Campus Fighter

CSEE 4840 Embedded System

Project Report

Haosen Wang, hw2363

Lei Wang, lw2464

Pan Deng, pd2389

Hongtao Li, hl2660

Pengyi Zhang, pnz2102

May 2012

● Overview

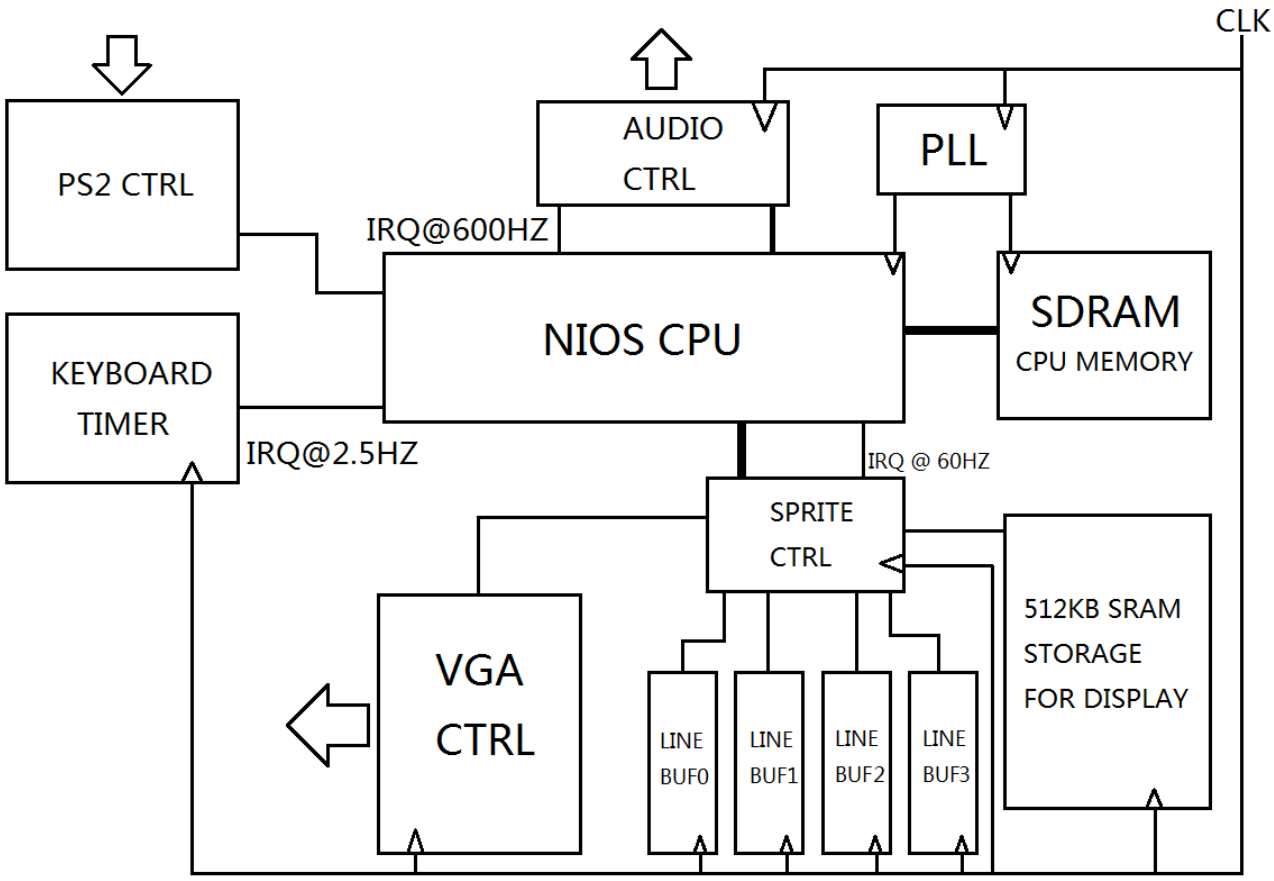
The campus fighter game is a fighting game based on DE2 board. In the game two players control their characters through keyboard and fight for the win. The game characters can make many different actions to move, attack or defense. Special effects are added to the graphic display for a better gaming experience.

This game is inspired by the famous fighting game Street Fighter. The game was first released in 1987 and grew dramatically to one of the most interesting and popular fighting game all over the world. The Street Fighter is easy for players to control, contains tens of characters with personalities. In the old gaming machine age, it demonstrated a really good display quality with many exciting special effects. This project aims to achieve as good display effects as that popular old game.

Generally, the project is composed by the hardware part written in VHDL and the software part written in C. DE2 FPGA board with Nios CPU entity is used as the project platform. Apart from the above two part, the game also contains a delicate storage structure and a big set of game data.

General Architecture

The most special part of a fighting game is the large number of game images included. While other old game can take full advantages of reusable tiles or sprites, a fighting game must deal with as many possible fighting situations as possible. And that means more different images. Other issues relating to a fighting game is the performance of input equipment, or the keyboard. With these considerations, our project architecture is shown below.



Four major hardware parts of the design are: sprite and display controller, data storage memory, keyboard controller and audio controller. These parts will be discussed in detail below.

● Project Data and Storage

Image Preparation

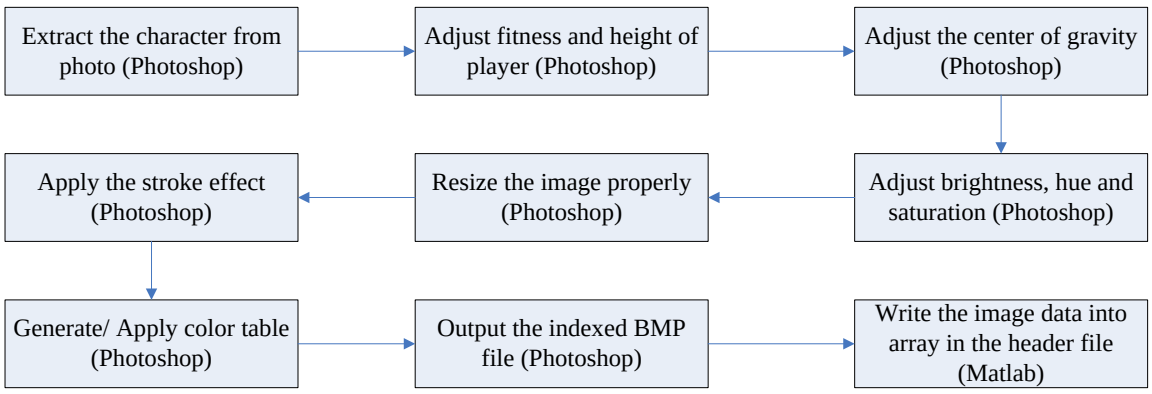
In our project, the images for the players' actions are all photographed by ourselves. These images are processed with the assistance of Photoshop to extract the player from the surroundings. Then, the players in different images are normalized in height and fitness, with the center of gravity adjusted.

As for our project, there are a large number of images to be stored. (The detailed image list is shown in the table below.) However, the SRAM is only of size 512 KB.To efficiently utilize the memory, we choose to use the color indexed images. The color indexed images could achieve satisfactory image quality with significant decrease in the memory usage. In our project, the color table is chosen to be composed of 16 colors, which means each pixel is represented by 4 bits. For the different sets of images, we utilize the different color tables to overcome the large color range difference among images.

Image Category		# of Images	Image size (KB)
1. Game Character			
a. Walk		6	50.4
b. Punch	Heavy	16	57
	Light		
	Jump		
	Low	9	50.7
c. Kick	Heavy		
	Light		
	Low light		
	Low heavy		
	Jump		
d. Defense	High	2	9.4
	Low		
e. Win		1	3.7

f. Lose		1	3.6
g. Standby		1	5
h. Wound		1	6
k. Knock down		1	3
2. Special skills	a. Dash	7	47.02
	b. Gun		
	c. Laser		
3. Background		1	140
4. Special effects	a. FireGun	2	6.43
	b. ball	1	0.76
	c. Laser	1	11.3
	d. PG	2	38.8
	e. Shadow	1	1.92
	f. UP	1	2.37
5. Scoring System	a. Health points	1	7
	b. Health point boundary	1	7
	c. Digits	10	15.5
	d. Player number	2	10
	e. Winner	1	7.5
	f. INF	1	1.2
For all images		70	485.6

The detailed flow for processing the image is shown as the figure below.

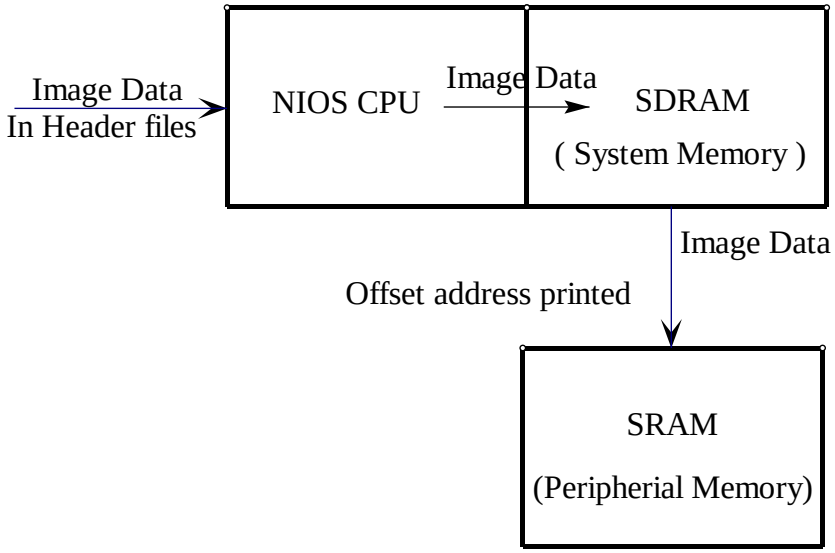


Use of SDRAM

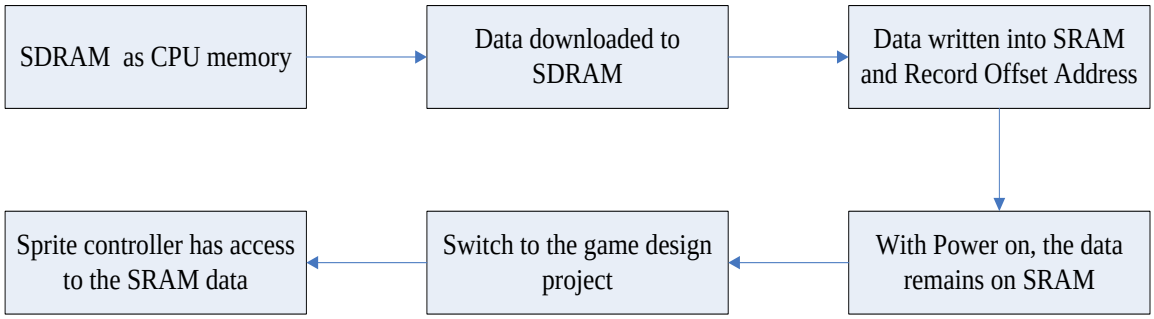
An additional Quartus project has been constructed to write the data to the SRAM indirectly through the SDRAM. In the SOPC builder for this project, the SDRAM is connected as the system memory. To construct the successful hardware, the phase

locked loop is required to generate the clock for the SDRAM, leading the NIOS system clock by 3ns. The image data is recorded as arrays in the C header files. Then, the image data information is downloaded to the system memory (the SDRAM) in a specific order. The offset address for each image is recorded for future access.

The structure for the hardware in this project is shown in the figure below.



The detailed process for writing to SRAM and reading from SRAM is shown as below.



● SPRITE-BASED VGA DISPLAY

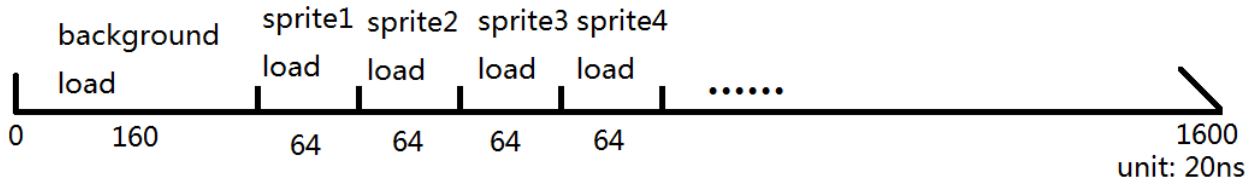
Sprite load structure

The fighting game is sprite based and must display different kinds of graphic elements in multiple sprites. The detailed arrangement is shown below.

SPRITE#	CONTENT	SPRITE#	CONTENT
0	P1 shadow	1	P2 shadow
2	P1 phantom	3	P2 phantom
4	P1 body	5	P2 body
6	P1 hit effect	7	P2 hit effect
8	P1 gun effect	9	P2 gun effect
10	P1 special effect	11	P2 special effect
12	P1 hp	13	P2 hp
14	P1 hp boundary	15	P2 hp boundary
16	Time tens	17	Time ones
18	Winner word	19	Winner

Sprite with bigger number index will overlap on the sprite with smaller number index. Each sprite image may contain some transparent points. To make a correct display, the controller should seek down to the lower sprite if a transparent point is met.

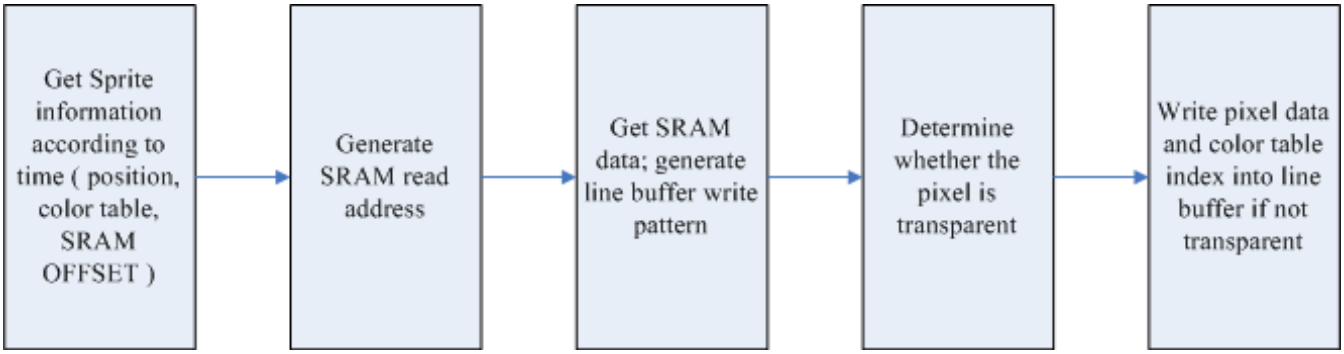
To implement high quality display of such large number of sprites, line buffer is used for a pre-load from SRAM. The general strategy is use the time of one VGA row scan period to make the line buffer ready and output line display data in the next row scan period. The VGA row scan contains 800 cycles in 25Mhz clock or 1600 cycles in 50Mhz clock. Make reasonable division over this time for each sprite and background can make the line buffer load action functional. As shown below.



The SRAM we used to store display data has a data port width of 16 bit. And as

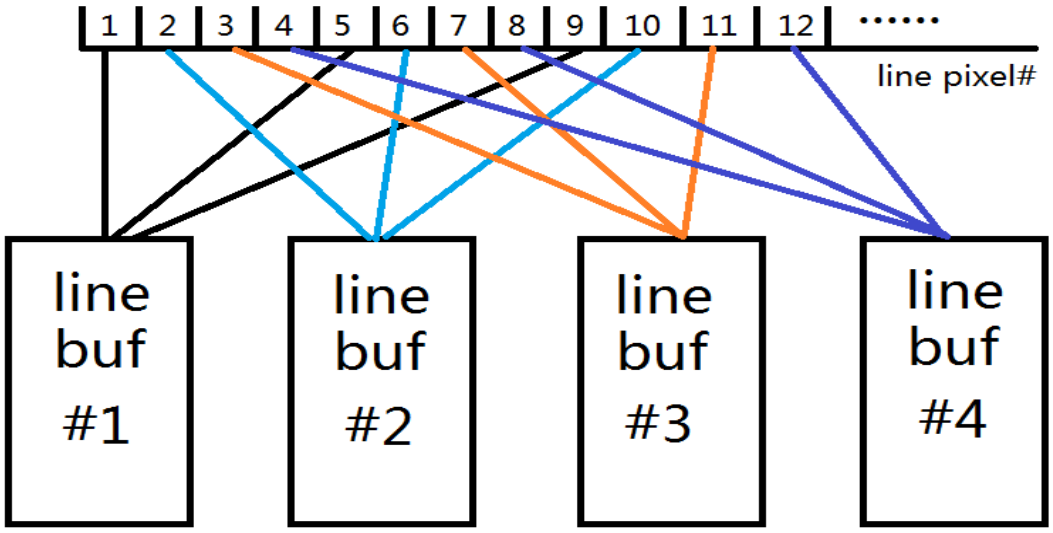
mentioned before, each pixel in our encoded indexed image is a 4-bit data. Thus if we want to achieve max read speed from SRAM, we must store 4 pixel data in a row and read out 4 continuing pixels within one SRAM access. Suppose image data is aligned line by line in the SRAM, in a 64 cycle period, we can access 256 pixels in a line of a single image. According to the sprite position information got from CPU, the data can be written into the correct position in the line buffer. Here we can also judge whether a pixel is transparent. If so, the line buffer point should not be changed. By repeating this process for all sprites, we can update the entire line buffer within one VGA row scan period.

The entire load process works in a pipeline style as shown below. This structure minimizes the element costs as well as the time costs.



Line buffer

One major issue in this part is that we cannot write 4 data points into line buffer within one cycle even if we can read the 4 data at the same time from the SRAM. If we write code as to allow 4 inputs within one cycle, the line buffer will not be generalized as SRAM cells, but as complex FF's. One the other hand, we cannot simply form a line buffer of width 16 because the start location of one certain sprite line is arbitrary rather than an integer multiply of 4. To solve this, we present a special line buffer structure as shown below.



This structure will work because the 4 pixels from SRAM in each cycle are always continuous. Thus each cycle each line buffer above can have exact one access. The space is the same, time is minimum and only a small control circuitry is added. There are actually two sets of the above line buffer structure so that pipeline style switch action can be done. **Also, each buffer cell actually contains 7 bits. The additional 3 bits are used to store the color table for a correct pixel color display.**

VGA Output

Once the line buffer is successfully loaded, VGA output controller can read each line pixel and output it on the screen. One thing special here is related to our indexed image data type. The pixel data got from the SRAM is only 4 bit index number. To make it a real colored point, the system should use the color table attach to the line buffers and read out real color values from the stored color tables. After that the display is straight forward.

● Keyboard

The PS2 keyboard will act as the only controller of the campus fighting video game. And the keyboard will and only will interface with the main game controller through several buffers which contains all the information of the user input.

There are 3 buffers for each player, respectively “key Buffer”, “time Buffer” and “hold Buffer”. “Key Buffer” is used to store the sequence of the user input, the 4 direction keys and 4 motion keys will be coded from 1 to 8 in the buffer. And the release code of each key is assigned to be 10 plus the key code. And the size of the hold buffer is 10. “time Buffer” which has a same length as the “key Buffer” is used to store the trigger time of each key event corresponding to the key stored in key buffer. These two buffers will refresh every time the controller finishes the motion judgment from their values. And the hold buffer is used to store the hold condition of each key. Since when a key is hold, the keyboard will continuously send the signal of this key’s code, the hold buffer is realized simply by counting the number of received signals of each key, and a count larger than 2 means a hold. When the release code of a key is received, its hold buffer will be cleared.

Since for such an action game, the reaction time is an important issue, it is vital for the buffered input data to be updated in time. And the buffer refresh is done through a PS2 keyboard interrupt, which is triggered by a key press or release event. And the timing of the key is done by introducing a time interrupt which increases every time unit equals the combo time interval. The reason why an interrupt timer is used is because that we want to try to get a hardware clock which runs in back ground and

not influenced by the software code.

The hardware setup for this keyboard controller is just to put the PS2 controller into SOPC and connect the eternal signals CLK and DATA to corresponding top-level board pins. The control region of player1 is the left half part of the keyboard whose keys use binary make code, and player2's keys uses long codes.

● Audio Design and Implement

Audio generation part

First of all, in our project, 2 kinds of the sound have been used: First part is the background music. It will play accompany the game from the beginning to the end; we choose the game sound as the background music. Before we use the sound, we need to do the preprocess. Using the matlab to quantity the sound (here we use 16 bits to represent 1 point and the sample frequency will be 10 kHz).

Second part is the effect of attaching and defending. People will make some sound during the fright, so that these sounds will come at special situation.

So that we need to do the pre treatment, for regular situation, 15 bits are used to store the data and the last bit is used to avoid the data flow when two effects add together.(Generally , the back ground music and the effect of the attack will fixed into together).

The hardware part of audio design:

The hardware part of the audio design is simple. The hardware structure of the audio is built in the SRAM, and it will connect with CPU, who connects the data in SDRM.

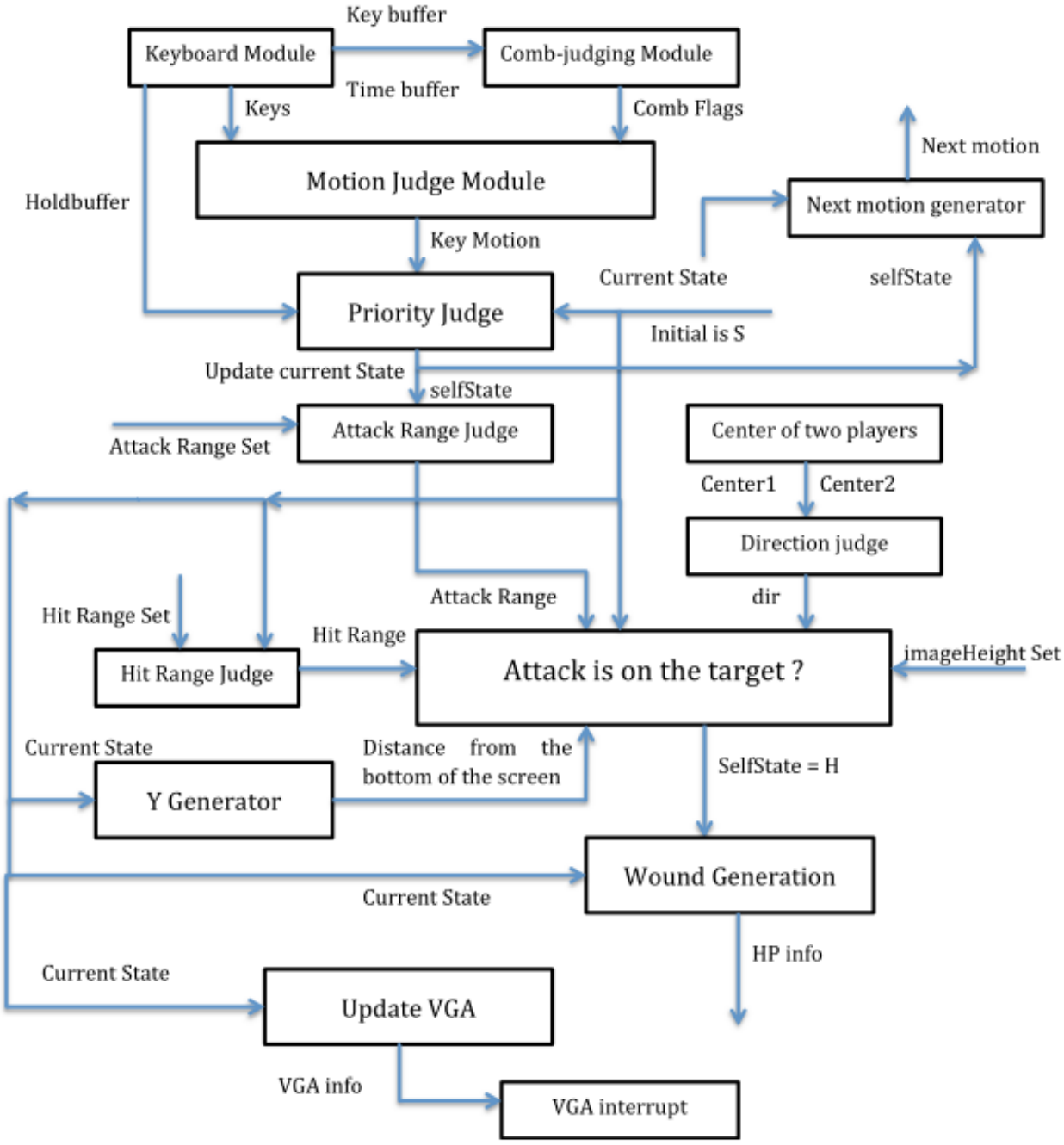
The biggest advantage of this structure is that the data can store in the CPU which connect with the SDRM. By that way, the memory will be not a problem for the music store and the CPU can call the music very easily.

The interruption signal is designed in the structure. After the interruption, the CPU will write the data. After that the data-buffer will be sent from the SDRM to the audio. In our design, the buffer size is 20. After the buffer has be read, a new interruption will be sent to the CPU again.

The software part of audio design:

For the software of the audio part design, we use the CPU to deal with the interruption signal. Generally, the background music will be sent to the audio to play every time when the interruption signal is coming. When the fighter makes some sound, the sound can add with the back ground sound, forming the new sound which can be written into the audio hardware.

Game Software



The structure of the software implementation is actually challenging, especially taking the large data information and complicated judgment into account. Interrupt and current state replacement are really very tricky in the design. However, based on the robust hardware system, software is much easier to be realized.

● Experience and Issues

Making a game of our own should be a dream project for boys. Indeed, we spent great amount of passion and efforts on this. Fortunately, the result is almost pleasant to us. But the story in between is never easy.

Our major issue at the beginning is to understand the design in a hardware manner rather in software coding manner. Our first several try on VHDL design doesn't turn out to be good because lack of consideration on timing and resource allocation. During the worst period, the FPGA chip is used up and the design cannot be implemented. Single compilation time is more than one hour… …If there is only one thing we did wisely, that is our determination of making a solid and fast hardware before any software work begins. With this point in mind, we change our storage structure and sprite controller several times for a perfect performance. Finally we achieved what we want---a solid display hardware design that can make the rest work no longer related to hardware modification.

The second issue we met is the gaming logic. Although the display is solid and easy to control, we should consider the timing of display update and keyboard input. With such limitations, software coding cannot be of large scale. We struggle to use concise code to implement the game logic, but fighting game is just the one that requires the most situation judgments. Several large FSM style function have to be used to make correct function. The debug really took us a lot of time. Finally, we finish the project.

To summarize the valuable experience, it should be "think deeply about resource and timing to make solid hardware" and "be really careful in software debug".

VHDL CODE FOR THE WHOLE STRUCTURE

```
# TCL File Generated by Component Editor 7.2 on:
# Mon Apr 09 22:47:53 EDT 2012
# DO NOT MODIFY

set_source_file "buf.vhd"
set_module "buf"
set_module_description ""
set_module_property "className" "buf"
set_module_property "group" ""
set_module_property  "libraries"  [  list  "ieee.std_logic_1164.all"  "ieee.numeric_std.all"
"std.standard.all" ]
set_module_property "synthesisFiles" "buf.vhd"

# Module parameters

# Interface clock
add_interface "clock" "clock" "sink" "asynchronous"
# Ports in interface clock
add_port_to_interface "clock" "reset" "reset"
add_port_to_interface "clock" "clk_50" "clk"

# Interface export_0
add_interface "export_0" "conduit" "start" "clock"
# Ports in interface export_0
add_port_to_interface "export_0" "clk_50_out" "export"
add_port_to_interface "export_0" "reset_out" "export"
add_port_to_interface "export_0" "write_out" "export"
add_port_to_interface "export_0" "writedata_out" "export"
add_port_to_interface "export_0" "address_out" "export"
add_port_to_interface "export_0" "irq" "export"

# Interface avalon_slave_0
add_interface "avalon_slave_0" "avalon" "slave" "clock"
set_interface_property "avalon_slave_0" "isNonVolatileStorage" "false"
set_interface_property "avalon_slave_0" "burstOnBurstBoundariesOnly" "false"
set_interface_property "avalon_slave_0" "readLatency" "0"
set_interface_property "avalon_slave_0" "holdTime" "0"
set_interface_property "avalon_slave_0" "printableDevice" "false"
set_interface_property "avalon_slave_0" "readWaitTime" "1"
set_interface_property "avalon_slave_0" "setupTime" "0"
set_interface_property "avalon_slave_0" "addressAlignment" "DYNAMIC"
```

```
set_interface_property "avalon_slave_0" "writeWaitTime" "0"
set_interface_property "avalon_slave_0" "timingUnits" "Cycles"
set_interface_property "avalon_slave_0" "minimumUninterruptedRunLength" "1"
set_interface_property "avalon_slave_0" "isMemoryDevice" "false"
set_interface_property "avalon_slave_0" "linewrapBursts" "false"
set_interface_property "avalon_slave_0" "maximumPendingReadTransactions" "0"
# Ports in interface avalon_slave_0
add_port_to_interface "avalon_slave_0" "address" "address"
add_port_to_interface "avalon_slave_0" "writedata" "writedata"
add_port_to_interface "avalon_slave_0" "write" "write"

# Interface interrupt_sender
add_interface "interrupt_sender" "interrupt" "sender" "clock"
set_interface_property "interrupt_sender" "associatedAddressablePoint" "avalon_slave_0"
# Ports in interface interrupt_sender
add_port_to_interface "interrupt_sender" "irq_out" "irq"



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

entity buf is
port(
    -- interface with CPU
    address   : in std_logic_vector(15 downto 0);-- higher 4 bit is sprite index, lower 6 bit
is ROM number
    writedata : in unsigned(31 downto 0);-- higher 10 bit is y value, lower 10 bit is x value
    write     : in std_logic;
    reset     : in std_logic;
    clk_50    : in std_logic;
    irq_out   : out std_logic;

    -- interface with sprite controller
    address_out   : out std_logic_vector(15 downto 0);-- higher 4 bit is sprite index, lower 6
bit is ROM number
    writedata_out : out unsigned(31 downto 0);-- higher 10 bit is y value, lower 10 bit is x value
    write_out     : out std_logic;
    reset_out     : out std_logic;
    clk_50_out    : out std_logic;
    irq           : in std_logic
    );
end buf;
```

```
architecture bypass of buf is
begin
    address_out <= address;
    writedata_out <= writedata;
    write_out <= write;
    reset_out <= reset;
    clk_50_out <= clk_50;
    irq_out <= irq;
end bypass;

# TCL File Generated by Component Editor 7.2 on:
# Mon Apr 23 17:05:07 EDT 2012
# DO NOT MODIFY

set_source_file "irTimer.vhd"
set_module "irTimer"
set_module_description ""
set_module_property "className" "irTimer"
set_module_property "group" ""
set_module_property  "libraries"  [  list  "ieee.std_logic_1164.all"  "ieee.numeric_std.all"
"std.standard.all" ]
set_module_property "synthesisFiles" "irTimer.vhd"

# Module parameters

# Interface clock
add_interface "clock" "clock" "sink" "asynchronous"
# Ports in interface clock
add_port_to_interface "clock" "clk" "clk"
add_port_to_interface "clock" "reset_n" "reset_n"

# Interface avalon_slave_0
add_interface "avalon_slave_0" "avalon" "slave" "clock"
set_interface_property "avalon_slave_0" "isNonVolatileStorage" "false"
set_interface_property "avalon_slave_0" "burstOnBurstBoundariesOnly" "false"
set_interface_property "avalon_slave_0" "readLatency" "0"
set_interface_property "avalon_slave_0" "holdTime" "0"
set_interface_property "avalon_slave_0" "printableDevice" "false"
set_interface_property "avalon_slave_0" "readWaitTime" "1"
set_interface_property "avalon_slave_0" "setupTime" "0"
set_interface_property "avalon_slave_0" "addressAlignment" "DYNAMIC"
set_interface_property "avalon_slave_0" "writeWaitTime" "0"
set_interface_property "avalon_slave_0" "timingUnits" "Cycles"
set_interface_property "avalon_slave_0" "minimumUninterruptedRunLength" "1"
```

```
set_interface_property "avalon_slave_0" "isMemoryDevice" "false"
set_interface_property "avalon_slave_0" "linewrapBursts" "false"
set_interface_property "avalon_slave_0" "maximumPendingReadTransactions" "0"
# Ports in interface avalon_slave_0
add_port_to_interface "avalon_slave_0" "chipselect" "chipselect"
add_port_to_interface "avalon_slave_0" "read" "read"
add_port_to_interface "avalon_slave_0" "write" "write"
add_port_to_interface "avalon_slave_0" "address" "address"
add_port_to_interface "avalon_slave_0" "readdata" "readdata"
add_port_to_interface "avalon_slave_0" "writedata" "writedata"

# Interface interrupt_sender
add_interface "interrupt_sender" "interrupt" "sender" "clock"
set_interface_property "interrupt_sender" "associatedAddressablePoint" "avalon_slave_0"
# Ports in interface interrupt_sender
add_port_to_interface "interrupt_sender" "irq" "irq"

-- Shangru Li
--
-- This component is basically a 26-bit counter
--
-- When the counter is 1, irq will be '1'
-- The irq is reset to '0' by the cpu, when the cpu wants to write to this
-- component, the irq will become '0'
--
-- The address, readdata, and writedata are effectively ignored

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

entity irTimer is port (
  clk, reset_n, chipselect, read, write, 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 data    : std_logic_vector(15 downto 0);
  signal counter : unsigned(22 downto 0);
begin

  process (clk)
```

```
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      data <= (others => '0');
    else
      if chipselect = '1' then
        if address = '1' then
          if write = '1' then
            data <= writedata;
          elsif read = '1' then
            readdata <= data;
          end if;
        end if;
      end if;
    end if;
  end if;
end process;

process (clk)
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      counter <= (others => '0');
    else
      counter <= counter +1 ;
    end if;
  end if;
end process;

process (clk)
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      irq <= '0';
    else
      if counter = 1 then
        irq <= '1';
      elsif write = '1' and chipselect = '1' then
        irq <= '0';  -- important to reset the irq
      end if;
    end if;
  end if;
end process;
```

```
end rtl;

# TCL File Generated by Component Editor 7.2 on:
# Thu May 10 02:19:32 EDT 2012
# DO NOT MODIFY

set_source_file "de2_ps2.vhd"
set_module "de2_ps2"
set_module_description ""
set_module_property "className" "de2_ps2"
set_module_property "group" ""
set_module_property  "libraries"  [  list  "ieee.std_logic_1164.all"  "ieee.numeric_std.all"
"std.standard.all" ]
set_module_property "synthesisFiles" "de2_ps2.vhd"

# Module parameters

# Interface clock
add_interface "clock" "clock" "sink" "asynchronous"
# Ports in interface clock
add_port_to_interface "clock" "avs_s1_clk" "clk"
add_port_to_interface "clock" "avs_s1_reset" "reset"

# Interface export_0
add_interface "export_0" "conduit" "start" "clock"
# Ports in interface export_0
add_port_to_interface "export_0" "PS2_Clk" "export"
add_port_to_interface "export_0" "PS2_Data" "export"

# Interface s1
add_interface "s1" "avalon" "slave" "clock"
set_interface_property "s1" "isNonVolatileStorage" "false"
set_interface_property "s1" "burstOnBurstBoundariesOnly" "false"
set_interface_property "s1" "readLatency" "0"
set_interface_property "s1" "holdTime" "0"
set_interface_property "s1" "printableDevice" "false"
set_interface_property "s1" "readWaitTime" "1"
set_interface_property "s1" "setupTime" "0"
set_interface_property "s1" "addressAlignment" "NATIVE"
set_interface_property "s1" "writeWaitTime" "0"
set_interface_property "s1" "timingUnits" "Cycles"
set_interface_property "s1" "minimumUninterruptedRunLength" "1"
set_interface_property "s1" "isMemoryDevice" "false"
set_interface_property "s1" "linewrapBursts" "false"
```

```
set_interface_property "s1" "maximumPendingReadTransactions" "0"
# Ports in interface s1
add_port_to_interface "s1" "avs_s1_address" "address"
add_port_to_interface "s1" "avs_s1_read" "read"
add_port_to_interface "s1" "avs_s1_chipselect" "chipselect"
add_port_to_interface "s1" "avs_s1_readdata" "readdata"

# Interface interrupt_sender
add_interface "interrupt_sender" "interrupt" "sender" "clock"
set_interface_property "interrupt_sender" "associatedAddressablePoint" "s1"
# Ports in interface interrupt_sender
add_port_to_interface "interrupt_sender" "keyIrq" "irq"

--------------------------------------------------------------------
--
-- 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.

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 unsigned(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 unsigned(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     : unsigned(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 unsigned(0 downto 0);
    avs_s1_read       : in std_logic;
    avs_s1_chipselect : in std_logic;
    avs_s1_readdata   : out unsigned(7 downto 0);

    PS2_Clk           : in std_logic;
    PS2_Data          : in std_logic;

    keyIrq : out std_logic
    --keyHandlerIrq : out std_logic
    );
end de2_ps2;

architecture rtl of de2_ps2 is

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

begin

  keyIrq <= DataAvailable;
  --keyHandlerIrq <= DataAvailable;

  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(0);
    end if;
  end process;

  process (Data, DataAvailable, avs_s1_address, avs_s1_chipselect)
  begin
    if avs_s1_chipselect = '1' then
      if avs_s1_address(0) = '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;
```

The C CODE FOR THE CORE PART

For the keyboard

```
#ifndef KEYBOARD_CTRL_H_
#define KEYBOARD_CTRL_H_

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

#include <sys/alt_irq.h> // the irq functions
#include <alt_types.h>

#define KB_RESET 0xFF
#define KB_SET_DEFAULT 0xF6
#define KB_DISABLE 0xF5
#define KB_ENABLE 0xF4
#define KB_SET_TYPE_RATE_DELAY 0xF3
```

```
/*key set for player 1*/
#define UP_1     0x1D
#define DOWN1    0x1B
#define LEFT1    0x1C
#define RIGHT1   0x23

#define ACTION_A1   0x35
#define ACTION_B1   0x3C
#define ACTION_C1   0x33
#define ACTION_D1   0x3B

/*key set for player 2*/
#define UP_2     0x75
#define DOWN2    0x72
#define LEFT2    0x6B
#define RIGHT2   0x74

#define ACTION_A2   0x70
#define ACTION_B2   0x6C
#define ACTION_C2   0x71
#define ACTION_D2   0x69

int P1_holdBuf[9];
int P2_holdBuf[9];



/**
 * @brief The Enum type for the type of keyboard code received
 **/
typedef enum
  {
    /** @brief --- Make Code that corresponds to an ASCII character.
    For example, the ASCII Make Code for letter <tt>A</tt> is 1C
     */
    KB_ASCII_MAKE_CODE = 1,
    /** @brief --- Make Code that corresponds to a non-ASCII character.
    For example, the Binary (Non-ASCII) Make Code for
    <tt>Left Alt</tt> is 11
     */
    KB_BINARY_MAKE_CODE = 2,
    /** @brief --- Make Code that has two bytes (the first byte is E0).
    For example, the Long Binary Make Code for <tt>Right Alt</tt>
    is "E0 11"
```

```
     */
    KB_LONG_BINARY_MAKE_CODE = 3,
    /** @brief --- Normal Break Code that has two bytes (the first byte is F0).
    For example, the Break Code for letter <tt>A</tt> is "F0 1C"
     */
    KB_BREAK_CODE = 4,
    /** @brief --- Long Break Code that has three bytes (the first two bytes
    are E0, F0). For example, the Long Break Code for <tt>Right Alt</tt>
    is "E0 F0 11"
     */
    KB_LONG_BREAK_CODE = 5,
    /** @brief --- Codes that the decode FSM cannot decode
     */
    KB_INVALID_CODE = 6,

    P1_PRESS_CODE=7,
    P1_RELEASE_CODE=8,

    P2_PRESS_CODE=9,
    P2_RELEASE_CODE=10

  } KB_CODE_TYPE;

/**
 * @brief Get the make code of the key when a key is pressed
 *
 * @param decode_mode -- indicates which type of code
 * (Make Code, Break Code, etc.) is received from the keyboard when the
 * key is pressed
 *
 * @param buf -- points to the location that stores the make code of
 *               the key pressed
 * @note For KB_LONG_BINARY_MAKE_CODE and KB_BREAK_CODE, only the
 *       second byte is retured. For KB_LONG_BREAK_CODE, only the
 *       third byte is returned
 *
 * @return \c PS2_TIMEOUT on timeout, or \c PS2_ERROR on error,
 *         otherwise \c PS2_SUCCESS
 **/
extern void read_make_code(KB_CODE_TYPE *decode_mode, int *buf);
void get_motion_code(alt_u8 *buf);

/**
 * @brief Set the repeat/delay rate of the keyboard
```

```
 *
 * @param rate -- an 8-bit number that represents the repeat/delay rate
 *                of the keyboard
 *
 * @return PS2_SUCCESS on success, otherwise PS2_ERROR
 **/
//extern alt_u32 set_keyboard_rate(alt_u8 rate);

/**
 * @brief Send the reset command to the keyboard
 *
 * @return \c PS2_SUCCESS on passing the BAT (Basic Assurance Test),
 *         otherwise \c PS2_ERROR
 **/
//extern alt_u32 reset_keyboard();

#endif
```

For vga_updata

```
#ifndef _VGA_UPDATE_H_
#define _VGA_UPDATE_H_
//---------basic parameters-----
#define FULL_TIME          90
#define SPEED               2
#define JSPEED              5
#define LIGHT_WOUND         4
#define MID_WOUND           8
#define HEAVY_WOUND         12
#define FULL_BLOOD          212

//-------color table------------
#define BACKGROUND                    0
#define BODY1                         1
#define BODY2                         2
#define ORANGE                        3
#define BLACK                          4
#define WHITE                          5
#define EFFECT1                       6
#define EFFECT2                       7

//-------fixed position---------
#define SHADOW_Y                   45
#define SHADOW_CENTER             40
#define HIT_CENTER                    30
```

```
#define GUN_CENTER                20
#define GUN_Y                    50
#define GUN_W                   16
#define HG_FIRE_W                 20
#define HG_FIRE_Y                50
#define HG_FIRE_TOP              400

//-------state index--------------
#define W1           0
#define W2           1
#define W3           2
#define W4          3
#define W5           4
#define W6          5
#define W7          6
#define W8         7
#define W9         8
#define W10       9
//------------
#define HP1          10
#define HP2          11
#define HP3          12
#define HP4          13
//------------
#define HK1          14
#define HK2          15
#define HK3         16
//------------
#define LP1          17
#define LP2          18
//------------
#define LK1          19
#define LK2          20
//------------
#define L1             21
//------------
#define LHP1         22
#define LHP2         23
#define LHP3         24
#define LHP4         25
//------------
#define LLP1         26
#define LLP2         27
//------------
```

```
#define LHK1          28
#define LHK2          29
#define LHK3         30
//------------
#define LLK1          31
#define LLK2          32
//------------
#define J1            33
#define J2           34
#define J3            35
#define J4            36
#define J5           37
#define J6           38
//------------
#define JK1           39
#define JK2           40
#define JK3          41
#define JK4          42
#define JK5         43
#define JK6          44
//------------
#define JP1           45
#define JP2          46
#define JP3          47
#define JP4          48
#define JP5          49
#define JP6          50
//------------
#define GUN1         51
#define GUN2         52
#define GUN3          53
#define GUN4          54
#define GUN5          55
//------------
#define DASH1        56
#define DASH2        57
#define DASH3        58
#define DASH4        59
#define DASH5        60
#define DASH6        61
//------------
#define LASER1        62
#define LASER2        63
#define LASER3        64
```

```
#define LASER4       65
#define LASER5       66
#define LASER6       67
//------------
#define UP1          68
#define UP2          69
#define UP3          70
#define UP4          71
#define UP5          72
//------------
#define HG1          73
#define HG2          74
#define HG3          75
#define HG4          76
//------------
#define S1             77
//------------
#define H1             78
#define H2             79
#define H3             80
//------------
#define D1             81
#define D2             82
#define D3             83
#define D4             84
//------------
#define LD1           85
#define LD2           86
#define LD3           87
#define LD4           88
//------------
#define WIN1        89
//------------
#define LOSE1       90
//-------------
#define KD1           91
#define KD2           92
#define KD3           93
#define KD4           94
//--------------
#define HG5          95
#define HG6          96
#define HG7          97
#define HG8          98
```

```
#define HG9          99
#define HG10          100
#define HG11          101
#define HG12          102
#define HG13          103
#define HG14          104
#define HG15          105
#define HG16          106
//----motion category---------------
#define LW           0
#define RW           1
#define HP            2
#define HK            3
#define LP             4
#define LK             5
#define L               6
#define LHP          7
#define LLP           8
#define LHK          9
#define LLK         10
#define J             11
#define LJ           12
#define RJ           13
#define JK           14
#define JP           15
#define GUN       16
#define DASH      17
#define LASER     18
#define UP           19
#define HG          20
#define S             21
#define H            22
#define D            23
#define LD          24
#define WIN       25
#define LOSE      26
#define KD     27

//-----------------------------------------------------------------
#define IOWR_SET_SPRITE(x, y, rom, sprite, flip, color) \
    IOWR_32DIRECT( SPRITE_BUF_BASE, 4*(rom + (sprite<<7)), x + (y<<10) + (flip<<30) + (color<<20) )

#define IOWR_LEFT_CUT(x, sprite) \
```

```
    IOWR_32DIRECT( SPRITE_BUF_BASE, 4*(sprite<<7), (1<<29) + x )

#define IOWR_RIGHT_CUT(x, sprite) \
    IOWR_32DIRECT( SPRITE_BUF_BASE, 4*(sprite<<7), (1<<28) + x )

#define IOWR_SET_BG_COLOR(color) \
    IOWR_32DIRECT( SPRITE_BUF_BASE, 4*(21<<7), (color<<20) )

#define IOWR_END_IRQ() \
    IOWR_32DIRECT( SPRITE_BUF_BASE, 0, (1<<31) + 1023 + (1023<<10) )

void sprite_refresh();
void fight_environment(int p1_hp_off, int p2_hp_off, int time);
void game_menu(int cursor_position);
void sprite_clear_all();

#endif /*VGA_UPDATE_H_*/



#include "keyboard_ctrl.h"
#include "time.h"

#define NUM_SCAN_CODES  102
//#define dRegMax 5
//#define motionNum 8

//char direction;
//int directionCount=0;
//int holdCount=0;

int keyCode;

////////////////////////////////////////////////////////////////////
// Table of scan code, make code and their corresponding values
// These data are useful for developing more features for the keyboard
//
alt_u8 *key_table[NUM_SCAN_CODES] = {
  "A", "B", "C", "D", "E", "F", "G", "H",
  "I", "J", "K", "L", "M", "N", "O", "P",
  "Q", "R", "S", "T", "U", "V", "W", "X",
  "Y", "Z", "0", "1", "2", "3", "4", "5",
  "6", "7", "8", "9", "`", "-", "=", "\\",
  "BKSP", "SPACE", "TAB", "CAPS", "L SHFT", "L CTRL", "L GUI", "L ALT",
  "R SHFT", "R CTRL", "R GUI", "R ALT", "APPS", "ENTER", "ESC", "F1",
```

```
  "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9",
  "F10", "F11", "F12", "SCROLL", "[", "INSERT", "HOME", "PG UP",
  "DELETE", "END", "PG DN", "U ARROW", "L ARROW", "D ARROW", "R ARROW", "NUM",
  "KP /", "KP *", "KP -", "KP +", "KP ENTER", "KP .", "KP 0", "KP 1",
  "KP 2", "KP 3", "KP 4", "KP 5", "KP 6", "KP 7", "KP 8", "KP 9",
  "]", ";", "'", ",", ".", "/"
};

alt_u8 ascii_codes[NUM_SCAN_CODES] = {
  'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
  'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
  'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
  'Y', 'Z', '0', '1', '2', '3', '4', '5',
  '6', '7', '8', '9', '`', '-', '=', 0,
  0x08, 0, 0x09, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0x0A, 0x1B,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, '[', 0, 0,
  0, 0x7F, 0, 0, 0, 0, 0, 0,
  0, '/', '*', '-', '+', 0x0A, '.', '0', '1',
  '2', '3', '4', '5', '6', '7', '8', '9',
  ']', ';', '\'', ',', '.', '/'
};

alt_u8 single_byte_make_code[NUM_SCAN_CODES] = {
  0x1C, 0x32, 0x21, 0x23, 0x24, 0x2B, 0x34, 0x33,
  0x43, 0x3B, 0x42, 0x4B, 0x3A, 0x31, 0x44, 0x4D,
  0x15, 0x2D, 0x1B, 0x2C, 0x3C, 0x2A, 0x1D, 0x22,
  0x35, 0x1A, 0x45, 0x16, 0x1E, 0x26, 0x25, 0x2E,
  0x36, 0x3D, 0x3E, 0x46, 0x0E, 0x4E, 0x55, 0x5D,
  0x66, 0x29, 0x0D, 0x58, 0x12, 0x14,    0, 0x11,
  0x59,    0,    0,    0,    0, 0x5A, 0x76, 0x05,
  0x06, 0x04, 0x0C, 0x03, 0x0B, 0x83, 0x0A, 0x01,
  0x09, 0x78, 0x07, 0x7E, 0x54,    0,    0,    0,
  0,    0,    0,    0,    0,    0,    0, 0x77,
  0, 0x7C, 0x7B, 0x79,    0, 0x71, 0x70, 0x69,
  0x72, 0x7A, 0x6B, 0x73, 0x74, 0x6C, 0x75, 0x7D,
  0x5B, 0x4C, 0x52, 0x41, 0x49, 0x4A };

alt_u8 multi_byte_make_code[NUM_SCAN_CODES] = {
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
```

```
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0x1F, 0,
  0, 0x14, 0x27, 0x11, 0x2F, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0x70, 0x6C, 0x7D,
  0x71, 0x69, 0x7A, 0x75, 0x6B, 0x72, 0x74, 0,
  0x4A, 0, 0, 0, 0x5A, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0 };
//////////////////////////////////////////////////////////////////////

// States for the Keyboard Decode FSM
typedef enum
  {
    STATE_INIT,
    STATE_LONG_BINARY_MAKE_CODE,
    STATE_BREAK_CODE ,
    STATE_DONE
  } DECODE_STATE;

//helper function for get_next_state
alt_u8 get_multi_byte_make_code_index(alt_u8 code)
{
  alt_u8 i;
  for (i = 0; i < NUM_SCAN_CODES; i++ ) {
    if ( multi_byte_make_code[i] == code )
      return i;
  }
  return NUM_SCAN_CODES;
}

//helper function for get_next_state
alt_u8 get_single_byte_make_code_index(alt_u8 code)
{
  alt_u8 i;
  for (i = 0; i < NUM_SCAN_CODES; i++ ) {
    if ( single_byte_make_code[i] == code )
      return i;
  }
  return NUM_SCAN_CODES;
}

//helper function for read_make_code
/* FSM Diagram (Main transitions)
```

```
 * Normal bytes: bytes that are not 0xF0 or 0xE0

 ------
 |     |
 |     |
 |    INIT ------ 0xF0 ----> BREAK CODE
 |     |                      /  |
 |     |                     /   |
 |    0xE0                  /    |
Normal |                    /    Normal
 |     |      ----0xF0--->       |
 |     V     /                   |
 |    LONG  /                    V
 | MAKE/BREAK --- Normal ----> DONE
 |    CODE                       ^
 X-------------------------------|

*/

DECODE_STATE get_next_state(DECODE_STATE state,
                alt_u8 byte,
                KB_CODE_TYPE *decode_mode,
                alt_u8 *buf)
{
  int P1_motion = (byte == UP_1 || byte == DOWN1 || byte == LEFT1 || byte == RIGHT1);
  int P1_action = (byte == ACTION_A1 || byte == ACTION_B1 || byte == ACTION_C1 || byte == ACTION_D1);

  int P2_motion = (byte == UP_2 || byte == DOWN2 || byte == LEFT2 || byte == RIGHT2);
  int P2_action = (byte == ACTION_A2 || byte == ACTION_B2 || byte == ACTION_C2 || byte == ACTION_D2);

  DECODE_STATE next_state = STATE_INIT;
  alt_u16 idx = NUM_SCAN_CODES;

  switch (state) {
  case STATE_INIT:
    if ( byte == 0xE0 ) {
      next_state = STATE_LONG_BINARY_MAKE_CODE;
    } else if (byte == 0xF0) {
      next_state = STATE_BREAK_CODE;
    } else {
      next_state = STATE_INIT;
      idx = get_single_byte_make_code_index(byte);

      if ( (idx < 40 || idx == 68 || idx > 79) && ( idx != NUM_SCAN_CODES ) ) {
           *decode_mode = KB_ASCII_MAKE_CODE;
```

```
          *buf= 'x';
           //printf("Key code is %d\n",keyCode);
           if(P1_action || P1_motion){
              //if(keyCode==P1_previousKey)
              P1_holdBuf[keyCode]+=1;

              *buf= keyCode;//ascii_codes[idx];
              next_state = STATE_DONE;

              *decode_mode = P1_PRESS_CODE;

              //P1_previousKey=keyCode;
           }

    }
    else {
         *decode_mode = KB_BINARY_MAKE_CODE;
         *buf = byte;
    }
    next_state = STATE_DONE;
  }
  break;
case STATE_LONG_BINARY_MAKE_CODE:
  if ( byte != 0xF0 && byte!= 0xE0) {
    *decode_mode = KB_LONG_BINARY_MAKE_CODE;
    *buf = 'x';

    if(P2_action || P2_motion){
        //if(keyCode==P2_previousKey)
        P2_holdBuf[keyCode]+=1;

        *buf= keyCode;//ascii_codes[idx];
        *decode_mode = P2_PRESS_CODE;

        //P2_previousKey=keyCode;
    }

    next_state = STATE_DONE;
  } else {
    next_state = STATE_BREAK_CODE;
  }
  break;
case STATE_BREAK_CODE:
  if ( byte != 0xF0 && byte != 0xE0) {
```

```
      *decode_mode = KB_BREAK_CODE;

      if(P1_action || P1_motion){
         P1_holdBuf[keyCode]=0;

         *buf= keyCode;
         next_state = STATE_DONE;

         *decode_mode = P1_RELEASE_CODE;
      }

      if(P2_action || P2_motion){
         P2_holdBuf[keyCode]=0;

         *buf= keyCode;
         next_state = STATE_DONE;

         *decode_mode = P2_RELEASE_CODE;
      }

      next_state = STATE_DONE;
    } else {
      next_state = STATE_BREAK_CODE;
    }
    break;
  default:
    *decode_mode = KB_INVALID_CODE;
    next_state = STATE_INIT;
  }
  return next_state;
}

/*
void get_motion_code(alt_u8 *buf){
    int i,j,matchCount;
    //printf("\n dReg is %s \n",dReg);
    *buf='0';
    for(i=0;i<motionNum;i++){
        matchCount=0;
        for(j=0;j<directionCount+1;j++){
            //printf("\n motion_table is %c \n",motion_table[i][j]);
            if (dReg[j]!= motion_table[i][j])
                break;
            matchCount++;
```

```
        }
        if(matchCount==directionCount+1){
            *buf=motion_code_table[i];
            break;
        }
    }
}
*/

void read_make_code(KB_CODE_TYPE *decode_mode, int *buf)
{
  alt_u8 byte = 0;
  //int status_read =0;
  *decode_mode = KB_INVALID_CODE;
  DECODE_STATE state = STATE_INIT;
  do {
    while (!IORD_8DIRECT(DE2_PS2_INST_BASE, 0));/* Poll the status */
       byte = IORD_8DIRECT(DE2_PS2_INST_BASE, 4);

    //printf("Byte is %x\n",byte);
    switch(byte){
       case UP_1:    keyCode=1;break;
       case DOWN1:  keyCode=2;break;
       case LEFT1:  keyCode=3;break;
       case RIGHT1: keyCode=4;break;
       case ACTION_A1: keyCode=5;break;
       case ACTION_B1: keyCode=6;break;
       case ACTION_C1: keyCode=7;break;
       case ACTION_D1: keyCode=8;break;


       case UP_2:    keyCode=1;break;
       case DOWN2:  keyCode=2;break;
       case LEFT2:  keyCode=3;break;
       case RIGHT2: keyCode=4;break;
       case ACTION_A2: keyCode=5;break;
       case ACTION_B2: keyCode=6;break;
       case ACTION_C2: keyCode=7;break;
       case ACTION_D2: keyCode=8;break;

       default:


       keyCode=0;
```

```
    }


    state = get_next_state(state, byte, decode_mode, buf);
  } while (state != STATE_DONE);
}




#include <io.h>
#include <system.h>
#include <stdio.h>
#include <sys/alt_irq.h> // the irq functions
#include <alt_types.h>
#include <keyboard_ctrl.h>
#include "vga_update.h"

//------------------------------
KB_CODE_TYPE decode_mode;

int time_count = 0;
int currentT;

int keyBuf1[10] ={0,0,0,0,0,0,0,0,0,0};
int timeBuf1[10]={0,0,0,0,0,0,0,0,0,0};
int buf1_e=0;

int keyBuf2[10] ={0,0,0,0,0,0,0,0,0,0};
int timeBuf2[10]={0,0,0,0,0,0,0,0,0,0};
int buf2_e=0;
//----------------------------------------
//-------------WIDTH-----------------
const int romImage[107] = {
    16, 17, 17, 18, 18, 19, 19, 20, 20, 21,     //WALK
    1, 1, 2, 3,                         //HP
    26, 27, 27,                     //HK
    5, 5,                               //LP
    4, 4,                             //LK
    7,                                //L
    12, 12, 13, 14,                     //LHP
    10, 11,                           //LLP
    36, 36, 36,                       //LHK
    8, 9,                           //LLK
```

```
    28, 30, 30, 30, 28, 28,              //J
    8, 8, 8, 8, 8, 8,              //JK
    29, 29, 29, 29, 29, 29,              //JP
    38, 39, 40, 41, 41,                  //GUN
    37, 37, 37, 37, 37, 37,                  //DASH
    42, 42, 43, 43, 43, 43,              //LASER
    31, 32, 32, 33, 33,                  //UP
    23, 23, 24, 25,                      //HG
    15,                                 //S
    22, 22, 22,                         //H
    0, 0, 0, 0,                             //D
    6, 6, 6, 6,                             //LD
    0,                                  //WIN
    35,                                 //LOSE
    34, 34, 34, 34,                         //KD
    25, 25, 25, 25,                         //HG2
    25, 25, 25, 25,                         //HG3
    25, 25, 25, 25                        //HG4
};
//-----------------------------------------
const int romWidth [28] =
{68, 68,  132, 136,  76,   84,  76, 112, 112, 152, 112,
 52, 52, 52, 112,  76, 104,  112, 116,  68, 100,  60,  72,
 64,   76,  44,  44,  152
 };
//--------------HEIGHT----------------
const int imageHeight [28] =
 {150, 150, 150, 150, 150, 150, 110, 110, 110,  88, 110,
  162, 162, 162, 110, 120, 150, 150, 150, 162, 162, 150, 150,
  150, 110, 150, 150, 37
  };
 //-------------CNETER-----------------
const int romCenter [28] =
{
40, 40, 50, 44, 22, 37, 40, 40, 34, 30, 40,
18, 18, 18, 40, 15, 54, 18, 43, 22, 43, 24, 43,
20, 40, 18, 18, 75
};

const int attackRangeX[28] = {
    0, 0, 78, 88, 52, 52, 0, 71, 50, 118,
   70, 0,  0,  0, 71, 56,43,  78, 70,  36,
   50, 0,  0,  0,  0,  0, 0,  0
};
```

```
const int attackRangeY[28] = {
    0, 0, 120, 98, 115, 22, 0, 90, 75, 56,
   18, 0,   0,  0,  18, 43, 120,  120,  110, 146,
    0, 0,   0,  0,   0,  0, 0,  0
};


const int hitRangeX1[28] = {
    15, 20, 25, 40, 20, 34, 30, 33, 30, 0,
    37, 18, 18, 18, 37, 15, 24, 18, 35, 22,
    30, 24, 22, 20, 37,  0, 13, 0
};

const int hitRangeX2[28] = {
    20, 15, 35, 30, 22, 17, 30, 24, 21, 0,
    26, 18, 18, 18, 31, 45, 26, 50, 40, 28,
    53, 28, 44, 40, 20,  0, 20, 0
};

const int hitRangeY[28] = {
   0,  0,  0, 15, 0, 0, 0, 0, 10, 0,
   2, 16, 16, 16, 0, 0, 0, 0,  5, 15,
  10,  8, 10,  0, 3, 0, 0, 0
};

const int Wound[28] = {
   0,  0,  8,  8, 4, 4, 0, 8, 4, 8,
   4, 0, 0, 0, 4, 4, 12, 12,  12, 12, 16,
   0, 0,  0, 0, 0, 0, 0
};

const int startMotion[28] = {
0, 0, 10, 14, 17, 19, 21, 22, 26, 28,
31, 33, 33, 33, 39, 45, 51, 56, 62, 68,
73, 77, 78, 81, 85, 89, 90, 91};
//---------------------------------------
int dir1=0,dir2=1;


//==================================
int time = FULL_TIME;
//int is_fighting, cursor_position, fight_time;
```

```
int p1_x[2], p1_y[2], p2_x[2], p2_y[2];
int p1_center[2], p2_center[2];
int p1_width[2], p2_width[2];
int p1_image[2], p2_image[2];
int p1_flip[2], p2_flip[2];
int p1_color[2], p2_color[2];
//================================
 int p1_hp_off[2], p2_hp_off[2];
  //-----------------------------------------
 int hit1[2], hit2[2];
 int hit1_x[2], hit1_y[2], hit2_x[2], hit2_y[2];
  //-----------------------------------------
 int gunn1[2], gunn2[2];
 int gun1_flip[2], gun2_flip[2];
 int gun1_x[2], gun1_y[2], gun2_x[2], gun2_y[2];
  //-------------------------------------------------
 int phantom1[2], phantom2[2];
  //-------------------------------------------------
 int effect1[2], effect2[2];
 int effect1_image[2], effect2_image[2];
 int effect1_x[2], effect1_y[2], effect2_x[2], effect2_y[2];
 //---------------------------------------
int ds=0;
//---------------------------------------
int key=0;
int key1=0, key2=0;

int motionJudge(int key,int* bufferhold);
int priority(int state, int input, int* bufferhold);
int attackRange(int state);
int nextState(int currentState);
int nextJK(int currentState);
int nextJP(int currentState);
int generateY(int currentMotion);
int bufferhold1[9];
int bufferhold2[9];
//-----------------------------------------
int attack_rangeX1_t;
int attack_rangeY1_t;
int attack_rangeX2_t;
int attack_rangeY2_t;
int attack_rangeX1;
int attack_rangeY1;
int attack_rangeX2;
```

```
int attack_rangeY2;
int bullet_attackX1;
int bullet_attackX2;
///---------------------------------------
int hit_rangeXL1_t;
int hit_rangeXR1_t;
int hit_rangeXL2_t;
int hit_rangeXR2_t;

int hit_rangeXL1;
int hit_rangeXR1;
int hit_rangeXL2;
int hit_rangeXR2;

//=======================================

extern int sixth_second;
extern int fifteen_second;

int main()
{
    int i;
    int fast_tick_buf = 1;
    int fast_tick = 11;
    int slow_tick_buf = 1;
    int slow_tick = 11;
    int keyMotion1, selfState1, currentState1 = S, currentMotion1 = S1;
    int keyMotion2, selfState2, currentState2 = S, currentMotion2 = S1;
    int buf_switch = 0;
    int y1, y2;
    int wound1=0, wound2=0;
    int center1 = 100, center2 = 500;
    int color1, color2;
    //=============================================
    int bullet_attackX1 = 0, bullet_attackX2 = 0;
    int bullet_attackY1 = 0, bullet_attackY2 = 0;
    int gun1 = 0, gun2 = 0;
    int gunX1 = 0, gunX2 = 0;
    int gun_dir1 = 0, gun_dir2 = 0;
    //=============================================
    int hg_fire1=0, hg_fire2=0;

    printf("start\n");
    void comb_judg(int *keybuffer, int *time, int* bufEnd, int* key, int dir );
```

```
   void reset_buffer9( int *buffer );
   static void timer_irqhandler (void * context, alt_u32 id);
   static void key_irqhandler (void * context, alt_u32 id);
//-----------------------------------------------------------------------
   alt_irq_register( SPRITE_BUF_IRQ, NULL, sprite_refresh );
   alt_irq_register( DE2_PS2_INST_IRQ, NULL,( void*)key_irqhandler ); // register the irq
   alt_irq_register( IRTIMER_INST_IRQ, NULL,( void*)timer_irqhandler );

   reset_buffer9( P1_holdBuf );
   reset_buffer9( P2_holdBuf );

   while(1){
            for (i=1;i<=8;i++){
             bufferhold1[i]=P1_holdBuf[i];
             bufferhold2[i]=P2_holdBuf[i];
            }

           if (sixth_second != slow_tick_buf){
                   slow_tick = 1;
                   slow_tick_buf = sixth_second;
           }else slow_tick = 0;
           if (fifteen_second != fast_tick_buf){
                   fast_tick = 1;
                   fast_tick_buf = fifteen_second;
                   buf_switch = 1-buf_switch;
                   ds = 1-buf_switch;
           }else fast_tick = 0;
       //---------------------------------------
       if(currentMotion1 == S1) currentState1 = S;
       if(currentMotion2 == S1) currentState2 = S;

       //---------------------------------------
       attack_rangeX1 = 0;
      bullet_attackX1 = 0;
       attack_rangeX2 = 0;
      bullet_attackX2 = 0;

       //-----------get key value------------------
       comb_judg( keyBuf1, timeBuf1, &buf1_e, &key1, dir1);
       comb_judg( keyBuf2, timeBuf2, &buf2_e, &key2, dir2);
       //----------decode key motion---------------
       if(time > 0){
           keyMotion1 = motionJudge(key1, bufferhold1);
           keyMotion2 = motionJudge(key2, bufferhold2);
```

```
        }else{
            keyMotion1 = S;
            keyMotion2 = S;
        }
//        hit1[buf_switch] = 0;
//        hit2[buf_switch] = 0;
         //-----------player color-----------------------
        color1 = BODY1;
        color2 = BODY2;

        //-----------------------------------------
        selfState1 = priority(currentState1 ,keyMotion1, bufferhold1);
        selfState2 = priority(currentState2 ,keyMotion2, bufferhold2);

        //--------direction change---------
        if(center2 < center1){ dir1 = 1; dir2 = 0;}
        else{ dir2 = 1; dir1 = 0;}
        //==================================
        //---------GUN attack range and position-----------------
        if(gun1 && fast_tick){
                bullet_attackY1 = GUN_Y+20;
                if(!gun_dir1) {
                    gunX1 += SPEED;
                    bullet_attackX1 = gunX1 + GUN_W;
                }else{
                    gunX1 -= SPEED;
                    bullet_attackX1 = gunX1 - GUN_W;
                }
        }
        if(gun2 && fast_tick){
                bullet_attackY2 = GUN_Y+20;
                if(!gun_dir2) {
                    gunX2 += SPEED;
                    bullet_attackX2 = gunX2 + GUN_W;
                }
                else{
                    gunX2 -= SPEED;
                    bullet_attackX2 = gunX2 - GUN_W;
                }
        }
        //==================================
//---------generate hg fire----------------------
       if(currentMotion1 >= HG5 && currentMotion1 <= HG16){
               hg_fire1 = 1;
```

```
            attack_rangeX1 += HG_FIRE_W ;
            attack_rangeY1 = HG_FIRE_Y;
            bullet_attackX1 = attack_rangeX1;
            bullet_attackY1 = HG_FIRE_TOP;
        }else    hg_fire1 = 0;
        if(currentMotion2 >= HG5 && currentMotion2 <= HG16){
            hg_fire2 = 1;
            attack_rangeX2 += HG_FIRE_W ;
            attack_rangeY2 = HG_FIRE_Y;
            bullet_attackX2 = attack_rangeX2;
            bullet_attackY2 = HG_FIRE_TOP;
        }else    hg_fire2 = 0;

         //-------attack range-----------------
          attack_rangeX1_t= attackRangeX[ currentState1 ];
          attack_rangeY1_t= attackRangeY[ currentState1 ];
          attack_rangeX2_t= attackRangeX[ currentState2 ];
          attack_rangeY2_t= attackRangeY[ currentState2 ];
          if(currentMotion1 == HP1 || currentMotion1 == HP2|| currentMotion1 == HP3||
          currentMotion1   ==   LHP1   ||   currentMotion1   ==   LHP2||   currentMotion1   ==
LHP3)attack_rangeX1_t = 0;
          if(currentMotion2 == HP1 || currentMotion2 == HP2|| currentMotion2 == HP3||
          currentMotion2   ==   LHP1   ||   currentMotion2   ==   LHP2||   currentMotion2   ==
LHP3)attack_rangeX2_t = 0;

          if ( !dir1 ) attack_rangeX1= center1 + attack_rangeX1_t;
              else attack_rangeX1= center1 - attack_rangeX1_t;
          if ( !dir2 ) attack_rangeX2= center2 + attack_rangeX2_t;
              else attack_rangeX2= center2 - attack_rangeX2_t;

        //=============================================
         //-------generate gun effect-------------
         if (currentMotion1 == GUN4) {
         gun1 = 1;
         gunX1 = attack_rangeX1;
         gun_dir1 = dir1;
         }
         if (currentMotion2 == GUN4){
         gun2 = 1;
         gunX2 = attack_rangeX2;
         gun_dir2 = dir2;
         }

        //----------hit range----------------------------
```

```
         hit_rangeXL1_t= hitRangeX1[ currentState1 ];
         hit_rangeXR1_t= hitRangeX2[ currentState1 ];
         hit_rangeXL2_t= hitRangeX1[ currentState2 ];
         hit_rangeXR2_t= hitRangeX2[ currentState2 ];

         hit_rangeXL1 = center1 - hit_rangeXL1_t;
         hit_rangeXR1 = center1 + hit_rangeXR1_t;
         hit_rangeXL2 = center2 - hit_rangeXL2_t;
         hit_rangeXR2 = center2 + hit_rangeXR2_t;
         //--------defence--------------------------------------
        if(attack_rangeX2_t && (( dir1 && selfState1 == RW)||(!dir1 && selfState1 ==LW)) )
selfState1 = D;
        if(attack_rangeX1_t && (( dir2 && selfState2 == RW)||(!dir2 && selfState2 ==LW)) )
selfState2 = D;
        if(attack_rangeX2_t && (( dir1 && selfState1 == RW)||(!dir1 && selfState1 ==LW)) &&
currentState1 == L) selfState1 = LD;
        if(attack_rangeX1_t && (( dir2 && selfState2 == RW)||(!dir2 && selfState2 ==LW)) &&
currentState2 == L) selfState2 = LD;

       //-----------attack & hit---------------------------------
       if (attack_rangeX1_t)
          if( ( currentState2 != D && currentState2 != LD) && ( attack_rangeX1 >= hit_rangeXL2)
&& (attack_rangeX1 <= hit_rangeXR2)
              &&  ( y1 + attack_rangeY1_t > y2) && (y1 + attack_rangeY1_t < y2 +
imageHeight[currentState2] - hitRangeY[ currentState2 ] ) )
               selfState2 = H;
       if ( gun1==1 )
            if(( bullet_attackX1 >=hit_rangeXL2 ) && (attack_rangeX1 <= hit_rangeXR2)
              && ( bullet_attackY1 > y2 ) && ( bullet_attackY1 < y2 + imageHeight[currentState2]
- hitRangeY[ currentState2 ] ))
              selfState2 = H;
//          else if( ( currentState2 != D && currentState2 != LD) && !dir1 &&( attack_rangeX1 >=
hit_rangeXL2) && (attack_rangeX1 <= hit_rangeXR2)
//                    &&  ( y1 + attack_rangeY1_t > y2) && (y1 + attack_rangeY1_t < y2 +
imageHeight[currentState2] - hitRangeY[ currentState2 ] )
//              || (( bullet_attackX1 >=hit_rangeXL2 ) && (attack_rangeX1 <= hit_rangeXR2)
//              && ( bullet_attackY1 > y2 ) && ( bullet_attackY1 < y2 + imageHeight[currentState2]
- hitRangeY[ currentState2 ] )) )
//               selfState2 = H;
       if(attack_rangeX2_t)
          if( ( currentState1 != D && currentState1 != LD) && ( attack_rangeX2 >= hit_rangeXL1)
&& (attack_rangeX2 <= hit_rangeXR1)
              &&  ( y2 + attack_rangeY2_t > y1) && (y2 +attack_rangeY2_t < y1+
imageHeight[currentState1] - hitRangeY[ currentState1 ] ) )
```

```
            selfState1 = H;
       if( gun2==1 )
           if(( bullet_attackX2 >=hit_rangeXL1 ) && (attack_rangeX2 <= hit_rangeXR1)
              && ( bullet_attackY2 > y1 ) && ( bullet_attackY2 < y1 + imageHeight[currentState1]
- hitRangeY[ currentState1 ] )))
              selfState1 = H;
//          else if( ( currentState1 != D && currentState1 != LD) && dir2 &&( attack_rangeX2 >=
hit_rangeXL1) && (attack_rangeX2 <= hit_rangeXR1)
//                     &&  ( y2+ attack_rangeY2_t  > y1)  &&  (y2  + attack_rangeY2_t  <  y1  +
imageHeight[currentState1] - hitRangeY[ currentState1 ] )
//              || (( bullet_attackX2>=hit_rangeXL1 ) && (attack_rangeX2 <= hit_rangeXR1)
//              && ( bullet_attackY2> y1 ) && ( bullet_attackY2 < y1 + imageHeight[currentState1]
- hitRangeY[ currentState1 ] )) )
//               selfState1 = H;


        //--------------wound generation--------------------
        if(selfState1 == H ) {
            if( fast_tick && dir1 && center1<630) center1++;
            if (fast_tick && !dir1 && center1>10) center1--;
            if (slow_tick) wound1 += Wound[currentState2];
            color1 = EFFECT2;
            if (gun2 && fast_tick) wound1+=12;
        }
        if(selfState2 == H ) {
            if( fast_tick && dir2 && center2<630) center2++;
            if (fast_tick && !dir2 && center2>10) center2--;
            color2 = EFFECT1;
            if(gun1 && fast_tick) wound2+=12;
            if (slow_tick ) wound2 += Wound[currentState1];
        }
        //================================================
                //---------gun effect or dissappear-----
        if( gun1&&(gunX1 >= 630 || gunX1 <= 10|| selfState2 == H) ) gun1 = 0;
        if( gun2&&(gunX2 >= 630 || gunX2 <= 10|| selfState1 == H) ) gun2 = 0;

        //-------------x position-----------------------------------
        if(center1>=10 && (selfState1 == LW)&&fast_tick) center1 -= SPEED;
        else if (center1 <=630 && (selfState1 == RW)&&fast_tick) center1 += SPEED;
        if(center2>=10 && (selfState2 == LW)&&fast_tick) center2 -= SPEED;
        else if (center2 <=630 &&(selfState2 == RW)&&fast_tick) center2 += SPEED;
        //-------------x jump position-------------------------------
        if(center1>=10 && (selfState1 == LJ)&&fast_tick) center1 -= JSPEED;
        else if (center1 <=630 && (selfState1 == RJ)&&fast_tick) center1 += JSPEED;
```

```
        if(center2>=10 && (selfState2 == LJ)&&fast_tick) center2 -= JSPEED;
        else if (center2 <=630 &&(selfState2 == RJ)&&fast_tick) center2 += JSPEED;
        //------------DASH x position----------------------------------
        if(center1>=10 && (selfState1 == DASH && dir1)&&fast_tick) center1 -= (JSPEED + SPEED);
        else if (center1 <=630 && (selfState1 == DASH && !dir1)&&fast_tick) center1 += (JSPEED
+ SPEED);
        if(center2>=10 && (selfState2 == DASH && dir2)&&fast_tick) center2 -= (JSPEED + SPEED);
        else if (center2 <=630 && (selfState2 == DASH && !dir2)&&fast_tick) center2 += (JSPEED
+ SPEED);
        //--------------generate next state--------------
        if(currentState1 == selfState1){
            if(slow_tick) currentMotion1 = nextState(currentMotion1);
        }
        else if( selfState1 == JK) currentMotion1 = nextJK(currentMotion1);
        else if( selfState1 == JP) currentMotion1 = nextJP(currentMotion1);
        else currentMotion1 = startMotion[selfState1];

        if(currentState2 == selfState2){
            if(slow_tick) currentMotion2 = nextState(currentMotion2);
        }
        else if(selfState2 == JK) currentMotion2 = nextJK(currentMotion2);
        else if(selfState2 == JP) currentMotion2 = nextJP(currentMotion2);
        else currentMotion2 = startMotion[selfState2];

        if(currentMotion1 > 106) currentMotion1 = S1;
        if(currentMotion2 > 106) currentMotion2 = S1;
        currentState1 = selfState1;
        currentState2 = selfState2;
        //--------------y position----------------------
        y1 = generateY(currentMotion1);
        y2 = generateY(currentMotion2);

        //---------------update display info-------------
        p1_image[buf_switch] = romImage[currentMotion1];
        p2_image[buf_switch] = romImage[currentMotion2];
        p1_flip[buf_switch] = dir1;
        p2_flip[buf_switch] = dir2;
        p1_x[buf_switch] = center1;
        p2_x[buf_switch] = center2;
        p1_y[buf_switch] = 480 - y1 - imageHeight[currentState1];
        p2_y[buf_switch] = 480 - y2 - imageHeight[currentState2];
        p1_center[buf_switch] = romCenter[currentState1];
        p2_center[buf_switch] = romCenter[currentState2];
        p1_width[buf_switch] = romWidth[currentState1];
```

```
p2_width[buf_switch] = romWidth[currentState2];
p1_color[buf_switch] = color1;
p2_color[buf_switch] = color2;
//-----------------------------------------
hit1[buf_switch] = (selfState1 == H) && attack_rangeX2_t && !gun1;
hit2[buf_switch] = (selfState2 == H) && attack_rangeX1_t && !gun2;
hit1_x[buf_switch] = attack_rangeX2;
hit1_y[buf_switch] =460 - y2 - attack_rangeY2_t;
hit2_x[buf_switch] = attack_rangeX1;
hit2_y[buf_switch] =460 - y1 - attack_rangeY1_t;
if(time > 0){
    p1_hp_off[buf_switch]= wound1;
    p2_hp_off[buf_switch]= wound2;
}
  //-----------------------------------------
 gunn1[buf_switch] = gun1;
 gunn2[buf_switch] = gun2;
 gun1_flip[buf_switch] = gun_dir1;
 gun2_flip[buf_switch] = gun_dir2;
 gun1_x[buf_switch] = gunX1;
 gun2_x[buf_switch] = gunX2;
 gun1_y[buf_switch] = 480 - GUN_Y;
 gun2_y[buf_switch] = 480 - GUN_Y;
//=========================================
phantom1[buf_switch] = (selfState1 == DASH);
phantom2[buf_switch] = (selfState2 == DASH);
//=========================================
effect1[buf_switch] = hg_fire1;
effect2[buf_switch] = hg_fire2;
if(hg_fire1){
    effect1_image[buf_switch] = 64 + fifteen_second%2;
    effect1_x[buf_switch] = center1 + HG_FIRE_W;
    effect1_y[buf_switch] = 480 - y1 - 300;
}
if(hg_fire2){
    effect2_image[buf_switch] = 64 + fifteen_second%2;
    effect2_x[buf_switch] = center2 + HG_FIRE_W;
    effect2_y[buf_switch] = 480 - y2 - 300;
}
//=========================================
if(wound1 >= FULL_BLOOD || wound2 >= FULL_BLOOD) time = 0;
if(time == 0) break;
//=========================================
//if(slow_tick){
```

```
            key1 = 0;
            key2 = 0;
    //}
    }
    while(1){}
    return 0;
}



int motionJudge(int key, int* bufferhold )
{
    int MotionState;

    if ( bufferhold[5] >0 )   MotionState=LP;
    else if ( bufferhold[6] >0 )   {MotionState=2; }
    else if ( bufferhold[7] >0 )   MotionState=LK;
    else if ( bufferhold[8] >0 )   MotionState=HK;
         if ( key == 9  )  { MotionState=GUN;  }
    else if ( key == 10 )  MotionState=DASH;
    else if ( key == 11 )  MotionState=HG;
    else if ( key == 12 )  MotionState=UP;
    else if ( key == 13 )  MotionState=LASER;
    else if ( bufferhold[1] >0 )     MotionState=J;
    else if ( bufferhold[2] >0 )     MotionState=L;
    else if ( bufferhold[3] >0 )     MotionState=LW;
    else if ( bufferhold[4] >0 )     MotionState=RW;
    else if (  bufferhold[1]==0 && bufferhold[2]==0 &&
                bufferhold[3]==0 && bufferhold[4]==0 &&
                bufferhold[5]==0 && bufferhold[6]==0 &&
                bufferhold[7]==0 && bufferhold[8]==0   )    MotionState= S;
    return MotionState;
}

int priority(int state, int input, int* bufferhold )
{
    int new_state;
    switch(state){
    case S:
        new_state = input;
        break;
    case LW:
        if(input == J) new_state = LJ;
        else if(bufferhold[3]==0) new_state = S;
```

```
        else if (input == LW) new_state = state;
        else new_state = input;
        break;
    case RW:
        if(input == J) new_state = RJ;
        else if(bufferhold[4]==0) new_state = S;
        else if (input == RW) new_state = state;
        else    new_state = input;
        break;
    case L:
         if(bufferhold[2]==0)     new_state=S;
        else if(input == S)     new_state= S;
        else if(input == LP)     new_state= LLP;
        else if(input == HP)     new_state= LHP;
        else if(input == LK)     new_state=LLK;
        else if(input == HK)     new_state= LHK;
        else new_state = state;
        break;
    case J:
             if(input==LP)      new_state=JP;
                 else if(input==HP)      new_state=JP;
                 else if(input==LK)     new_state=JK;
                 else if(input==HK)     new_state=JK;
                 else new_state = state;
            break;
    case RJ:
             if(input==LP)      new_state=JP;
                else if(input==HP)     new_state=JP;
                else if(input==LK)     new_state=JK;
                else if(input==HK)     new_state=JK;
                else new_state = state;
            break;
    case LJ:
             if(input==LP)      new_state=JP;
                 else if(input==HP)      new_state=JP;
                 else if(input==LK)     new_state=JK;
                 else if(input==HK)     new_state=JK;
                 else new_state = state;
            break;

    default: new_state = state;
    }
    return new_state;
}
```

```
//int attackRange(int state)
//{
//  int range;
//  switch(state){
//      case
//  }
//}
int nextState(int currentState)
{
    int nextState;
    switch (currentState){
        case W1: nextState = W2;                    break;
        case W2: nextState = W3;                    break;
        case W3: nextState = W4;                    break;
        case W4: nextState = W5;                    break;
        case W5: nextState = W6;                    break;
        case W6: nextState = W7;                    break;
        case W7: nextState = W8;                    break;
        case W8: nextState = W9;                    break;
        case W9: nextState = W10;                    break;
        case W10: nextState = W1;                    break;
     //-----------------------------------
        case HP1: nextState = HP2;                    break;
        case HP2: nextState = HP3;                    break;
        case HP3: nextState = HP4;                    break;
        case HP4: nextState = S1;                    break;
    //-----------------------------------
        case HK1: nextState = HK2;                    break;
        case HK2: nextState = HK3;                    break;
        case HK3: nextState = S1;                    break;
    //-----------------------------------
        case LP1: nextState = LP2;                    break;
        case LP2: nextState = S1;                    break;
    //-----------------------------------
        case LK1: nextState = LK2;                    break;
        case LK2: nextState = S1;                    break;
    //-----------------------------------
        case L1: nextState = L1;                    break;
    //-----------------------------------
        case LHP1: nextState = LHP2;                    break;
        case LHP2: nextState = LHP3;                    break;
        case LHP3: nextState = LHP4;                    break;
        case LHP4: nextState = S1;                    break;
     //-----------------------------------
```

```
    case LLP1: nextState = LLP2;                    break;
    case LLP2: nextState = S1;                      break;
 //-----------------------------------
    case LHK1: nextState = LHK2;                    break;
    case LHK2: nextState = LHK3;                    break;
    case LHK3: nextState = S1;                      break;
//-----------------------------------
    case LLK1: nextState = LLK2;                    break;
    case LLK2: nextState = S1;                      break;
//-----------------------------------
    case J1: nextState = J2;                    break;
    case J2: nextState = J3;                    break;
    case J3: nextState = J4;                    break;
    case J4: nextState = J5;                    break;
    case J5: nextState = J6;                    break;
    case J6: nextState = S1;                    break;
//-----------------------------------
    case JK1: nextState = JK2;                    break;
    case JK2: nextState = JK3;                    break;
    case JK3: nextState = JK4;                    break;
    case JK4: nextState = JK5;                    break;
    case JK5: nextState = JK6;                    break;
    case JK6: nextState = S1;                     break;
//-----------------------------------
    case JP1: nextState = JP2;                    break;
    case JP2: nextState = JP3;                    break;
    case JP3: nextState = JP4;                    break;
    case JP4: nextState = JP5;                    break;
    case JP5: nextState = JP6;                    break;
    case JP6: nextState = S1;                     break;
//-----------------------------------
    case GUN1: nextState = GUN2;                    break;
    case GUN2: nextState = GUN3;                    break;
    case GUN3: nextState = GUN4;                    break;
    case GUN4: nextState = GUN5;                    break;
    case GUN5: nextState = S1;                      break;
//-----------------------------------
    case DASH1: nextState = DASH2;                    break;
    case DASH2: nextState = DASH3;                    break;
    case DASH3: nextState = DASH4;                    break;
    case DASH4: nextState = DASH5;                    break;
    case DASH5: nextState = DASH6;                    break;
    case DASH6: nextState = S1;                       break;
//-----------------------------------
```

```
    case LASER1: nextState = LASER2;                    break;
    case LASER2: nextState = LASER3;                    break;
    case LASER3: nextState = LASER4;                    break;
    case LASER4: nextState = LASER5;                    break;
    case LASER5: nextState = LASER6;                    break;
    case LASER6: nextState = S1;                        break;
 //-----------------------------------
    case UP1: nextState = UP2;                        break;
    case UP2: nextState = UP3;                        break;
    case UP3: nextState = UP4;                        break;
    case UP4: nextState = UP5;                        break;
    case UP5: nextState = S1;                         break;
//-----------------------------------
    case HG1: nextState = HG2;                        break;
    case HG2: nextState = HG3;                        break;
    case HG3: nextState = HG4;                        break;
    case HG4: nextState = HG5;                        break;
    case HG5: nextState = HG6;                        break;
    case HG6: nextState = HG7;                        break;
    case HG7: nextState = HG8;                        break;
    case HG8: nextState = HG9;                         break;
    case HG9: nextState = HG10;                        break;
    case HG10: nextState = HG11;                        break;
    case HG11: nextState = HG12;                        break;
    case HG12: nextState = HG13;                        break;
    case HG13: nextState = HG14;                        break;
    case HG14: nextState = HG15;                        break;
    case HG15: nextState = HG16;                        break;
    case HG16: nextState = S1;                         break;
//-----------------------------------
    case H1: nextState = H2;                       break;
    case H2: nextState = H3;                       break;
    case H3: nextState = S1;                       break;
//-----------------------------------
    case D1: nextState = D2;                       break;
    case D2: nextState = D3;                       break;
    case D3: nextState = D4;                       break;
    case D4: nextState = S1;                       break;
//-----------------------------------
    case LD1: nextState = LD2;                         break;
    case LD2: nextState = LD3;                         break;
    case LD3: nextState = LD4;                         break;
    case LD4: nextState = S1;                          break;
//-----------------------------------
```

```
        case S1: nextState = S1;                    break;
    //-------------------------------------
        case WIN1: nextState = WIN1;                break;
    //-------------------------------------
        case LOSE1: nextState = LOSE1;              break;
    //-------------------------------------
        case KD1: nextState = KD2;                  break;
        case KD2: nextState = KD3;                  break;
        case KD3: nextState = KD4;                  break;
        case KD4: nextState = KD4;                  break;
        default:  nextState = S1;
    }
    return nextState;
}

int nextJK(int currentState)
{
    int nextState;
    switch (currentState){
    case J1: nextState = JK2;       break;
    case J2: nextState = JK3;       break;
    case J3: nextState = JK4;       break;
    case J4: nextState = JK5;       break;
    case J5: nextState = JK6;       break;
    case J6: nextState = S1;        break;
    default: nextState = S1;
    }
    return nextState;
}

int nextJP(int currentState)
{
    int nextState;
    switch (currentState){
    case J1: nextState = JP2;       break;
    case J2: nextState = JP3;       break;
    case J3: nextState = JP4;       break;
    case J4: nextState = JP5;       break;
    case J5: nextState = JP6;       break;
    case J6: nextState = S1;        break;
    default: nextState = S1;
    }
    return nextState;
}
```

```
int generateY(int currentMotion)
{
    int y,y0,y1,y2,y3;
    y0 = 50;
    y1 = 120;
    y2 = 180;
    y3 = 210;
    switch (currentMotion){
        case J1: y = y1;            break;
        case J2: y = y2;            break;
        case J3: y = y3;            break;
        case J4: y = y2;            break;
        case J5: y = y1;            break;
        case J6: y = y0;            break;
        case JK1: y = y1;            break;
        case JK2: y = y2;            break;
        case JK3: y = y3;            break;
        case JK4: y = y2;            break;
        case JK5: y = y1;            break;
        case JK6: y = y0;            break;
        case JP1: y = y1;            break;
        case JP2: y = y2;            break;
        case JP3: y = y3;            break;
        case JP4: y = y2;            break;
        case JP5: y = y1;            break;
        case JP6: y = y0;            break;
     default : y = y0;
    }
    return y;
}

void reset_buffer9( int *buffer )
  {
   int i;
   for (i=0; i<=8 ;i++)
    {  buffer[i]=0;  }
  }

static void timer_irqhandler (void * context, alt_u32 id)
{
    IOWR_16DIRECT(IRTIMER_INST_BASE, 0, 0); // reset request
    time_count += 1;
```

```
}

//void printbuf(int *buf, int length){
//    int i;
//    for(i=0;i<length;i++)
//        printf("%d ",buf[i]);
//}
static void key_irqhandler (void * context, alt_u32 id)
{

    read_make_code (&decode_mode, &key);

      //printf( "key is%d\n",key1);
        if(decode_mode==P1_PRESS_CODE){
            currentT=time_count;

            keyBuf1[buf1_e]=key;
            timeBuf1[buf1_e]=currentT;
            buf1_e++;
        }

        else if(decode_mode==P2_PRESS_CODE){
            currentT=time_count;
            keyBuf2[buf2_e]=key;
            timeBuf2[buf2_e]=currentT;
            buf2_e++;
        }

//    printf("\nkeyBuf2 is :");
//            printbuf(keyBuf2,10);
//            printf("\ntimeBuf2 is :");
//            printbuf(timeBuf2,10);
//
}


queue_control
#include "keyboard_ctrl.h"
#include "time.h"
#include "vga_update.h"

extern int buf1_e;
extern int buf2_e;
```

```
void reset_buffer( int *time, int *keybuffer, int* bufEnd )
  {
   int i;
   for (i=0; i<=10 ;i++)
    {  keybuffer[i]=0;  time[i]=0; }
   *bufEnd = 0;
  }

void update_buffer( int *time, int *keybuffer, int timeVal, int keyVal, int* bufEnd)
  {
   int i;
   for (i=0; i<=10 ;i++)
    {  keybuffer[i]=0;  time[i]=0; }
    keybuffer[0]=keyVal;
    time[0]=timeVal;
    *bufEnd = 1;
  }


void comb_judg(int *keybuffer, int *time, int* bufEnd, int* key, int dir )
{


  if ( time[3] == 0 && time[0] != 0 && time[1] != 0 && time[2] != 0
                          && time[1]-time[0]<=2 && time[2]-time[1]<=2 && keybuffer[2]>4)   //
internal time of 3 keys are short enough
     {
        if ( ( (keybuffer[0] == 2 && keybuffer[1] == 4 && dir==0) || (keybuffer[0] == 2 &&
keybuffer[1] == 3 && dir==1) )&& ( keybuffer[2] == 5 || keybuffer[2] == 6 )  ) // determine if
it is one of the combinations (DRP)

          {
                 *key = 9;
                // printf("test!!!!!!!!!!!: %d\n", *key );

              reset_buffer(time , keybuffer, bufEnd);

printf( "DRP_flag" );
             // printf("buffer end is %d n",*bufEnd);
          }
         else if ( ((keybuffer[0] == 2 && keybuffer[1] == 3 && dir==0) || (keybuffer[0] == 2 &&
keybuffer[1] == 4 && dir==1) )
                                              && ( keybuffer[2] == 7 || keybuffer[2] == 8 ))
// determine if it is one of the combinations (DLK)
```

```
          {
                 *key = 10;

              reset_buffer(time , keybuffer, bufEnd);
              printf( "DLK_flag" );
          }
         else   // internal time is short enough while combinations of keys do not match
          {

              reset_buffer(time , keybuffer, bufEnd);
             // printf( "Not a comb" );
          }
     }
  else if ( time[4] == 0 && time[0] != 0 && time[1] != 0 && time[2] != 0 && time[3] != 0
         && time[1]-time[0]<=2 && time[2]-time[1]<=2 && time[3]-time[2]<=2 && keybuffer[3]>4)
// internal time of 4 keys are short enough
        {
            if ( ( ( keybuffer[0] == 4 && keybuffer[1] == 2 && keybuffer[2] == 3 && dir==0) ||
( keybuffer[0] == 3 && keybuffer[1] == 2 && keybuffer[2] == 4 && dir==1) )
                                          && ( keybuffer[3] == 7 || keybuffer[3] == 8 ) ) //
determine if it is one of the combinations (RDLK)

             {
                 *key = 11;

               reset_buffer(time , keybuffer, bufEnd);
               //printf( "RDLK_flag" );
             }

         else if ( ( ( keybuffer[0] == 3 && keybuffer[1] == 2 && keybuffer[2] == 4 && dir==0 )
|| ( keybuffer[0] == 4 && keybuffer[1] == 2 && keybuffer[2] == 3 && dir==1 ) )
                                               && ( keybuffer[3] == 5 || keybuffer[3] == 6 ) )
// determine if it is one of the combinations (LDRP)

            {
                 *key = 12;

               reset_buffer(time , keybuffer, bufEnd);
              // printf( "LDRP_flag" );
            }
          else  // internal time is short enough while combinations of keys do not match
          {
```

```
            reset_buffer(time , keybuffer, bufEnd);
           // printf("Not a comb" );
         }
       }
  else if ( time[0] != 0 && time[1] != 0 && time[2] != 0 && time[3] != 0 && time[4] != 0
        &&   time[1]-time[0]<=2   &&   time[2]-time[1]<=2   &&   time[3]-time[2]<=2   &&
time[4]-time[3]<=2 ) // internal time of 5 keys are short enough
      {
          if ( ( (keybuffer[0] == 2 && keybuffer[1] == 4 && keybuffer[2] == 2 && keybuffer[3]
== 4 && dir==0) || (keybuffer[0] == 2 && keybuffer[1] == 3 && keybuffer[2] == 2 && keybuffer[3]
== 3 && dir==1) )
                           &&( keybuffer[4] == 5 || keybuffer[4] == 6 ) ) // determine if it
is one of the combinations (DRDRP)
           {
              *key = 13;

            reset_buffer(time , keybuffer, bufEnd);
           //printf(" DRDRP_flag " );
           }
          else  // internal time is short enough while combinations of keys do not match
           {

            reset_buffer(time , keybuffer, bufEnd);
          //  printf("Not a comb" );
           }
      }

  // cases where  previous internal time is short enouth
  // while time between last key and current key is too long


  else if ( ( time[0] != 0 || keybuffer[0]!=0 ) && time[1] != 0 && time[2] == 0
                                  && time[1]-time[0]>2 )
      {
          if ( keybuffer[0] != 0 && keybuffer[1] != 0 && keybuffer[2] == 0 )
           {

            //reset_buffer(time , key);
            update_buffer(time , keybuffer, time[1], keybuffer[1], bufEnd);
           // printf("Not a comb" );
           }
      }
  else if ( time[3] == 0 && time[0] != 0 && time[1] != 0 && time[2] != 0
                       && time[1]-time[0]<=2 && time[2]-time[1]>2 )
```

```
       {
            if ( keybuffer[0] != 0 && keybuffer[1] != 0 && keybuffer[2] != 0 && keybuffer[3] ==
0)
             {

              //reset_buffer(time , keybuffer);
              update_buffer(time , keybuffer, time[2], keybuffer[2], bufEnd);
             //printf("Not a comb" );
             }
       }

   else if ( time[4] == 0 && time[0] != 0 && time[1] != 0 && time[2] != 0 && time[3] != 0
                          && time[1]-time[0]<=2 && time[2]-time[1]<=2 && time[3]-time[2]>2)
       {
            if ( keybuffer[0] != 0 && keybuffer[1] != 0 && keybuffer[2] != 0 && keybuffer[3] !=
0 && keybuffer[4] == 0)
             {

              //reset_buffer(time , key);
              update_buffer(time , keybuffer, time[3], keybuffer[3], bufEnd);
             // printf("Not a comb" );
             }
       }
   else if (time[0] != 0 && time[1] != 0 && time[2] != 0 && time[3] != 0 && time[4] != 0
                          && time[1]-time[0]<=2 && time[2]-time[1]<=2 && time[3]-time[2]<=2 &&
time[4]-time[3]>2)
       {
            if ( keybuffer[0] != 0 && keybuffer[1] != 0 && keybuffer[2] != 0 && keybuffer[3] !=
0 && keybuffer[4] != 0 )
             {

              //reset_buffer(time , key);
              update_buffer(time , keybuffer, time[4], keybuffer[4], bufEnd);
             // printf("Not a comb" );
             }
       }
}


#include <io.h>
#include <system.h>
#include <stdio.h>
#include "vga_update.h"
//-----------------------------------------
```

```
static extern int p1_x[2], p1_y[2], p2_x[2], p2_y[2];
static extern int p1_hp_off[2], p2_hp_off[2];
static extern int p1_center[2], p2_center[2];
static extern int p1_width[2], p2_width[2];
static extern int p1_image[2], p2_image[2];
static extern int p1_flip[2], p2_flip[2];
static extern int p1_color[2], p2_color[2];
static extern int hit1[2], hit2[2];
static extern int hit1_x[2], hit1_y[2], hit2_x[2], hit2_y[2];
static extern int gun1[2], gun2[2];
static extern int gun1_flip[2], gun2_flip[2];
static extern int gun1_x[2], gun1_y[2], gun2_x[2], gun2_y[2];
static extern int phantom1[2], phantom2[2];
static extern int effect1[2], effect2[2];
static extern int effect1_image[2], effect2_image[2];
static extern int effect1_x[2], effect1_y[2], effect2_x[2], effect2_y[2];
static extern int ds;
int vga_cnt = 0;
int second = 0;
int sixth_second = 0;
int minute = 0;

//-----------------------------------------------
void sprite_refresh()
{
//close interrupt for this cycle and clear sprites
    IOWR_END_IRQ();
    sprite_clear_all();

//update timer variables
    if(vga_cnt < 60)    vga_cnt++;
    else {vga_cnt = 0;    second++; sixth_second++;}
    second = second%60;
    sixth_second = sixth_second%10;
    if(second == 59) minute++;

//display undate main process
    if(is_fighting){
        //update player shadows
        IOWR_SET_SPRITE( p1_x[ds] - SHADOW_CENTER, SHADOW_Y, SHORT_SHADOW, 0, 0, BLACK);
        IOWR_SET_SPRITE( p2_x[ds] - SHADOW_CENTER, SHADOW_Y, SHORT_SHADOW, 1, 0, BLACK);

        //update player body images and phantoms
        if(!p1_flip[ds]){
```

```
            if(phantom1[ds])      IOWR_SET_SPRITE( p1_x[ds] - p1_center[ds] - 5, p1_y[ds],
p1_image[ds], 2, p1_flip[ds], PHANTOM);
            IOWR_SET_SPRITE( p1_x[ds] - p1_center[ds], p1_y[ds], p1_image[ds], 4, p1_flip[ds],
p1_color[ds]);
        }else{
            if(phantom1[ds])    IOWR_SET_SPRITE( p1_x[ds] - p1_width[ds] + p1_center[ds] + 5,
p1_y[ds], p1_image[ds], 2, p1_flip[ds], PHANTOM);
            IOWR_SET_SPRITE( p1_x[ds] - p1_width[ds] + p1_center[ds], p1_y[ds], p1_image[ds],
4, p1_flip[ds], p1_color[ds] );
        }
        if(!p2_flip[ds]){
            if(phantom2[ds])      IOWR_SET_SPRITE( p2_x[ds] - p2_center[ds] - 5, p2_y[ds],
p2_image[ds], 3, p2_flip[ds], PHANTOM);
            IOWR_SET_SPRITE( p2_x[ds] - p2_center[ds], p2_y[ds], p2_image[ds], 5, p2_flip[ds],
p2_color[ds] );
        }else{
            if(phantom2[ds])    IOWR_SET_SPRITE( p2_x[ds] - p2_width[ds] + p2_center[ds] + 5,
p2_y[ds], p2_image[ds], 4, p2_flip[ds], PHANTOM);
            IOWR_SET_SPRITE( p2_x[ds] - p2_width[ds] + p2_center[ds], p2_y[ds], p2_image[ds],
5, p2_flip[ds], p2_color[ds] );
        }

        //update hit effects
        if(hit1[ds])      IOWR_SET_SPRITE( hit1_x[ds] - HIT_CENTER, hit1_y[ds], HIT, 6, 0,
ORANGE);
        if(hit2[ds])      IOWR_SET_SPRITE( hit2_x[ds] - HIT_CENTER, hit2_y[ds], HIT, 7, 0,
ORANGE);

        //update energy gun bullets
        if(gun1[ds])       IOWR_SET_SPRITE( gun1_x[ds] - GUN_CENTER, gun1_y[ds], GUN, 8,
gun1_flip[ds], EFFECT1);
        if(gun2[ds])       IOWR_SET_SPRITE( gun2_x[ds] - GUN_CENTER, gun2_y[ds], GUN, 9,
gun2_flip[ds], EFFECT2);

        //update combo effects
        if(effect1[ds])       IOWR_SET_SPRITE( effect1_x[ds] - HIT_CENTER, effect1_y[ds],
effect1_image[ds], 6, 10, EFFECT1);
        if(effect2[ds])       IOWR_SET_SPRITE( effect2_x[ds] - HIT_CENTER, effect2_y[ds],
effect2_image[ds], 7, 11, EFFECT2);
    }else   game_menu(cursor_position);

}

void fight_environment(int p1_hp_off, int p2_hp_off, int time)
```

```
{
    IOWR_SET_BG_COLOR(BACKGROUND); //turn on background image
    IOWR_SET_SPRITE( 12, 10, 34, 15, 1, ORANGE);// update P1 hp boundary
    IOWR_SET_SPRITE( 12, 10, 33, 14, 1, ORANGE);// update P1 hp value
    IOWR_LEFT_CUT( 16 + p1_hp_off, 14 );         // update current P1 hp off value
    IOWR_SET_SPRITE( 380, 10, 34, 17, 0, ORANGE);// update P2 hp boundary
    IOWR_SET_SPRITE( 380, 10, 33, 16, 0, ORANGE);// update P2 hp value
    IOWR_RIGHT_CUT( 16 + p2_hp_off, 16 );        // update current P2 hp off value
    IOWR_SET_SPRITE( 273, 5, 23+(time/10), 12, 0, ORANGE);//update time second tens
    IOWR_SET_SPRITE( 316, 5, 23+(time%10), 13, 0, ORANGE);//update time second ones
    if (time == 0){
        IOWR_SET_SPRITE( 197, 150, 36, 19, 0, ORANGE);
        if(p1_hp_off > p2_hp_off)   IOWR_SET_SPRITE( 264, 220, 38, 18, 0, ORANGE);
        else    IOWR_SET_SPRITE( 264, 220, 37, 18, 0, ORANGE);
    }// show up winner words when the fight ends
}

void game_menu(int cursor_position)
{
    IOWR_SET_BG_COLOR(BLACK);
}// show game menu image

void sprite_clear_all()
{
    int i;
    for(i = 0; i<21; i++) IOWR_SET_SPRITE( 1023, 1023, 0, i, 0, 0 );
}//clear all sprite before a new cycle



#ifndef VGA_UPDATE_H_
#define VGA_UPDATE_H_
//---------basic parameters-----
#define SPEED               3

//-------color table------------
#define BACKGROUND            0
#define BODY1                       1
#define BODY2                       2
#define ORANGE                    3
#define BLACK                        4
#define PHANTOM                  5
#define EFFECT1                     6
#define EFFECT2                     7
```

```
//-------fixed position---------
#define SHADOW_Y                  100
#define SHADOW_CENTER         50
#define HIT_CENTER                    30
#define GUN_CENTER                 20

//-------state index--------------
#define W1           0
#define W2           1
#define W3           2
#define W4          3
#define W5           4
#define W6          5
#define W7          6
#define W8         7
#define W9         8
#define W10       9
//------------
#define HP1          10
#define HP2          11
#define HP3          12
#define HP4          13
//------------
#define HK1          14
#define HK2          15
#define HK3         16
//------------
#define LP1          17
#define LP2          18
//------------
#define LK1          19
#define LK2          20
//------------
#define L1             21
//------------
#define LHP1         22
#define LHP2         23
#define LHP3         24
#define LHP4         25
//------------
#define LLP1         26
#define LLP2         27
//------------
#define LHK1         28
```

```
#define LHK2         29
#define LHK3        30
//------------
#define LLK1         31
#define LLK2         32
//------------
#define J1           33
#define J2          34
#define J3           35
#define J4           36
#define J5          37
#define J6          38
//------------
#define JK1          39
#define JK2          40
#define JK3         41
#define JK4         42
#define JK5        43
#define JK6         44
//------------
#define JP1          45
#define JP2         46
#define JP3         47
#define JP4         48
#define JP5         49
#define JP6         50
//------------
#define GUN1        51
#define GUN2        52
#define GUN3         53
#define GUN4         54
#define GUN5         55
//------------
#define DASH1       56
#define DASH2       57
#define DASH3       58
#define DASH4       59
#define DASH5       60
#define DASH6       61
//------------
#define LASER1       62
#define LASER2       63
#define LASER3       64
#define LASER4       65
```

```
#define LASER5       66
#define LASER6       67
//------------
#define UP1          68
#define UP2          69
#define UP3          70
#define UP4          71
#define UP5          72
//------------
#define HG1          73
#define HG2          74
#define HG3          75
#define HG4          76
//------------
#define S1             77
//------------
#define H1             78
#define H2             79
#define H3             80
//------------
#define D1             81
#define D2             82
#define D3             83
#define D4             84
//------------
#define LD1           85
#define LD2           86
#define LD3           87
#define LD4           88
//------------
#define WIN1        89
//------------
#define LOSE1       90
//-------------
#define KD1           91
#define KD2           92
#define KD3           93
#define KD4           94

//----motion category---------------
#define LW           0
#define RW           1
#define HP            2
#define HK            3
```

```
#define LP              4
#define LK              5
#define L                6
#define LHP           7
#define LLP            8
#define LHK           9
#define LLK          10
#define J              11
#define LJ            12
#define RJ            13
#define JK            14
#define JP            15
#define GUN        16
#define DASH       17
#define LASER      18
#define UP            19
#define HG           20
#define S              21
#define H             22
#define D             23
#define LD           24
#define WIN        25
#define LOSE       26
#define KD     27
//--------------WIDTH-----------------
#define int romImage[95] = {
    18, 19, 19, 20, 20, 21, 21, 22, 22, 23,      //WALK
    3, 3, 4, 5,                          //HP
    28, 29, 29,                       //HK
    7, 7,                                //LP
    10, 10,                              //LK
    9,                                //L
    14, 14, 15, 16,                      //LHP
    12, 13,                           //LLP
    38, 38, 38,                       //LHK
    10, 11,                           //LLK
    30, 30, 32, 32, 30, 30,              //J
    11, 11, 11, 11, 11, 11,              //JK
    31, 31, 31, 31, 31, 31,              //JP
    40, 41, 42, 43, 43,                  //GUN
    39, 39, 39, 39, 39, 39,                  //DASH
    44, 44, 45, 45, 45, 45,              //LASER
    33, 34, 34, 35, 35,                  //UP
    25, 25, 26, 27,                      //HG
```

```
    17,                             //S
    24, 24, 24,                     //H
    2, 2, 2, 2,                         //D
    8, 8, 8, 8,                         //LD
    0,                              //WIN
    37,                             //LOSE
    36, 36, 36, 36                      //KD
}
//-----------------------------------------
#define int romWidth [28] =
{68, 68,  132, 136,  76,   84,  76, 112, 112, 152, 112,
 52, 52, 52, 112,  76, 104,  112, 116,  68, 100,  60,  72,
 64,   76,  44,  44,  152
 }
//--------------HEIGHT----------------
 #define int imageHeight [28] =
 {150, 150, 150, 150, 150, 150, 110, 110, 110,  88, 110,
  162, 162, 162, 110, 120, 150, 150, 150, 162, 162, 150, 150,
  150, 110, 150, 150, 37
  }
 //--------------CNETER-------------------
  #define int romCenter [28] =
{
40, 40, 50, 44, 22, 37, 40, 40, 34, 30, 40,
18, 18, 18, 40, 15, 54, 18, 43, 22, 43, 24, 43,
20, 40, 18, 18, 75
}

//-----------rom index--------------
#define SHORT_SHADOW           0
#define GUN                    0
#define HIT                    0

//-------------------------------------------------------------
static int startMotion[28] = {
0, 0, 10, 14, 17, 19, 21, 22, 26, 28,
31, 33, 33, 33, 39, 45, 51, 56, 62, 68,
73, 77, 78, 81, 85, 89, 90, 91}
//---------------------------------------------------------------
#define IOWR_SET_SPRITE(x, y, rom, sprite, flip, color) \
    IOWR_32DIRECT( SPRITE_BUF_BASE,  4*(rom  +  (sprite<<7)),  x  +  (y<<10)  +  (flip<<30)  +
(color<<20) )

#define IOWR_LEFT_CUT(x, sprite) \
```

```
    IOWR_32DIRECT( SPRITE_BUF_BASE, 4*(sprite<<7), (1<<29) + x )

#define IOWR_RIGHT_CUT(x, sprite) \
    IOWR_32DIRECT( SPRITE_BUF_BASE, 4*(sprite<<7), (1<<28) + x )

#define IOWR_SET_BG_COLOR(color) \
    IOWR_32DIRECT( SPRITE_BUF_BASE, 4*(21<<7), (color<<20) )

#define IOWR_END_IRQ() \
    IOWR_32DIRECT( SPRITE_BUF_BASE, 0, (1<<31) + 1023 + (1023<<10) )

void sprite_refresh();
void fight_environment(int p1_hp_off, int p2_hp_off, int time);
void game_menu(int cursor_position);
void sprite_clear_all();

#endif /*VGA_UPDATE_H_*/
```

VHDL FOR Audio

```
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;

-- Shangru Li
--
-- This component is basically a 26-bit counter
--
-- When the counter is 1, irq will be '1'
-- The irq is reset to '0' by the cpu, when the cpu wants to write to this
-- component, the irq will become '0'
--
-- The address, readdata, and writedata are effectively ignored

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

entity irTimer is port (
```

```
  clk, reset_n, chipselect, read, write, 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 data    : std_logic_vector(15 downto 0);
  signal counter : unsigned(21 downto 0);
begin

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        data <= (others => '0');
      else
        if chipselect = '1' then
          if address = '1' then
            if write = '1' then
              data <= writedata;
            elsif read = '1' then
              readdata <= data;
            end if;
          end if;
        end if;
      end if;
    end if;
  end process;

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        counter <= (others => '0');
      else
        counter <= counter +1 ;
      end if;
    end if;
  end process;

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

```
      if reset_n = '0' then
        irq <= '0';
      else
        if counter = 1 then
          irq <= '1';
        elsif write = '1' and chipselect = '1' then
          irq <= '0';  -- important to reset the irq
        end if;
      end if;
    end if;
  end process;

end rtl;



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

entity de2_wm8731_audio is
port (
    -- interface with CPU
    chipselect: in std_logic;
    address   : in unsigned(6 downto 0);-- higher 4 bit is sprite index, lower 6 bit is ROM number
    writedata : in unsigned(15 downto 0);-- bit 31 is IRQ switch, bit 30~16 is y value, bit 15~0
is x value
    readdata : out unsigned(15 downto 0);

    write     : in std_logic;
    read      : in std_logic;
    reset     : in std_logic;
    clk_50    : in std_logic;
    irq       : out std_logic;

    clk : in std_logic;       --  Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
    reset_n : in std_logic;
    test_mode : in std_logic;       --    Audio CODEC controller test mode
    audio_request : out std_logic;  --    Audio controller request new data
    data : in unsigned(15 downto 0);

    -- 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(12 downto 0);
    signal bclk_divider : unsigned(3 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 sin_out     : unsigned(15 downto 0);
    signal sin_counter : unsigned(6 downto 0);

    type ram_type is array (integer range 0 to 19) of unsigned(15 downto 0);
    signal audioRAM : ram_type
    :=(
"1000100010001000","1000100010001000","1000100010001000","1000100010001000",
"1000100010001000","1000100010001000","1000100010001000","1000100010001000",
"1000100010001000","1000100010001000","1000100010001000","1000100010001000",
"1000100010001000","1000100010001000","1000100010001000","1000100010001000",
"1000100010001000","1000100010001000","1000100010001000","1000100010001000"
    );

begin

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        lrck_divider <= (others => '0');
      elsif lrck_divider = "0000100010000"  then        -- "C0" minus 1
        lrck_divider <=    "0000000000000";
```

```
    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"B" or set_lrck = '1'  then
      bclk_divider <= X"0";
    else
      bclk_divider <= bclk_divider + 1;
    end if;
  end if;
end process;

  process ( lrck_divider )
  begin
      if ( lrck_divider = "00000010000") then
          set_lrck <= '1';
      else
          set_lrck <= '0';
      end if;
  end process;


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;

-- BCLK divider
set_bclk <= '1' when bclk_divider(3 downto 0) = "0101" else '0';
clr_bclk <= '1' when bclk_divider(3 downto 0) = "1011" 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 set_lrck = '1' then
      if test_mode = '1' then
        shift_out <= sin_out;
      else
        shift_out <= audioRAM(to_integer(sin_counter));
      end if;
    elsif clr_bclk = '1' then
      shift_out <= shift_out (14 downto 0) & '0';
    end if;
  end if;
end process;

  -- Audio outputs

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

  -- Self test with Sin wave

  process(clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
```

```
        sin_counter <= (others => '0');
    elsif lrck_lat = '1' and lrck = '0'  then
      if (sin_counter = 100) then
         sin_counter <= (others => '0');
         irq <='1';
      else

        if write = '1' then
         irq <= '0';  -- important to reset the irq
        end if;

        sin_counter <= sin_counter + 1;

      end if;
    end if;

  end if;
end process;

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

process(clk)
begin
  if rising_edge(clk) then
    if write='1' and chipselect='1' then
       audioRAM(to_integer(address))<=writedata;
    elsif read='1' and chipselect='1' then
       readdata <= audioRAM(to_integer(address));
    end if;
  end if;
end process;




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

```
        if lrck_lat = '1' and lrck = '0' then
          audio_request <= '1';
        else
          audio_request <= '0';
        end if;
      end if;
    end process;

end architecture;
```

The C code for The audio

```
#include <alt_types.h>


    0xfccd ,
    0xff82

};

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

#include <sys/alt_irq.h> // the irq functions
#include <alt_types.h>
#include <keyboard_ctrl.h>
#include "DM9000A.h"

#define IOWR_LED_DATA(base, offset, data) \
  IOWR_16DIRECT(base, (offset) * 2, data)
#define IORD_LED_DATA(base, offset) \
  IORD_16DIRECT(base, (offset) * 2)
#define IOWR_LED_SPEED(base, data) \
  IOWR_16DIRECT(base + 32, 0, data)


#define bufSize 150000

FILE * f;

void printbuf(int *buf, int length);
```

```
KB_CODE_TYPE decode_mode;
//alt_u8 key = 0;
int key = 0;
int time_count = 0;

int audio_irq_count=0;

int offset = -100;

int currentT;

//extern int audio;

int keyBuf1[10] ={0,0,0,0,0,0,0,0,0,0};
int timeBuf1[10]={0,0,0,0,0,0,0,0,0,0};
int buf1_s=0, buf1_e=0;

int keyBuf2[10] ={0,0,0,0,0,0,0,0,0,0};
int timeBuf2[10]={0,0,0,0,0,0,0,0,0,0};
int buf2_s=0, buf2_e=0;

extern alt_u16 audio [1192];

#define MAX_MSG_LENGTH 128

// Ethernet MAC address.  Choose the last three bytes yourself
unsigned char mac_address[6] = { 0x01, 0x60, 0x6E, 0x11, 0x02, 0x0F  };

unsigned int receive_buffer_length;
unsigned char receive_buffer[1600];

#define UDP_PACKET_PAYLOAD_OFFSET 42
#define UDP_PACKET_LENGTH_OFFSET 38

#define UDP_PACKET_PAYLOAD (transmit_buffer + UDP_PACKET_PAYLOAD_OFFSET)
#define UDP_RECEIVE_PAYLOAD (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)

unsigned char transmit_buffer[] = {
  // Ethernet MAC header
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination MAC address
  0x01, 0x60, 0x6E, 0x11, 0x02, 0x0F, // Source MAC address
  0x08, 0x00,                         // Packet Type: 0x800 = IP
```

```
  // IP Header
  0x45,                 // version (IPv4), header length = 20 bytes
  0x00,                 // differentiated services field
  0x00,0x9C,            // total length: 20 bytes for IP header +
                        // 8 bytes for UDP header + 128 bytes for payload
  0x3d, 0x35,           // packet ID
  0x00,                 // flags
  0x00,                 // fragment offset
  0x80,                 // time-to-live
  0x11,                 // protocol: 11 = UDP
  0xa3,0x43,            // header checksum: incorrect
  0xc0,0xa8,0x01,0x01,  // source IP address
  0xc0,0xa8,0x01,0xff,  // destination IP address

  // UDP Header
  0x67,0xd9, // source port port (26585: garbage)
  0x27,0x2b, // destination port (10027: garbage)
  0x00,0x88, // length (136: 8 for UDP header + 128 for data)
  0x00,0x00, // checksum: 0 = none

  // UDP payload
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67
};

static void ethernet_interrupt_handler() {
  unsigned int receive_status;

  receive_status = ReceivePacket(receive_buffer, &receive_buffer_length);
```

```
  if (receive_status == DMFE_SUCCESS) {
    //printf("\nPacket received is 0x%.2X",UDP_RECEIVE_PAYLOAD[0]);
  }

  dm9000a_iow(ISR, 0x3F);
  dm9000a_iow(IMR, INTR_set);
}


static void audio_irqhandler (void * context, alt_u32 id)
{
    //printf("\n Audio irq!\n");
    IOWR_16DIRECT(AUDIO_INST_BASE, 0, 0); // reset request
    offset += 20;
    audio_irq_count+=1;
    int i;
    for (i=0;i<20;i++){
        if((offset+i)>bufSize)
            offset-=bufSize;
            IOWR_16DIRECT(AUDIO_INST_BASE, i*2, 2);
            f=fopen("data.txt",'at');
  if(audio_irq_count==20){
    for (i=0;i<20;i++){
        int d=IORD_16DIRECT(AUDIO_INST_BASE, i*2);
        printf("%d",d);
        printf("a\n");
    }
  }

}

static void timer_irqhandler (void * context, alt_u32 id)
{
    IOWR_16DIRECT(IRTIMER_INST_BASE, 0, 0); // reset request
    time_count += 1;
    //printf("\n Audio irq %d!\n",audio_irq_count);
}

static void key_irqhandler (void * context, alt_u32 id)
{
    read_make_code (&decode_mode, &key);

    if(decode_mode==P1_RELEASE_CODE || decode_mode==P2_RELEASE_CODE)
        key=key+10;
```

```
    //printf("Key code is %d\n",key);

    if(decode_mode==P1_PRESS_CODE || decode_mode==P1_RELEASE_CODE){
        currentT=time_count;

        keyBuf1[buf1_e]=key;
        timeBuf1[buf1_e]=currentT;
        buf1_e++;
    }

    if(decode_mode==P2_PRESS_CODE || decode_mode==P2_RELEASE_CODE){
        currentT=time_count;

        keyBuf2[buf2_e]=key;
        timeBuf2[buf2_e]=currentT;
        buf2_e++;
    }

    UDP_PACKET_PAYLOAD[0]=key >> 8;
    UDP_PACKET_PAYLOAD[1]=key & 0xff;
    TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET+2);


        //printf("\nkeyBuf1 is :");
        //printbuf(keyBuf1,10);
        //printf("\ntimeBuf1 is :");
        //printbuf(timeBuf1,10);

        //printf("\nkeyBuf2 is :");
        //printbuf(keyBuf2,10);
        //printf("\ntimeBuf2 is :");
        //printbuf(timeBuf2,10);

        //printf("\nHoldBuf1 is :");
        //printbuf(P1_holdBuf,8);
        //printf("\nHoldBuf2 is :");
        //printbuf(P2_holdBuf,8);
}
/*
void printbuf(int *buf, int length){
    int i;
    for(i=0;i<length;i++)
        printf("%d ",buf[i]);
```

```
}
*/

int main()
{
  //int i;
  printf("Hello Michael\n");

  // Initalize the DM9000 and the Ethernet interrupt handler
  DM9000_init(mac_address);
  alt_irq_register(DM9000A_IRQ, NULL, (void*)ethernet_interrupt_handler);
  alt_irq_register( DE2_PS2_INST_IRQ, NULL,( void*)key_irqhandler ); // register the irq
  alt_irq_register( IRTIMER_INST_IRQ, NULL,( void*)timer_irqhandler );
  alt_irq_register(   AUDIO_INST_IRQ, NULL,( void*)audio_irqhandler );

  while(1){
    //printf("Goodbye!\n");
  }
  /* Get received byte */

  printf("Goodbye!\n");

  return 0;
}


CODE FOR PROJECT SDRAM
#include <stdio.h>
#include <system.h>
#include <io.h>
#include <bg.h>
#include <DEFENCE.h>
#include <HEAVY_P1.h>
#include <HEAVY_P2.h>
#include <HEAVY_P3.h>
#include <LIGHT_K.h>
#include <LIGHT_P.h>
#include <LOW1.h>
#include <LOW2.h>
#include <LOWK1.h>
#include <LOWK2.h>
#include <LOWP1.h>
#include <LOWP2.h>
#include <LOWP3.h>
```

```
#include <LOWP4.h>
#include <LOWP5.h>
#include <STANDBY.h>
#include <WALK1.h>
#include <WALK2.h>
#include <WALK3.h>
#include <WALK4.h>
#include <WALK5.h>
#include <WALK6.h>
#include <WOUND.h>

#include <down_p1.h>
#include <down_p2.h>
#include <down_p3.h>
#include <high_k1.h>
#include <high_k2.h>
#include <jump.h>
#include <jump_down_p.h>
#include <jump_low.h>
#include <jump_p1.h>
#include <jump_p2.h>
#include <jump_p3.h>
#include <knock_down.h>
#include <loss.h>
#include <low_heavy_k.h>
#include <rush.h>
#include <wave1_1.h>
#include <wave1_2.h>
#include <wave1_3.h>
#include <wave1_4.h>
#include <wave2_1.h>
#include <wave2_2.h>

#include <d0.h>
#include <d1.h>
#include <d2.h>
#include <d3.h>
#include <d4.h>
#include <d5.h>
#include <d6.h>
#include <d7.h>
#include <d8.h>
#include <d9.h>
#include <hp.h>
```

```
#include <hp_b.h>
#include <winner.h>
#include <p1.h>
#include <p2.h>
#include <inf.h>

#include <ball.h>
#include <firegun1.h>
#include <firegun2.h>
#include <laser.h>
#include <pg1.h>
#include <pg2.h>
#include <pg3.h>
#include <up.h>
#include <shadow.h>

#define IOWR_SRAM_DATA(base, offset, data) \
IOWR_16DIRECT(base, (offset) * 2, data)

int main()
{
    int i, offset;
    offset = 0;

    for (i = offset; i<76800+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, BG[i-offset]);
    offset = i;   printf("BG: %d\n",offset);  //Print the intial address for BG

    for (i = offset; i<2400+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, DEFENCE[i-offset]);
    offset = i;   printf("DEFENCE: %d\n",offset);  //Print the intial address for DEFENCE

    for (i = offset; i<4950+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, HEAVY_P1[i-offset]);
    offset = i;   printf("HEAVY_P1: %d\n",offset);  //Print the intial address for HEAVY_P1

    for (i = offset; i<4950+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, HEAVY_P2[i-offset]);
    offset = i;   printf("HEAVY_P2: %d\n",offset);  //Print the intial address for HEAVY_P2

    for (i = offset; i<4950+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, HEAVY_P3[i-offset]);
    offset = i;   printf("HEAVY_P3: %d\n",offset);  //Print the intial address for HEAVY_P3

    for (i = offset; i<3150+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LIGHT_K[i-offset]);
    offset = i;   printf("LIGHT_K: %d\n",offset);  //Print the intial address for LIGHT_K

    for (i = offset; i<2850+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LIGHT_P[i-offset]);
    offset = i;   printf("LIGHT_P: %d\n",offset);  //Print the intial address for LIGHT_P
```

```
for (i = offset; i<2090+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOW1[i-offset]);
offset = i;   printf("LOW1: %d\n",offset);  //Print the intial address for LOW1

for (i = offset; i<2090+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOW2[i-offset]);
offset = i;   printf("LOW2: %d\n",offset);  //Print the intial address for LOW2

for (i = offset; i<3080+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOWK1[i-offset]);
offset = i;   printf("LOWK1: %d\n",offset);  //Print the intial address for LOWK1

for (i = offset; i<3080+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOWK2[i-offset]);
offset = i;   printf("LOWK2: %d\n",offset);  //Print the intial address for LOWK2

for (i = offset; i<3080+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOWP1[i-offset]);
offset = i;   printf("LOWP1: %d\n",offset);  //Print the intial address for LOWP1

for (i = offset; i<3080+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOWP2[i-offset]);
offset = i;   printf("LOWP2: %d\n",offset);  //Print the intial address for LOWP2

for (i = offset; i<3080+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOWP3[i-offset]);
offset = i;   printf("LOWP3: %d\n",offset);  //Print the intial address for LOWP3

for (i = offset; i<3080+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOWP4[i-offset]);
offset = i;   printf("LOWP4: %d\n",offset);  //Print the intial address for LOWP4

for (i = offset; i<3080+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, LOWP5[i-offset]);
offset = i;   printf("LOWP5: %d\n",offset);  //Print the intial address for LOWP5

for (i = offset; i<2250+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, STANDBY[i-offset]);
offset = i;   printf("STANDBY: %d\n",offset);  //Print the intial address for STANDBY

for (i = offset; i<2550+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, WALK1[i-offset]);
offset = i;   printf("WALK1: %d\n",offset);  //Print the intial address for WALK1

for (i = offset; i<2550+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, WALK2[i-offset]);
offset = i;   printf("WALK2: %d\n",offset);  //Print the intial address for WALK2

for (i = offset; i<2550+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, WALK3[i-offset]);
offset = i;   printf("WALK3: %d\n",offset);  //Print the intial address for WALK3

for (i = offset; i<2550+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, WALK4[i-offset]);
offset = i;   printf("WALK4: %d\n",offset);  //Print the intial address for WALK4

for (i = offset; i<2550+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, WALK5[i-offset]);
```

```
    offset = i;   printf("WALK5: %d\n",offset);  //Print the intial address for WALK5

    for (i = offset; i<2550+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, WALK6[i-offset]);
    offset = i;   printf("WALK6: %d\n",offset);  //Print the intial address for WALK6

    for (i = offset; i<2700+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, WOUND[i-offset]);
    offset = i;   printf("WOUND: %d\n",offset);  //Print the intial address for WOUND

    for (i = offset; i<4050+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, down_p1[i-offset]);
    offset = i;   printf("DOWN_P1: %d\n",offset);  //Print the intial address for DOWN_P1

    for (i = offset; i<4050+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, down_p2[i-offset]);
    offset = i;   printf("DOWN_P2: %d\n",offset);  //Print the intial address for DOWN_P2

    for (i = offset; i<4050+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, down_p3[i-offset]);
    offset = i;   printf("DOWN_P3: %d\n",offset);  //Print the intial address for DOWN_P3

    for (i = offset; i<5100+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, high_k1[i-offset]);
    offset = i;   printf("HIGH_K1: %d\n",offset);  //Print the intial address for HIGH_K1

    for (i = offset; i<5100+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, high_k2[i-offset]);
    offset = i;   printf("HIGH_K2: %d\n",offset);  //Print the intial address for HIGH_K2

    for (i = offset; i<2106+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, jump[i-offset]);
    offset = i;   printf("JUMP: %d\n",offset);  //Print the intial address for JUMP

    for (i = offset; i<2280+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, jump_down_p[i-offset]);
    offset  =  i;     printf("JUMP_DOWN_P:  %d\n",offset);   //Print  the  intial  address  for
JUMP_DOWN_P

    for (i = offset; i<1520+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, jump_low[i-offset]);
    offset = i;   printf("JUMP_LOW: %d\n",offset);  //Print the intial address for JUMP_LOW

    for (i = offset; i<2754+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, jump_p1[i-offset]);
    offset = i;   printf("jump_p1: %d\n",offset);  //Print the intial address for jump_p1

    for (i = offset; i<2754+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, jump_p2[i-offset]);
    offset = i;   printf("jump_p2: %d\n",offset);  //Print the intial address for jump_p2

    for (i = offset; i<2754+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, jump_p3[i-offset]);
    offset = i;   printf("jump_p3: %d\n",offset);  //Print the intial address for jump_p3

    for (i = offset; i<1406+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, knock_down[i-offset]);
    offset = i;   printf("knock_down: %d\n",offset);  //Print the intial address for knock_down
```

```
    for (i = offset; i<1650+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, loss[i-offset]);
    offset = i;   printf("loss: %d\n",offset);  //Print the intial address for loss

    for (i = offset; i<3344+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, low_heavy_k[i-offset]);
    offset = i;     printf("low_heavy_k: %d\n",offset);   //Print the intial address for
low_heavy_k

    for (i = offset; i<4200+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, rush[i-offset]);
    offset = i;   printf("rush: %d\n",offset);  //Print the intial address for rush

    for (i = offset; i<3900+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, wave1_1[i-offset]);
    offset = i;   printf("wave1_1: %d\n",offset);  //Print the intial address for wave1_1

    for (i = offset; i<3900+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, wave1_2[i-offset]);
    offset = i;   printf("wave1_2: %d\n",offset);  //Print the intial address for wave1_2

    for (i = offset; i<3900+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, wave1_3[i-offset]);
    offset = i;   printf("wave1_3: %d\n",offset);  //Print the intial address for wave1_3

    for (i = offset; i<3900+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, wave1_4[i-offset]);
    offset = i;   printf("wave1_4: %d\n",offset);  //Print the intial address for wave1_4

    for (i = offset; i<4350+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, wave2_1[i-offset]);
    offset = i;   printf("wave2_1: %d\n",offset);  //Print the intial address for wave2_1

    for (i = offset; i<4350+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, wave2_2[i-offset]);
    offset = i;   printf("wave2_2: %d\n",offset);  //Print the intial address for wave2_2



    for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d0[i-offset]);
    offset = i;   printf("d0: %d\n",offset);  //Print the intial address for d0

    for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d1[i-offset]);
    offset = i;   printf("d1: %d\n",offset);  //Print the intial address for d1

    for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d2[i-offset]);
    offset = i;   printf("d2: %d\n",offset);  //Print the intial address for d2

    for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d3[i-offset]);
    offset = i;   printf("d3: %d\n",offset);  //Print the intial address for d3

    for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d4[i-offset]);
```

```
offset = i;   printf("d4: %d\n",offset);  //Print the intial address for d4

for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d5[i-offset]);
offset = i;   printf("d5: %d\n",offset);  //Print the intial address for d5

for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d6[i-offset]);
offset = i;   printf("d6: %d\n",offset);  //Print the intial address for d6

for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d7[i-offset]);
offset = i;   printf("d7: %d\n",offset);  //Print the intial address for d7

for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d8[i-offset]);
offset = i;   printf("d8: %d\n",offset);  //Print the intial address for d8

for (i = offset; i<624+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, d9[i-offset]);
offset = i;   printf("d9: %d\n",offset);  //Print the intial address for d9

for (i = offset; i<3410+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, hp[i-offset]);
offset = i;   printf("hp: %d\n",offset);  //Print the intial address for hp

for (i = offset; i<3410+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, hp_b[i-offset]);
offset = i;   printf("hp_b: %d\n",offset);  //Print the intial address for hp_b

for (i = offset; i<312+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, inf[i-offset]);
offset = i;   printf("inf: %d\n",offset);  //Print the intial address for inf

for (i = offset; i<4464+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, winner[i-offset]);
offset = i;   printf("winner: %d\n",offset);  //Print the intial address for winner

for (i = offset; i<2016+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, p1[i-offset]);
offset = i;   printf("p1: %d\n",offset);  //Print the intial address for p1

for (i = offset; i<2016+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, p2[i-offset]);
offset = i;   printf("p2: %d\n",offset);  //Print the intial address for p2

for (i = offset; i<320+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, ball[i-offset]);
offset = i;   printf("hitball: %d\n",offset);  //Print the intial address for hitball

for (i = offset; i<1037+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, firegun1[i-offset]);
offset = i;   printf("bullet1: %d\n",offset);  //Print the intial address for bullet1

for (i = offset; i<1037+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, firegun2[i-offset]);
offset = i;   printf("bullet2: %d\n",offset);  //Print the intial address for bullet2
```

```
    for (i = offset; i<2880+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, laser[i-offset]);
    offset = i;   printf("laser: %d\n",offset);  //Print the intial address for laser

    for (i = offset; i<6220+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, pg1[i-offset]);
    offset = i;   printf("pg_fire1: %d\n",offset);  //Print the intial address for pg_fire1

    for (i = offset; i<6220+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, pg2[i-offset]);
    offset = i;   printf("pg_fire2: %d\n",offset);  //Print the intial address for pg_fire2

    for (i = offset; i<476+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, shadow[i-offset]);
    offset = i;   printf("shadow: %d\n",offset);  //Print the intial address for shadow

    for (i = offset; i<576+offset; i++)  IOWR_SRAM_DATA(SRAM_BASE, i, up[i-offset]);
    offset = i;   printf("up: %d\n",offset);  //Print the intial address for up

  return 0;
}
```