(6);
                        ball_map_address(3 downto 0) <= "0110";
                --elsif ball_mask_flag1(7) = '1' and (not(BallMask(ball_mask_addr(7) + 1) =
"00")) then    --ball number 8
                elsif ball_mask_flag1(7) = '1' and (not(BallMask(ball_mask_addr2(7)) = "00"))
then    --ball number 8
                        ball_map_address(13 downto 4) <= ball_map_addr(7);
                        ball_map_address(3 downto 0) <= "0111";
                --elsif ball_mask_flag1(8) = '1' and (not(BallMask(ball_mask_addr(8) + 1) =
"00")) then    --ball number 9
                elsif ball_mask_flag1(8) = '1' and (not(BallMask(ball_mask_addr2(8)) = "00"))
then    --ball number 9
                        ball_map_address(13 downto 4) <= ball_map_addr(8);
                        ball_map_address(3 downto 0) <= "1000";
                --elsif ball_mask_flag1(9) = '1' and (not(BallMask(ball_mask_addr(9) + 1) =
"00")) then    --ball number 10
                elsif ball_mask_flag1(9) = '1' and (not(BallMask(ball_mask_addr2(9)) = "00"))
then    --ball number 10
                        ball_map_address(13 downto 4) <= ball_map_addr(9);
                        ball_map_address(3 downto 0) <= "1001";
                --elsif ball_mask_flag1(10) = '1' and (not(BallMask(ball_mask_addr(10) + 1) =
"00")) then    --ball number 11
```

```
                elsif ball_mask_flag1(10) = '1' and (not(BallMask(ball_mask_addr2(10)) = "00"))
then    --ball number 11
                        ball_map_address(13 downto 4) <= ball_map_addr(10);
                        ball_map_address(3 downto 0) <= "1010";
                --elsif ball_mask_flag1(11) = '1' and (not(BallMask(ball_mask_addr(11) + 1) =
"00")) then    --ball number 12
                elsif ball_mask_flag1(11) = '1' and (not(BallMask(ball_mask_addr2(11)) = "00"))
then    --ball number 12
                        ball_map_address(13 downto 4) <= ball_map_addr(11);
                        ball_map_address(3 downto 0) <= "1011";
                --elsif ball_mask_flag1(12) = '1' and (not(BallMask(ball_mask_addr(12) + 1) =
"00")) then    --ball number 13
                elsif ball_mask_flag1(12) = '1' and (not(BallMask(ball_mask_addr2(12)) = "00"))
then    --ball number 13
                        ball_map_address(13 downto 4) <= ball_map_addr(12);
                        ball_map_address(3 downto 0) <= "1100";
                --elsif ball_mask_flag1(13) = '1' and (not(BallMask(ball_mask_addr(13) + 1) =
"00")) then    --ball number 14
                elsif ball_mask_flag1(13) = '1' and (not(BallMask(ball_mask_addr2(13)) = "00"))
then    --ball number 14
                        ball_map_address(13 downto 4) <= ball_map_addr(13);
                        ball_map_address(3 downto 0) <= "1101";
                --elsif ball_mask_flag1(14) = '1' and (not(BallMask(ball_mask_addr(14) + 1) =
"00")) then    --ball number 15
                elsif ball_mask_flag1(14) = '1' and (not(BallMask(ball_mask_addr2(14)) = "00"))
then    --ball number 15
                        ball_map_address(13 downto 4) <= ball_map_addr(14);
                        ball_map_address(3 downto 0) <= "1110";
                else
                        ball_map_address <= "00000000000000";
                end if;

                ball_mask_flag2 <= ball_mask_flag1;
        end if;
  end process Ball_map_AddrGen2;

  Ball_Map_inst : Ball_Map_rom port map(
        address => std_logic_vector(ball_map_address),
        clock => clk,
        q => ball_data
        );



  Ball_mask_colorGen : process(clk)
        begin
                if rising_edge(clk) then
                        if reset = '1' then
                                ball_mask_color <= 0;
                        elsif (ball_mask_flag2(15) = '1' and (not(BallMask(ball_mask_addr(15)) =
"00"))) then  -- ball num 16
                                ball_mask_color <= to_integer(BallMask(ball_mask_addr(15))) +
26;
                        elsif (ball_mask_flag2(0) = '1' and (not(BallMask(ball_mask_addr(0)) =
"00"))) then  -- ball num 1
                                ball_mask_color <= to_integer(BallMask(ball_mask_addr(0))) + 2;
                        elsif (ball_mask_flag2(1) = '1' and (not(BallMask(ball_mask_addr(1)) =
"00"))) then  -- ball num 2
                                ball_mask_color <= to_integer(BallMask(ball_mask_addr(1))) + 5;
                        elsif (ball_mask_flag2(2) = '1' and (not(BallMask(ball_mask_addr(2)) =
"00"))) then  -- ball num 3
                                ball_mask_color <= to_integer(BallMask(ball_mask_addr(2))) + 8;
                        elsif (ball_mask_flag2(3) = '1' and (not(BallMask(ball_mask_addr(3)) =
"00"))) then  -- ball num 4
                                ball_mask_color <= to_integer(BallMask(ball_mask_addr(3))) +
11;
                        elsif (ball_mask_flag2(4) = '1' and (not(BallMask(ball_mask_addr(4)) =
"00"))) then  -- ball num 5
                                ball_mask_color <= to_integer(BallMask(ball_mask_addr(4))) +
14;
                        elsif (ball_mask_flag2(5) = '1' and (not(BallMask(ball_mask_addr(5)) =
"00"))) then  -- ball num 6
                                ball_mask_color <= to_integer(BallMask(ball_mask_addr(5))) +
17;
                        elsif (ball_mask_flag2(6) = '1' and (not(BallMask(ball_mask_addr(6)) =
"00"))) then  -- ball num 7
                                ball_mask_color <= to_integer(BallMask(ball_mask_addr(6))) +
20;
```

```
            elsif (ball_mask_flag2(7) = '1' and (not(BallMask(ball_mask_addr(7)) =
"00"))) then  -- ball num 8
                ball_mask_color <= to_integer(BallMask(ball_mask_addr(7))) +
23;
            elsif (ball_mask_flag2(8) = '1' and (not(BallMask(ball_mask_addr(8)) =
"00"))) then  -- ball num 9
                ball_mask_color <= to_integer(BallMask(ball_mask_addr(8))) + 2;
            elsif (ball_mask_flag2(9) = '1' and (not(BallMask(ball_mask_addr(9)) =
"00"))) then  -- ball num 10
                ball_mask_color <= to_integer(BallMask(ball_mask_addr(9))) + 5;
            elsif (ball_mask_flag2(10) = '1' and (not(BallMask(ball_mask_addr(10)) =
"00"))) then  -- ball num 11
                ball_mask_color <= to_integer(BallMask(ball_mask_addr(10))) +
8;
            elsif (ball_mask_flag2(11) = '1' and (not(BallMask(ball_mask_addr(11)) =
"00"))) then  -- ball num 12
                ball_mask_color <= to_integer(BallMask(ball_mask_addr(11))) +
11;
            elsif (ball_mask_flag2(12) = '1' and (not(BallMask(ball_mask_addr(12)) =
"00"))) then  -- ball num 13
                ball_mask_color <= to_integer(BallMask(ball_mask_addr(12))) +
14;
            elsif (ball_mask_flag2(13) = '1' and (not(BallMask(ball_mask_addr(13)) =
"00"))) then  -- ball num 14
                ball_mask_color <= to_integer(BallMask(ball_mask_addr(13))) +
17;
            elsif (ball_mask_flag2(14) = '1' and (not(BallMask(ball_mask_addr(14)) =
"00"))) then  -- ball num 15
                ball_mask_color <= to_integer(BallMask(ball_mask_addr(14))) +
20;
            else
                ball_mask_color <= 0;
            end if;
        end if;
    end process Ball_mask_colorGen;

 Level2_ActiveFGen : process(clk)
    begin
    if rising_edge(clk) then
```

```
                if reset = '1' then
                        level2_active <= '0';
                else
                  level2_active <= ball_mask_flag2(0) or ball_mask_flag2(1) or
ball_mask_flag2(2) or ball_mask_flag2(3) or
                                                                        ball_mask_flag2(4) or
ball_mask_flag2(5) or ball_mask_flag2(6) or ball_mask_flag2(7) or
                                                                        ball_mask_flag2(8) or
ball_mask_flag2(9) or ball_mask_flag2(10) or ball_mask_flag2(11) or
                                                                        ball_mask_flag2(12) or
ball_mask_flag2(13) or ball_mask_flag2(14) or ball_mask_flag2(15);
                end if;
        end if;
  end process Level2_ActiveFGen;

  Level2_Gen : process(clk)
        begin
        if rising_edge(clk) then
                if reset = '1' then
                        level2_vga <= '0';
                elsif ((level2_active = '1') and (not (ball_mask_color = 0))) then
--              elsif (level2_active = '1') then
                  level2_vga <= '1';
                else
                        level2_vga <= '0';
                end if;

                if reset = '1' then
                        ball_data_vga <= 0;
                elsif ball_data(1) = '1' then
                        if ball_data(0) = '0' then
                                ball_data_vga <= 24;  --black
                        elsif ball_data(0) = '1' then
                                ball_data_vga <= 29;  --while
                        end if;
                else --ball_data(1) = '0'
                        ball_data_vga <= ball_mask_color;
                end if;
        end if;
end process Level2_Gen;

 ---------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
--Level 3

--level 3
        PoolCueRAM : process(clk)
          begin
          if rising_edge(clk) then
                  if reset = '1' then
                          CUERAM_addr <= "000000000"; -- 9 bits std_logic_vector
                          cue_line_begin <= "0000000000000000";
                          cue_line_end <= "0000000000000000";
                  else
                          CUERAM_addr <= std_logic_vector(y_coord(8 downto 0));
                          cue_line_begin <= unsigned(CUERAM_q(31 downto 16));
                          cue_line_end <= unsigned(CUERAM_q(15 downto 0));
                  end if;
          end if;
        end process PoolCueRAM;

        Level3_Gen : process(clk)
          begin
          if rising_edge(clk) then
                  if reset = '1' then
                          level3_vga <= '0';
                          --cue_line_begin <= "0000"
                  else
                          if (x_coord >= cue_line_begin and x_coord < cue_line_end) then
                                  level3_vga <= '1';
                          else
                                  level3_vga <= '0';
                          end if;

                  end if;
          end if;
        end process Level3_Gen;
```

```
      ------------------------------------------------------------------------------------------
 ------------------------------------------------------------------------------------------
 --Level 4

  Words_rom_inst : Words_rom port map(
      address => std_logic_vector(Word_address),
      clock => clk,
      q => Word_data
      );

      Word_FLAG_Gen : process(clk)
  begin
      if rising_edge(clk) then
          if reset = '1' then
              Word_flag1 <= (others => '0');
          else
              for I in 0 to 7 loop
                  if(Word_en(I) = '1' and x_coord + 2 >= Word_start_x(I) and
x_coord + 2 < Word_start_x(I) + 40 and y_coord >= Word_start_y(I) and y_coord <
Word_start_y(I) + 16) then
                      Word_flag1(I) <= '1';
                  else
                      Word_flag1(I) <= '0';
                  end if;
              end loop;
          end if;

      end if;
 end process Word_FLAG_Gen;

 Word_AddrGen1 : process(clk)
 begin
      if rising_edge(clk) then
          if reset = '1' then
              for I in 0 to 7 loop
                  Word_eachaddr(I) <= (others => '0');
              end loop;
          else
```

```
                        for I in 0 to 7 loop
                          if(x_coord = Word_start_x(I) + 40 and y_coord = Word_start_y(I) + 16)
then
                                        Word_eachaddr(I) <= (others => '0');
                                elsif(x_coord + 1 >= Word_start_x(I) and x_coord + 1 <
Word_start_x(I) + 40 and y_coord >= Word_start_y(I) and y_coord < Word_start_y(I) + 16)
then
                                        Word_eachaddr(I) <= Word_eachaddr(I) + 1;
                                end if;
                        end loop;

                end if;
        end if;
 end process Word_AddrGen1;

 Word_AddrGen2 : process(clk)
 begin
        if rising_edge(clk) then
                --        Generate ball address
                if reset = '1' then
                        Word_address <= (others => '0');
                elsif word_flag1(0) = '1' then
                        Word_address(12 downto 3) <= Word_eachaddr(0);
                        Word_address(2 downto 0) <= "000";
                        Word_hl <= Word_hl_en(0);
                elsif word_flag1(1) = '1' then
                        Word_address(12 downto 3) <= Word_eachaddr(1);
                        Word_address(2 downto 0) <= "001";
                        Word_hl <= Word_hl_en(1);
                elsif word_flag1(2) = '1' then
                        Word_address(12 downto 3) <= Word_eachaddr(2);
                        Word_address(2 downto 0) <= "010";
                        Word_hl <= Word_hl_en(2);
                elsif word_flag1(3) = '1' then
                        Word_address(12 downto 3) <= Word_eachaddr(3);
                        Word_address(2 downto 0) <= "011";
                        Word_hl <= Word_hl_en(3);
                elsif word_flag1(4) = '1' then
                        Word_address(12 downto 3) <= Word_eachaddr(4);
                        Word_address(2 downto 0) <= "100";
                        Word_hl <= Word_hl_en(4);
                elsif word_flag1(5) = '1' then
                        Word_address(12 downto 3) <= Word_eachaddr(5);
                        Word_address(2 downto 0) <= "101";
                        Word_hl <= Word_hl_en(5);
                elsif word_flag1(6) = '1' then
                        Word_address(12 downto 3) <= Word_eachaddr(6);
                        Word_address(2 downto 0) <= "110";
                        Word_hl <= Word_hl_en(6);
                elsif word_flag1(7) = '1' then
                        Word_address(12 downto 3) <= Word_eachaddr(7);
                        Word_address(2 downto 0) <= "111";
                        Word_hl <= Word_hl_en(7);
                else
                        Word_address <= (others => '0');
                        Word_hl <= '0';
                end if;

                Word_flag2 <= Word_flag1;
        end if;
  end process Word_AddrGen2;

  Level4_ActiveFGen : process(clk)
  begin
        if rising_edge(clk) then
                if reset = '1' then
                        level4_active <= '0';
                else
                  level4_active <= word_flag2(0) or word_flag2(1) or word_flag2(2) or
                                                                        word_flag2(3) or
word_flag2(4) or word_flag2(5) or
                                                                        word_flag2(6) or
word_flag2(7);

                end if;
        end if;
  end process Level4_ActiveFGen;
```

```
Level4_Gen : process(clk)
        begin
        if rising_edge(clk) then
                if reset = '1' then
                        level4_vga <= '0';
                        Word_color <= 0;
                elsif level4_active = '1' then
                        if Word_data = "1" then
                                level4_vga <= '1';
                                Word_color <= 24;
                        elsif Word_hl = '1' then
                                level4_vga <= '1';
                                Word_color <= 15;
                        else
                                level4_vga <= '0';
                        end if;
                else
                        level4_vga <= '0';
                end if;

        end if;
end process Level4_Gen;


-- Registered video signals going to the video DAC


VideoOut: process (clk, reset)
begin
         VGA_R(1 downto 0) <= "00";
         VGA_G(1 downto 0) <= "00";
         VGA_B(1 downto 0) <= "00";
  if reset = '1' then
    VGA_R(9 downto 2) <= "00000000";
    VGA_G(9 downto 2) <= "00000000";
    VGA_B(9 downto 2) <= "00000000";
  elsif clk'event and clk = '1' then
                if vga_hblank = '1' or vga_vblank ='1' then
                  VGA_R(9 downto 2) <= "00000000";
        VGA_G(9 downto 2) <= "00000000";
        VGA_B(9 downto 2) <= "00000000";
                    elsif level4_vga = '1' then
                      VGA_R(9 downto 2) <= COLOR_TABLE_R(Word_color);
        VGA_G(9 downto 2) <= COLOR_TABLE_G(Word_color);
        VGA_B(9 downto 2) <= COLOR_TABLE_B(Word_color);
                    elsif level3_vga = '1' then
                      VGA_R(9 downto 2) <= COLOR_TABLE_R(2);
        VGA_G(9 downto 2) <= COLOR_TABLE_G(2);
        VGA_B(9 downto 2) <= COLOR_TABLE_B(2);
                    elsif level2_vga = '1' then
                      VGA_R(9 downto 2) <= COLOR_TABLE_R(ball_data_vga);
        VGA_G(9 downto 2) <= COLOR_TABLE_G(ball_data_vga);
        VGA_B(9 downto 2) <= COLOR_TABLE_B(ball_data_vga);
                    elsif level1_vga = '1' then
                      VGA_R(9 downto 2) <=
COLOR_TABLE_R(to_integer(unsigned(level1_data_vga)));
        VGA_G(9 downto 2) <= COLOR_TABLE_G(to_integer(unsigned(level1_data_vga)));
        VGA_B(9 downto 2) <= COLOR_TABLE_B(to_integer(unsigned(level1_data_vga)));
      elsif level0_active = '1' then
        VGA_R(9 downto 2) <= COLOR_TABLE_R(to_integer(colorcode_level0));
        VGA_G(9 downto 2) <= COLOR_TABLE_G(to_integer(colorcode_level0));
        VGA_B(9 downto 2) <= COLOR_TABLE_B(to_integer(colorcode_level0));

      else --background color
        VGA_R(9 downto 2) <= COLOR_TABLE_R(5);
        VGA_G(9 downto 2) <= COLOR_TABLE_G(5);
        VGA_B(9 downto 2) <= COLOR_TABLE_B(5);

      end if;
    end if;
  end process VideoOut;

  VGA_CLK <= clk;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);
```

```
end rtl;

-- Pool Cue Ram Controller
--Editor: Jiawan Zhang
--Data: 2013

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PoolCue_ram_controller is port(
                chipselect : in std_LOGIC;
                writedata               : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
                wraddress               : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                clk_50          : IN STD_LOGIC;
                write           : IN STD_LOGIC;
                CUE_rdaddress                   : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                CUE_rdclock             : IN STD_LOGIC ;
                CUE_q                   : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
end entity PoolCue_ram_controller;

architecture blockram of PoolCue_ram_controller is

component PoolCue_ram IS
        PORT
        (
                data            : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
                rdaddress               : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                rdclock         : IN STD_LOGIC ;
                wraddress               : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                wrclock                 : IN STD_LOGIC;
                wren            : IN STD_LOGIC;
                q               : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
end component PoolCue_ram;

signal inter_CUE_rdaddress : std_logic_vector(8 downto 0);
signal inter_CUE_rdclock : std_logic;
signal inter_CUE_q : std_logic_vector(31 downto 0);

begin
        the_PoolCue_ram : PoolCue_ram port map(
                data => writedata,
                wraddress => wraddress,
                wrclock => clk_50,
                wren => write,
                rdaddress => inter_CUE_rdaddress,
                rdclock => inter_CUE_rdclock,
                q => inter_CUE_q
                );
        inter_CUE_rdaddress <= CUE_rdaddress;
        inter_CUE_rdclock <= CUE_rdclock;
        CUE_q <= inter_CUE_q;

end blockram;
```

-- **Keyboard Controller**

```
-------------------------------------------------------------------
--
-- Simple (receive-only) PS/2 controller for the Altera Avalon bus
--
-- Presents a two-word interface:
--
-- Byte 0: LSB is a status bit: 1 = data received, 0 = no new data
-- Byte 4: least significant byte is received data,
--        reading it clears the input register
--
-- Make sure "Slave addressing" in the interfaces tab of SOPC Builder's
-- "New Component" dialog is set to "Register" mode.
--
--
-- Stephen A. Edwards and Yingjian Gu
-- Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Bert Cuzeau
-- (c) ALSE. http://www.alse-fr.com
--
```

```
---------------------------------------------------------------------------

-- ----------------------------------------------------
--   Simplified PS/2 Controller  (kbd, mouse...)
-- ----------------------------------------------------
-- Only the Receive function is implemented !
-- (c) ALSE. http://www.alse-fr.com
-- Author : Bert Cuzeau.
-- Fully synchronous solution, same Filter on PS2_Clk.
-- Still as compact as "Plain_wrong"...
-- Possible improvement : add TIMEOUT on PS2_Clk while shifting
-- Note: PS2_Data is resynchronized though this should not be
-- necessary (qualified by Fall_Clk and does not change at that time).
-- Note the tricks to correctly interpret 'H' as '1' in RTL simulation.

-----------------------------------------------------------------------------------
-- Editor Zeshi Wang
-- Data: 2013


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PS2_Ctrl is
  port(
    Clk       : in  std_logic;  -- System Clock
    Reset     : in  std_logic;  -- System Reset
    PS2_Clk   : in  std_logic;  -- Keyboard Clock Line
    PS2_Data  : in  std_logic;  -- Keyboard Data Line
    DoRead    : in  std_logic;  -- From outside when reading the scan code
    Scan_Err  : out std_logic;  -- To outside : Parity or Overflow error
    Scan_DAV  : out std_logic;  -- To outside when a scan code has arrived
    Scan_Code : out std_logic_vector(7 downto 0) -- Eight bits Data Out
    );
end PS2_Ctrl;

architecture rtl of PS2_Ctrl is
```

```
  signal PS2_Datr  : std_logic;

  subtype Filter_t is std_logic_vector(7 downto 0);
  signal Filter    : Filter_t;
  signal Fall_Clk  : std_logic;
  signal Bit_Cnt   : unsigned (3 downto 0);
  signal Parity    : std_logic;
  signal Scan_DAVi : std_logic;

  signal S_Reg     : std_logic_vector(8 downto 0);

  signal PS2_Clk_f : std_logic;

  Type   State_t is (Idle, Shifting);
  signal State : State_t;

begin

  Scan_DAV <= Scan_DAVi;

-- This filters digitally the raw clock signal coming from the keyboard :
--  * Eight consecutive PS2_Clk=1 makes the filtered_clock go high
--  * Eight consecutive PS2_Clk=0 makes the filtered_clock go low
-- Implies a (FilterSize+1) x Tsys_clock delay on Fall_Clk wrt Data
-- Also in charge of the re-synchronization of PS2_Data

  process (Clk)
  begin
    if rising_edge(Clk) then
      if Reset = '1' then
        PS2_Datr  <= '0';
        PS2_Clk_f <= '0';
        Filter    <= (others => '0');
        Fall_Clk  <= '0';
      else
        PS2_Datr <= PS2_Data and PS2_Data; -- also turns 'H' into '1'
        Fall_Clk <= '0';
        Filter   <= (PS2_Clk and PS2_CLK) & Filter(Filter'high downto 1);
        if Filter = Filter_t'(others=>'1') then
```

```
          PS2_Clk_f <= '1';
        elsif Filter = Filter_t'(others=>'0') then
          PS2_Clk_f <= '0';
          if PS2_Clk_f = '1' then
            Fall_Clk <= '1';
          end if;
        end if;
      end if;
    end if;
  end process;

-- This simple State Machine reads in the Serial Data
-- coming from the PS/2 peripheral.

  process(Clk)
  begin
    if rising_edge(Clk) then
      if Reset = '1' then
        State     <= Idle;
        Bit_Cnt   <= (others => '0');
        S_Reg     <= (others => '0');
        Scan_Code <= (others => '0');
        Parity    <= '0';
        Scan_DAVi <= '0';
        Scan_Err  <= '0';
      else

        if DoRead = '1' then
          Scan_DAVi <= '0'; -- note: this assgnmnt can be overriden
        end if;

        case State is

          when Idle =>
            Parity  <= '0';
            Bit_Cnt <= (others => '0');
            -- note that we do not need to clear the Shift Register
            if Fall_Clk='1' and PS2_Datr='0' then -- Start bit
              Scan_Err <= '0';
              State <= Shifting;
            end if;

          when Shifting =>
            if Bit_Cnt >= 9 then
              if Fall_Clk = '1' then -- Stop Bit
                -- Error is (wrong Parity) or (Stop='0') or Overflow
                Scan_Err  <= (not Parity) or (not PS2_Datr) or Scan_DAVi;
                Scan_Davi <= '1';
                Scan_Code <= S_Reg(7 downto 0);
                State <= Idle;
              end if;
            elsif Fall_Clk = '1' then
              Bit_Cnt <= Bit_Cnt + 1;
              S_Reg <= PS2_Datr & S_Reg (S_Reg'high downto 1); -- Shift right
              Parity <= Parity xor PS2_Datr;
            end if;

          when others => -- never reached
            State <= Idle;

        end case;

        --Scan_Err <= '0'; -- to create a deliberate error

      end if;

    end if;

  end process;

end rtl;

-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
entity de2_ps2 is

  port (
    avs_s1_clk        : in std_logic;
    avs_s1_reset      : in std_logic;
    avs_s1_address    : in std_logic;
    avs_s1_read       : in std_logic;
    avs_s1_chipselect : in std_logic;
    avs_s1_readdata   : out std_logic_vector(7 downto 0);

    PS2_Clk           : in std_logic;
    PS2_Data          : in std_logic
    );
end de2_ps2;

architecture rtl of de2_ps2 is

  signal Data          : std_logic_vector(7 downto 0);
  signal DataAvailable : std_logic;
  signal DoRead        : std_logic;

begin

  U1: entity work.PS2_CTRL port map(
    Clk       => avs_s1_clk,
    Reset     => avs_s1_reset,
    DoRead    => DoRead,
    PS2_Clk   => PS2_Clk,
    PS2_Data  => PS2_Data,
    Scan_Code => Data,
    Scan_DAV  => DataAvailable );

  process (avs_s1_clk)
  begin
    if rising_edge(avs_s1_clk) then
      DoRead <= avs_s1_read and avs_s1_chipselect and avs_s1_address;
    end if;
  end process;
  process (Data, DataAvailable, avs_s1_address, avs_s1_chipselect)
  begin
    if avs_s1_chipselect = '1' then
      if avs_s1_address = '1' then
        avs_s1_readdata <= Data;
      else
        avs_s1_readdata <= "0000000" & DataAvailable;
      end if;
    else
      avs_s1_readdata <= "00000000";
    end if;
  end process;

end rtl;
```

-- **Audio Controller**

```
--Editor: Zeshi Wang
--Data: 2013

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_wm8731_audio is
port (
    clk : in std_logic;       --  Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
    reset_n : in std_logic;
    data : in std_logic_vector(7 downto 0) := "00000000";
             write      : in std_logic;
             chipselect : in std_logic;

    -- Audio interface signals
    AUD_ADCLRCK  : out  std_logic;   --    Audio CODEC ADC LR Clock
    AUD_ADCDAT   : in   std_logic;   --    Audio CODEC ADC Data
    AUD_DACLRCK  : out  std_logic;   --    Audio CODEC DAC LR Clock
    AUD_DACDAT   : out  std_logic;   --    Audio CODEC DAC Data
    AUD_BCLK     : inout std_logic  --    Audio CODEC Bit-Stream Clock
```

```
 );
end  de2_wm8731_audio;

architecture rtl of de2_wm8731_audio is

   signal lrck : std_logic;
   signal bclk : std_logic;
   signal xck  : std_logic;

   signal lrck_divider : unsigned(15 downto 0);
   signal bclk_divider : unsigned(11 downto 0);

   signal set_bclk : std_logic;
   signal set_lrck : std_logic;
   signal clr_bclk : std_logic;
   signal lrck_lat : std_logic;

   signal shift_out : unsigned(15 downto 0);

          signal start_audio : std_logic_vector(3 downto 0);
          type count_type is array(0 to 3) of unsigned(11 downto 0);
          signal count          : count_type;
          --signal ram_address : count_type;
          signal data_in : std_logic_vector(15 downto 0);
          signal ram_address : std_logic_vector(13 downto 0);


component  sound_rom IS
        PORT
        (
                address         : IN STD_LOGIC_VECTOR (13 DOWNTO 0);
                clock           : IN STD_LOGIC  := '1';
                q               : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
        );
END component;


begin
```

```
     ROM  : sound_rom port map(
           address         =>  ram_address,
           clock           =>  clk,
           q                 =>  data_in
     );

 -- LRCK divider
 -- Audio chip main clock is 18.432MHz / Sample rate 48KHz
 -- Divider is 18.432 MHz / 48KHz = 192 (X"C0")
 -- Left justify mode set by I2C controller




--Set start_audio signal
process (clk)
begin
 if rising_edge(clk) then
       if reset_n = '0' then
    start_audio <= (others => '0');
       else
                      for I in 0 to 3 loop
                            if (write = '1' and chipselect = '1' and data(I) = '1') then
                                  start_audio(I) <= '1';
                            elsif count(I) = x"08ae" then
                                  start_audio(I) <= '0';
                            end if;
                      end loop;
           end if;
     end if;
end process;


process (clk)
begin
 if rising_edge(clk) then
  if reset_n = '0' then
   lrck_divider <= (others => '0');
```

```
      elsif lrck_divider = X"08FF"  then         -- "C0" minus 1
        lrck_divider <= X"0000";
      else
        lrck_divider <= lrck_divider + 1;
      end if;
    end if;
  end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        bclk_divider <= (others => '0');
      elsif bclk_divider = X"08F" or set_lrck = '1'  then
        bclk_divider <= X"000";
      else
        bclk_divider <= bclk_divider + 1;
      end if;
    end if;
  end process;

  set_lrck <= '1' when lrck_divider = X"08FF" else '0';

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        lrck <= '0';
      elsif set_lrck = '1' then
        lrck <= not lrck;
      end if;
    end if;
  end process;

process(clk)
    begin
      if rising_edge(clk) then
        lrck_lat <= lrck;
      end if;
```

```
 end process;


 -- BCLK divider
 set_bclk <= '1' when bclk_divider(11 downto 0) = X"047" else '0';
 clr_bclk <= '1' when bclk_divider(11 downto 0) = X"08F" else '0';

 process (clk)
 begin
  if rising_edge(clk) then
   if reset_n = '0' then
    bclk <= '0';
   elsif set_lrck = '1' or clr_bclk = '1' then
    bclk <= '0';
   elsif set_bclk = '1' then
    bclk <= '1';
   end if;
  end if;
 end process;

 -- Audio data shift output

process (clk)
 begin
  if rising_edge(clk) then
   if reset_n = '0' then
    shift_out <= (others => '0');
            elsif (start_audio(0) or start_audio(1) or start_audio(2) or start_audio(3)) = '1' then
    if set_lrck = '1' then
      shift_out <= unsigned(data_in);
             elsif clr_bclk = '1' then
                        shift_out <= shift_out (14 downto 0) & '0';
             end if;
            else
              shift_out <= (others => '0');
            end if;

  end if;
 end process;
```

```
  -- Audio outputs

  AUD_ADCLRCK  <= lrck;
  AUD_DACLRCK  <= lrck;
  AUD_DACDAT   <= shift_out(15);
  AUD_BCLK     <= bclk;

  -- ram address counter
       -- counter for ball_hit audio
-- Update count
      process(clk)
  begin
   if rising_edge(clk) then
    if reset_n = '0' then
      for I in 0 to 3 loop
                              count(I) <= (others => '0');
                        end loop;
    elsif lrck_lat = '1' and lrck = '0'  then
                        for I in 0 to 3 loop
                              if count(I) = x"8ae" then
                                    count(I) <= (others => '0');
                              elsif (start_audio(I) = '1') then
                                    count(I) <= count(I) + 1;
                              end if;

                        end loop;

    end if;
   end if;
  end process;

      --Update Address
      process(clk)
      begin
            if rising_edge(clk) then
                  if reset_n = '0' then
                        ram_address <= (others => '0');
                  elsif start_audio(0) = '1' then
```

```
                ram_address(13 downto 2) <= std_logic_vector(count(0));
                ram_address(1 downto 0) <= "00";
            elsif start_audio(1) = '1' then
                ram_address(13 downto 2) <= std_logic_vector(count(1));
                ram_address(1 downto 0) <= "01";
            elsif start_audio(2) = '1' then
                ram_address(13 downto 2) <= std_logic_vector(count(2));
                ram_address(1 downto 0) <= "10";
            elsif start_audio(3) = '1' then
                ram_address(13 downto 2) <= std_logic_vector(count(3));
                ram_address(1 downto 0) <= "11";
            else
                ram_address <= (others => '0');
            end if;
        end if;
    end process;

end architecture;
```

-- SRAM_Controller

```
--Legal Notice: (C)2007 Altera Corporation. All rights reserved.  Your
--use of Altera Corporation's design tools, logic functions and other
--software and tools, and its AMPP partner logic functions, and any
--output files any of the foregoing (including device programming or
--simulation files), and any associated documentation or information are
--expressly subject to the terms and conditions of the Altera Program
--License Subscription Agreement or other applicable license agreement,
--including, without limitation, that your use is for the sole purpose
--of programming logic devices manufactured by Altera and sold by Altera
--or its authorized distributors.  Please refer to the applicable
--agreement for further details.


-- turn off superfluous VHDL processor warnings
-- altera message_level Level1
-- altera message_off 10034 10035 10036 10037 10230 10240 10030

-- Editor: Zeshi Wang
-- Data: 2013
```

```
library altera;
use altera.altera_europa_support_lib.all;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity sram is
        port (
              -- inputs:
                 signal address : IN STD_LOGIC_VECTOR (17 DOWNTO 0);
                 signal byteenable : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
                 signal chipselect : IN STD_LOGIC;
                 signal read : IN STD_LOGIC;
                 signal write : IN STD_LOGIC;
                 signal writedata : IN STD_LOGIC_VECTOR (15 DOWNTO 0);

              -- outputs:
                 signal SRAM_ADDR : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
                 signal SRAM_CE_N : OUT STD_LOGIC;
                 signal SRAM_DQ : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
                 signal SRAM_LB_N : OUT STD_LOGIC;
                 signal SRAM_OE_N : OUT STD_LOGIC;
                 signal SRAM_UB_N : OUT STD_LOGIC;
                 signal SRAM_WE_N : OUT STD_LOGIC;
                 signal readdata : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
              );
end entity sram;


architecture europa of sram is
component de2_sram_controller is
           port (
                 -- inputs:
                    signal address : IN STD_LOGIC_VECTOR (17 DOWNTO 0);
                    signal byteenable : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
                    signal chipselect : IN STD_LOGIC;
                    signal read : IN STD_LOGIC;
                    signal write : IN STD_LOGIC;
                    signal writedata : IN STD_LOGIC_VECTOR (15 DOWNTO 0);

                  -- outputs:
                    signal SRAM_ADDR : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
                    signal SRAM_CE_N : OUT STD_LOGIC;
                    signal SRAM_DQ : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
                    signal SRAM_LB_N : OUT STD_LOGIC;
                    signal SRAM_OE_N : OUT STD_LOGIC;
                    signal SRAM_UB_N : OUT STD_LOGIC;
                    signal SRAM_WE_N : OUT STD_LOGIC;
                    signal readdata : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
                  );
end component de2_sram_controller;

                signal internal_SRAM_ADDR :  STD_LOGIC_VECTOR (17 DOWNTO 0);
                signal internal_SRAM_CE_N :  STD_LOGIC;
                signal internal_SRAM_LB_N :  STD_LOGIC;
                signal internal_SRAM_OE_N :  STD_LOGIC;
                signal internal_SRAM_UB_N :  STD_LOGIC;
                signal internal_SRAM_WE_N :  STD_LOGIC;
                signal internal_readdata :  STD_LOGIC_VECTOR (15 DOWNTO 0);

begin

  --the_de2_sram_controller, which is an e_instance
  the_de2_sram_controller : de2_sram_controller
    port map(
      SRAM_ADDR => internal_SRAM_ADDR,
      SRAM_CE_N => internal_SRAM_CE_N,
      SRAM_DQ => SRAM_DQ,
      SRAM_LB_N => internal_SRAM_LB_N,
      SRAM_OE_N => internal_SRAM_OE_N,
      SRAM_UB_N => internal_SRAM_UB_N,
      SRAM_WE_N => internal_SRAM_WE_N,
      readdata => internal_readdata,
      address => address,
      byteenable => byteenable,
      chipselect => chipselect,
      read => read,
      write => write,
      writedata => writedata
    );


  --vhdl renameroo for output signals
  SRAM_ADDR <= internal_SRAM_ADDR;
  --vhdl renameroo for output signals
  SRAM_CE_N <= internal_SRAM_CE_N;
  --vhdl renameroo for output signals
  SRAM_LB_N <= internal_SRAM_LB_N;
  --vhdl renameroo for output signals
  SRAM_OE_N <= internal_SRAM_OE_N;
  --vhdl renameroo for output signals
  SRAM_UB_N <= internal_SRAM_UB_N;
  --vhdl renameroo for output signals
  SRAM_WE_N <= internal_SRAM_WE_N;
  --vhdl renameroo for output signals
  readdata <= internal_readdata;

end europa;

--****************************************************
library ieee;
use ieee.std_logic_1164.all;

entity de2_sram_controller is

  port (
    signal chipselect : in std_logic;
    signal write, read : in std_logic;
    signal address  :  in std_logic_vector(17 downto 0);
    signal readdata : out std_logic_vector(15 downto 0);
    signal writedata : in std_logic_vector(15 downto 0);
    signal byteenable : in std_logic_vector(1 downto 0);

    signal SRAM_DQ : inout std_logic_vector(15 downto 0);
    signal SRAM_ADDR : out std_logic_vector(17 downto 0);
    signal SRAM_UB_N, SRAM_LB_N : out std_logic;
    signal SRAM_WE_N, SRAM_CE_N : out std_logic;
    signal SRAM_OE_N            : out std_logic
    );

end de2_sram_controller;

architecture dp of de2_sram_controller is
begin

  SRAM_DQ <= writedata when write = '1'
             else (others => 'Z');
  readdata <= SRAM_DQ;
  SRAM_ADDR <= address;
  SRAM_UB_N <= not byteenable(1);
  SRAM_LB_N <= not byteenable(0);
  SRAM_WE_N <= not write;
  SRAM_CE_N <= not chipselect;
  SRAM_OE_N <= not read;

end dp;
```

-- irTimer

```
-- Editor: Jiawan Zhang
-- Data: 2013


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity irTimer is port (
  clk_50 : in std_logic;
  reset : in std_logic;
  chipselect : in std_logic;
  read : in std_logic;
```

```vhdl
  write : in std_logic;
  address : in std_logic;
  readdata  : out std_logic_vector(15 downto 0);
  writedata : in  std_logic_vector(15 downto 0);
  irq       : out std_logic);
end irTimer;

architecture rtl of irTimer is
  signal counter : unsigned(15 downto 0);
  --signal ms : unsigned(15 downto 0);
  signal data    : std_logic_vector(15 downto 0);
begin

  process(clk_50)
  begin
	if rising_edge(clk_50) then
		if reset = '1' then
			readdata <= (others => '0');
			counter <= (others => '0');
		else
			if chipselect = '1' and address = '0' then
				if write = '1' then
					counter <= unsigned(writedata);
				elsif read = '1' then
					readdata <= std_logic_vector(counter);
					if not (counter = x"0000") then
						counter <= counter - x"0001";
					end if;
				end if;
			else
				if not (counter = x"0000") then
					counter <= counter - x"0001";
				end if;
			end if;
		end if;
	end if;
  end process;
  process (clk_50)
  begin
    if rising_edge(clk_50) then
      if reset = '1' then
        irq <= '0';
      else
        if counter = 0 then
          irq <= '1';
        elsif chipselect = '1' and write = '1' then
          irq <= '0';
        end if;
      end if;
    end if;
  end process;

end rtl;
```

Software Codes:

-- Main function for Pool Game

```
//Editors: XunChi Wu; Yuhan Zhang; Jiawan Zhang; Zeshi Wang
//Data: 2013

#include <stdio.h>
#include <io.h>
#include <system.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "poolcue.h"
#include "poolball.h"
#include "keyboard.h"

#define VC_MAX 3200
#define ACC 100
#define pi 3.1415926
#define angle_trans 0.05236   //
```

```
#define zero 0.0001
#define n_zero -0.0001
#define sq_zero 0.01
#define PRODUCT 0.34    // PRODUCT = dx * vc , when dx = min(zero), vc = vc_max
#define edge_acc 0.95


//Audio
//1 = loud; 2 = low
#define IOWR_AUDIO_EN(data)\
                IOWR_8DIRECT(AUDIO_0_BASE, 0, data);

//Write to vga ram
#define IOWR_VGA_STRENGTHBAR(data) \
        IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, 128, data)
#define SERVELINE_EN(flag)\
        IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, 130, flag)

//Words
//Enable
//6:WIN! ; 5: GAME ; 4: NEW ; 3: network ; 2: single ; 1: PLR 2 ; 0: PLR1
#define IOWR_VGA_WORD_EN(data) \
        IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, 132, data)
#define IOWR_VGA_WORD_HL_EN(data) \
        IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, 134, data)
#define IOWR_VGA_WORD_POS_X(n, data) \
        IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, (68 + 2*n)*2, data)
#define IOWR_VGA_WORD_POS_Y(n, data) \
        IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, (69 + 2*n)*2, data)

struct balltype
{
        int pos_x,pos_y;
        int vc_x, vc_y;
        int count_x, count_y;
        short int dir_x, dir_y;
        short int flag;
        float dx, dy;
        int force_move;
        int bias_x, bias_y;
}ball[16];


int hole[6][2]={{24,158},{320,154},{615,158},
                                {24,461},{320,465},{615,461}};

int player=0;
int change=1;
int oops = 0;
int plball[2]={0,0};
int i_0 =0;
int i_1 =0;
int ex = 1;
int cue_ready = 0;
int angle = 0;
int strength = 0;
int release = 0;
int begin_flag;
float swtich_d;
int dir_strength = 1;

void decide_movement(int ball_num)
{
        if((ball[ball_num].dx * ball[ball_num].dx + ball[ball_num].dy * ball[ball_num].dy) >=
zero) //if the ball speed hasn't reached the stopping threshold
        {
                //decide the moving direction of X axis(either rolling forward or backward)
                if(ball[ball_num].dx >= zero)
                {
                        ball[ball_num].vc_x = (int)(PRODUCT / ball[ball_num].dx);
        //velocity of X axis
                        ball[ball_num].dir_x = 1;               //ball is rolling forward
                }
                else if (ball[ball_num].dx <= n_zero)
                {
                        ball[ball_num].vc_x = - (int)(PRODUCT / ball[ball_num].dx);
                        ball[ball_num].dir_x = -1;              //ball is rolling backward
                }
```

```
            else                //indicating ball stops in X axis
            {
                ball[ball_num].vc_x = VC_MAX + 1;
                ball[ball_num].dir_x = 0;
                ball[ball_num].dx = 0;
            }
            //decide the moving direction of Y axis(either rolling forward or backward)
            if(ball[ball_num].dy >= zero)
            {
                ball[ball_num].vc_y = (int)(PRODUCT / ball[ball_num].dy);
        //velocity of Y axis
                ball[ball_num].dir_y = 1;               //ball is rolling forward
            }
            else if (ball[ball_num].dy <= n_zero)
            {
                ball[ball_num].vc_y = - (int)(PRODUCT / ball[ball_num].dy);
                ball[ball_num].dir_y = -1;              //ball is rolling backward
            }
            else                //indicating ball stops in Y axis
            {
                ball[ball_num].vc_y = VC_MAX + 1;
                ball[ball_num].dir_y = 0;
                ball[ball_num].dy = 0;
            }
        }
        else                                //if the ball speed has reached the stopping threshold, then
    both direction stopped moving
        {
            ball[ball_num].vc_x = VC_MAX + 1;
            ball[ball_num].dir_x = 0;
            ball[ball_num].dx = 0;
            ball[ball_num].vc_y = VC_MAX + 1;
            ball[ball_num].dir_y = 0;
            ball[ball_num].dy = 0;
        }

        //assigning the count for the ball
        if(ball[ball_num].vc_x < ball[ball_num].count_x)
        {
```

```
        ball[ball_num].count_x = ball[ball_num].vc_x;
    }

    if(ball[ball_num].vc_y < ball[ball_num].count_y)
    {
        ball[ball_num].count_y = ball[ball_num].vc_y;
    }

}
void dir_change(int i, char axle, int change_to)   //force the direction to be the one we
want(change_to)
{
    if(axle == 'x')
    {
        if((change_to == 1 && ball[i].dx < 0) || (change_to == -1 && ball[i].dx > 0))
        {
            ball[i].dx = -ball[i].dx;
        }
    }
    else if(axle == 'y')
    {
        if((change_to == 1 && ball[i].dy < 0) || (change_to == -1 && ball[i].dy > 0))
        {
            ball[i].dy = -ball[i].dy;
        }
    }
}

void detect_bound_edge(int i)
{
    int hit_flag = 0;
    float dis_v_sq;

      if(ball[i].pos_x < 45 && ball[i].pos_y < 179)  //left_up pocket area
      {
          if(39 - ball[i].pos_x > 179 - ball[i].pos_y)
          {
              swtich_d = ball[i].dx;
              ball[i].dx = ball[i].dy;
              ball[i].dy = swtich_d;
              dir_change(i, 'x', 1);
              dir_change(i, 'y', -1);
              ball[i].dx *= edge_acc;
              ball[i].dy *= edge_acc;
              decide_movement (i);
              hit_flag = 1;
          }
          else if(173 - ball[i].pos_y > 45 - ball[i].pos_x)
          {
              swtich_d = ball[i].dx;
              ball[i].dx = ball[i].dy;
              ball[i].dy = swtich_d;
              dir_change(i, 'x', -1);
              dir_change(i, 'y', 1);
              ball[i].dx *= edge_acc;
              ball[i].dy *= edge_acc;
              decide_movement (i);
              hit_flag = 1;
          }
      }
      else if(ball[i].pos_x > 594 && ball[i].pos_y < 179)  //right_up pocket area
      {
          if(ball[i].pos_x - 600 > 179 - ball[i].pos_y)
          {
              swtich_d = ball[i].dx;
              ball[i].dx = ball[i].dy;
              ball[i].dy = swtich_d;
              dir_change(i, 'x', -1);
              dir_change(i, 'y', -1);
              ball[i].dx *= edge_acc;
              ball[i].dy *= edge_acc;
              decide_movement (i);
              hit_flag = 1;
          }
          else if(173 - ball[i].pos_y > ball[i].pos_x - 594)
          {
              swtich_d = ball[i].dx;
              ball[i].dx = ball[i].dy;
          ball[i].dy = swtich_d;
          dir_change(i, 'x', 1);
          dir_change(i, 'y', 1);
          ball[i].dx *= edge_acc;
          ball[i].dy *= edge_acc;
          decide_movement (i);
          hit_flag = 1;
      }

}
else if(ball[i].pos_x < 45 && ball[i].pos_y > 440)  //left_down pocket area
{
    if(39 - ball[i].pos_x > ball[i].pos_y - 440)
    {
        swtich_d = ball[i].dx;
        ball[i].dx = ball[i].dy;
        ball[i].dy = swtich_d;
        dir_change(i, 'x', 1);
        dir_change(i, 'y', 1);
        ball[i].dx *= edge_acc;
        ball[i].dy *= edge_acc;
        decide_movement (i);
        hit_flag = 1;

    }
    else if(ball[i].pos_y - 446 > 45 - ball[i].pos_x)
    {
        swtich_d = ball[i].dx;
        ball[i].dx = ball[i].dy;
        ball[i].dy = swtich_d;
        dir_change(i, 'x', -1);
        dir_change(i, 'y', -1);
        ball[i].dx *= edge_acc;
        ball[i].dy *= edge_acc;
        decide_movement (i);
        hit_flag = 1;
    }

}
```

```
else if(ball[i].pos_x > 594 && ball[i].pos_y > 440)  //right_down pocket area
{
    if(ball[i].pos_x - 600 > ball[i].pos_y - 440)
    {
        swtich_d = ball[i].dx;
        ball[i].dx = ball[i].dy;
        ball[i].dy = swtich_d;
        dir_change(i, 'x', -1);
        dir_change(i, 'y', 1);
        ball[i].dx *= edge_acc;
        ball[i].dy *= edge_acc;
        decide_movement (i);
        hit_flag = 1;

    }
    else if(ball[i].pos_y - 446 > ball[i].pos_x - 594)
    {
        swtich_d = ball[i].dx;
        ball[i].dx = ball[i].dy;
        ball[i].dy = swtich_d;
        dir_change(i, 'x', 1);
        dir_change(i, 'y', -1);
        ball[i].dx *= edge_acc;
        ball[i].dy *= edge_acc;
        decide_movement (i);
        hit_flag = 1;
    }
}
else
{
    if(ball[i].pos_x <= 39)  //left edge
    {
        ball[i].pos_x = 39;
        dir_change(i, 'x', 1);
        ball[i].dx *= edge_acc;
        //ball[i].dy *= edge_acc;
        decide_movement (i);
        hit_flag = 1;
    }
```

```
if(ball[i].pos_x >= 600)  //right edge
{
    ball[i].pos_x = 600;
    dir_change(i, 'x', -1);
    ball[i].dx *= edge_acc;
    //ball[i].dy *= edge_acc;
    decide_movement (i);
    hit_flag = 1;
}

if(ball[i].pos_y <= 173)   //up edge
{
    if(ball[i].pos_y > 168 ) //up_mid pocket area
    {
        if(ball[i].pos_x < 336 && ball[i].pos_x > 304)
        {
            if(173 - ball[i].pos_y > ball[i].pos_x - 304)  //slide edge
            {
                swtich_d = ball[i].dx;
                ball[i].dx = ball[i].dy;
                ball[i].dy = swtich_d;
                dir_change(i, 'x', 1);
                dir_change(i, 'y', 1);

                ball[i].dx *= edge_acc;
                ball[i].dy *= edge_acc;
                decide_movement (i);
            }
            else if(173 - ball[i].pos_y > 336 - ball[i].pos_x)
            {
                swtich_d = ball[i].dx;
                ball[i].dx = ball[i].dy;
                ball[i].dy = swtich_d;
                dir_change(i, 'x', -1);
                dir_change(i, 'y', 1);

                ball[i].dx *= edge_acc;
                ball[i].dy *= edge_acc;
                decide_movement (i);
            }
        }
        else  //up edge normal area
        {
            ball[i].pos_y = 173;
            dir_change(i, 'y', 1);
            //ball[i].dx *= edge_acc;
            ball[i].dy *= edge_acc;
            decide_movement (i);
            hit_flag = 1;
         }
    }
    else   //if the ball lands near the upper mid-hole and not pocket, debug
    {
        if(ball[i].pos_x >= 325)   //vertical edge of the upper mid-hole
        {
            ball[i].pos_x = 325;
            dir_change(i, 'x', -1);
            ball[i].dx *= edge_acc;
            //ball[i].dy *= edge_acc;
            decide_movement (i);
            hit_flag = 1;
        }
        else if(ball[i].pos_x <= 315)
        {
            //ball[i].pos_x = 315;
            dir_change(i, 'x', 1);
            ball[i].dx *= edge_acc;
            //ball[i].dy *= edge_acc;
            decide_movement (i);
            hit_flag = 1;
        }
    }
}

if(ball[i].pos_y >= 446)   //down edge
{
    if(ball[i].pos_y < 451) //down_mid pocket area
    {
```

```
if(ball[i].pos_x < 336 && ball[i].pos_x > 304)
{
        if(ball[i].pos_y - 446 > ball[i].pos_x - 304)
        {
                swtich_d = ball[i].dx;
                ball[i].dx = ball[i].dy;
                ball[i].dy = swtich_d;
                dir_change(i, 'x', 1);
                dir_change(i, 'y', -1);

                ball[i].dx *= edge_acc;
                ball[i].dy *= edge_acc;
                decide_movement (i);
                hit_flag = 1;

        }
        else if(ball[i].pos_y - 446 > 336 - ball[i].pos_x)
        {
                swtich_d = ball[i].dx;
                ball[i].dx = ball[i].dy;
                ball[i].dy = swtich_d;
                dir_change(i, 'x', -1);
                dir_change(i, 'y', -1);

                ball[i].dx *= edge_acc;
                ball[i].dy *= edge_acc;
                decide_movement (i);
                hit_flag = 1;
        }

}
else  //normal
{
        ball[i].pos_y = 446;
        dir_change(i, 'y', -1);
        //ball[i].dx *= edge_acc;
        ball[i].dy *= edge_acc;
        decide_movement (i);
        hit_flag = 1;
```

```
                }
            }
            else
            {
                if(ball[i].pos_x >= 325)
                {
                    //ball[i].pos_x = 325;
                    dir_change(i, 'x', -1);
                    ball[i].dx *= edge_acc;
                    //ball[i].dy *= edge_acc;
                    decide_movement (i);
                    hit_flag = 1;
                }
                else if(ball[i].pos_x <= 315)
                {
                    ball[i].pos_x = 315;
                    dir_change(i, 'x', 1);
                    ball[i].dx *= edge_acc;
                    //ball[i].dy *= edge_acc;
                    decide_movement (i);
                    hit_flag = 1;
                }
            }
        }
    }
    if(hit_flag == 1)    // display the audio
    {
        dis_v_sq = ball[i].dx * ball[i].dx + ball[i].dy * ball[i].dy;
        if(dis_v_sq > 0.05)
        {
            IOWR_AUDIO_EN(4);
        }
        else if(dis_v_sq > 0)
        {
            IOWR_AUDIO_EN(8);
        }
    }
}
```

```
void bound_balls(int b1,int b2)     //main function for ball collision
{
        float s1=100,s2=100,s,d1,d2,x,y,dx1,dx2,dy1,dy2;
        float dis_v_sq;
        int dis1_x, dis2_x;
        int dis1_y, dis2_y;

        x = ball[b2].pos_x - ball[b1].pos_x;
        y = ball[b2].pos_y - ball[b1].pos_y;

        if(!(fabs(ball[b1].dx) < zero && fabs(ball[b1].dy) < zero))
        {
                s1 = atan2(ball[b1].dy,ball[b1].dx);
        }

        if(!(fabs(ball[b2].dx) < zero && fabs(ball[b2].dy) < zero))
        {
                s2 = atan2(ball[b2].dy,ball[b2].dx);
        }
        s = atan2(y,x);

        if(s1!=100 && fabs(s-s1) < pi/2)
        {
                d1=sqrt(ball[b1].dx * ball[b1].dx + ball[b1].dy * ball[b1].dy)*cos(s-s1);
                dx1= d1*cos(s);
                dy1= d1*sin(s);
                ball[b1].dx -= dx1;
                ball[b1].dy -= dy1;
                ball[b2].dx += dx1;
                ball[b2].dy += dy1;

                dis_v_sq = dx1*dx1 + dy1*dy1;     //display the audio
                if(dis_v_sq > 0.02)
                {
                        IOWR_AUDIO_EN(1);
                }
                else if(dis_v_sq > 0)
                {
                        IOWR_AUDIO_EN(2);
```

```
    }
}

if(s2 != 100 && fabs(s-s2) > pi/2)
{
    d2=sqrt(ball[b2].dx * ball[b2].dx + ball[b2].dy * ball[b2].dy)*cos(pi-(s-s2));
    dx2= d2*cos(pi-s);
    dy2= d2*sin(pi-s);
    ball[b1].dx += dx2;
    ball[b1].dy += dy2;
    ball[b2].dx -= dx2;
    ball[b2].dy -= dy2;

    dis_v_sq = dx1*dx1 + dy1*dy1;
    if(dis_v_sq > 0.05)
    {
        IOWR_AUDIO_EN(1);
    }
    else if(dis_v_sq > 0)
    {
        IOWR_AUDIO_EN(2);
    }
}
//velocity decrease
ball[b1].dx *= 0.90;
ball[b1].dy *= 0.90;
ball[b2].dx *= 0.90;
ball[b2].dy *= 0.90;
decide_movement (b1);
decide_movement (b2);
//if two balls overlap, then force both balls move one pixel.
if(x*x + y*y < 196)
{
    if(x > 0)
    {
        dis2_x = 1;
        dis1_x = -1;
    }
    else
```

```
        {
            dis2_x = -1;
            dis1_x = 1;
        }
        if(y > 0)
        {
            dis2_y = 1;
            dis1_y = -1;
        }
        else
        {
            dis2_y = -1;
            dis1_y = 1;
        }
       moveball(b1, &ball[b1].pos_x, &ball[b1].pos_y, dis1_x, dis1_y, &ball[b1].bias_x,
&ball[b1].bias_y);
       moveball(b2, &ball[b2].pos_x, &ball[b2].pos_y, dis2_x, dis2_y, &ball[b2].bias_x,
&ball[b2].bias_y);
        ball[b2].force_move = 1;
        IOWR_AUDIO_EN(2);
    }
}

void win(int b)     // main function for winning condition, and player changing rule.
{
    int i,m=0;
    if(b==7)    // if the black ball pockets
    {
        ex = 0;
        ball[b].pos_x = 120;
        ball[b].pos_y = 48;
        placeball(b, ball[b].pos_x, ball[b].pos_y, 0, 0, &ball[b].bias_x, &ball[b].bias_y);
        if(plball[player]==0)    // if player directly pocket the black ball without pocketing
his balls first
        {
            IOWR_VGA_WORD_EN(67);
            IOWR_VGA_WORD_POS_X(0, 52);
            IOWR_VGA_WORD_POS_Y(0, 60);
            IOWR_VGA_WORD_POS_X(1, 52);
            IOWR_VGA_WORD_POS_Y(1, 90);
            //IOWR_VGA_WORD_POS_X(2, 550);
            //IOWR_VGA_WORD_POS_Y(2, 40);
            if(player == 0)
            {
                IOWR_VGA_WORD_HL_EN(64);
                IOWR_VGA_WORD_POS_X(6, 10);
                IOWR_VGA_WORD_POS_Y(6, 90);
            }
            else
            {
                IOWR_VGA_WORD_HL_EN(64);
                IOWR_VGA_WORD_POS_X(6, 10);
                IOWR_VGA_WORD_POS_Y(6, 60);
            }
        }
        else
        {
            for(i = plball[player]-1; i < plball[player]+6; i++)
            {
                if(ball[i].flag==1)  // if player doesn't pocket all his balls before
pocketing black ball
                {
                    m=1;
                    break;
                }
            }
            if(m)
            {
                printf("win player: %d",!player);
                IOWR_VGA_WORD_EN(67);
                IOWR_VGA_WORD_POS_X(0, 52);
                IOWR_VGA_WORD_POS_Y(0, 60);
                IOWR_VGA_WORD_POS_X(1, 52);
                IOWR_VGA_WORD_POS_Y(1, 90);
                //IOWR_VGA_WORD_POS_X(2, 550);
                //IOWR_VGA_WORD_POS_Y(2, 40);
                if(player == 0)
                {
```

```
                IOWR_VGA_WORD_HL_EN(64);
                IOWR_VGA_WORD_POS_X(6, 10);
                IOWR_VGA_WORD_POS_Y(6, 60);
            }
            else
            {
                IOWR_VGA_WORD_HL_EN(64);
                IOWR_VGA_WORD_POS_X(6, 10);
                IOWR_VGA_WORD_POS_Y(6, 90);
            }
        }
        else
        {
            printf("win player: %d",player);
            IOWR_VGA_WORD_EN(71);
            IOWR_VGA_WORD_POS_X(0, 52);
            IOWR_VGA_WORD_POS_Y(0, 60);
            IOWR_VGA_WORD_POS_X(1, 52);
            IOWR_VGA_WORD_POS_Y(1, 90);
            //IOWR_VGA_WORD_POS_X(2, 550);
            //IOWR_VGA_WORD_POS_Y(2, 40);
            if(player == 0)
            {
                IOWR_VGA_WORD_HL_EN(64);
                IOWR_VGA_WORD_POS_X(6, 10);
                IOWR_VGA_WORD_POS_Y(6, 90);
            }
            else
            {
                IOWR_VGA_WORD_HL_EN(64);
                IOWR_VGA_WORD_POS_X(6, 10);
                IOWR_VGA_WORD_POS_Y(6, 60);
            }
        }
    }
}
else if(b==15)        //if the cue ball pockets
{
    oops = 1;
```

```
        ball[b].pos_x = 100;
        ball[b].pos_y = 48;
        placeball(b, ball[b].pos_x, ball[b].pos_y, 0, 0, &ball[b].bias_x, &ball[b].bias_y);
    }
    else if(plball[player]==0)      //if player first pockets a ball, then he has to pocket the
same type balls afterwards
    {
        if(b >=0 && b < 7)
        {
            plball[player]=1;
            plball[!player]=9;
            if(player == 0)
            {
                ball[b].pos_x = 100 + 17 * i_0;
                ball[b].pos_y = 68;
                placeball(b, ball[b].pos_x, ball[b].pos_y, 0, 0, &ball[b].bias_x,
&ball[b].bias_y);
                i_0 ++;
            }
            else
            {
                ball[b].pos_x = 100 + 17 * i_1;
                ball[b].pos_y = 98;
                placeball(b, ball[b].pos_x, ball[b].pos_y, 0, 0, &ball[b].bias_x,
&ball[b].bias_y);
                i_1 ++;
            }
        }
        else if(b >= 8&&b<15)
        {
            plball[player]=9;
            plball[!player]=1;
            if(player == 0)
            {
                ball[b].pos_x = 100 + 17 * i_0;
                ball[b].pos_y = 68;
                placeball(b, ball[b].pos_x, ball[b].pos_y, 0, 0, &ball[b].bias_x,
&ball[b].bias_y);
                i_0 ++;
            }
            else
            {
                ball[b].pos_x = 100 + 17 * i_1;
                ball[b].pos_y = 98;
                placeball(b, ball[b].pos_x, ball[b].pos_y, 0, 0, &ball[b].bias_x,
&ball[b].bias_y);
                i_1 ++;
            }
        }
        change=0;
    }
}

void place_cue_ball()    //function for placing the cue ball along the serve line. press space to
confirm placement
{
    int i;
    int f;
    int buff;
    SERVELINE_EN(1);

    angle = 60;
    buff = strength;
    while(cue_ready == 0)
    {
        get_key(&strength, &dir_strength, &angle, &release);
        ball[15].pos_x = 172;
        ball[15].pos_y = 446 + (int)(-2.3 * angle);
        placeball(15, ball[15].pos_x, ball[15].pos_y, 0, 0, &ball[15].bias_x,
&ball[15].bias_y);
        if(strength != buff)
        {
            f = 1;
            for(i = 0; i < 15; i++)
            {
                if(ball[i].pos_x >= 158 && ball[i].pos_x <= 186)   // if the cue ball
you want to place overlap with other balls, then placement is forbidden
                {
                    if(ball[i].pos_y >= (ball[15].pos_y - 14) && ball[i].pos_y <= (ball[15].pos_y + 14))
                    {
                        f = 0;
                        buff = strength;
                    }
                }
            }
            if(f)
            {
                cue_ready =1;
                begin_flag = 0;
                ball[15].flag = 1;
                SERVELINE_EN(0);
                print_poolcue(ball[15].pos_x, ball[15].pos_y, 60);
            }

        }

    }

    angle = 60;
}

void ball_initial()  // initialize all the balls on the table.
{
    int i;
    int ballxy_triangle[15][2]={{3,4},{1,-3},{-1,2},{-3,-1},
                                {3,-4},{3,-2},{1,-1},{-1,0},
                                {-5,0},{1,1},{3,0},{-3,1},
                                {-1,-2},{1,3},{3,2},};

    // initialize the position of the balls
    for(i = 0; i < 15; i++)             // initialize the position of the 15 balls
    {
        ball[i].pos_x = 500 + ballxy_triangle[i][0]*8;
        ball[i].pos_y = 310 + ballxy_triangle[i][1]*8;
        placeball(i, ball[i].pos_x, ball[i].pos_y, 0, 0, &ball[i].bias_x, &ball[i].bias_y);
        ball[i].flag = 1;
```

```
		ball[i].force_move = 0;
	}
	ball[15].pos_x = 172;
	ball[15].pos_y = 310;
	placeball(15, ball[15].pos_x, ball[15].pos_y, 0, 0, &ball[15].bias_x, &ball[15].bias_y);
	ball[15].flag = 1;
	// initialize the speed and direction of the balls
	for(i = 0; i < 16; i++)
	{
		ball[i].vc_x = VC_MAX + 1;
		ball[i].vc_y = VC_MAX + 1;
		ball[i].count_x = 0;
		ball[i].count_y = 0;
		ball[i].dir_x = 1;
		ball[i].dir_y = 1;
		ball[i].dx = 0;
		ball[i].dy = 0;
	}
	// initialize the strength bar
	strength = 16;
	IOWR_VGA_STRENGTHBAR(16);

	//initialize pool cue
	print_poolcue(ball[15].pos_x, ball[15].pos_y, 60);
}

int main()
{
	printf("Hello from Nios II!\n");

	int i;
	int j;
	int ff;
	float temp_angle;
	int acc_count = 0;
	short int move_flag;
	short int bias_x;
	short int bias_y;
	int dis_x;
```

```
    int dis_y;
    int hole_dis_x;
    int hole_dis_y;


while(1)
{
    //start:
        angle = 60;
        strength = 0;
        release = 0;
        cue_ready = 0;
        i_0 = 0;
        i_1 = 0;
        begin_flag = 0;
        oops = 0;
        //display "NEW GAME"
        IOWR_VGA_WORD_EN(48);
        IOWR_VGA_WORD_HL_EN(48);
        IOWR_VGA_WORD_POS_X(4, 280);
        IOWR_VGA_WORD_POS_Y(4, 60);
        IOWR_VGA_WORD_POS_X(5, 320);
        IOWR_VGA_WORD_POS_Y(5, 60);

        ball_initial();

    //start2:     wait for the user to press the enter to confirm game start.
        while(release == 0)
        {
            get_key(&strength, &dir_strength, &angle, &release);
            //if(release == 0)  //wait for player to press "return"
            //{
            //goto start2;
            //}
        }
        release = 0;

        //display main screen printouts
        IOWR_VGA_WORD_EN(3);
        IOWR_VGA_WORD_HL_EN(1);
        IOWR_VGA_WORD_POS_X(0, 52);
        IOWR_VGA_WORD_POS_Y(0, 60);
        IOWR_VGA_WORD_POS_X(1, 52);
        IOWR_VGA_WORD_POS_Y(1, 90);
        //IOWR_VGA_WORD_POS_X(2, 550);
        //IOWR_VGA_WORD_POS_Y(2, 40);
        //IOWR_VGA_WORD_POS_X(3, 550);
        //IOWR_VGA_WORD_POS_Y(3, 60);
        // wait for the user to place the cue ball, press space to confirm.
        place_cue_ball();

        while(cue_ready)
        {
            while(ex)   // if the game doesn't meet termination condition.
            {
                while(begin_flag == 0)   // wait for user to adjust the cue ball
direction and strength.press enter to confirm
                {
                    cx = ball[15].pos_x;
                    cy = ball[15].pos_y;
                    release = 0;
                    get_key(&strength, &dir_strength, &angle, &release);
                    print_poolcue(cx, cy, angle);
                    IOWR_VGA_STRENGTHBAR(strength);
                    if (release == 1)
                    {
                        begin_flag = 1;
                        temp_angle = angle_trans * angle;
                        ball[15].dx= - (strength + 1) * sq_zero *
cos(temp_angle);
                        ball[15].dy= - (strength + 1) * sq_zero *
sin(temp_angle);
                        decide_movement (15);
                        ball[15].count_x = ball[15].vc_x;
                        ball[15].count_y = ball[15].vc_y;
                    }
                }
                // cue ball release, game enter the physical collision and movement
phase.
                while(begin_flag == 1)
                {
                    release = 0;
                    for(i = 15; i >= 0; i--)
                    {
                        if(ball[i].flag != 0)
                        {
                            move_flag = 0;
                            //Update (Position update) counter. the
larger the count number, the slower the ball moves.
                                    if(ball[i].count_x > 0)
                                    {
                                        ball[i].count_x --;
                                        bias_x = 0;
                                    }
                                    else if(ball[i].vc_x <=
VC_MAX)

                                    {
                                        ball[i].count_x =
ball[i].vc_x;

                                        move_flag = 1;
                                        if(ball[i].dir_x == 1)
                                        {
                                            bias_x = 1;
                                        }
                                        else if(ball[i].dir_x ==
-1)

                                        {
                                            bias_x = -1;
                                        }
                                        else
                                        {
                                            bias_x = 0;
                                        }
                                    }
                                    else
                                    {
                                        bias_x = 0;
                                    }

                                    if(ball[i].count_y > 0)
                                    {
                                        ball[i].count_y --;
                                        bias_y = 0;
                                    }
                                    else if(ball[i].vc_y <=
VC_MAX)

                                    {
                                        ball[i].count_y =
ball[i].vc_y;

                                        move_flag = 1;
                                        if(ball[i].dir_y == 1)
                                        {
                                            bias_y = 1;
                                        }
                                        else if(ball[i].dir_y ==
-1)

                                        {
                                            bias_y = -1;
                                        }
                                        else
                                        {
                                            bias_y = 0;
                                        }
                                    }
                                    else
                                    {
                                        bias_y = 0;
                                    }


                                    //Update positions
                                    if(move_flag ||
ball[i].force_move)

                                    {
                                        if(move_flag)
                                        {
                                            moveball(i,
&ball[i].pos_x, &ball[i].pos_y, bias_x, bias_y, &ball[i].bias_x, &ball[i].bias_y);
                                        }

                                        ball[i].force_move =
0;
                                        //detect and handle
the situation when a ball hits edge
                                        detect_bound_edge(i);

                                        //detect and handle
the situation when a ball hits other balls
                                        for(j = 15; j >= 0; j--)
                                        {
                                            if(j != i)
                                            {
                                                dis_x =
abs(ball[i].pos_x - ball[j].pos_x);
                                                dis_y =
abs(ball[i].pos_y - ball[j].pos_y);

    if(dis_x < 14 || dis_y < 14)
                                                {

    if(dis_x * dis_x + dis_y * dis_y <= 196)

    {

    bound_balls(i, j);
                                                   }
                                                }
                                            }
                                        }
                                        //detect and handle
the situation when a ball pockets
                                        for(j = 0; j < 6; j++)
                                        {
```

```
                                                                    hole_dis_x =
ball[i].pos_x - hole[j][0];
                                                                    hole_dis_y =
ball[i].pos_y - hole[j][1];
                                                                    if(hole_dis_x
* hole_dis_x + hole_dis_y * hole_dis_y <= 144)
                                                                    {

     //printf("pocketed!\n");

     ball[i].flag = 0;

     if(plball[player]!=0 && oops == 0) // if player pocket before and it's not the cue ball
pocketed
                                                                         {

     if(i >= (plball[player]-1) && i <= plball[player]+5) // if player is pocketing his balls, then
no need to switch turn

     {

     change=0;

     if(player == 0)

     {

          ball[i].pos_x = 100 + 17 * i_0;

          ball[i].pos_y = 68;

          placeball(i, ball[i].pos_x, ball[i].pos_y, 0, 0, &ball[i].bias_x, &ball[i].bias_y);

          i_0 ++;

     }

     else
```

```
        {

                ball[i].pos_x = 100 + 17 * i_1;

                ball[i].pos_y = 98;

                placeball(i, ball[i].pos_x, ball[i].pos_y, 0, 0, &ball[i].bias_x, &ball[i].bias_y);

                i_1 ++;

        }
                                                                                                        }

        else if (i >= (plball[!player]-1) && i <= plball[!player]+5)          // if player is
pocketing other's balls, then need to switch turn

        {

        change=1;

        if(player == 0)

        {

                ball[i].pos_x = 100 + 17 * i_1;

                ball[i].pos_y = 98;

                placeball(i, ball[i].pos_x, ball[i].pos_y, 0, 0, &ball[i].bias_x, &ball[i].bias_y);

                i_1 ++;

        }

        else

        {
```

```
        ball[i].pos_x = 100 + 17 * i_0;

        ball[i].pos_y = 68;

        placeball(i, ball[i].pos_x, ball[i].pos_y, 0, 0, &ball[i].bias_x, &ball[i].bias_y);

        i_0 ++;

    }
                                                   }
                                                }
                                                win(i);

                                            }
                                        }
                                    }
                        }

                        //this function is used to regulate the overall loop
time to be a fixed value.
                        while(IORD_16DIRECT(IRTIMER_0_BASE,
0) != 0)
                        {

                        }

                        IOWR_16DIRECT(IRTIMER_0_BASE, 0, 3000);

                    } //16 ball

                    //Update Speed (dx,dy)
                    if(acc_count >= ACC)
                    {
                        acc_count = 0;
                        for(i = 16; i >= 0; i--)
                        {
                            ball[i].dx = ball[i].dx * 0.95;
                            ball[i].dy = ball[i].dy * 0.95;
                            decide_movement(i);
                        }
                    }
                    else
                    {
                        acc_count ++;
                    }


                    ff=1;

                    for(j = 15; j >= 0; j--)
                    {
                        if(!(ball[j].dx == 0 && ball[j].dy ==0) &&
ball[j].flag == 1)      //if there're still balls moving
                        {
                            ff=0;
                            break;
                        }
                    }
                    if(ff)          //if all the balls have stopped
                    {
                        begin_flag = 0;
                        release = 0;
                        if(ex == 1)
                        {
                            if(change == 1)
                            {
                                player=!player;
                                strength = 16;
                                dir_strength = 1;

                                if(player == 0)
                                {

    IOWR_VGA_WORD_HL_EN(5);
                                }
                                else
                                {
```

```
    IOWR_VGA_WORD_HL_EN(6);
                                }
                            }
                        }
                        change=1;
                        if(oops == 1)           //see if the cue ball is
pocketed.
                        {
                            cue_ready = 0;
                            place_cue_ball();
                            change=1;
                            oops = 0;
                        }
                    }

                }

            } //End: while(ex)
            get_key(&strength, &dir_strength, &angle, &release);
            if(release == 1)
            {

                release = 0;
                ex = 1;
                break;
                //goto start;
            }
            //}
        }  //End: while(cue_ready)
   }
    return 0;
}
```

-- Keyboard.h

```
/*
 * keyboard.h
 *
 *  Created on: Apr 21, 2013
 *      Author: Zeshi Wang
 */

#ifndef KEYBOARD_H_
#define KEYBOARD_H_

#include <io.h>
#include <system.h>
#include <stdio.h>

unsigned char code;

int cx,cy;
int strength_increase = 1;

void get_key(int *strength, int *dir_strength, int *angle, int *release)
{
            int temp_strength = 0;
            int temp_angle = 0;
            int temp_release = 0;

            strength_increase = *dir_strength;


            temp_strength = *strength;
            temp_angle = *angle;
            temp_release = *release;

            while(IORD_8DIRECT(DE2_PS2_0_BASE,1) != 0x5a)
            {

                                    while(!IORD_8DIRECT(DE2_PS2_0_BASE, 0));

                                    code = IORD_8DIRECT(DE2_PS2_0_BASE,1);
                                    //printf("polled status is %x \n" , IORD_8DIRECT(DE2_PS2_0_BASE,
1));

                                    //printf("code is %x \n", code);
                      switch(code)
```

```
{
     case 0x29: // Space

     if(temp_strength == 31)
     {
              strength_increase = -1;
     }
     else if(temp_strength == 0)
     {
        strength_increase = 1;
     }
     temp_strength = temp_strength + strength_increase;

       break;
     case 0x6B: // LEFT Key
        temp_angle = temp_angle - 1;
                      if(temp_angle < 0)
                                      temp_angle = 119;
                      if(temp_angle > 119)
                                      temp_angle = 0;
       break;
     case 0x75: // UP Key
              temp_angle = temp_angle + 10;
                     if(temp_angle < 0)
                           temp_angle = 119;
                     if(temp_angle > 119)
                           temp_angle = 0;
                     break;
     case 0x72: // DOWN Key
              temp_angle = temp_angle - 10;
                     if(temp_angle < 0)
                           temp_angle = 119;
                     if(temp_angle > 119)
                           temp_angle = 0;
                     break;
     case 0x74: // RIGHT Key
       temp_angle = temp_angle + 1;
        if(temp_angle < 0)
                     temp_angle = 119;
                     if(temp_angle > 119)
                               temp_angle = 0;

                    break;
                   case 0x5a: //enter
                       temp_release = 1;
                    break;
                   case 0x76: //escape

                   break;

                  default:

                    break;
                 }
             break;
        }


       *strength = temp_strength;
       *angle = temp_angle;
       *release = temp_release;
       *dir_strength = strength_increase;

}


#endif /* KEYBOARD_H_ */

-- poolcue.h
/*
 * poolcue.h
 *
 *  Created on: Apr 3, 2013
 *      Author: Jiawan Zhang
 */

#ifndef POOLCUE_H_
#define POOLCUE_H_
```

```
//-------------------------------------------

#include <io.h>
#include <system.h>
#include <stdio.h>

#define IOWR_CUERAM(line,data)\
        IOWR_32DIRECT(POOLCUE_RAM_CONTROLLER_0_BASE, line*4, data)

alt_u16 CueBegin[31][355] = {

        {205,77,39,15,15,15,15,39,77,205,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0},

        {0,0,0,0,15,15,15,15,34,39,53,72,77,91,110,129,148,167,186,205,205,224,243,262,282,3
01,320,339,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,15,15,15,15,19,29,38,39,48,57,67,76,76,86,95,105,114,124,133,143,152,162,171
,181,190,200,203,209,219,228,238,247,257,266,276,285,295,304,314,323,333,342,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,15,15,15,15,21,27,33,38,39,46,52,58,65,71,76,77,84,90,96,103,109,115,121,12
8,134,140,147,153,159,166,172,178,185,191,197,202,203,210,216,222,229,235,241,248,254,260
,267,273,279,286,292,298,305,311,317,323,330,336,342,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0},

          {0,0,0,0,0,15,15,15,14,17,21,26,31,35,38,40,45,49,54,59,63,68,73,75,78,82,87,92,96,101,
106,110,115,120,125,129,134,139,143,148,153,157,162,167,172,176,181,186,190,195,200,200,2
04,209,214,219,223,228,233,237,242,247,251,256,261,266,270,275,280,284,289,294,299,303,30
8,313,317,322,327,331,336,341,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

          {0,0,0,0,0,0,15,15,14,14,18,22,25,29,33,36,37,40,44,48,51,55,59,62,66,70,74,74,77,81,85
,88,92,96,100,103,107,111,115,118,122,126,129,133,137,141,144,148,152,156,159,163,167,171,
174,178,182,185,189,193,197,197,200,204,208,211,215,219,223,226,230,234,238,241,245,249,2
53,256,260,264,267,271,275,279,282,286,290,294,297,301,305,309,312,316,320,323,327,331,33
5,338,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

          {0,0,0,0,0,0,15,15,14,14,15,19,22,25,28,31,34,37,37,40,43,46,49,52,55,58,61,65,68,71,72
,74,77,80,83,86,89,92,95,98,101,104,107,111,114,117,120,123,126,129,132,135,138,141,144,14
7,151,154,157,160,163,166,169,172,175,178,181,184,187,191,194,194,197,200,203,206,209,212
,215,218,221,224,227,230,233,237,240,243,246,249,252,255,258,261,264,267,270,274,277,280,
283,286,289,292,295,298,301,304,307,310,314,317,320,323,326,329,332,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,15,14,14,14,16,19,22,24,27,29,32,35,36,37,40,42,45,47,50,53,55,58,60,63,
66,68,71,71,73,76,79,81,84,86,89,92,94,97,99,102,105,107,110,112,115,118,120,123,125,128,13
1,133,136,138,141,144,146,149,151,154,157,159,162,164,167,170,172,175,177,180,183,185,188
,190,190,193,196,198,201,203,206,209,211,214,216,219,222,224,227,229,232,235,237,240,242,
245,248,250,253,255,258,261,263,266,268,271,274,276,279,282,284,287,289,292,295,297,300,3
02,305,308,310,313,315,318,321,323,326,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,15,14,14,13,15,17,19,21,24,26,28,30,33,35,35,37,39,41,44,46,48,50,53,55,
57,59,62,64,66,68,69,70,73,75,77,79,82,84,86,88,91,93,95,97,100,102,104,106,109,111,113,115,
118,120,122,124,127,129,131,133,136,138,140,142,145,147,149,151,154,156,158,160,162,165,1
67,169,171,174,176,178,180,183,185,186,187,189,191,194,196,198,200,203,205,207,209,212,21
4,216,218,221,223,225,227,230,232,234,236,239,241,243,245,248,250,252,254,257,259,261,263
,266,268,270,272,275,277,279,281,284,286,288,290,293,295,297,299,302,304,306,308,311,313,
315,317,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0},

{0,0,0,0,0,0,0,0,14,14,13,13,15,17,19,21,23,25,27,29,31,33,34,35,37,39,41,43,45,46,48,5
0,52,54,56,58,60,62,64,66,68,68,70,72,74,76,78,80,82,84,85,87,89,91,93,95,97,99,101,103,105,1
07,109,111,113,115,117,119,121,123,125,127,129,131,133,135,137,138,140,142,144,146,148,15
0,152,154,156,158,160,162,164,166,168,170,172,174,176,178,180,181,181,183,185,187,189,191
,193,195,197,199,201,203,205,207,209,211,213,215,217,219,221,223,225,227,229,230,232,234,
236,238,240,242,244,246,248,250,252,254,256,258,260,262,264,266,268,270,272,274,276,278,2
80,282,283,285,287,289,291,293,295,297,299,301,303,305,307,309,311,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,14,13,13,14,16,17,19,21,23,24,26,28,30,31,33,33,35,36,38,40,41,43,45,
47,48,50,52,54,55,57,59,60,62,64,66,65,67,69,71,72,74,76,78,79,81,83,84,86,88,90,91,93,95,97,
98,100,102,104,105,107,109,110,112,114,116,117,119,121,123,124,126,128,130,131,133,135,13
```

```
6,138,140,142,143,145,147,149,150,152,154,155,157,159,161,162,164,166,168,169,171,173,175
,176,176,178,179,181,183,185,186,188,190,192,193,195,197,199,200,202,204,205,207,209,211,
212,214,216,218,219,221,223,224,226,228,230,231,233,235,237,238,240,242,244,245,247,249,2
50,252,254,256,257,259,261,263,264,266,268,270,271,273,275,276,278,280,282,283,285,287,28
9,290,292,294,296,297,299,301,302,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,14,13,13,13,14,16,17,19,21,22,24,25,27,28,30,31,32,33,34,36,37,39,40,
42,43,45,46,48,50,51,53,54,56,57,59,60,62,63,63,65,66,68,69,71,72,74,75,77,78,80,82,83,85,86,
88,89,91,92,94,95,97,99,100,102,103,105,106,108,109,111,112,114,115,117,119,120,122,123,12
5,126,128,129,131,132,134,135,137,139,140,142,143,145,146,148,149,151,152,154,155,157,159
,160,162,163,165,166,168,169,170,171,172,174,175,177,178,180,181,183,184,186,188,189,191,
192,194,195,197,198,200,201,203,204,206,208,209,211,212,214,215,217,218,220,221,223,224,2
26,228,229,231,232,234,235,237,238,240,241,243,245,246,248,249,251,252,254,255,257,258,26
0,261,263,265,266,268,269,271,272,274,275,277,278,280,281,283,285,286,288,289,291,292,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,14,13,13,12,13,15,16,17,19,20,22,23,24,26,27,28,30,31,31,32,34,35,36,
38,39,40,42,43,45,46,47,49,50,52,53,54,56,57,58,60,61,61,62,64,65,66,68,69,70,72,73,75,76,77,
79,80,81,83,84,86,87,88,90,91,92,94,95,97,98,99,101,102,103,105,106,108,109,110,112,113,114
,116,117,119,120,121,123,124,126,127,128,130,131,132,134,135,137,138,139,141,142,143,145,
146,148,149,150,152,153,154,156,157,159,160,161,163,164,164,165,166,168,169,171,172,173,1
75,176,177,179,180,182,183,184,186,187,188,190,191,193,194,195,197,198,200,201,202,204,20
5,206,208,209,211,212,213,215,216,217,219,220,222,223,224,226,227,228,230,231,233,234,235
,237,238,239,241,242,244,245,246,248,249,250,252,253,255,256,257,259,260,261,263,264,266,
267,268,270,271,272,274,275,277,278,279,281,282,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,13,13,12,12,14,15,16,17,18,20,21,22,23,25,26,27,28,30,30,30,32,33,3
4,35,37,38,39,40,42,43,44,45,47,48,49,50,51,53,54,55,56,58,59,59,60,61,62,63,65,66,67,68,70,7
1,72,73,75,76,77,78,80,81,82,83,84,86,87,88,89,91,92,93,94,96,97,98,99,101,102,103,104,105,1
07,108,109,110,112,113,114,115,117,118,119,120,122,123,124,125,126,128,129,130,131,133,13
4,135,136,138,139,140,141,143,144,145,146,147,149,150,151,152,154,155,156,157,157,158,159
```

```
,161,162,163,164,166,167,168,169,171,172,173,174,175,177,178,179,180,182,183,184,185,187,
188,189,190,192,193,194,195,196,198,199,200,201,203,204,205,206,208,209,210,211,213,214,2
15,216,217,219,220,221,222,224,225,226,227,229,230,231,232,234,235,236,237,238,240,241,24
2,243,245,246,247,248,250,251,252,253,255,256,257,258,259,261,262,263,264,266,267,268,269
,271,273,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,13,12,11,13,14,15,16,17,18,19,20,21,23,24,25,26,27,28,29,29,30,31
,32,33,34,36,37,38,39,40,41,42,43,44,45,47,48,49,50,51,52,53,54,55,56,56,57,58,60,61,62,63,64,
65,66,67,68,70,71,72,73,74,75,76,77,78,80,81,82,83,84,85,86,87,88,90,91,92,93,94,95,96,97,98,
100,101,102,103,104,105,106,107,108,110,111,112,113,114,115,116,117,118,120,121,122,123,1
24,125,126,127,128,130,131,132,133,134,135,136,137,138,140,141,142,143,144,145,146,147,14
8,150,151,150,151,152,154,155,156,157,158,159,160,161,162,164,165,166,167,168,169,170,171
,172,174,175,176,177,178,179,180,181,182,184,185,186,187,188,189,190,191,192,194,195,196,
197,198,199,200,201,202,204,205,206,207,208,209,210,211,212,214,215,216,217,218,219,220,2
21,222,224,225,226,227,228,229,230,231,232,234,235,236,237,238,239,240,241,242,244,245,24
6,247,248,249,250,251,252,254,255,256,257,258,259,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,13,12,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,27,28
,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,53,54,55,56,57,58,
59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,
90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,
115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,1
37,138,139,140,141,142,143,143,143,144,145,146,147,148,149,150,151,152,153,154,155,156,15
7,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179
,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,
202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,2
24,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,24
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,13,12,11,11,12,13,14,14,15,16,17,18,19,20,21,22,23,23,24,25,26,26
,27,28,28,29,30,31,32,33,34,35,36,37,37,38,39,40,41,42,43,44,45,46,46,47,48,49,50,51,50,51,52,
53,54,55,56,57,58,59,59,60,61,62,63,64,65,66,67,68,68,69,70,71,72,73,74,75,76,77,78,78,79,80,
81,82,83,84,85,86,87,87,88,89,90,91,92,93,94,95,96,96,97,98,99,100,101,102,103,104,105,105,1
06,107,108,109,110,111,112,113,114,114,115,116,117,118,119,120,121,122,123,123,124,125,12
```

```
6,127,128,129,130,131,132,132,133,134,135,136,136,136,137,138,139,140,141,142,143,144,145
,145,146,147,148,149,150,151,152,153,154,154,155,156,157,158,159,160,161,162,163,163,164,
165,166,167,168,169,170,171,172,173,173,174,175,176,177,178,179,180,181,182,182,183,184,1
85,186,187,188,189,190,191,191,192,193,194,195,196,197,198,199,200,200,201,202,203,204,20
5,206,207,208,209,209,210,211,212,213,214,215,216,217,218,218,219,220,221,222,223,224,225
,226,227,227,228,229,230,231,232,233,235,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0},

            {0,0,0,0,0,0,0,0,0,0,0,0,12,10,10,11,12,13,13,14,15,16,17,17,18,19,20,21,21,22,23,24,25,
24,25,26,27,27,28,29,30,31,32,32,33,34,35,36,36,37,38,39,40,40,41,42,43,44,44,45,46,47,48,47,
48,49,50,50,51,52,53,54,55,55,56,57,58,59,59,60,61,62,63,63,64,65,66,67,67,68,69,70,71,72,72,
73,74,75,76,76,77,78,79,80,80,81,82,83,84,84,85,86,87,88,89,89,90,91,92,93,93,94,95,96,97,97,
98,99,100,101,101,102,103,104,105,106,106,107,108,109,110,110,111,112,113,114,114,115,116
,117,118,118,119,120,121,122,123,123,124,125,126,127,127,127,128,129,129,130,131,132,133,
133,134,135,136,137,137,138,139,140,141,141,142,143,144,145,146,146,147,148,149,150,150,1
51,152,153,154,154,155,156,157,158,158,159,160,161,162,163,163,164,165,166,167,167,168,16
9,170,171,171,172,173,174,175,176,176,177,178,179,180,180,181,182,183,184,184,185,186,187
,188,188,189,190,191,192,193,193,194,195,196,197,197,198,199,200,201,201,202,203,204,205,
205,206,207,208,209,210,210,211,212,213,214,214,215,216,217,218,218,219,221,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

            {0,0,0,0,0,0,0,0,0,0,0,0,12,10,9,10,11,12,12,13,14,15,15,16,17,17,18,19,20,20,21,22,23,2
3,23,23,24,25,26,26,27,28,29,29,30,31,31,32,33,34,34,35,36,37,37,38,39,39,40,41,42,42,43,44,4
5,44,45,45,46,47,48,48,49,50,51,51,52,53,53,54,55,56,56,57,58,59,59,60,61,61,62,63,64,64,65,6
6,67,67,68,69,69,70,71,72,72,73,74,75,75,76,77,77,78,79,80,80,81,82,83,83,84,85,85,86,87,88,8
8,89,90,91,91,92,93,93,94,95,96,96,97,98,99,99,100,101,101,102,103,104,104,105,106,106,107,
108,109,109,110,111,112,112,113,114,114,115,116,117,117,118,119,119,119,120,121,121,122,1
23,123,124,125,126,126,127,128,129,129,130,131,131,132,133,134,134,135,136,136,137,138,13
9,139,140,141,142,142,143,144,144,145,146,147,147,148,149,150,150,151,152,152,153,154,155
,155,156,157,158,158,159,160,160,161,162,163,163,164,165,166,166,167,168,168,169,170,171,
171,172,173,174,174,175,176,176,177,178,179,179,180,181,182,182,183,184,184,185,186,187,1
87,188,189,190,190,191,192,192,193,194,195,195,196,197,198,198,199,200,200,201,202,203,20
3,204,206,208,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

            {0,0,0,0,0,0,0,0,0,0,0,0,0,10,9,9,10,11,11,12,13,13,14,15,15,16,17,17,18,19,19,20,20,21,2
2,21,22,23,23,24,24,25,26,26,27,28,28,29,30,30,31,32,32,33,34,34,35,36,36,37,37,38,39,39,40,4
```

```
1,41,41,41,42,43,43,44,45,45,46,47,47,48,49,49,50,51,51,52,53,53,54,54,55,56,56,57,58,58,59,6
0,60,61,62,62,63,64,64,65,65,66,67,67,68,69,69,70,71,71,72,73,73,74,75,75,76,77,77,78,78,79,8
0,80,81,82,82,83,84,84,85,86,86,87,88,88,89,90,90,91,91,92,93,93,94,95,95,96,97,97,98,99,99,1
00,101,101,102,103,103,104,104,105,106,106,107,108,108,109,110,110,110,110,111,112,112,11
3,114,114,115,116,116,117,118,118,119,120,120,121,121,122,123,123,124,125,125,126,127,127
,128,129,129,130,131,131,132,132,133,134,134,135,136,136,137,138,138,139,140,140,141,142,
142,143,144,144,145,145,146,147,147,148,149,149,150,151,151,152,153,153,154,155,155,156,1
57,157,158,158,159,160,160,161,162,162,163,164,164,165,166,166,167,168,168,169,170,170,17
1,171,172,173,173,174,175,175,176,177,177,178,179,179,180,181,181,182,182,183,184,184,185
,186,186,187,188,188,189,191,191,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,10,9,9,9,10,10,11,12,12,13,13,14,15,15,16,16,17,17,18,19,19,20
,20,20,20,21,21,22,23,23,24,24,25,25,26,27,27,28,28,29,30,30,31,31,32,32,33,34,34,35,35,36,36,
37,38,38,38,38,39,39,40,41,41,42,42,43,43,44,45,45,46,46,47,47,48,49,49,50,50,51,51,52,53,53,
54,54,55,56,56,57,57,58,58,59,60,60,61,61,62,62,63,64,64,65,65,66,66,67,68,68,69,69,70,71,71,
72,72,73,73,74,75,75,76,76,77,77,78,79,79,80,80,81,81,82,83,83,84,84,85,86,86,87,87,88,88,89,
90,90,91,91,92,92,93,94,94,95,95,96,97,97,98,98,99,99,100,101,101,101,101,102,102,103,103,1
04,105,105,106,106,107,107,108,109,109,110,110,111,112,112,113,113,114,114,115,116,116,11
7,117,118,118,119,120,120,121,121,122,122,123,124,124,125,125,126,127,127,128,128,129,129
,130,131,131,132,132,133,133,134,135,135,136,136,137,138,138,139,139,140,140,141,142,142,
143,143,144,144,145,146,146,147,147,148,148,149,150,150,151,151,152,153,153,154,154,155,1
55,156,157,157,158,158,159,159,160,161,161,162,162,163,163,164,165,165,166,166,167,168,16
8,169,169,170,170,171,172,172,173,173,174,176,177,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,10,8,8,9,9,10,10,11,11,12,12,13,13,14,14,15,15,16,16,17,17,18,
18,18,18,19,19,20,20,21,21,22,22,23,23,24,24,25,25,26,26,27,27,28,28,29,29,30,30,31,31,32,32,
33,33,34,34,35,34,35,35,36,36,37,37,38,38,39,39,40,41,41,42,42,43,43,44,44,45,45,46,46,47,47,
48,48,49,49,50,50,51,51,52,52,53,53,54,54,55,55,56,56,57,57,58,58,59,59,60,60,61,61,62,62,63,
63,64,64,65,65,66,66,67,68,68,69,69,70,70,71,71,72,72,73,73,74,74,75,75,76,76,77,77,78,78,79,
79,80,80,81,81,82,82,83,83,84,84,85,85,86,86,87,87,88,88,89,89,90,90,91,91,92,91,92,92,93,93,
94,94,95,95,96,96,97,97,98,98,99,100,100,101,101,102,102,103,103,104,104,105,105,106,106,1
07,107,108,108,109,109,110,110,111,111,112,112,113,113,114,114,115,115,116,116,117,117,11
8,118,119,119,120,120,121,121,122,122,123,123,124,124,125,125,126,127,127,128,128,129,129
,130,130,131,131,132,132,133,133,134,134,135,135,136,136,137,137,138,138,139,139,140,140,
141,141,142,142,143,143,144,144,145,145,146,146,147,147,148,148,149,149,150,150,151,151,1
52,153,153,154,154,155,155,156,156,157,157,158,159,161,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
```

{0,0,0,0,0,0,0,0,0,0,0,0,0,10,8,7,8,8,9,9,10,10,11,11,11,12,12,13,13,14,14,15,15,15,16,16,
17,16,17,17,18,18,18,19,19,20,20,21,21,22,22,22,23,23,24,24,25,25,26,26,26,27,27,28,28,29,29,
30,30,30,31,31,31,31,32,32,32,33,33,34,34,35,35,36,36,36,37,37,38,38,39,39,40,40,40,41,41,42,
42,43,43,44,44,44,45,45,46,46,47,47,48,48,48,49,49,50,50,51,51,52,52,52,53,53,54,54,55,55,56,
56,56,57,57,58,58,59,59,60,60,61,61,61,62,62,63,63,64,64,65,65,65,66,66,67,67,68,68,69,69,69,
70,70,71,71,72,72,73,73,73,74,74,75,75,76,76,77,77,77,78,78,79,79,80,80,81,81,81,82,82,82,82,
83,83,83,84,84,85,85,86,86,87,87,87,88,88,89,89,90,90,91,91,91,92,92,93,93,94,94,95,95,95,96,
96,97,97,98,98,99,99,99,100,100,101,101,102,102,103,103,103,104,104,105,105,106,106,107,10
7,107,108,108,109,109,110,110,111,111,112,112,112,113,113,114,114,115,115,116,116,116,117
,117,118,118,119,119,120,120,120,121,121,122,122,123,123,124,124,124,125,125,126,126,127,
127,128,128,128,129,129,130,130,131,131,132,132,132,133,133,134,134,135,135,136,136,136,1
37,137,138,138,139,139,140,140,140,141,141,143,144,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,7,7,8,8,8,9,9,10,10,10,11,11,11,12,12,13,13,13,14,14,15,15,
15,15,15,15,16,16,17,17,17,18,18,18,19,19,20,20,20,21,21,22,22,22,23,23,23,24,24,25,25,25,26,
26,27,27,27,28,27,27,28,28,29,29,29,30,30,30,31,31,32,32,32,33,33,34,34,34,35,35,35,36,36,37,
37,37,38,38,39,39,39,40,40,40,41,41,42,42,42,43,43,44,44,44,45,45,45,46,46,47,47,47,48,48,49,
49,49,50,50,50,51,51,52,52,52,53,53,53,54,54,55,55,55,56,56,57,57,57,58,58,58,59,59,60,60,60,
61,61,62,62,62,63,63,63,64,64,65,65,65,66,66,67,67,67,68,68,68,69,69,70,70,70,71,71,72,72,72,
73,72,72,73,73,74,74,74,75,75,75,76,76,77,77,77,78,78,79,79,79,80,80,80,81,81,82,82,82,83,83,
84,84,84,85,85,85,86,86,87,87,87,88,88,89,89,89,90,90,90,91,91,92,92,92,93,93,93,94,94,95,95,
95,96,96,97,97,97,98,98,98,99,99,100,100,100,101,101,102,102,102,103,103,103,104,104,105,1
05,105,106,106,107,107,107,108,108,108,109,109,110,110,110,111,111,112,112,112,113,113,11
3,114,114,115,115,115,116,116,117,117,117,118,118,118,119,119,120,120,120,121,121,122,122
,122,123,123,123,124,124,126,126,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,6,7,7,7,8,8,8,9,9,9,10,10,10,11,11,11,12,12,12,12,13,13,13,1
3,13,13,14,14,14,15,15,15,16,16,16,17,17,17,18,18,18,19,19,19,20,20,20,21,21,21,22,22,22,22,2
3,23,23,24,24,24,24,24,24,25,25,25,26,26,26,27,27,27,28,28,28,29,29,29,30,30,30,31,31,31,31,3
2,32,32,33,33,33,34,34,34,35,35,35,36,36,36,37,37,37,38,38,38,39,39,39,40,40,40,41,41,41,42,4
2,42,43,43,43,44,44,44,44,45,45,45,46,46,46,47,47,47,48,48,48,49,49,49,50,50,50,51,51,51,52,5
2,52,53,53,53,54,54,54,55,55,55,56,56,56,57,57,57,57,58,58,58,59,59,59,60,60,60,61,61,61,62,6
2,62,63,63,62,63,63,63,64,64,64,65,65,65,66,66,66,67,67,67,67,68,68,68,69,69,69,70,70,70,71,7
1,71,72,72,72,73,73,73,74,74,74,75,75,75,76,76,76,77,77,77,78,78,78,79,79,79,80,80,80,80,81,8
1,81,82,82,82,83,83,83,84,84,84,85,85,85,86,86,86,87,87,87,88,88,88,89,89,89,90,90,90,91,91,9
1,92,92,92,93,93,93,93,94,94,94,95,95,95,96,96,96,97,97,97,98,98,98,99,99,99,100,100,100,101,

```
101,101,102,102,102,103,103,103,104,104,104,105,105,105,106,106,106,106,107,107,107,109,1
10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,6,6,6,7,7,7,7,8,8,8,8,9,9,9,9,10,10,10,11,11,11,11,12,11,11,1
1,12,12,12,12,13,13,13,14,14,14,14,15,15,15,15,16,16,16,16,17,17,17,18,18,18,18,19,19,19,19,2
0,20,20,21,20,20,20,21,21,21,21,22,22,22,22,23,23,23,24,24,24,24,25,25,25,25,26,26,26,26,27,2
7,27,28,28,28,28,29,29,29,29,30,30,30,30,31,31,31,32,32,32,32,33,33,33,33,34,34,34,34,35,35,3
5,36,36,36,36,37,37,37,37,38,38,38,39,39,39,39,40,40,40,40,41,41,41,41,42,42,42,43,43,43,43,4
4,44,44,44,45,45,45,45,46,46,46,47,47,47,47,48,48,48,48,49,49,49,49,50,50,50,51,51,51,51,52,5
2,52,52,53,52,52,52,53,53,53,54,54,54,54,55,55,55,55,56,56,56,56,57,57,57,58,58,58,58,59,59,5
9,59,60,60,60,61,61,61,61,62,62,62,62,63,63,63,63,64,64,64,65,65,65,65,66,66,66,66,67,67,67,6
7,68,68,68,69,69,69,69,70,70,70,70,71,71,71,72,72,72,72,73,73,73,73,74,74,74,74,75,75,75,76,7
6,76,76,77,77,77,77,78,78,78,78,79,79,79,80,80,80,80,81,81,81,81,82,82,82,82,83,83,83,84,84,8
4,84,85,85,85,85,86,86,86,87,87,87,87,88,88,88,88,89,89,89,89,90,91,91,93,0,0,0,0,0,0,0,0,0,0,0
},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,5,5,6,6,6,6,6,7,7,7,7,8,8,8,8,8,9,9,9,9,9,10,10,10,9,9,10,10,1
0,10,11,11,11,11,11,12,12,12,12,12,13,13,13,13,14,14,14,14,14,15,15,15,15,15,16,16,16,16,16,1
7,17,16,16,17,17,17,17,17,18,18,18,18,18,19,19,19,19,20,20,20,20,20,21,21,21,21,21,22,22,22,2
2,22,23,23,23,23,24,24,24,24,24,25,25,25,25,25,26,26,26,26,27,27,27,27,27,28,28,28,28,28,29,2
9,29,29,29,30,30,30,30,31,31,31,31,31,32,32,32,32,32,33,33,33,33,34,34,34,34,34,35,35,35,35,3
5,36,36,36,36,37,37,37,37,37,38,38,38,38,38,39,39,39,39,39,40,40,40,40,41,41,41,41,41,42,42,4
2,42,42,42,42,42,42,43,43,43,43,43,44,44,44,44,44,45,45,45,45,45,46,46,46,46,47,47,47,47,47,4
8,48,48,48,48,49,49,49,49,50,50,50,50,50,51,51,51,51,51,52,52,52,52,52,53,53,53,53,54,54,54,5
4,54,55,55,55,55,55,56,56,56,56,57,57,57,57,57,58,58,58,58,58,59,59,59,59,60,60,60,60,60,61,6
1,61,61,61,62,62,62,62,62,63,63,63,63,64,64,64,64,64,65,65,65,65,65,66,66,66,66,67,67,67,67,6
7,68,68,68,68,68,69,69,69,69,70,70,70,70,70,71,71,71,71,71,72,72,72,73,73,75,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,5,5,5,5,5,6,6,6,6,6,6,7,7,7,7,7,7,7,8,8,8,8,8,7,8,8,8,8,8,8,9,
9,9,9,9,9,9,10,10,10,10,10,10,11,11,11,11,11,11,12,12,12,12,12,12,12,13,13,13,13,13,12,13,13,1
3,13,13,13,14,14,14,14,14,14,14,15,15,15,15,15,15,16,16,16,16,16,16,17,17,17,17,17,17,18,18,1
8,18,18,18,18,19,19,19,19,19,19,20,20,20,20,20,20,21,21,21,21,21,21,21,22,22,22,22,22,22,23,2
3,23,23,23,23,24,24,24,24,24,24,24,25,25,25,25,25,25,26,26,26,26,26,26,27,27,27,27,27,27,27,2
8,28,28,28,28,28,29,29,29,29,29,29,30,30,30,30,30,30,30,31,31,31,31,31,31,32,32,32,32,32,31,3
2,32,32,32,32,32,32,33,33,33,33,33,33,34,34,34,34,34,34,35,35,35,35,35,35,35,36,36,36,36,36,3
6,37,37,37,37,37,37,38,38,38,38,38,38,39,39,39,39,39,39,39,40,40,40,40,40,40,41,41,41,41,41,4
1,42,42,42,42,42,42,42,43,43,43,43,43,43,44,44,44,44,44,44,45,45,45,45,45,45,45,46,46,46,46,4
6,46,47,47,47,47,47,47,48,48,48,48,48,48,48,49,49,49,49,49,49,50,50,50,50,50,50,51,51,51,51,5
1,51,51,52,52,52,52,52,52,53,53,53,53,53,53,54,54,54,54,54,55,56,57,0,0,0,0}, 
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,6,7,6,6,6,6,6,6,6,6,
6,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10
,10,10,11,11,11,11,11,11,11,11,11,12,12,12,12,12,12,12,12,12,13,13,13,13,13,13,13,13,13,14,
14,14,14,14,14,14,14,14,14,15,15,15,15,15,15,15,15,15,16,16,16,16,16,16,16,16,16,17,17,17,
17,17,17,17,17,17,18,18,18,18,18,18,18,18,18,18,19,19,19,19,19,19,19,19,20,20,20,20,20,20,
20,20,20,20,21,21,21,21,21,21,21,21,21,22,22,22,22,21,21,21,21,21,21,22,22,22,22,22,22,22,
22,23,23,23,23,23,23,23,23,23,23,24,24,24,24,24,24,24,24,25,25,25,25,25,25,25,25,25,25,26,
26,26,26,26,26,26,26,26,27,27,27,27,27,27,27,27,27,27,28,28,28,28,28,28,28,28,29,29,29,29,
29,29,29,29,29,29,30,30,30,30,30,30,30,30,30,31,31,31,31,31,31,31,31,31,32,32,32,32,32,32,
32,32,32,33,33,33,33,33,33,33,33,33,33,34,34,34,34,34,34,34,34,34,35,35,35,35,35,35,35,35,
35,36,36,36,36,36,36,36,36,37,39,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,11,11,1
1,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,1
1,11,11,11,11,11,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,13,13,13,13,13,13,1
3,13,13,13,13,13,13,13,13,13,13,13,13,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,1
4,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,16,16,16,16,16,16,16,16,16,16,16,1
6,16,16,16,16,16,16,16,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,18,18,18,18,1
8,18,18,18,18,18,18,18,18,19,19,20},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,2}

};

alt_u16 CueEnd[31][355] = {

        {352,354,355,355,355,355,355,355,354,352,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0},

        {0,0,0,0,48,86,110,129,148,167,186,225,244,263,282,301,320,339,352,354,355,355,355,
355,354,354,353,351,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,19,29,47,57,67,86,96,105,115,124,134,143,153,162,172,181,191,206,219,229,23
8,248,257,267,276,286,295,305,314,324,334,343,351,352,352,353,353,353,353,353,353,350,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,21,27,35,46,53,59,65,73,84,91,97,103,109,116,122,128,135,141,147,154,160,1
66,173,179,185,192,199,211,217,223,230,236,242,248,255,261,267,274,280,286,293,299,305,31
2,318,324,331,337,343,348,349,350,351,351,351,351,350,350,347,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,17,22,26,31,38,45,50,55,59,64,69,76,83,88,92,97,102,107,111,116,121,125,13
0,135,140,144,149,154,158,163,168,172,177,182,187,191,196,205,210,215,220,224,229,234,238
,243,248,253,257,262,267,271,276,281,285,290,295,300,304,309,314,318,323,328,332,337,342,
345,347,347,348,348,347,347,347,347,343,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

```
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,18,22,26,30,33,40,45,48,52,56,60,63,67,71,77,82,86,89,93,97,101,104,108,1
12,116,119,123,127,131,134,138,142,145,149,153,157,160,164,168,172,175,179,183,187,190,19
4,201,205,209,213,217,220,224,228,231,235,239,243,246,250,254,258,261,265,269,272,276,280
,284,287,291,295,299,302,306,310,314,317,321,325,328,332,336,340,341,343,343,344,343,343,
343,343,341,339,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0},

        {0,0,0,0,0,0,16,19,22,25,28,31,35,41,44,47,50,53,56,59,62,65,69,72,78,81,84,87,90,93,96
,100,103,106,109,112,115,118,121,124,127,130,133,136,140,143,146,149,152,155,158,161,164,
167,170,173,176,180,183,186,189,192,196,201,204,207,210,214,217,220,223,226,229,232,235,2
38,241,244,247,250,254,257,260,263,266,269,272,275,278,281,284,287,290,294,297,300,303,30
6,309,312,315,318,321,324,327,330,334,337,336,338,339,338,338,338,337,337,335,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,17,20,22,25,28,30,33,35,41,43,46,49,51,54,56,59,62,64,67,69,74,77,80,83,
85,88,90,93,96,98,101,103,106,109,111,114,116,119,122,124,127,129,132,135,137,140,143,145,
148,150,153,156,158,161,163,166,169,171,174,176,179,182,184,187,189,194,197,200,203,205,2
08,210,213,216,218,221,223,226,229,231,234,236,239,242,244,247,250,252,255,257,260,263,26
5,268,270,273,276,278,281,283,286,289,291,294,296,299,302,304,307,309,312,315,317,320,322
,325,328,330,330,332,332,332,332,332,331,331,328,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,16,18,20,22,25,27,29,31,34,37,40,43,45,47,49,52,54,56,58,61,63,65,67,70,
74,77,79,81,83,86,88,90,92,95,97,99,101,104,106,108,110,113,115,117,119,122,124,126,128,13
1,133,135,137,139,142,144,146,148,151,153,155,157,160,162,164,166,169,171,173,175,178,180
,182,184,187,191,194,196,198,200,203,205,207,209,212,214,216,218,221,223,225,227,230,232,
234,236,239,241,243,245,248,250,252,254,256,259,261,263,265,268,270,272,274,277,279,281,2
83,286,288,290,292,295,297,299,301,304,306,308,310,313,315,317,319,322,324,324,325,326,32
6,325,325,324,324,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,16,18,20,22,24,26,28,30,32,34,38,40,42,44,46,48,50,52,54,56,58,60,62,6
4,66,68,71,74,76,78,80,82,83,85,87,89,91,93,95,97,99,101,103,105,107,109,111,113,115,117,11
9,121,123,125,127,129,131,133,135,136,138,140,142,144,146,148,150,152,154,156,158,160,162
,164,166,168,170,172,174,176,178,180,182,185,188,190,192,194,196,198,200,201,203,205,207,
209,211,213,215,217,219,221,223,225,227,229,231,233,235,237,239,241,243,245,247,249,251,2
53,254,256,258,260,262,264,266,268,270,272,274,276,278,280,282,284,286,288,290,292,294,29
6,298,300,302,304,305,307,309,311,313,315,316,317,317,318,318,317,317,316,314,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,17,19,20,22,24,25,27,29,31,32,34,38,40,41,43,45,47,48,50,52,53,55,57,
59,60,62,64,66,67,71,73,74,76,78,80,81,83,85,87,88,90,92,94,95,97,99,100,102,104,106,107,109
,111,113,114,116,118,120,121,123,125,126,128,130,132,133,135,137,139,140,142,144,146,147,
149,151,152,154,156,158,159,161,163,165,166,168,170,171,173,175,177,178,182,184,186,187,1
89,191,193,194,196,198,199,201,203,205,206,208,210,212,213,215,217,219,220,222,224,225,22
7,229,231,232,234,236,238,239,241,243,244,246,248,250,251,253,255,257,258,260,262,264,265
,267,269,270,272,274,276,277,279,281,283,284,286,288,290,291,293,295,296,298,300,302,303,
305,307,308,308,309,310,309,308,308,307,306,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,16,17,19,20,22,23,25,26,28,29,31,33,35,37,39,41,42,44,45,47,48,50,51,
53,54,56,57,59,61,62,64,65,69,70,72,73,75,76,78,79,81,82,84,85,87,89,90,92,93,95,96,98,99,101
,102,104,106,107,109,110,112,113,115,116,118,119,121,122,124,126,127,129,130,132,133,135,
136,138,139,141,142,144,146,147,149,150,152,153,155,156,158,159,161,162,164,166,167,169,1
```

```
70,172,173,177,178,180,181,183,184,186,187,189,190,192,194,195,197,198,200,201,203,204,20
6,207,209,211,212,214,215,217,218,220,221,223,224,226,227,229,231,232,234,235,237,238,240
,241,243,244,246,247,249,251,252,254,255,257,258,260,261,263,264,266,267,269,271,272,274,
275,277,278,280,281,283,284,286,288,289,291,292,294,295,297,298,298,300,299,300,299,299,2
98,297,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,15,16,17,19,20,21,23,24,26,27,28,30,31,32,36,37,38,40,41,42,44,45,47,
48,49,51,52,53,55,56,58,59,60,62,63,65,68,69,70,72,73,74,76,77,79,80,81,83,84,85,87,88,90,91,
92,94,95,96,98,99,101,102,103,105,106,107,109,110,112,113,114,116,117,118,120,121,123,124,
125,127,128,129,131,132,134,135,136,138,139,140,142,143,145,146,147,149,150,152,153,154,1
56,157,158,160,161,163,164,165,167,168,171,172,174,175,177,178,179,181,182,183,185,186,18
8,189,190,192,193,195,196,197,199,200,201,203,204,206,207,208,210,211,212,214,215,217,218
,219,221,222,223,225,226,228,229,230,232,233,234,236,237,239,240,241,243,244,245,247,248,
250,251,252,254,255,256,258,259,261,262,263,265,266,267,269,270,272,273,274,276,277,278,2
80,281,283,284,285,287,288,288,289,289,290,289,288,288,286,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,15,16,17,19,20,21,22,24,25,26,27,29,30,31,33,35,36,38,39,40,41,42,4
4,45,46,47,49,50,51,52,54,55,56,57,59,60,61,63,65,66,68,69,70,71,72,74,75,76,77,79,80,81,82,8
4,85,86,87,89,90,91,92,93,95,96,97,98,100,101,102,103,105,106,107,108,110,111,112,113,114,1
16,117,118,119,121,122,123,124,126,127,128,129,131,132,133,134,135,137,138,139,140,142,14
3,144,145,147,148,149,150,152,153,154,155,156,158,159,160,161,164,165,167,168,169,170,172
,173,174,175,177,178,179,180,182,183,184,185,186,188,189,190,191,193,194,195,196,198,199,
200,201,203,204,205,206,207,209,210,211,212,214,215,216,217,219,220,221,222,223,225,226,2
27,228,230,231,232,233,235,236,237,238,240,241,242,243,244,246,247,248,249,251,252,253,25
4,256,257,258,259,261,262,263,264,265,267,268,269,270,272,273,274,275,277,278,277,279,278
,278,278,277,276,274,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,15,16,17,19,20,21,22,23,24,25,26,27,29,30,31,33,34,36,37,38,39,40
,41,42,43,44,46,47,48,49,50,51,52,53,54,56,57,58,59,61,63,64,65,66,67,68,69,70,72,73,74,75,76,
77,78,79,80,82,83,84,85,86,87,88,89,90,91,93,94,95,96,97,98,99,100,101,103,104,105,106,107,1
08,109,110,111,113,114,115,116,117,118,119,120,121,123,124,125,126,127,128,129,130,131,13
```

```
3,134,135,136,137,138,139,140,141,143,144,145,146,147,148,149,150,151,153,154,155,156,159
,160,161,162,163,164,165,166,167,169,170,171,172,173,174,175,176,177,179,180,181,182,183,
184,185,186,187,189,190,191,192,193,194,195,196,197,198,200,201,202,203,204,205,206,207,2
08,210,211,212,213,214,215,216,217,218,220,221,222,223,224,225,226,227,228,230,231,232,23
3,234,235,236,237,238,240,241,242,243,244,245,246,247,248,250,251,252,253,254,255,256,257
,258,260,261,262,263,264,265,266,266,267,267,267,266,265,263,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,31,33,34,35,36,37
,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,59,60,61,62,63,64,65,66,67,68,69,
70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100
,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,
123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,1
45,146,147,148,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,16
9,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191
,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,
214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,2
36,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,253,254,254,25
4,253,252,250,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,14,14,15,16,17,18,19,20,21,22,23,23,24,25,26,27,28,29,31,32,33,34
,35,36,36,37,38,39,40,41,42,43,44,45,46,46,47,48,49,50,51,52,53,54,55,57,58,59,59,60,61,62,63,
64,65,66,67,68,68,69,70,71,72,73,74,75,76,77,77,78,79,80,81,82,83,84,85,86,86,87,88,89,90,91,
92,93,94,95,95,96,97,98,99,100,101,102,103,104,104,105,106,107,108,109,110,111,112,113,113
,114,115,116,117,118,119,120,121,122,122,123,124,125,126,127,128,129,130,131,131,132,133,
134,135,136,137,138,139,140,140,142,144,145,145,146,147,148,149,150,151,152,153,154,154,1
55,156,157,158,159,160,161,162,163,163,164,165,166,167,168,169,170,171,172,172,173,174,17
5,176,177,178,179,180,181,181,182,183,184,185,186,187,188,189,190,190,191,192,193,194,195
,196,197,198,199,199,200,201,202,203,204,205,206,207,208,208,209,210,211,212,213,214,215,
216,217,217,218,219,220,221,222,223,224,225,226,226,227,228,229,230,231,232,233,234,235,2
35,236,237,238,239,240,241,240,241,241,241,240,239,238,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,14,14,15,16,17,18,19,19,20,21,22,23,23,24,25,26,27,27,30,30,31,
32,33,34,34,35,36,37,38,38,39,40,41,42,42,43,44,45,46,47,47,48,49,50,51,51,52,54,55,56,57,58,
58,59,60,61,62,62,63,64,65,66,66,67,68,69,70,70,71,72,73,74,75,75,76,77,78,79,79,80,81,82,83,
83,84,85,86,87,87,88,89,90,91,92,92,93,94,95,96,96,97,98,99,100,100,101,102,103,104,104,105,
106,107,108,109,109,110,111,112,113,113,114,115,116,117,117,118,119,120,121,121,122,123,1
24,125,126,126,127,128,129,130,130,131,132,133,135,136,137,137,138,139,140,141,141,142,14
3,144,145,145,146,147,148,149,150,150,151,152,153,154,154,155,156,157,158,158,159,160,161
,162,162,163,164,165,166,167,167,168,169,170,171,171,172,173,174,175,175,176,177,178,179,
179,180,181,182,183,184,184,185,186,187,188,188,189,190,191,192,192,193,194,195,196,196,1
97,198,199,200,201,201,202,203,204,205,205,206,207,208,209,209,210,211,212,213,213,214,21
5,216,217,218,218,219,220,221,222,222,223,224,225,226,226,227,227,228,227,227,226,224,222
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

            {0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,15,16,17,17,18,19,19,20,21,22,22,23,24,25,25,26,28,29,
29,30,31,32,32,33,34,35,35,36,37,37,38,39,40,40,41,42,42,43,44,45,45,46,47,48,48,49,51,52,52,
53,54,55,55,56,57,58,58,59,60,60,61,62,63,63,64,65,66,66,67,68,68,69,70,71,71,72,73,74,74,75,
76,76,77,78,79,79,80,81,82,82,83,84,84,85,86,87,87,88,89,90,90,91,92,92,93,94,95,95,96,97,97,
98,99,100,100,101,102,103,103,104,105,105,106,107,108,108,109,110,111,111,112,113,113,114
,115,116,116,117,118,119,119,120,121,121,122,123,124,124,125,127,128,129,129,130,131,131,
132,133,134,134,135,136,137,137,138,139,139,140,141,142,142,143,144,145,145,146,147,147,1
48,149,150,150,151,152,152,153,154,155,155,156,157,158,158,159,160,160,161,162,163,163,16
4,165,166,166,167,168,168,169,170,171,171,172,173,174,174,175,176,176,177,178,179,179,180
,181,182,182,183,184,184,185,186,187,187,188,189,190,190,191,192,192,193,194,195,195,196,
197,198,198,199,200,200,201,202,203,203,204,205,206,206,207,208,208,209,210,211,211,212,2
13,212,213,213,213,212,210,209,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

            {0,0,0,0,0,0,0,0,0,0,0,0,0,13,14,14,15,16,16,17,17,18,19,19,20,21,21,22,23,23,24,25,26,2
7,28,28,29,30,30,31,32,32,33,34,34,35,36,36,37,37,38,39,39,40,41,41,42,43,43,44,45,45,46,48,4
8,49,50,50,51,52,52,53,54,54,55,56,56,57,58,58,59,59,60,61,61,62,63,63,64,65,65,66,67,67,68,6
9,69,70,70,71,72,72,73,74,74,75,76,76,77,78,78,79,80,80,81,82,82,83,83,84,85,85,86,87,87,88,8
9,89,90,91,91,92,93,93,94,95,95,96,96,97,98,98,99,100,100,101,102,102,103,104,104,105,106,1
06,107,108,108,109,109,110,111,111,112,113,113,114,115,115,116,117,118,119,120,120,121,12
2,122,123,124,124,125,126,126,127,128,128,129,129,130,131,131,132,133,133,134,135,135,136
,137,137,138,139,139,140,141,141,142,142,143,144,144,145,146,146,147,148,148,149,150,150,
151,152,152,153,154,154,155,155,156,157,157,158,159,159,160,161,161,162,163,163,164,165,1
65,166,166,167,168,168,169,170,170,171,172,172,173,174,174,175,176,176,177,178,178,179,17
9,180,181,181,182,183,183,184,185,185,186,187,187,188,189,189,190,191,191,192,192,193,194
,194,195,196,196,197,198,198,198,198,198,197,196,193,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,12,13,13,14,15,15,16,16,17,17,18,19,19,20,20,21,21,22,23,23,2
5,25,26,27,27,28,28,29,30,30,31,31,32,32,33,34,34,35,35,36,36,37,38,38,39,39,40,41,41,42,42,4
3,44,45,46,46,47,47,48,49,49,50,50,51,51,52,53,53,54,54,55,56,56,57,57,58,58,59,60,60,61,61,6
2,62,63,64,64,65,65,66,66,67,68,68,69,69,70,71,71,72,72,73,73,74,75,75,76,76,77,77,78,79,79,8
0,80,81,81,82,83,83,84,84,85,86,86,87,87,88,88,89,90,90,91,91,92,92,93,94,94,95,95,96,97,97,9
8,98,99,99,100,101,101,102,102,103,103,104,105,105,106,106,107,107,108,110,110,111,112,11
2,113,113,114,114,115,116,116,117,117,118,118,119,120,120,121,121,122,122,123,124,124,125
,125,126,127,127,128,128,129,129,130,131,131,132,132,133,133,134,135,135,136,136,137,138,
138,139,139,140,140,141,142,142,143,143,144,144,145,146,146,147,147,148,148,149,150,150,1
51,151,152,153,153,154,154,155,155,156,157,157,158,158,159,159,160,161,161,162,162,163,16
3,164,165,165,166,166,167,168,168,169,169,170,170,171,172,172,173,173,174,174,175,176,176
,177,177,178,178,179,180,180,181,181,182,183,182,183,183,183,181,179,178,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,12,12,13,13,14,14,15,15,16,16,17,17,18,18,19,19,20,20,21,21,2
2,23,24,24,25,25,26,26,27,27,28,28,29,30,30,31,31,32,32,33,33,34,34,35,35,36,36,37,37,38,38,3
9,39,40,40,42,42,43,43,44,44,45,45,46,46,47,47,48,48,49,49,50,51,51,52,52,53,53,54,54,55,55,5
6,56,57,57,58,58,59,59,60,60,61,61,62,62,63,63,64,64,65,65,66,66,67,67,68,68,69,69,70,70,71,7
1,72,72,73,73,74,74,75,75,76,76,77,78,78,79,79,80,80,81,81,82,82,83,83,84,84,85,85,86,86,87,8
7,88,88,89,89,90,90,91,91,92,92,93,93,94,94,95,95,96,96,97,97,98,98,99,101,101,102,102,103,1
03,104,104,105,105,106,106,107,107,108,108,109,109,110,110,111,111,112,112,113,113,114,11
4,115,115,116,116,117,117,118,118,119,119,120,120,121,121,122,122,123,123,124,124,125,126
,126,127,127,128,128,129,129,130,130,131,131,132,132,133,133,134,134,135,135,136,136,137,
137,138,138,139,139,140,140,141,141,142,142,143,143,144,144,145,145,146,146,147,147,148,1
48,149,149,150,150,151,151,152,153,153,154,154,155,155,156,156,157,157,158,158,159,159,16
0,160,161,161,162,162,163,163,164,164,165,165,166,166,167,166,167,166,167,165,163,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,11,11,12,12,13,13,14,14,15,15,15,16,16,17,17,18,18,19,19,19,2
0,20,22,22,23,23,24,24,25,25,25,26,26,27,27,28,28,29,29,29,30,30,31,31,32,32,33,33,33,34,34,3
5,35,36,36,37,38,39,39,39,40,40,41,41,42,42,43,43,43,44,44,45,45,46,46,47,47,47,48,48,49,49,5
0,50,51,51,51,52,52,53,53,54,54,55,55,55,56,56,57,57,58,58,59,59,59,60,60,61,61,62,62,63,63,6
3,64,64,65,65,66,66,67,67,67,68,68,69,69,70,70,71,71,71,72,72,73,73,74,74,75,75,75,76,76,77,7
7,78,78,79,79,80,80,80,81,81,82,82,83,83,84,84,84,85,85,86,86,87,87,88,88,88,89,89,90,91,92,9
2,93,93,94,94,94,95,95,96,96,97,97,98,98,98,99,99,100,100,101,101,102,102,102,103,103,104,1
04,105,105,106,106,106,107,107,108,108,109,109,110,110,110,111,111,112,112,113,113,114,11
4,114,115,115,116,116,117,117,118,118,118,119,119,120,120,121,121,122,122,122,123,123,124
,124,125,125,126,126,126,127,127,128,128,129,129,130,130,130,131,131,132,132,133,133,134,
134,134,135,135,136,136,137,137,138,138,138,139,139,140,140,141,141,142,142,142,143,143,1
44,144,145,145,146,146,146,147,147,148,148,149,149,150,150,150,150,150,150,150,148,146,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

            {0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,11,12,12,12,13,13,13,14,14,15,15,15,16,16,17,17,17,18,18,
18,19,20,21,21,21,22,22,23,23,23,24,24,25,25,25,26,26,26,27,27,28,28,28,29,29,30,30,30,31,31,
31,32,32,33,33,33,35,35,36,36,36,37,37,37,38,38,39,39,39,40,40,41,41,41,42,42,42,43,43,44,44,
44,45,45,46,46,46,47,47,47,48,48,49,49,49,50,50,51,51,51,52,52,52,53,53,54,54,54,55,55,56,56,
56,57,57,57,58,58,59,59,59,60,60,61,61,61,62,62,62,63,63,64,64,64,65,65,66,66,66,67,67,67,68,
68,69,69,69,70,70,71,71,71,72,72,72,73,73,74,74,74,75,75,76,76,76,77,77,77,78,78,79,79,79,80,
80,82,82,82,83,83,83,84,84,85,85,85,86,86,87,87,87,88,88,88,89,89,90,90,90,91,91,92,92,92,93,
93,93,94,94,95,95,95,96,96,97,97,97,98,98,98,99,99,100,100,100,101,101,102,102,102,103,103,
103,104,104,105,105,105,106,106,107,107,107,108,108,108,109,109,110,110,110,111,111,112,1
12,112,113,113,113,114,114,115,115,115,116,116,116,117,117,118,118,118,119,119,120,120,12
0,121,121,121,122,122,123,123,123,124,124,125,125,125,126,126,126,127,127,128,128,128,129
,129,130,130,130,131,131,131,132,132,133,133,133,134,133,133,134,133,131,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0},

            {0,0,0,0,0,0,0,0,0,0,0,0,0,0,10,11,11,11,11,12,12,12,13,13,13,14,14,14,15,15,15,16,16,16,
17,17,17,19,19,19,20,20,20,21,21,21,22,22,22,23,23,23,24,24,24,25,25,25,26,26,26,27,27,27,27,
28,28,28,29,29,29,30,31,31,32,32,32,33,33,33,34,34,34,35,35,35,36,36,36,37,37,37,38,38,38,39,
39,39,40,40,40,41,41,41,42,42,42,43,43,43,43,44,44,44,45,45,45,46,46,46,47,47,47,48,48,48,49,
49,49,50,50,50,51,51,51,52,52,52,53,53,53,54,54,54,55,55,55,56,56,56,56,57,57,57,58,58,58,59,
59,59,60,60,60,61,61,61,62,62,62,63,63,63,64,64,64,65,65,65,66,66,66,67,67,67,68,68,68,69,69,
69,69,70,70,70,72,72,72,73,73,73,74,74,74,75,75,75,76,76,76,77,77,77,78,78,78,79,79,79,80,80,
80,81,81,81,82,82,82,83,83,83,84,84,84,85,85,85,85,86,86,86,87,87,87,88,88,88,89,89,89,90,90,
90,91,91,91,92,92,92,93,93,93,94,94,94,95,95,95,96,96,96,97,97,97,97,98,98,98,99,99,99,100,10
0,100,101,101,101,102,102,102,103,103,103,104,104,104,105,105,105,106,106,106,107,107,107
,108,108,108,109,109,109,110,110,110,110,111,111,111,112,112,112,113,113,113,114,114,114,
115,115,115,116,116,116,117,117,116,117,115,114,111,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

            {0,0,0,0,0,0,0,0,0,0,0,0,0,0,10,10,10,10,11,11,11,11,12,12,12,13,13,13,13,14,14,14,14,15,
15,15,16,17,17,17,18,18,18,18,19,19,19,19,20,20,20,21,21,21,21,22,22,22,22,23,23,23,24,24,24,
24,25,25,25,25,26,26,26,26,28,28,28,29,29,29,29,30,30,30,30,31,31,31,32,32,32,32,33,33,33,33,
34,34,34,34,35,35,35,36,36,36,36,37,37,37,37,38,38,38,38,39,39,39,40,40,40,40,41,41,41,41,42,
42,42,43,43,43,43,44,44,44,44,45,45,45,45,46,46,46,47,47,47,47,48,48,48,48,49,49,49,49,50,50,
50,51,51,51,51,52,52,52,52,53,53,53,53,54,54,54,55,55,55,55,56,56,56,56,57,57,57,58,58,58,58,
59,59,59,59,60,60,60,60,62,62,62,63,63,63,63,64,64,64,64,65,65,65,66,66,66,66,67,67,67,67,68,
68,68,68,69,69,69,70,70,70,70,71,71,71,71,72,72,72,72,73,73,73,74,74,74,74,75,75,75,75,76,76,
76,76,77,77,77,78,78,78,78,79,79,79,79,80,80,80,81,81,81,81,82,82,82,82,83,83,83,83,84,84,84,
85,85,85,85,86,86,86,86,87,87,87,87,88,88,88,89,89,89,89,90,90,90,90,91,91,91,91,92,92,92,93,
93,93,93,94,94,94,94,95,95,95,96,96,96,96,97,97,97,97,98,98,98,98,99,99,99,100,99,99,98,98,94
,0,0,0,0,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,9,9,9,10,10,10,10,11,11,11,11,11,12,12,12,12,12,13,13,13,13,
14,14,14,15,15,16,16,16,16,16,17,17,17,17,18,18,18,18,18,19,19,19,19,19,20,20,20,20,21,21,21,
21,21,22,22,22,22,22,23,23,24,24,24,25,25,25,25,26,26,26,26,26,27,27,27,27,27,28,28,28,28,29,
29,29,29,29,30,30,30,30,30,31,31,31,31,32,32,32,32,32,33,33,33,33,33,34,34,34,34,34,35,35,35,
35,36,36,36,36,36,37,37,37,37,37,38,38,38,38,39,39,39,39,39,40,40,40,40,41,41,41,41,42,42,
42,42,42,43,43,43,43,43,44,44,44,44,44,45,45,45,45,46,46,46,46,46,47,47,47,47,47,48,48,48,48,
49,49,49,49,49,50,50,50,50,50,52,52,52,52,53,53,53,53,53,54,54,54,54,54,55,55,55,55,55,56,56,
56,56,57,57,57,57,57,58,58,58,58,58,59,59,59,59,60,60,60,60,60,61,61,61,61,61,62,62,62,62,63,
63,63,63,63,64,64,64,64,64,65,65,65,65,65,66,66,66,66,67,67,67,67,67,68,68,68,68,68,69,69,69,
69,70,70,70,70,70,71,71,71,71,71,72,72,72,72,72,73,73,73,73,74,74,74,74,74,75,75,75,75,75,76,
76,76,76,77,77,77,77,77,78,78,78,78,78,79,79,79,79,80,80,80,80,80,81,81,81,81,81,82,82,81,81,
81,81,77,0,0,0,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,9,9,9,9,9,9,10,10,10,10,10,10,11,11,11,11,11,11,12,12,12,12
,12,13,14,14,14,14,14,14,14,15,15,15,15,15,15,16,16,16,16,16,16,17,17,17,17,17,17,17,18,18,18,
18,18,18,19,19,19,19,19,20,21,21,21,21,21,21,21,22,22,22,22,22,22,23,23,23,23,23,23,24,24,24,
24,24,24,24,25,25,25,25,25,25,26,26,26,26,26,26,27,27,27,27,27,27,28,28,28,28,28,28,28,29,29,
29,29,29,29,30,30,30,30,30,30,31,31,31,31,31,31,31,32,32,32,32,32,32,33,33,33,33,33,33,34,34,
34,34,34,34,34,35,35,35,35,35,35,36,36,36,36,36,36,37,37,37,37,37,37,37,38,38,38,38,38,38,39,
39,39,39,39,39,40,40,40,40,40,41,42,42,42,42,42,42,42,43,43,43,43,43,43,44,44,44,44,44,44,45,
45,45,45,45,45,45,46,46,46,46,46,46,47,47,47,47,47,47,48,48,48,48,48,48,48,49,49,49,49,49,49,
50,50,50,50,50,50,51,51,51,51,51,51,51,52,52,52,52,52,52,53,53,53,53,53,53,54,54,54,54,54,54,
54,55,55,55,55,55,55,56,56,56,56,56,56,57,57,57,57,57,57,57,58,58,58,58,58,58,59,59,59,59,59,
59,60,60,60,60,60,60,61,61,61,61,61,61,61,62,62,62,62,62,62,63,63,63,63,63,63,64,64,64,64,64,
63,63,63,61,0,0,0,0},

        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,8,8,8,9,9,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10,10,11,12,
12,12,12,12,12,12,12,12,13,13,13,13,13,13,13,13,14,14,14,14,14,14,14,14,14,14,15,15,15,15,
15,15,15,15,15,17,17,17,17,17,17,17,17,17,17,18,18,18,18,18,18,18,18,18,19,19,19,19,19,19,19,
19,19,19,20,20,20,20,20,20,20,20,20,21,21,21,21,21,21,21,21,21,22,22,22,22,22,22,22,22,22,
23,23,23,23,23,23,23,23,23,23,24,24,24,24,24,24,24,24,24,25,25,25,25,25,25,25,25,25,25,26,26,
26,26,26,26,26,26,26,27,27,27,27,27,27,27,27,27,28,28,28,28,28,28,28,28,29,29,29,29,29,
29,29,29,29,29,30,30,30,31,31,31,31,31,31,32,32,32,32,32,32,32,32,32,33,33,33,33,33,33,33,
33,33,34,34,34,34,34,34,34,34,34,34,35,35,35,35,35,35,35,35,35,36,36,36,36,36,36,36,36,36,
37,37,37,37,37,37,37,37,37,38,38,38,38,38,38,38,38,38,38,39,39,39,39,39,39,39,39,39,40,40,40,
40,40,40,40,40,40,40,41,41,41,41,41,41,41,41,41,42,42,42,42,42,42,42,42,42,42,43,43,43,43,43,
43,43,43,43,44,44,44,44,44,44,44,44,44,44,45,45,45,45,45,45,45,45,45,46,46,46,46,46,46,46,46,
45,45,45,0,0},

		{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,10,10,10,10,10,
10,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,12,12,12,
12,12,13,13,13,13,13,13,13,13,13,13,13,13,13,13,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,
14,14,14,14,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,16,16,16,16,16,16,16,16,
16,16,16,16,16,16,16,16,16,16,16,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,18,
18,18,18,18,18,18,18,18,18,18,18,18,18,18,18,18,18,18,19,19,19,19,19,19,19,19,19,19,19,19,19,
19,19,19,19,19,19,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,22,22,22,22,22,22,
22,22,22,22,22,22,22,22,22,22,22,22,22,23,23,23,23,23,23,23,23,23,23,23,23,23,23,23,23,23,23,
23,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,24,25,25,25,25,25,25,25,25,25,25,
25,25,25,25,25,25,25,25,25,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,26,27,27,27,
27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,27,28,28,28,28,28,28,28,28,28,28,28,28,27,27,26,
24},

		{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,
8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
9,9,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,9,9,8}

};

//angle >=120 : clean pool cue
void print_poolcue(int x, int y, int angle)
{
	//(x, y) position of the top of the cue; angle 0-119, clockwise
	int line;
	int line_addr;
	int datain;
	int new_angle;
```

```
    if(angle >= 0 && angle < 30)
    {
        line_addr = 0;
        for(line = y - 5; line < 0; line++)
        {
            line_addr++;
        }

        for(line = 0; line < y - 5; line++)
        {
            IOWR_CUERAM(line, 0);
            //printf("Write %d to %d\n", datain, line);
        }

        while((line < y + 350) & (line < 480))
        {
            if(CueEnd[angle][line_addr] != 0)
            {
                if (CueBegin[angle][line_addr] + x < 5)
                {
                    datain = 0;
                    if (CueEnd[angle][line_addr] + x >= 5)
                    {
                        datain = (datain << 16) + CueEnd[angle][line_addr]
+ x - 5;
                    }

                }
                else if (CueEnd[angle][line_addr] + x > 645)
                {

                    if (CueBegin[angle][line_addr] + x > 645)
                    {
                        datain = 640;
                    }
                    else
                    {
                        datain = CueBegin[angle][line_addr] + x - 5;
                    }
                    datain = (datain << 16) + 640;
                }
                else
                {
                    datain = CueBegin[angle][line_addr] + x - 5;
                    datain = (datain << 16) + CueEnd[angle][line_addr] + x - 5;
                }
            }
            else
            {
                datain = 0;
            }

            IOWR_CUERAM(line, datain);
            line ++;
            line_addr++;

            //printf("Write %d to %d\n", datain, line);

        }

        while(line < 480)
        {
            IOWR_CUERAM(line, 0);
            //printf("Write %d to %d\n", datain, line);
            line ++;

        }

    }
    else if(angle >= 30 && angle < 60)
    {
        new_angle = 60 - angle;
        line_addr = 0;
        for(line = y - 5; line < 0; line++)
        {
            line_addr++;
        }
```

```
        for(line = 0; line < y - 5; line++)
        {
            IOWR_CUERAM(line, 0);
            //printf("Write %d to %d\n", datain, line);
        }

        while((line < y + 350) & (line < 480))
        {
            if(CueEnd[new_angle][line_addr] != 0)
            {
                if (CueEnd[new_angle][line_addr] - x > 6)
                {
                    datain = 0;
                    if (CueBegin[new_angle][line_addr] - x <= 6)
                    {
                        datain = (datain << 16) + x -
CueBegin[new_angle][line_addr] + 6;
                    }

                }
                else if (x - CueBegin[new_angle][line_addr] > 635)
                {
                    if (x - CueEnd[new_angle][line_addr] > 635)
                    {
                        datain = 640;
                    }
                    else
                    {
                        datain = x - CueEnd[new_angle][line_addr] + 6;
                    }

                    datain = (datain << 16) + 640;
                }
                else
                {
                    datain = x - CueEnd[new_angle][line_addr] + 6;
                    datain = (datain << 16) + x -
CueBegin[new_angle][line_addr] + 6;
```

```
            }
        }
        else
        {
            datain = 0;
        }

        IOWR_CUERAM(line, datain);
        line ++;
        line_addr++;

        //printf("Write %d to %d\n", datain, line);

    }

    while(line < 480)
    {
        IOWR_CUERAM(line, 0);
        //printf("Write %d to %d\n", datain, line);
        line ++;

    }

}
else if(angle >= 60 && angle < 90)
{
    new_angle = angle - 60;
    line_addr = 0;
    for(line = y + 5; line >= 480; line--)
    {
        line_addr++;
    }

    for(line = 479; line > y + 5; line--)
    {
        IOWR_CUERAM(line, 0);
        //printf("Write %d to %d\n", datain, line);
    }
```

```
        while((line > y - 350) & (line > -1))
        {
            if(CueEnd[new_angle][line_addr] != 0)
            {
                if (CueEnd[new_angle][line_addr] - x > 6)
                {
                    datain = 0;
                    if (CueBegin[new_angle][line_addr] - x <= 6)
                    {
                        datain = (datain << 16) + x -
CueBegin[new_angle][line_addr] + 6;
                    }

                }
                else if (x - CueBegin[new_angle][line_addr] > 635)
                {
                    if (x - CueEnd[new_angle][line_addr] > 635)
                    {
                        datain = 640;
                    }
                    else
                    {
                        datain = x - CueEnd[new_angle][line_addr] + 6;
                    }

                    datain = (datain << 16) + 640;
                }
                else
                {
                    datain = x - CueEnd[new_angle][line_addr] + 6;
                    datain = (datain << 16) + x -
CueBegin[new_angle][line_addr] + 6;
                }
            }
            else
            {
                datain = 0;
            }
```

```
                IOWR_CUERAM(line, datain);
                line --;
                line_addr++;

                //printf("Write %d to %d\n", datain, line);

            }

            while(line >= 0)
            {
                IOWR_CUERAM(line, 0);
                //printf("Write %d to %d\n", datain, line);
                line --;

            }

        }
        else if(angle >= 90 && angle < 120)
        {
            line_addr = 0;
            new_angle = 120 - angle;
            for(line = y + 5; line >= 480; line--)
            {
                line_addr++;
            }

            for(line = 479; line > y + 5; line--)
            {
                IOWR_CUERAM(line, 0);
                //printf("Write %d to %d\n", datain, line);
            }

            while((line > y - 350) & (line > -1))
            {
                if(CueEnd[new_angle][line_addr] != 0)
                {
                    if (CueBegin[new_angle][line_addr] + x < 5)
                    {
                        datain = 0;
                        if (CueEnd[new_angle][line_addr] + x >= 5)
                        {
                            datain = (datain << 16) + 
    CueEnd[new_angle][line_addr] + x - 5;
                        }

                    }
                    else if (CueEnd[new_angle][line_addr] + x > 645)
                    {

                        if (CueBegin[new_angle][line_addr] + x > 645)
                        {
                            datain = 640;
                        }
                        else
                        {
                            datain = CueBegin[new_angle][line_addr] + x - 5;
                        }
                        datain = (datain << 16) + 640;
                    }
                    else
                    {
                        datain = CueBegin[new_angle][line_addr] + x - 5;
                        datain = (datain << 16) + CueEnd[new_angle][line_addr] + 
    x - 5;
                    }
                }
                else
                {
                    datain = 0;
                }

                IOWR_CUERAM(line, datain);
                line --;
                line_addr++;

                //printf("Write %d to %d\n", datain, line);

            }
            while(line >= 0)
            {
                IOWR_CUERAM(line, 0);
                //printf("Write %d to %d\n", datain, line);
                line --;

            }

        }
        else
        {
            for(line = 0; line < 480; line ++)
            {
                IOWR_CUERAM(line, 0);
            }
        }



    }



    //------------------------------------------------
    #endif /* POOLCUE_H_ */
```

-- poolball.h

```
/*
 * poolball.h
 *
 *  Created on: Apr 9, 2013
 *      Author: Jiawan Zhang
 */

#ifndef POOLBALL_H_
#define POOLBALL_H_
```

```
//****************************************
#include <io.h>
#include <system.h>
#include <stdio.h>

#define IOWR_VGA_BALL_POSITION_X(ballnum, data) \
  IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, ((ballnum)*4 + 0)*2, data)
#define IOWR_VGA_BALL_POSITION_Y(ballnum, data) \
  IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, ((ballnum)*4 + 1)*2, data)
#define IOWR_VGA_BALL_BIAS_X(ballnum, bias_x) \
  IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, ((ballnum)*4 + 2)*2, bias_x)
#define IOWR_VGA_BALL_BIAS_Y(ballnum, bias_y) \
  IOWR_16DIRECT(DE2_VGA_RASTER_0_BASE, ((ballnum)*4 + 3)*2, bias_y)

//int BALL_BIAS_X[16];
//int BALL_BIAS_Y[16];

void placeball(int ballnum, int pos_x, int pos_y, int bias_x, int bias_y, int* BiasX, int* BiasY)
{
	//BALL_BIAS_X[ballnum] = bias_x;
	//BALL_BIAS_Y[ballnum] = bias_y;

	IOWR_VGA_BALL_POSITION_X(ballnum, pos_x);
	IOWR_VGA_BALL_POSITION_Y(ballnum, pos_y);
	IOWR_VGA_BALL_BIAS_X(ballnum, bias_x);
	IOWR_VGA_BALL_BIAS_Y(ballnum, bias_y);

	*BiasX = bias_x;
	*BiasY = bias_y;

}


void moveball(int ballnum, int *pos_x, int *pos_y, int flag_x, int flag_y, int* BiasX, int* BiasY)
{
//flag_x = 0, no moving in x direction; = 1 move right 1 pixel in x direction; = -1 move left 1 pixel in x direction
//flag_y = 0, no moving in y direction; = 1 move down 1 pixel in y direction; = -1 move up 1 pixel in y direction
```

```
//ori_x, x position before moving; ori_y, y position before moving;
        int bias_x;
        int bias_y;
        int new_pos_x;
        int new_pos_y;

        bias_x = *BiasX;
        bias_y = *BiasY;
        new_pos_x = *pos_x + flag_x;
        new_pos_y = *pos_y + flag_y;

        bias_x = bias_x - flag_x;
        bias_y = bias_y - flag_y;


        if(bias_x < 0)
        {
                bias_x = 13;
        }
        else if(bias_x > 13)
        {
                bias_x = 0;
        }


        if(bias_y < 0)
        {
                bias_y = 13;
        }
        else if(bias_y > 13)
        {
                bias_y = 0;
        }


        IOWR_VGA_BALL_POSITION_X(ballnum, new_pos_x);
        IOWR_VGA_BALL_POSITION_Y(ballnum, new_pos_y);
        IOWR_VGA_BALL_BIAS_X(ballnum, bias_x);
        IOWR_VGA_BALL_BIAS_Y(ballnum, bias_y);
        *BiasX = bias_x;
        *BiasY = bias_y;
        *pos_x = new_pos_x;
        *pos_y = new_pos_y;


}

//*****************************************
#endif /* POOLBALL_H_ */
```