Save Edwards

CSEE 4840 Embedded System Design

WEI WEI WW2313

ZEYANG YANG ZY2171

ZHE CAO ZC2237

GE ZHAO GZ2196

May 16th, 2013

Contents

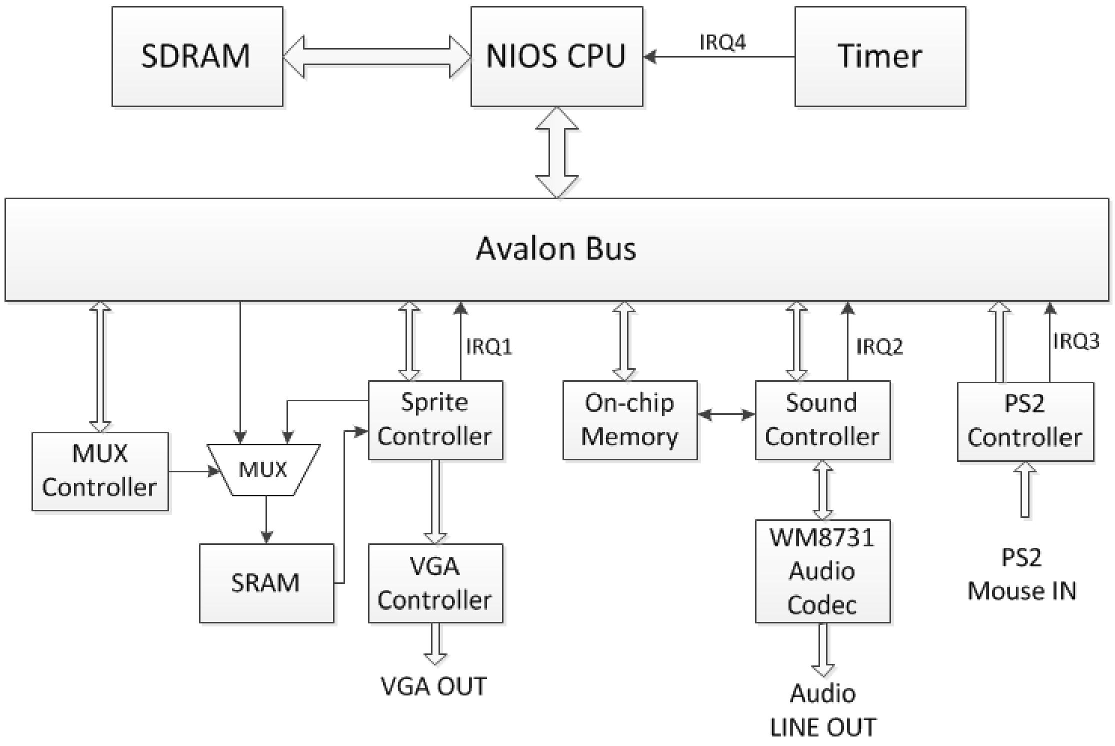
Overview

In this project, we intend to implement a tower defense video game. Basically, the player needs to set up defense facilities beside the path, in order to kill the attackers before they get through the path to the destination. Any attackers who survive from our towers and arrive at the destination will do certain amount of damage to our boss, prof. Edwards, and the game will be over if our professor runs out of health points. Otherwise, if the professor is still alive when all the attackers are killed by the towers, you win and can head to next scene! We will have several facilities with different features which need to be selected each time you build a tower, and of course it has different price. We have a unique money earning and paying system which could better control the speed and the difficulty of this game. What's more, the defense facilities could be upgraded to have a stronger power.

There are some key points of this project with which we need to handle carefully. First is the VGA display, since there will be lots of different figures, special effects and complex background, we need to use proper methods to implement the dynamic and complicated display of the game. Second is the algorithm issue. For the attackers, we need to set them walking along the designed path and think of the different behavior after being attacked by different defense facilities. For the defense facilities, we need to design different characteristics according to different types. And how the tower judges whether there are any attackers in its attack range and decide which one to attack need to be considered carefully as well. The third one is the operation of mouse, which was briefly mentioned in lab2 and need to be revised for this specific project. The last key point is how we deal with the sound effect using the FPGA. We have been in touch with the audio problem in lab3 already, and in this project we need to play some certain sound waveform in specific cases.

A high-level hardware structure

Our hardware structure is as below:



Four major hardware parts of the design: Display Module including sprite controller and VGA controller, Data Storage module including SDRAM for system memory, SRAM for image storage and on-chip RAM for audio storage, PS/2 Mouse Controller and Audio controller. These parts will be discussed in detail below.

Image preparation

All image resources of our project are extracted from the creeps HD game. In order to use these images in our project, we first did following settings of images.

Image category and size

Background images are resized to 640 x 480 using Photoshop. The image contains a path for monsters to go, blank areas to build towers, obstacles that cannot build towers, and a prompt bar with money sign, wave sign, score sign, speed button, menu button and play/pause button.

Tower images are resized to 32 x 32. Because towers can attack monsters 360^0 around, we implemented green tower and glue towers with different angles every 15^0 which will display continuous rotation to our eyes. These two towers images are saved in 9 angles from 5^0 to 85^0. Images can be flipped or mirrored in Sprite Controller, which let image rotate in all phases and can save lots of memory. Each angle has 3 images to display attacking animation. The glue tower also contains 3 slow effect images to display slow effect animation. The other two towers have are fixed in angle but have bullets rotating every 60^0. All these towers have effects of 3 images in animation. The detailed list is shown in the table below.

Each map has 5 types of monsters. 3 are smaller monsters of size 32 x 32, which are normal, fast and slow monsters. 2 larger monsters of size 32 x 40 are bosses. Each monster has 3 images displaying consequently to form animation.

Buttons of play/pause, speed, mute and unmute, upgrade and sell tower, max sign, attack range, quit and restart are drawn using Photoshop in size of 32 x 32.

Numbers are generated in size of 8 x 16 for displaying money, score, wave and lives.

Color space

VGA controller in the DE2 board uses 30 bits to represent RGB values, which requires too large memory. To save memory, one option is to use fewer bits for RGB values, like 16-bit RGB. We adopted another method, using indexed color. We use an 8-bit Color Look Up Table (CLUT) which supports 256 colors. The CLUT we use is shown in the figure below. Since we need transparent pixels in Sprite images, we define the last index 255 or 0xFF, which is white, as transparent pixel. Once the find the color of 0xFF, it will not be displayed in the Sprite Controller. If we want to display white pixels, we should manually draw the pixel to index 246 or 0xF6, which is milky white but cannot be distinguished from white on the lab screen. All images described above are transformed to 8-bit indexed color using Photoshop. Then use Matlab to convert these images into files that can be used in header files. The number images can change color in Sprite Controller by substituting non-transparent pixels to other color indices.

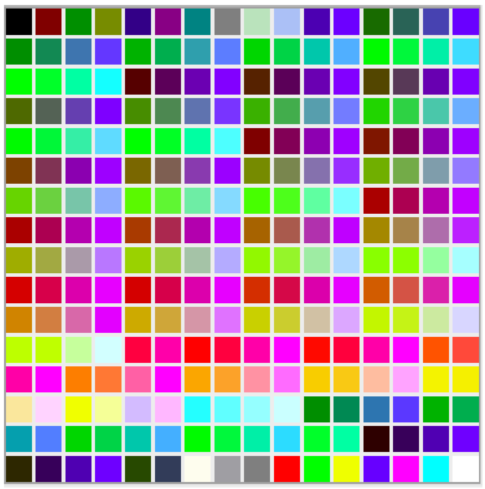


Image Category	file name	# of images	Single image size (pixels)	Total Size (KB)
Background	background1	1	640 x 480	300
	background2	1	640 x 480	300
Monsters	appear effect	3	32 x 32	3
Monsters For Background 1	normal	3	32 x 32	3
	fast	3	32 x 32	3
	slow	3	32 x 32	3
	boss1	3	32 x 40	3.012
	boss2	3	32 x 40	3.012
Monsters For Background 2	normal	3	32 x 32	3
	fast	3	32 x 32	3
	slow	3	32 x 32	3
	boss1	3	32 x 40	3.012
	boss2	3	32 x 40	3.012
Green Tower	big bullet	1	32 x 32	1
	small bullet	1	32 x 32	1
	hit effect	3	32 x 32	3
	level1 tower	9 x 3 = 27	32 x 32	27
	level2 tower	9 x 3 = 27	32 x 32	27
Glue Tower	bullet	3	32 x 32	3
	slow down effect	3	32 x 32	3
	level1 tower	9 x 3 = 27	32 x 32	27
	level2 tower	9 x 3 = 27	32 x 32	27
Bomb Tower	bomb effect	4	32 x 32	4
	level 1 tower	3	32 x 32	3
	level 1 bullet	6	32 x 32	6
	level 2 tower	3	32 x 32	3
	level 2 bullet	6	32 x 32	6

Dart Tower	level 1 tower	3	32 x 32	3
	level 1 bullet	6	32 x 32	6
	level 2 tower	3	32 x 32	3
	level 2 bullet	6	32 x 32	6
Buttons	Tower selection	1	32 x 128	
	attack range	1	32 x 32	1
	play	1	32 x 32	1
	pause	1	32 x 32	1
	speed 1	1	32 x 32	1
	speed 2	1	32 x 32	1
	mute	1	32 x 32	1
	unmute	1	32 x 32	1
	upgrade	2	32 x 32	2
	max	1	32 x 32	1
	sell	3	32 x 32	3
	restart & quit	1	64 x 32	2
Numbers	0 to 9	10	8 x 16	1.25
TOTAL		111		812.30

Audio preparing

Our project will play sound effects for tower attacks, monsters roaring and Edwards' screaming. Sounds playing at the same time can overlap. In double speed mode, sound effect will also speed up by playing the first half of the samples.

The source audios are extracted from a tower-defend game *Carrot Fantasy*. To ensure real-time overlapping multiple sounds (may up to a dozen), we use on-chip memory to store sound waves. Therefore, the memory is limited and we have compress sounds.

Sound size is related to sampling rate and quantization level. We finally choose to down-sample wave to 8k with 16-bit quantization. When we sample it lower than 8k, the effect will not be identified. When we quantize it with 8-bit, the noise is too large due to quantization error. Even waves of 16k and 8-bit have worse quality by hearing than 8k and 16-bit.

There are three simple overlapping algorithms: First is to add multiple sounds directly. This is consistent with sound overlap in reality, but in digital world, it may overflow and the sound will be distorted. The second is to calculate the average of multiple sounds. This can avoid overflow but is not consistent with the reality. When one sound is silent and overlapped with another sound, their average will be quieter than the second sound itself, which is unrealistic. The third method considers all these conditions. To overlap A and B, the result is A + B –AB. But this can only apply when samples are all positive. The wave playing in the codec have positive and negative samples and make no sense with this algorithm.

We tried both the first two algorithm while testing our project. At last, we choose the first one – summing up all concurrent audio samples directly. To avoid overflow, we lower the amplitude of all sound waves under 0.1 of the maximum amplitude. Now 16-bit quantization is important to guarantee quality of small amplitude. While playing the game, the total number of concurrent audio seldom exceed 10. Therefore, this overlapping algorithm works well.

We have four sounds for four towers, one sound for monsters and one sound for Edwards' screaming. The sound lengths and sizes are shown below.

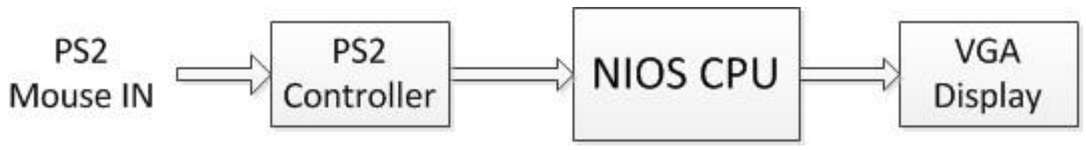
Sound name	Length (s)	Samples	Size (KB)
Green tower	0.192	1536	3.072
Glue tower	0.128	1024	2.048
Bomb tower	0.544	4352	8.704
Dart tower	0.544	4352	8.704
Monster	0.608	4864	9.728
Edwards' screaming	0.32	2560	5.12
total	2.336	18688	37.376

Input control: PS2 Mouse

Conventionally, the human-machine interface of embedded system is based on text menu mode keyboard operation which is inconvenient to control a tower defense game. PS2 mouse is used as the user input in our video game allowing us design a better graphic user interface.

Basic implementation

Once reset the FPGA, a reset command is send to the mouse by host and the host then judge the returned acknowledge signal. If correct, the host successfully initializes the mouse and begins to receive the data packet sending from mouse. The data packet includes information of the state of left button, middle button and right button and the X,Y coordinate movement. The initialization of mouse and the mouse data transfer and processing is done in hardware. The calculated mouse position corresponding to the screen and the mouse button status will be sent to the Nios processor.



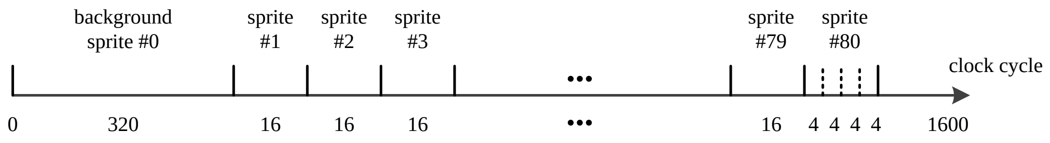
The exact position of mouse is calculated using its X,Y coordinate movement value and the calculated position is displayed as a hand icon by VGA in real time. The minimum and maximum boundary for X coordinate is 0 and 624. The minimum and maximum boundary for Y coordinate is 0 and 464. Note the maximum boundary for both coordinate is set to be a little bit smaller than the VGA display boundary in order to show at least half of the mouse icon and provide necessary visual feedback. The current position of the mouse can be updated on VGA by generating interrupt each time a movement of mouse is detected, however this way may occupies larger resources. It would be interesting to find out a way to display the mouse icon in real time without losing performance. Considering the display speed is limited. It would be unnecessary to update mouse position that cannot be captured by VGA display, but the position should be updated each time the whole scene that will be displayed by VGA is generated. The way in our design is that the mouse doesn’t generate interrupt when a movement is detected. Instead, the mouse position is updated once a VGA interrupt occurs.

The left button click and release event must be captured and updated in time. In order to get those statuses in real time, an interrupt is used to help updating mouse button status without losing any event. The interrupt signal is asserted once the left button or right button is pressed or released. Once a mouse button event occurs and the interrupt is generated, the mouse position is first analyzed and the function corresponding to the status of the mouse button and the position of the mouse is then triggered.

Sprite controller and VGA display

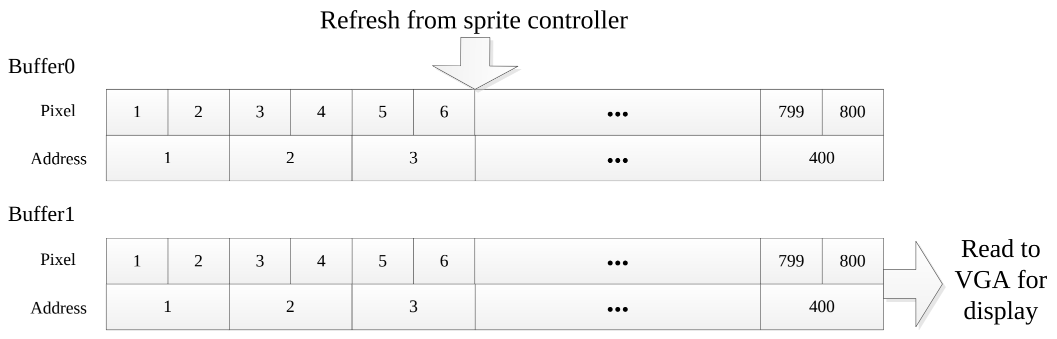
Since our tower defense video game has up to 10 monsters and up to 10 towers, who have their individual characteristics and motions, and they must be able to appear in the game at the same time. What's more, we have tens of other elements, such as the live bar, the numbers, the menu toolbar and etc. If we choose to deal with the image data directly in the VGA module, there will be exponentially increasing pain as we increase the number of the sprites. Hence, we decide to design a sprite controller, to build a platform, with which we can easily handle different sprites with different color information, position information, flip information and so on.

Obviously, it is impractical to refresh all the pixel information sprite by sprite. For two reasons: first, we need to have a huge buffer to save the pixel information of the entire screen; second, it will take a very long time after we refresh all the sprites, and it will be further longer if the number of sprites increases, which will degrade the performance of VGA display significantly. Hence the main design idea of our sprite controller and VGA controller is to multiplex the clock cycle so that we can calculate the pixel information and display onto the screen "at the same time" every row. To explain in detail, when display one row on the VGA under 25MHz frequency, we need 800 clock cycles, meanwhile we have 1600 clock cycles under 50MHz in the sprite control module. So that we can do the computation at both sides at the same time to enhance the efficiency and performance. And the 1600 clock cycles are distributed as follows:

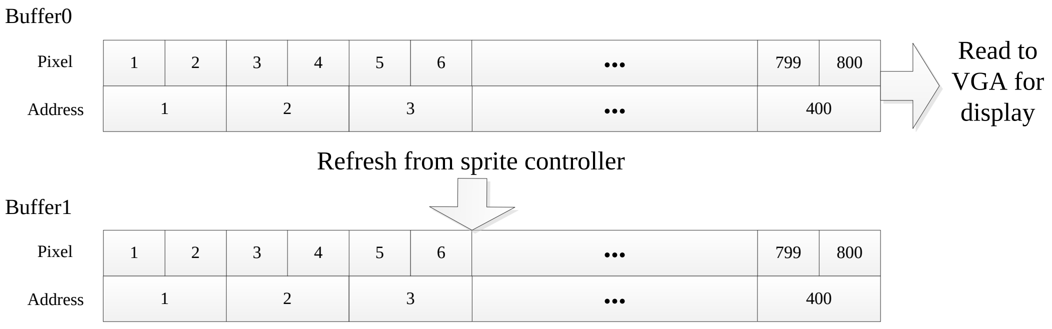


As the figure shows, at the first 320 clock cycles of 1600 cycles, the sprite controller computes the first row of background. And every 16 cycles after the first 320 cycles, the sprite controller computes the first row of corresponding sprites and if it should be shown, overwrites the previous sprites.

Since we cannot actually compute and display the same row at the same time for the reason that we have to sweep all the sprites before we display the corresponding row to guarantee the correctness. What we do is to have two sets of buffers and alternatively write to and read from them. The follow figure could explain the mechanism:
When clock cycle counter is even,



When clock cycle counter is odd,



By using this method, we are actually refreshing the pixel information of current row and display previous row. The process of computing in sprite controller and displaying in VGA is at the same time, yet on different buffers, so that there will not be data contention.

Sprite controller

In our design, we have 88 sprites in total, here is a list of the sprites:

Sprite category	Amount	Sprite category	Amount
Mouse pointer	1	Heath bar	13
Selects	2	Towers	13
Buttons	9	Bullets	13
Monsters	13	Numbers	11
Glue Effects	13	TOTAL	88

Since we do not have enough clock cycle to wait for the finish of the previous pixel computation to start the next one, we decide to use a pipeline to achieve the required throughput. Every stage does different judge and combinational logic. In the first stage, the sprite controller get the current sprite information such as x position, y position, width, height, flip signal, mirror signal and so on. Based on the above information, the sprite controller compute the corresponding SRAM offset address of the current pixel and send the address and read enable signal to the SRAM. In third stage, sprite controller gets the data from SRAM and computes the address of the buffer to write in. According to the parity of the clock cycle counter, the sprite controller rearranges the sequence of the 16-bit data to be written to the correct address in the buffer in stage #4. In the last stage, sprite controller writes the computed pixel information into the buffer and outputs the pixel information from the corresponding address in another buffer.

By rearranging the sequence of the pixel information or the offset address, our sprite controller has many functions. We are able to flip the sprite, to display the sprite at any position, to decide which sprite to show and which not. Furthermore, as we designed, if the color of the pixel is pure white, it would be recognized as transparent so that the multiple sprites can be overlapped perfect.

VGA Display

Thanks to the sprite controller, the VGA display module saves a lot of work. What it needs to do is to get the pixel information from sprit controller, which has been dealt with already, instead of from the SRAM, and display every pixel on the screen. To save the memory, we use 8-bit 256 color display strategy, so that the input pixel information is 8-bit. To display successfully, we write a color look-up table in the VGA display module to convert 8-bit pixel information to 24-bit RGB pixel information. The rests are the same as an ordinary VGA display module.

Audio

Since we have 10 monsters and 10 towers who can generate different sound effects, and most time there will be many monsters and towers on the ground at the same time. And the sound effect does keep for a period of time, which means there is a great chance that there are multiple objects generating different sound effects at the same time. Simply throwing all the sound data into the WM8731 codec will definitely mess all the things up. So we design an audio controller module to pre-compute the sound data, and then send the final value to the codec.

The algorithm we follow in merging sound is simple yet effective, which is add all the data up and divide the sum by the number of the data. Thanks to the successful experience in implementing the sprite controller, we found that the sound control is somehow like the sprite control, since each piece of sound effect can be considered as a "sprite" and to add all the sound up is just to iterate all the "sprites".

Actually, there is some difference between sound control and sprite control. In sprite control, every time we just need to put the new data into the buffer and replace the old one, so pipeline will be more effective. However, in sound control, we need to first fetch the old data from the buffer and do the addition or the division, and then put the result back to the same buffer. As assigning the same variable at different stages is a little complicated in pipeline and we have enough time for us to do the computation on the sound data, we choose to use a finite state machine (FSM) instead of pipeline in our audio module. Second difference in our audio part is the enable and disable of the sound effect. In our design, we employ the safest way to control the enable signal – software enables and hardware disables. Every time the enable signal in software flips, the hardware considers it as an enable signal so that we solve the communication problem between software and hardware.

The sound data is at 8kHz sample rate and stored in on-chip memory. We mean to generate an interrupt request signal to CPU every 256 sound data. So we need a cumulative counter to indicate the offset of the current data. The FSM has 22 states, first state is initial state and last state is hold state, the rest 20 states correspond to 10 monsters and 10 towers. After running 256 cycles in each state except the first state and the last one, the FSM will go to the next state and merge the next object's sound.

Software

User interface

A start interface is shown on the screen after download the program and running it. Two maps can be chose. The background is changed to the gaming map after choosing either one of the two maps. The top of the map is information bar which shows the current money, wave and score information. Several interactive buttons are also displayed on the top, which are mute/unmute button, game speed select button, menu button, and play/pause button. Below the information bar is the main body of the map. Towers can be built in the clear ground, tower build on path and obstacles are not allowed.

When the software runs, the image and audio data are first written in SRAM and on-chip memory, respectively. We use finite state machines to operation functions.

Interrupt handlers

We have four interrupt handlers. In Sprite control interrupt handler, we send the current sprite information including x & y coordinates and SRAM indices to sprite controller.
In audio interrupt handler, we send the current audio information including lengths of audio clips and on-chip RAM indices to audio controller.

In mouse interrupt handler, we read out the status of mouse button and position using mouse state machine to call different functions. Play/pause, game speed, mute/unmute, menu, tower build, upgrade and sell are main functions controlled by mouse input.

In the timer interrupt handler, an interrupted is asserted to let the main functions run frame by frame. We are expecting tower attacking effects faster than monster movements. So monster and tower actions are called in different timing.

Main function

In our design there are two maps. Each has 20 waves of monsters, the monsters are generated automatically, and towers are built by mouse control. The main monster and tower update related functions are shown below.

- Monster and tower generate functions are used to generate new monsters and towers respectively.
- Tower upgrade function is used to upgrade the tower and update the characters of the corresponding tower.
- Tower sell function is used to delete a tower including all its information.
- Monster movement function is used to route monsters from the door to Edwards.

Monsters have 6 main different attributes including type, speed, health, killed/not killed phase and position. Towers have 7 main different attributes, which are type, attack speed, attack range, busy/not busy, current target, phase and position. In each frame, each tower detects whether any monsters entering its range and always choose the foremost one that is in its attacking range. The attack function is then triggered. The main attacking related functions are shown below

- Monster movement function is used to route monsters from the door to Edwards.
- Status update function is used to update the attacking effects of each tower.
- Health bar update function is used to update the health information of monsters by displaying health bars above the monsters under attack.
- Bullet update function is used to update the bullet display from the tower to monsters.
- The mainline state machine is used to compute majority operations of the game, displaying sprites on the screen and playing audio. A mouse handler machine is used to handle mouse

Experience and issues

During the entire design, we definitely encountered all kinds of problems and issues. Here are some examples from which we learned some significant design principles and rules helping us with the following design and debug.

Incoherent in display caused by data contention

When we were designing the sprite controller, we were faced with a problem that the screen was flickering. At last we found that the problem was that we were reading and writing data at the same time, so there was a data contention issue. So we added an interrupt and just refreshed the data during the interrupt to avoid the contention. And the problem solved

Real time violation in audio processing

When implementing the audio computation, we first wanted to do the computation in the software part. However, the fact was that when we did the computation in software, we could get the correct function at a cost of the degeneration of the game performance. Since there was lots of computation which cost very long time, the game speed was unacceptably slow. Then we decided to implement the audio control module in hardware although this would be a little bit harder. After finishing the construction of the hardware platform of the audio control, we solved the problem and achieved perfect game performance.

Inappropriate IRQ of mouse events

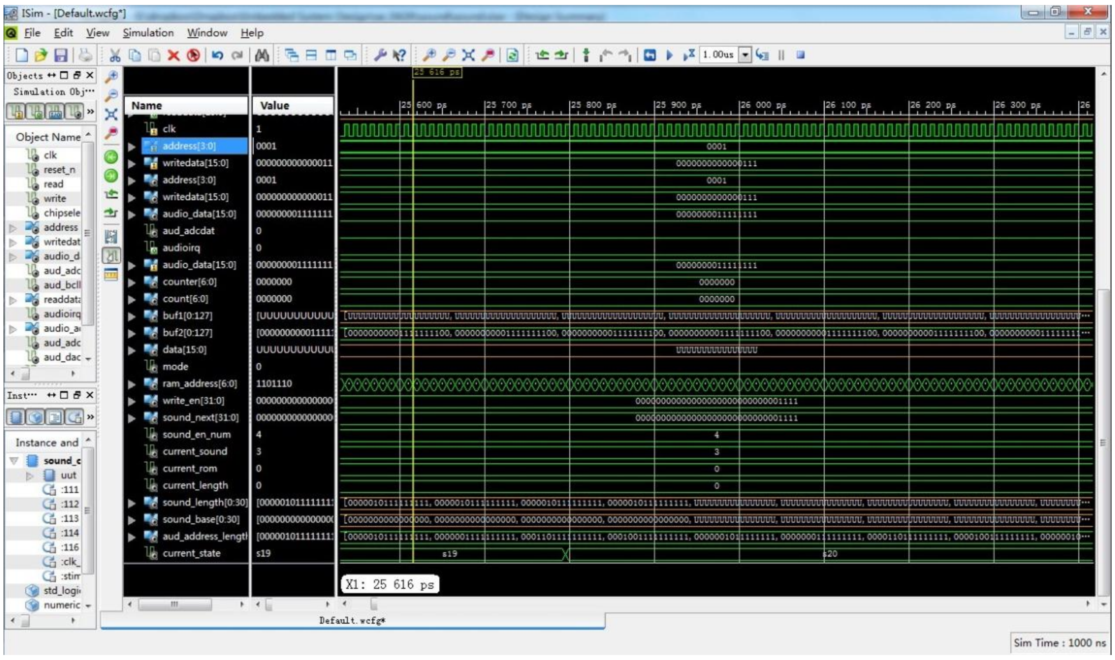
When doing the mouse part, we first fetched information from mouse by polling the mouse if there was a click. However, we found that we always missed the click and found that the problem was caused by the way of inquiry of the mouse. Then we changed the inquiry way to sending an interrupt to the software real-time and the problem solved.

Lesson learned

After one semester's study and work on the FPGA DE2 board, we are familiarized with the architecture of the hardware on FPGA board. And we increase our skills in VHDL coding and C program coding. We are familiar with some CAD tools such as Quartus, NISO II, ModelSim and so on.
Resource allocation – use hardware controllers to share computation
Scheduling optimization – remove slacked operations away from critical path

Debug methods

1. Write testbenches to simulate the RTL behavior, then find the bugs.
   When we encounter some bugs which are not that easy to solved, the best way to find out the bugs and to solve them is to run the simulation and check if the waveform follows the design. It is always the most efficient way to debug. For example, when we are designing the audio module, we could not find what bugs made our design failed all night long. And then we wrote a testbench and ran the simulation as follow, we easily found that an subtle judge error made the state transition of our FSM go wrong, and the problem solved.



Name	Value
clk	1
address[3:0]	0001
writedata[15:0]	0000000000000011
address[3:0]	0001
writedata[15:0]	0000000000000011
audio_data[15:0]	0000000001111111
aud_adcdat	0
audioirq	0
audio_data[15:0]	0000000001111111
counter[6:0]	0000000
count[6:0]	0000000
buf1[0:127]	[UUUUUUUUUUUU
buf2[0:127]	[000000000011111
data[15:0]	UUUUUUUUUUUU
mode	0
ram_address[6:0]	1101110
write_en[31:0]	0000000000000000
sound_next[31:0]	0000000000000000
sound_en_num	4
current_sound	3
current_rom	0
current_length	0
sound_length[0:30]	[000001011111111
sound_base[0:30]	[000000000000000
aud_address_lengtl	[000001011111111
current_state	s19

2. Use LEDs and LCD screen to indicate the value of the certain wire
   When we want to check the value of some certain wire or some certain bit, we could output the testing wire or bit to a LED or display them on LCD screen to find out if they were behaving as we expected them to.
3. Use console in NIOS II to debug software
   We can use "printf" sentence to display the data we want to test on the console screen. In that way, we can easily find if the values are what we want.

Advice for future work

1. Add more images including more monsters and towers
2. Sprite reuse to increase the limited number of sprites
3. Enhance the resolution
4. Optimize the VHDL code and software code to shorten the computation time
5. Add mouse selection attack mode, making the tower attack the monster selected by the mouse

Contribution

Wei Wei (ww2313): Software, PS2 mouse module, part of the audio module, part of the sprite control module;
Zeyang Yang (zy2171): Image and audio preprocessing, part of audio module
Zhe Cao (zc2237): Sprite control module, part of audio module
Ge Zhao (gz2196): Part of PS2 mouse module, image preprocessing

VHDL code

Top level vhdl

```
--
-- DE2 top-level module that includes the simple VGA raster generator
--
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--

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

entity top_level is

  port (
    -- Clocks

    CLOCK_27,                                                    -- 27 MHz
    CLOCK_50,                                                    -- 50 MHz
    EXT_CLOCK : in std_logic;                                -- External Clock

    -- Buttons and switches

    KEY : in std_logic_vector(3 downto 0);               -- Push buttons
    SW : in std_logic_vector(17 downto 0);               -- DPDT switches

    -- LED displays

    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
       : out std_logic_vector(6 downto 0);
    LEDG : out std_logic_vector(8 downto 0);             -- Green LEDs
    LEDR : out std_logic_vector(17 downto 0);            -- Red LEDs

    -- RS-232 interface

    UART_TXD : out std_logic;                                -- UART transmitter
    UART_RXD : in std_logic;                                 -- UART receiver

    -- IRDA interface
--      IRDA_TXD : out std_logic;                               -- IRDA Transmitter
    IRDA_RXD : in std_logic;                                -- IRDA Receiver

    -- SDRAM

    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                              -- Low-byte Data Mask
    DRAM_UDQM,                                              -- High-byte Data Mask
    DRAM_WE_N,                                              -- Write Enable
    DRAM_CAS_N,                                             -- Column Address Strobe
    DRAM_RAS_N,                                             -- Row Address Strobe
    DRAM_CS_N,                                              -- Chip Select
    DRAM_BA_0,                                              -- Bank Address 0
    DRAM_BA_1,                                              -- Bank Address 0
    DRAM_CLK,                                               -- Clock
    DRAM_CKE : out std_logic;                               -- Clock Enable

    -- FLASH

    FL_DQ : inout std_logic_vector(7 downto 0);             -- Data bus
    FL_ADDR : out std_logic_vector(21 downto 0);            -- Address bus
    FL_WE_N,                                                -- Write Enable
    FL_RST_N,                                               -- Reset
    FL_OE_N,                                                -- Output Enable
    FL_CE_N : out std_logic;                                -- Chip Enable

    -- SRAM

    SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
    SRAM_UB_N,                                              -- High-byte Data Mask
    SRAM_LB_N,                                              -- Low-byte Data Mask
    SRAM_WE_N,                                              -- Write Enable
    SRAM_CE_N,                                              -- Chip Enable
    SRAM_OE_N : out std_logic;                              -- Output Enable

    -- USB controller

    OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
    OTG_ADDR : out std_logic_vector(1 downto 0);            -- Address
    OTG_CS_N,                                               -- Chip Select
    OTG_RD_N,                                               -- Write
```

```
OTG_WR_N,                                   -- Read
OTG_RST_N,                                  -- Reset
OTG_FSPEED,                   -- USB Full Speed, 0 = Enable, Z = Disable
OTG_LSPEED : out std_logic;   -- USB Low Speed, 0 = Enable, Z = Disable
OTG_INT0,                                   -- Interrupt 0
OTG_INT1,                                   -- Interrupt 1
OTG_DREQ0,                                  -- DMA Request 0
OTG_DREQ1 : in std_logic;                   -- DMA Request 1
OTG_DACK0_N,                                -- DMA Acknowledge 0
OTG_DACK1_N : out std_logic;                -- DMA Acknowledge 1

-- 16 X 2 LCD Module

LCD_ON,                       -- Power ON/OFF
LCD_BLON,                     -- Back Light ON/OFF
LCD_RW,                       -- Read/Write Select, 0 = Write, 1 = Read
LCD_EN,                       -- Enable
LCD_RS : out std_logic;       -- Command/Data Select, 0 = Command, 1 = Data
LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits

-- SD card interface

SD_DAT,                       -- SD Card Data
SD_DAT3,                      -- SD Card Data 3
SD_CMD : inout std_logic;     -- SD Card Command Signal
SD_CLK : out std_logic;       -- SD Card Clock

-- USB JTAG link

TDI,                          -- CPLD -> FPGA (data in)
TCK,                          -- CPLD -> FPGA (clk)
TCS : in std_logic;           -- CPLD -> FPGA (CS)
TDO : out std_logic;          -- FPGA -> CPLD (data out)

-- I2C bus

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

-- PS/2 port

PS2_DAT : inout std_logic;    -- Data
PS2_CLK : inout std_logic;    -- Clock
```

```
-- VGA output

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

--   Ethernet Interface

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

-- Audio CODEC

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

-- Video Decoder

TD_DATA : in std_logic_vector(7 downto 0);    -- Data bus 8 bits
TD_HS,                                              -- H_SYNC
TD_VS : in std_logic;                           -- V_SYNC
TD_RESET : out std_logic;                       -- Reset

-- General-purpose I/O

GPIO_0,                                                 -- GPIO Connection 0
GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
);
```

```
end top_level;

architecture datapath of top_level is

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

   signal counter : unsigned(4 downto 0);
   signal reset_n : std_logic;

   signal BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
   signal DQM : STD_LOGIC_VECTOR(1 DOWNTO 0);
   signal pll_c1: STD_LOGIC;
   signal ram_addr    : std_logic_vector(17 downto 0);
   signal ram_data    : std_logic_vector(15 downto 0);
   signal vga_addr    : std_logic_vector(17 downto 0);
   signal vga_data    : std_logic_vector(15 downto 0);
   signal read_ready: std_logic_vector(15 downto 0);
   signal SRAM_address_io: std_logic_vector(17 downto 0);
   signal SRAM_byteenable_io: std_logic_vector(1 downto 0);
   signal SRAM_chipselect_io: std_logic;
   signal SRAM_read_io: std_logic;
   signal SRAM_write_io: std_logic;
   signal SRAM_writedata_io: std_logic_vector(15 downto 0);
   signal clk25 : std_logic := '0';
   signal sprite_pixel : unsigned(7 downto 0);
   signal sprite_y_blank :    std_logic;
   signal sprite_x : unsigned(9 downto 0);
   signal sprite_y : unsigned(9 downto 0);
   signal sprite_irq : std_logic;
   signal sprite_write :    std_logic;
   signal sprite_writedata : std_logic_vector(31 downto 0);
   signal sprite_address : std_logic_vector(15 downto 0);
   signal mouse_data : std_logic_vector(31 downto 0);
   signal audio_data : std_logic_vector(15 downto 0);
   signal audio_address : std_logic_vector(16 downto 0);
   signal audio_clock : unsigned(1 downto 0) := "00";
   signal x_cnt      : unsigned(11 downto 0);
```

```
begin

process (CLOCK_50)
begin
   if rising_edge(CLOCK_50) then
        clk25 <= not clk25;
      if counter = "1111" then
         reset_n <= '1';
      else
         reset_n <= '0';
         counter <= counter + 1;
      end if;
   end if;
end process;

AUD_XCK <= clk25;

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

nios : entity work.nios_system port map (
   clk_0                                    => pll_c1,
   reset_n                                  => '1',

    AUD_ADCDAT_to_the_sound_ctrl            => AUD_ADCDAT,
    AUD_ADCLRCK_from_the_sound_ctrl         => AUD_ADCLRCK,
    AUD_BCLK_to_and_from_the_sound_ctrl => AUD_BCLK,
    AUD_DACDAT_from_the_sound_ctrl          => AUD_DACDAT,
    AUD_DACLRCK_from_the_sound_ctrl         => AUD_DACLRCK,
    audio_address_from_the_sound_ctrl       => audio_address,
    audio_data_to_the_sound_ctrl            => audio_data,

   SRAM_address_from_the_sram_ctrl       => SRAM_address_io,
   SRAM_byteenable_from_the_sram_ctrl => SRAM_byteenable_io,
   SRAM_chipselect_from_the_sram_ctrl => SRAM_chipselect_io,
   SRAM_read_from_the_sram_ctrl          => SRAM_read_io,
   SRAM_write_from_the_sram_ctrl         => SRAM_write_io,
   SRAM_writedata_from_the_sram_ctrl     => SRAM_writedata_io,
        rdaddress_to_the_ram_ctrl                  => audio_address(14 downto 0),
        q_from_the_ram_ctrl                               => audio_data,

        zs_addr_from_the_sdram_ctrl              => DRAM_ADDR,
        zs_ba_from_the_sdram_ctrl                  => BA,
        zs_cas_n_from_the_sdram_ctrl            => DRAM_CAS_N,
        zs_cke_from_the_sdram_ctrl                => DRAM_CKE,
        zs_cs_n_from_the_sdram_ctrl              => DRAM_CS_N,
        zs_dq_to_and_from_the_sdram_ctrl      => DRAM_DQ,
        zs_dqm_from_the_sdram_ctrl                => DQM,
        zs_ras_n_from_the_sdram_ctrl            => DRAM_RAS_N,
        zs_we_n_from_the_sdram_ctrl              => DRAM_WE_N,

       address_out_from_the_sprite_ctrl     => sprite_address,
       write_out_from_the_sprite_ctrl              => sprite_write,
       writedata_out_from_the_sprite_ctrl => sprite_writedata,
       irq_to_the_sprite_ctrl                          => sprite_irq,

        mousedata_to_the_mouse_ctrl                     => mouse_data,

        outp_from_the_mux_ctrl                       => read_ready
       );


mouse: entity work.de2_ps2 port map(
       clkin                   => CLOCK_50,
        rst             => not reset_n,
       ps2_clk              => PS2_CLK,
        ps2_data            => PS2_DAT,
       lefbut                 => mouse_data(0),
        rigbut                => mouse_data(1),
        midbut               => mouse_data(2),
       x_mov                => mouse_data(12 downto 3),
        y_mov                => mouse_data(22 downto 13),
        mouse_irq          => mouse_data(23)
       );


mux_hw_sw: entity work.mux_hw_sw port map(
       clk              => CLOCK_50,
       reset_n         => '1',
       inp              => read_ready,
       chipselect      => SRAM_chipselect_io,
       read                   => SRAM_read_io,
	 write			=> SRAM_write_io,

	address			=> SRAM_address_io,
	writedata		=> SRAM_writedata_io,
	byteenable	=> SRAM_byteenable_io,

	readdata		=> ram_data,

	write_inp		=> '0',
	 read_inp		=> '1',
	 writedata_inp	=> (others => '0'),
	 address_inp	=> ram_addr,

	SRAM_DQ		=> SRAM_DQ,
	SRAM_ADDR		=> SRAM_ADDR,
	SRAM_UB_N		=> SRAM_UB_N,
	 SRAM_LB_N		=> SRAM_LB_N,
	SRAM_WE_N		=> SRAM_WE_N,
	 SRAM_CE_N		=> SRAM_CE_N,
	SRAM_OE_N		=> SRAM_OE_N,
	 outp			=> LEDR(17)
	);


sprite : entity work.sprite port map (
	clk_50			=> CLOCK_50,
	 address		=> sprite_address,
	write			=> sprite_write,
	writedata		=> unsigned(sprite_writedata),
	 irq			=> sprite_irq,

	 pixel			=> sprite_pixel,
	 x				=> sprite_x,
	 y				=> sprite_y,
	 y_blank		=> sprite_y_blank,
	 x_count		=> x_cnt,

	 sram_addr		=> ram_addr,
	 sram_datain		=> ram_data
);



vga_display: entity work.de2_vga_raster port map(
      reset           => '0',
      clk_50          => CLOCK_50,

       pixel          => sprite_pixel,
       x_index        => sprite_x,
       y_index        => sprite_y,
       y_blank        => sprite_y_blank,
       x_count        => x_cnt,
      inp            => read_ready,

      VGA_CLK         => VGA_CLK,
      VGA_HS              => VGA_HS,
      VGA_VS              => VGA_VS,
      VGA_BLANK  => VGA_BLANK,
      VGA_SYNC    => VGA_SYNC,
      VGA_R           => VGA_R,
      VGA_G           => VGA_G,
      VGA_B           => VGA_B
      );

neg_3ns: entity work.sdram_pll port map(
      inclk0 => CLOCK_50,
      c0         => DRAM_CLK,
      c1         => pll_c1);

   HEX7         <= "0001001"; -- Leftmost
   HEX6         <= "0000110";
   HEX5         <= "1000111";
   HEX4         <= "1000111";
   HEX3         <= "1000000";
   HEX2         <= (others => '1');
   HEX1         <= (others => '1');
   HEX0         <= (others => '1');                    -- Rightmost
   --LEDG          <= (others => '1');
   --LEDR         <= (others => '1');
   LCD_ON      <= '1';
   LCD_BLON <= '1';
   LCD_RW <= '1';
   LCD_EN <= '0';
   LCD_RS <= '0';

   SD_DAT3 <= '1';
   SD_CMD <= '1';
   SD_CLK <= '1';
```

```
--SRAM_DQ <= (others => 'Z');
--SRAM_ADDR <= (others => '0');
--SRAM_UB_N <= '1';
--SRAM_LB_N <= '1';
--SRAM_CE_N <= '1';
--SRAM_WE_N <= '1';
--SRAM_OE_N <= '1';

UART_TXD <= '0';
--DRAM_ADDR <= (others => '0');
DRAM_LDQM <= DQM(0);
DRAM_UDQM <= DQM(1);
--DRAM_WE_N <= '1';
--DRAM_CAS_N <= '1';
--DRAM_RAS_N <= '1';
--DRAM_CS_N <= '1';
DRAM_BA_0 <= BA(0);
DRAM_BA_1 <= BA(1);
--DRAM_CLK <= '0';
--DRAM_CKE <= '0';

FL_ADDR <= (others => '0');
FL_WE_N <= '1';
FL_RST_N <= '0';
FL_OE_N <= '1';
FL_CE_N <= '1';
OTG_ADDR <= (others => '0');
OTG_CS_N <= '1';
OTG_RD_N <= '1';
OTG_RD_N <= '1';
OTG_WR_N <= '1';
OTG_RST_N <= '1';
OTG_FSPEED <= '1';
OTG_LSPEED <= '1';
OTG_DACK0_N <= '1';
OTG_DACK1_N <= '1';

TDO <= '0';

ENET_CMD <= '0';
ENET_CS_N <= '1';
ENET_WR_N <= '1';
ENET_RD_N <= '1';
```

```
   ENET_RST_N <= '1';
   ENET_CLK <= '0';

   TD_RESET <= '0';

   --I2C_SCLK <= '1';

   --AUD_DACDAT <= '1';
   --AUD_XCK <= '1';

   -- Set all bidirectional ports to tri-state
   --DRAM_DQ         <= (others => 'Z');
   FL_DQ           <= (others => 'Z');
   --SRAM_DQ         <= (others => 'Z');
   OTG_DATA        <= (others => 'Z');
   LCD_DATA        <= (others => 'Z');
   SD_DAT          <= 'Z';
   I2C_SDAT        <= 'Z';
   ENET_DATA       <= (others => 'Z');
   AUD_ADCLRCK <= 'Z';
   AUD_DACLRCK <= 'Z';
   AUD_BCLK        <= 'Z';
   GPIO_0          <= (others => 'Z');
   GPIO_1          <= (others => 'Z');

end datapath;
```

Mux controller code (connect hardware and software bus to the SRAM)

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

entity mux_hw_sw is

   port (
      clk           : in    std_logic;
      reset_n       : in    std_logic;
      signal inp                 : in    std_logic_vector(15 downto 0);
      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 write_inp, read_inp : in std_logic;
         signal writedata_inp : in std_logic_vector(15 downto 0);
         signal address_inp    :    in std_logic_vector(17 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;
         outp : out std_logic
        );

end mux_hw_sw;

architecture dp of mux_hw_sw is
begin

mux: process(inp, chipselect, write, read, address, writedata, byteenable, write_inp,
read_inp, writedata_inp, address_inp)
begin
                if inp = "0000000000001100" then
                    if write_inp = '1' then
                     SRAM_DQ <= writedata;
                     else
                 SRAM_DQ <= (others => 'Z');
                     end if;
                     readdata <= SRAM_DQ;
                     SRAM_ADDR <= address_inp;
                     SRAM_UB_N <= '0';
                     SRAM_LB_N <= '0';
                     SRAM_WE_N <= not write_inp;
                     SRAM_CE_N <= '0';
                     SRAM_OE_N <= not read_inp;
                     outp <= '1';

                else
                    if write = '1' then
                     SRAM_DQ <= writedata;
                     else
                 SRAM_DQ <= (others => 'Z');
                     end if;
                     readdata <= (others => '0');
                     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;
                     outp <= '0';
                end if;
end process;
end dp;
```

Sprite controller vhdl code

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



entity sprite is
port(
     address     : in std_logic_vector(15 downto 0);-- higher 6 bit is sprite index, lower 6 bit
is ROM number
     writedata : in unsigned(31 downto 0);-- bit 31 is IRQ switch, bit 30~16 is y value, bit
15~0 is x value
     write       : in std_logic;
     clk_50      : in std_logic;
     irq         : out std_logic;

    pixel    : out unsigned(7 downto 0); -- current pixel code
     x        :    in unsigned(9 downto 0);
     y          : in unsigned(9 downto 0);
     y_blank: in std_logic;
     x_count: in unsigned(11 downto 0);

     sram_addr      : out std_logic_vector(17 downto 0);
     sram_datain : in    std_logic_vector(15 downto 0)
     );
end sprite;

architecture decode of sprite is

constant SPRITENO: integer:=92;
constant IMAGENO: integer:=214;

-- sprite width and hight
```

```
type rom_size_type is array(integer range 0 to IMAGENO) of integer range 1 to 256;
constant rom_width : rom_size_type := (
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32,
32, 32, 32,
32, 32, 32,
32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32,

32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32,
32, 32, 32,
32, 32, 32,
32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32,

32, 32, 32,
32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32,
32, 32, 32, 32,
32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32,
8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
32, 32, 32,
32, 32, 32,
32, 32, 32, 32
);

constant rom_height : rom_size_type := (
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32,
32, 32, 32,
32, 32, 32,
32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32,

32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32,
32, 32, 32,
32, 32, 32,
32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32,

32, 32, 32,
32, 32, 32,
4, 4, 4, 4, 4, 4, 4, 4,
32, 32, 32, 32, 32, 32,
32, 32, 32, 32,
32, 32, 32, 32, 32, 32,
40, 40, 40, 40, 40, 40,
16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
64, 64, 64,
32, 32, 32,
32, 64, 96, 128
);

--sram information store
type sram_offset_type is array(integer range 0 to IMAGENO) of integer range 0 to 262143;
constant sram_offset : sram_offset_type := (
153600, 154112, 154624, 155136, 155648, 156160, 156672, 157184, 157696, --monster
158208, 158720, 159232, 159744, 160256, 160768, 161280, 161792, 162304,
162816, 163328, 163840, 164352, 164864, 165376, 165888, 166400, 166912,
167424, 167936, 168448, 168960, 169472, 169984, 170496, 171008, 171520,  --glu tower
172032, 172544, 173056, 173568, 174080, 174592, 175104, 175616, 176128,
176640, 177152, 177664, 178176, 178688, 179200, 179712, 180224, 180736,
181248, 181760, 182272, 182784, 183296, 183808, 184320, 184832, 185344, --green tower
185856, 186368, 186880, --dart tower
187392, 187904, 188416, --bomb tower
188928, 189440, 189952, --bullet glu green
190464, 190976, 191488, 192000, 192512, 193024, --bullet dart
193536, 194048, 194560, 195072, 195584, 196096, --bullet bomb

196608, 197120, 197632, 198144, 198656, 199168, 199680, 200192, 200704,
201216, 201728, 202240, 202752, 203264, 203776, 204288, 204800, 205312,
205824, 206336, 206848, 207360, 207872, 208384, 208896, 209408, 209920, --glu tower
210432, 210944, 211456, 211968, 212480, 212992, 213504, 214016, 214528,
215040, 215552, 216064, 216576, 217088, 217600, 218112, 218624, 219136,
219648, 220160, 220672, 221184, 221696, 222208, 222720, 223232, 223744, --green tower
224256, 224768, 225280, --dart tower
225792, 226304, 226816, --bomb tower
227328, 227840, 228352, --bullet glu green
228864, 229376, 229888, 230400, 230912, 231424, --bullet dart
231936, 232448, 232960, 233472, 233984, 234496, --bullet bomb

235008, 235520, 236032, --glu effect
236544, 237056, 237568, --select
238080, 238144, 238208, 238272, 238336, 238400, 238464, 238528, --health bar
238592, 239104, 239616, 240128, 240640, 241152, --button
241664, 242176, 242688, 243200, --tower upgrade
243712, 244224, 244736, 245248, 245760, 246272,--bullet effect
246784, 247424, 248064, 248704, 249344, 249984,--boss
250624, 250688, 250752, 250816, 250880, 250944, 251008, 251072, 251136, 251200,
--number
251264, 252288, 253312,--quit
254336, 254848, 255360,
257952, 257952, 257952, 257952--tower select
--211
);

----input buffers
signal irq_mask : std_logic := '0';
signal x_buf    : unsigned(9 downto 0);
signal y_buf : unsigned(9 downto 0);
signal sprite_index_buf : integer range 0 to SPRITENO;
signal rom_buf : integer range 0 to IMAGENO;

--store the sprite status
type sprite_pos_type is array(integer range 0 to SPRITENO) of unsigned(9 downto 0);
signal      sprite_width,
                  sprite_x,
                 sprite_y_end,
                  sprite_y : sprite_pos_type;
signal sprite_show: std_logic_vector(SPRITENO downto 0) := (others => '0');
signal sprite_flip : std_logic_vector(SPRITENO downto 0);
signal sprite_mirrow : std_logic_vector(SPRITENO downto 0);

type color_type is array(integer range 0 to SPRITENO) of unsigned(1 downto 0);
signal sprite_color : color_type;

type sprite_addr_offset_type is array(integer range 0 to SPRITENO) of unsigned(17 downto
0);
signal sprite_addr_offset : sprite_addr_offset_type;

-- pipeline storage
signal current_sprite     : integer range 0 to 92;
signal x_stage1                 : unsigned(9 downto 0);
signal y_stage1                 : unsigned(9 downto 0);
signal y_stage1_end             : unsigned(9 downto 0);
signal width_stage1             : unsigned(9 downto 0);
signal offset_stage1            : unsigned(8 downto 0);
signal show_stage1              : std_logic;
signal addr_offset_stage1 : unsigned(17 downto 0);
signal flip_stage1        : std_logic;
signal color_stage1             : unsigned(1 downto 0);
signal mirrow_stage1            : std_logic;



signal position_stage2          : unsigned(9 downto 0);
signal position_stage2_plus : unsigned(9 downto 0);
signal show_stage2              : std_logic;
signal flip_stage2        : std_logic;
signal mirrow_stage2            : std_logic;
signal color_stage2             : unsigned(1 downto 0);
signal sram_addr_buf            : unsigned(17 downto 0);



signal sram_in_buf              : std_logic_vector(15 downto 0);
signal write_addr_buf     : unsigned(17 downto 0);
signal show_stage3              : std_logic;
signal flip_stage3        : std_logic;
signal mirrow_stage3            : std_logic;
signal position_roll      : std_logic;
signal buf_stage4          : unsigned(15 downto 0);
signal show_en_stage4        : std_logic;
signal write4_addr_buf: unsigned(17 downto 0);


signal write_en_buf                    : std_logic_vector(1 downto 0);
signal addr_0, addr_1   : unsigned(17 downto 0);
signal di                       : unsigned(15 downto 0);
signal write_en1, write_en0               : std_logic_vector(1 downto 0);
signal dataout_buf              : unsigned(7 downto 0);
type do_type is array(integer range 0 to 1) of unsigned(7 downto 0);
signal do_1, do_0                : do_type;
signal irq_f : integer;
signal irq_test: std_logic;
signal wr_out    : std_logic_vector(15 downto 0);
signal mode : std_logic;


--interface signal:
--writedata:
--bit 0 ~9    : x position
--bit 10~19 : y position
--bit 29       : mirrow
--bit 30       : flip
--bit 31       : irq release
begin

    irq <= irq_mask and y_blank;
    x_buf <= writedata(9 downto 0);
    y_buf <= writedata(19 downto 10);
    sprite_index_buf <= to_integer( unsigned( address(6 downto 0) ) );
    rom_buf <= to_integer( unsigned( address(14 downto 7) ) );
    sram_addr <= std_logic_vector(sram_addr_buf);

    UPDATE: process(clk_50)
    begin
        if rising_edge(clk_50) then
            if y_blank = '0' then
                irq_mask <= '1';
            end if;
            if write = '1' then
                if y_blank = '1' then
                    if writedata(31) = '1' then
```

```
                                        irq_mask <= '0';
                                end if;

                                if x_buf = "1111111111" and y_buf = "1111111111" then
                                        sprite_show( sprite_index_buf ) <= '0';
                                else
                                        sprite_flip( sprite_index_buf ) <= writedata(20);
                                        sprite_mirrow( sprite_index_buf ) <= writedata(21);
                                        sprite_color(  sprite_index_buf  )  <=  unsigned(writedata(23
downto 22));
                                        sprite_show( sprite_index_buf ) <= '1';
                                        sprite_x( sprite_index_buf ) <= x_buf;
                                        sprite_y( sprite_index_buf ) <= y_buf;
                                        sprite_y_end(      sprite_index_buf      )      <=      y_buf      +
to_unsigned( rom_height(rom_buf), 10 );
                                        sprite_width(              sprite_index_buf              )              <=
to_unsigned( rom_width(rom_buf), 10);
                                        sprite_addr_offset(           sprite_index_buf           )           <=
to_unsigned( sram_offset(rom_buf), 18 );
                                end if;
                        end if;
                end if;                 --update  the  new  sprite  location  only  when  the  vga
refreshing is blank between two frames
        end if;
    end process UPDATE;

    -- firt 12 sprite used for numbers display, 4 cycles for each, others use 16 cycles
    current_sprite  <=  to_integer(  x_count(11  downto  4))  -  11  when  x_count  >  368  else
to_integer(x_count(11  downto  2))  -  80;  --to_integer(  x_count(11  downto  4)  )  -  20;  --when
mode = '1' else to_integer( x_count(11 downto 4) ) + 60;
    BUF_REFRESH_1 : process(clk_50)
    begin
        if rising_edge(clk_50) then
            if    x_count < 320 then
                offset_stage1 <= x_count(8 downto 0);
                width_stage1 <= to_unsigned(640, 10);
                x_stage1              <= to_unsigned(    0, 10);
                y_stage1              <= to_unsigned(    0, 10);
                y_stage1_end          <= to_unsigned(479, 10);
                flip_stage1     <= '0';
                mirrow_stage1      <= '0';
                color_stage1  <= "00";
                show_stage1  <= '1';
                addr_offset_stage1 <= to_unsigned(0, 18);
            elsif current_sprite <= 11 and current_sprite >= 0 then
                offset_stage1 <= "0000000" & x_count(1 downto 0);
                x_stage1            <= sprite_x(current_sprite);
                y_stage1            <= sprite_y(current_sprite);
                y_stage1_end        <= sprite_y_end(current_sprite);
                width_stage1        <= sprite_width(current_sprite);
                flip_stage1     <= sprite_flip(current_sprite);
                mirrow_stage1 <= sprite_mirrow(current_sprite);
                color_stage1  <= sprite_color(current_sprite);
                show_stage1         <= sprite_show(current_sprite);
                addr_offset_stage1 <= sprite_addr_offset(current_sprite);
            elsif current_sprite <= (SPRITENO-1) and current_sprite > 11 then
                offset_stage1 <= "00000" & x_count(3 downto 0);
                x_stage1            <= sprite_x(current_sprite);
                y_stage1            <= sprite_y(current_sprite);
                y_stage1_end        <= sprite_y_end(current_sprite);
                width_stage1        <= sprite_width(current_sprite);
                flip_stage1     <= sprite_flip(current_sprite);
                mirrow_stage1 <= sprite_mirrow(current_sprite);
                color_stage1  <= sprite_color(current_sprite);
                show_stage1         <= sprite_show(current_sprite);
                addr_offset_stage1 <= sprite_addr_offset(current_sprite);
            else
                show_stage1  <= '0';
            end if;
        end if;
    end process BUF_REFRESH_1;

    BUF_REFRESH_2 : process(clk_50)
    begin
        if rising_edge(clk_50) then
            if y >= y_stage1 and y < y_stage1_end then
                    if (flip_stage1 = '1' and    mirrow_stage1 = '0') then
                        sram_addr_buf <= addr_offset_stage1 - offset_stage1 + (1 + y(8
downto 0) - y_stage1(8 downto 0)) * width_stage1(9 downto 1);
                    elsif (flip_stage1 = '0' and    mirrow_stage1 = '1') then
                        sram_addr_buf  <=  addr_offset_stage1  +  offset_stage1  +  (31  -
y(8 downto 0) + y_stage1(8 downto 0)) * width_stage1(9 downto 1);
                    elsif (flip_stage1 = '1' and    mirrow_stage1 = '1') then
                        sram_addr_buf <= addr_offset_stage1 - offset_stage1 + (32 - y(8
downto 0) + y_stage1(8 downto 0)) * width_stage1(9 downto 1);
                    else
                        sram_addr_buf  <=  addr_offset_stage1  +  offset_stage1  +  (y(8
downto 0) - y_stage1(8 downto 0)) * width_stage1(9 downto 1);
```

```
                    end if;
                    position_stage2 <= x_stage1 + (offset_stage1&'0');
                    position_stage2_plus <= x_stage1 + (offset_stage1&'0') + 2;
                    show_stage2 <= show_stage1;
                    flip_stage2 <= flip_stage1;
                    color_stage2 <= color_stage1;
                else
                    show_stage2 <= '0';
                end if;
            end if;
    end process BUF_REFRESH_2;

    BUF_REFRESH_3 : process(clk_50)
    begin
            if rising_edge(clk_50) then
                flip_stage3 <= flip_stage2;
                show_stage3 <= show_stage2;
                position_roll <= position_stage2(0);
                if  color_stage2  =  "01"  and  sram_datain(15  downto  8)  <  X"ff"  and
sram_datain(7 downto 0) = X"ff" then
                    sram_in_buf <= X"8DFF";
                elsif  color_stage2  =  "01"  and  sram_datain(7  downto  0)  <  X"ff"  and
sram_datain(15 downto 8) = X"ff" then
                    sram_in_buf <= X"FF8D";
                elsif  color_stage2  =  "01"  and  sram_datain(7  downto  0)  <  X"ff"  and
sram_datain(15 downto 8) < X"ff" then --and sram_datain > X"0000" then
                    sram_in_buf <= X"8D8D";
                elsif  color_stage2  =  "10"  and  sram_datain(15  downto  8)  <  X"ff"  and
sram_datain(7 downto 0) = X"ff" then
                    sram_in_buf <= X"F6FF";
                elsif  color_stage2  =  "10"  and  sram_datain(7  downto  0)  <  X"ff"  and
sram_datain(15 downto 8) = X"ff" then
                    sram_in_buf <= X"FFF6";
                elsif  color_stage2  =  "10"  and  sram_datain(7  downto  0)  <  X"ff"  and
sram_datain(15 downto 8) < X"ff" then --and sram_datain > X"0000" then
                    sram_in_buf <= X"F6F6";
                else
                    sram_in_buf <= sram_datain;
                end if;
                case position_stage2(0) is
                    when  '0'  =>write_addr_buf  <=  position_stage2(9  downto  1)  &
position_stage2(9 downto 1);
                    when  '1'  =>write_addr_buf  <=  position_stage2_plus(9  downto  1)  &
position_stage2(9 downto 1);
```

```
                end case;
            end if;
        end process BUF_REFRESH_3;

        BUF_REFRESH_4 : process(clk_50)
        begin
            if rising_edge(clk_50) then
                if flip_stage3 = '0' then
                    case position_roll is
                        when '0' => buf_stage4 <= unsigned(sram_in_buf);
                        when '1' => buf_stage4 <= unsigned(sram_in_buf(7 downto 0)&
sram_in_buf(15 downto 8));
                    end case;
                else
                    case position_roll is
                        when '0' => buf_stage4 <= unsigned(sram_in_buf(7 downto 0)&
sram_in_buf(15 downto 8));
                        when '1' => buf_stage4 <= unsigned(sram_in_buf);
                    end case;
                end if;
                write4_addr_buf <= write_addr_buf;
                show_en_stage4 <= show_stage3;
            end if;
        end process BUF_REFRESH_4;

        BUF_REFRESH_5 : process(clk_50)
        begin
            if rising_edge(clk_50) then
            di <= buf_stage4;
                if y(0) = '1' then
                    if show_en_stage4 = '1' then
                        write_en1(0) <= not (buf_stage4(15) and buf_stage4(14) and
buf_stage4(13) and buf_stage4(12) and
                                                buf_stage4(11) and buf_stage4(10) and
buf_stage4( 9) and buf_stage4( 8));
                        write_en1(1) <= not (buf_stage4( 7) and buf_stage4( 6) and
buf_stage4( 5) and buf_stage4( 4) and
                                                buf_stage4( 3) and buf_stage4( 2) and
buf_stage4( 1) and buf_stage4( 0));
                    else
                        write_en1 <= (others => '0');
                    end if;
                    write_en0 <= (others => '0');
                    addr_1 <= write4_addr_buf;
```

```
                        addr_0 <= x(9 downto 1) & x(9 downto 1);
                else
                        if show_en_stage4 = '1' then
                                write_en0(0)   <=   not   (buf_stage4(15)   and   buf_stage4(14)   and
buf_stage4(13) and buf_stage4(12) and
                                                        buf_stage4(11)    and    buf_stage4(10)    and
buf_stage4( 9) and buf_stage4( 8));
                                write_en0(1)   <=   not   (buf_stage4(   7)   and   buf_stage4(   6)   and
buf_stage4( 5) and buf_stage4( 4) and
                                                        buf_stage4(   3)    and    buf_stage4(   2)    and
buf_stage4( 1) and buf_stage4( 0));
                        else
                                write_en0 <= (others => '0');
                        end if;
                        write_en1 <= (others => '0');
                        addr_0 <= write4_addr_buf;
                        addr_1 <= x(9 downto 1) & x(9 downto 1);
                end if;
        end if;
    end process BUF_REFRESH_5;

    dataout_buf <=    do_0(0) when x(0)='1' and y(0) = '1' else
                            do_0(1) when x(0)='0' and y(0) = '1' else
                            do_1(0) when x(0)='1' and y(0) = '0' else
                            do_1(1);
    pixel <= dataout_buf(7 downto 0);

    -- two buffers
    LINE_BUF_0m0 : entity work.line_buf port map(
        clk    => clk_50, we=> write_en0(0),
        a=> addr_0(17 downto 9), do=>do_0(0),
        di(7 downto 0)=> di(15 downto 8));
    LINE_BUF_0m1 : entity work.line_buf port map(
        clk    => clk_50, we=> write_en0(1),     do=>do_0(1),
        a=> addr_0(8 downto 0),
        di(7 downto 0)=> di(7 downto 0));

    LINE_BUF_1m0 : entity work.line_buf port map(
        clk    => clk_50, we=> write_en1(0), do=>do_1(0),
        a=> addr_1(17 downto 9),
        di(7 downto 0)=> di(15 downto 8));
    LINE_BUF_1m1 : entity work.line_buf port map(
        clk    => clk_50, we=> write_en1(1),     do=>do_1(1),
        a=> addr_1(8 downto 0),
            di(7 downto 0)=> di(7 downto 0));

end decode;
```

Audio controller vhdl code

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

entity sound_controller is
port (

     clk            : in    std_logic;
     reset_n        : in    std_logic;
     read           : in    std_logic;
     write          : in    std_logic;
     chipselect : in    std_logic;
     address        : in    std_logic_vector(3 downto 0);
     readdata       : out std_logic_vector(15 downto 0);
     writedata      : in    std_logic_vector(15 downto 0);
     audioIrq       : out std_logic;

      audio_data   : in    std_logic_vector(15 downto 0);
      audio_address : out    std_logic_vector(16 downto 0);

     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 sound_controller;

  architecture rtl of sound_controller is

  signal clk_25:std_logic;
  signal counter: unsigned (7 downto 0);
  signal count: unsigned (7 downto 0);
  type    ram_type is array (0 to 255) of unsigned (17 downto 0);
  signal buf1 : ram_type;
  signal buf2 : ram_type;
  signal data: unsigned (15 downto 0);
```

```
   signal mode:std_logic := '0';
   signal ram_address : unsigned(7 downto 0) := "00000000";
   signal audio_offset : unsigned(15 downto 0);
   signal base_address : unsigned(15 downto 0);
   signal write_en: std_logic_vector(31 downto 0) := X"00000000";
   signal sound_next: std_logic_vector(31 downto 0) := X"00000000";
   signal sound_en_num: unsigned(4 downto 0) := "00000";
   signal current_sound: integer range 0 to 30;
   signal current_rom: integer range 0 to 8;
   signal current_length: integer range 0 to 15;
   signal current_play: integer range 0 to 30;

   type aud_address_type is array(integer range 0 to 30) of unsigned(15 downto 0);
   signal aud_address : aud_address_type;
   signal sound_length : aud_address_type;
   signal sound_base : aud_address_type;

   type aud_address_base_type is array(integer range 0 to 7) of unsigned(15 downto 0);
   constant aud_address_base : aud_address_base_type := (
   X"0000", X"0600", X"0A00", X"2600",
   X"3A00", X"4D00", X"0000", X"0000"
   );

   type aud_address_length_type is array(integer range 0 to 15) of unsigned(15 downto 0);
   signal aud_address_length : aud_address_length_type := (
   X"05FF", X"03FF", X"13FF", X"13FF",
   X"02FF", X"01FF", X"07FF", X"09FF",
   X"02FF", X"01FF", X"05FF", X"07FF",
   X"12FF", X"097F", X"097F", X"09FF"
   );

   type                                    state                                    is
(s0,s1,s2,s3);--,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,s19,s20,s21,s22);
   signal current_state : state;

    component de2_wm8731_audio is port(
              clk              :  in   std_logic;          --   Audio  CODEC  Chip  Clock
AUD_XCK (18.43 MHz)
               reset_n         : in std_logic;
              data             : in unsigned(15 downto 0);
               counter         : buffer unsigned (7 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 component de2_wm8731_audio;

begin

        audio_address(15 downto 0) <= std_logic_vector(audio_offset + base_address);

        current_sound <= to_integer(unsigned(writedata(5 downto 1)));
        current_rom <= to_integer(unsigned(writedata(9 downto 6)));
        current_length <= to_integer(unsigned(writedata(13 downto 10)));

        process (current_state)
        begin
                case current_state is
                when s1 =>
                        audio_offset <= aud_address(current_play);
                        base_address <= sound_base(current_play);
                when others =>
                end case;
        end process;

        process (clk)
        begin
                if rising_edge (clk) then
                        clk_25 <= not clk_25;
                end if;
        end process;

        process (clk)
        begin
                if rising_edge(clk) then
                        if reset_n = '0' then
                                audioIrq <= '0';
                        else

                                -- generate irq
                                if count = "11111111" and counter = "00000000" then
                                        audioIrq <= '1';
                                        mode <= not mode;
                                        ram_address <= (others => '0');
                    current_state <= s0;
                    current_play <= 0;

                -- clear irq
                elsif write = '1' and chipselect = '1' then
                    audioIrq <= '0';
                    if (sound_next(current_sound) = '0' and writedata(0) = '1') or
(sound_next(current_sound) = '1' and writedata(0) = '0') then
                        write_en(current_sound) <= '1';
                        sound_base(current_sound) <=
aud_address_base(current_rom);
                        sound_length(current_sound) <=
aud_address_length(current_length);
                    end if;
                    sound_next(current_sound) <= writedata(0);
                end if;
                count <= counter;

                case current_state is
                    when s0 =>
                        if ram_address = "11111111" then
                            current_state <= s1;
                        end if;
                        ram_address <= ram_address + 1;
                        sound_en_num <= (others => '0');

                    when s1 =>
                        if aud_address(current_play) >= sound_length(current_play)
then
                            aud_address(current_play) <= (others => '0');
                            write_en(current_play) <= '0';
                        end if;

                        if ram_address = "11111111" then
                            if write_en(current_play) = '1' and current_play > 9 then
                                sound_en_num <= sound_en_num + 1;
                            end if;
                            current_play <= current_play + 1;
                        else
                            if write_en(current_play) = '1' then
                                aud_address(current_play) <=
aud_address(current_play) + 1;
                            end if;
                        end if;
```

```
                                        if current_play = 26 then
                                                current_state <= s2;
                                        end if;
                                        ram_address <= ram_address + 1;

                                when s2 =>
                                        if ram_address = "11111111" then
                                                current_state <= s3;
                                        end if;
                                        ram_address <= ram_address + 1;

                                when s3 =>
                        end case;
                end if;
        end if;
    end process;


    data <= buf1(to_integer (counter))(15 downto 0) when mode = '0' else buf2(to_integer
(counter))(15 downto 0);
    process (clk)
    begin
        if rising_edge(clk) then
                case current_state is
                        when s0 =>
                                if mode = '0' then
                                        buf2 (to_integer (unsigned(ram_address))) <= (others => '0');
                                elsif mode = '1' then
                                        buf1 (to_integer (unsigned(ram_address))) <= (others => '0');
                                end if;
                        when s1 =>
                                if write_en(current_play) = '1' and mode = '0' then
                                        buf2  (to_integer  (unsigned(ram_address)))  <=  buf2  (to_integer
(unsigned(ram_address)))
                                        +        unsigned(audio_data(15)&audio_data(15)&audio_data(15
downto 0));
                                elsif write_en(current_play) = '1' and mode = '1' then
                                        buf1  (to_integer  (unsigned(ram_address)))  <=  buf1  (to_integer
(unsigned(ram_address)))
                                        +        unsigned(audio_data(15)&audio_data(15)&audio_data(15
downto 0));
                                end if;
                        when s2 =>
                        case sound_en_num is
                            when "00010" =>
                                if mode = '0' then
                                    buf2  (to_integer  (unsigned(ram_address)))  <=  "0"  &
buf2 (to_integer (unsigned(ram_address)))(17 downto 1);
                                elsif mode = '1' then
                                    buf1  (to_integer  (unsigned(ram_address)))  <=  "0"  &
buf1 (to_integer (unsigned(ram_address)))(17 downto 1);
                                end if;
                            when "00011" =>
                                if mode = '0' then
                                    buf2  (to_integer  (unsigned(ram_address)))  <=  "00"  &
buf2 (to_integer (unsigned(ram_address)))(17 downto 2);
                                elsif mode = '1' then
                                    buf1  (to_integer  (unsigned(ram_address)))  <=  "00"  &
buf1 (to_integer (unsigned(ram_address)))(17 downto 2);
                                end if;
                            when "00100" =>
                                if mode = '0' then
                                    buf2  (to_integer  (unsigned(ram_address)))  <=  "00"  &
buf2 (to_integer (unsigned(ram_address)))(17 downto 2);
                                elsif mode = '1' then
                                    buf1  (to_integer  (unsigned(ram_address)))  <=  "00"  &
buf1 (to_integer (unsigned(ram_address)))(17 downto 2);
                                end if;
                            when others =>
                        end case;
                    when s3 =>
                end case;
            end if;
    end process;

    V1: de2_wm8731_audio port map (
    clk                 => clk_25,
    reset_n             => '1',
    data                =>   data,
     counter            => counter,

    -- Audio interface signals
    AUD_ADCLRCK         => AUD_ADCLRCK,
    AUD_ADCDAT          => AUD_ADCDAT,
    AUD_DACLRCK         => AUD_DACLRCK,
    AUD_DACDAT          => AUD_DACDAT,
    AUD_BCLK            => AUD_BCLK
```

```
   );

   end architecture;
```

PS2 mouse drive vhdl code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity de2_ps2 is
    port(clkin,rst              :in std_logic;
            ps2_clk,ps2_data :inout std_logic;
            lefbut,rigbut,midbut: buffer std_logic;
              mouse_irq : buffer std_logic;
              s,l : out std_logic;
            x_mov,y_mov: out std_logic_vector(9 downto 0));
end entity;
architecture behav of de2_ps2 is
--
signal byte_cnt,delay :std_logic_vector(3 downto 0);
signal lef_latch,rig_latch,mid_latch :std_logic;
signal x_latch,y_latch :std_logic_vector(9 downto 0);
signal ps2_clk_in,ps2_clk_out,ps2_data_in,ps2_data_out :std_logic;
signal shift_reg :std_logic_vector(32 downto 0);
signal clk_div :std_logic_vector(8 downto 0);
signal clk,ce,de :std_logic;
signal dout_reg :std_logic_vector(9 downto 0);
signal cnt :std_logic_vector(7 downto 0);
--enable instruction 0xF4;
constant enable_data :std_logic_vector(9 downto 0):="0111101000";
signal ct :std_logic_vector(5 downto 0);
--
signal
ps2_clk_syn0,ps2_clk_syn1,ps2_dat_syn0,ps2_dat_syn1,dout_syn1,dout_syn0 :std_logic;
--type define
type st is (listen,pullclk,pulldata,trans);
signal state :st ;
signal lef_flag : std_logic;

begin

x_mov <= x_latch(9 downto 0);
y_mov <= y_latch(9 downto 0);
```

```
process
begin
   wait until clkin='1';
   clk_div<=clk_div+1;
end process;
clk<=clk_div(8);
--sampling frame data;
process(ps2_clk_in,state)
begin
    if state=listen then
         if ps2_clk_in'event and ps2_clk_in='0' then
              shift_reg(31 downto 0)<=shift_reg(32 downto 1);
              shift_reg(32)<=ps2_data_in;
         end if;
    end if;
end process;
--after the controller send a command to device ,it'll sendback a response,with 11 bit.
--so when ct's value    less than 16,no load activity.
process(ps2_clk_in,cnt,state)
begin
    if cnt="11111111" then
         ct<=(others=>'0');
    elsif state=listen then
         if rising_edge(ps2_clk_in) then
              ct<=ct+1;
         end if;
    end if;
end process;
---idle state counter, watchdog like activity;
process(clk)
begin
    if clk'event and clk='1' then
         if (ps2_data_in='1' and ps2_clk_in='1') then
              cnt<=cnt+1;
         else
              cnt<=(others=>'0');
         end if;
    end if;
end process;
----

----latch data from shift_reg;output is of 2's complement;
process(clk)
```

```
begin
    if clk'event and clk='1' then
            if mouse_irq = '1' then
                    mouse_irq <= '0';
            end if;
         if cnt="00011110" and (ct(5)='1'or ct(4)='1') then
            s <= shift_reg(1);
            l <= lefbut;
         lef_latch<=shift_reg(1);
         rig_latch<=shift_reg(2);
         mid_latch<=shift_reg(3);
            if shift_reg(1) = '1' then
                    mouse_irq <= '1';
            end if;
                    if ((x_latch >= 617 and shift_reg(5)='0') or (x_latch <= 2 and shift_reg(5)='1'))
then
                            x_latch <= x_latch;
                    else
                            x_latch <= x_latch + (shift_reg(5) & shift_reg(5) & shift_reg(18 downto 12)
& '0');
                            if ((x_latch <= 2 or x_latch >= 639) and shift_reg(5)='1') then
                                    x_latch <= "0000000010";
                            elsif ((x_latch >= 623) and shift_reg(5)='0') then
                                    x_latch <= "1001011111";
                            end if;
                    end if;
                    if ((y_latch >= 463 and shift_reg(6)='1') or (y_latch <= 2 and shift_reg(6)='0'))
then
                            y_latch <= y_latch;
                    else
                            y_latch <= y_latch + (not (shift_reg(6) & shift_reg(6) & shift_reg(29
downto 23) & '0') + "1");
                            if ((y_latch <= 2 or y_latch >= 479) and shift_reg(6)='0') then
                                    y_latch <= "0000000010";
                            elsif ((y_latch >= 463) and shift_reg(6)='1') then
                                    y_latch <= "0110111111";
                            end if;
                    end if;
            end if;
    end if;
end process;
lefbut<=lef_latch;
rigbut<=rig_latch;
midbut<=mid_latch;
```

```
--
ps2_clk_syn0<=ps2_clk;
ps2_dat_syn0<=ps2_data;
--tristate output for ps2_clk and ps2_data;
process(ps2_clk,ce,ps2_clk_out)
begin
    if ce='1' then
        ps2_clk<=ps2_clk_out;
    else
        ps2_clk<='Z';
    end if;
end process;
--tristate output of ps2_data;
process(ps2_data,de,ps2_data_out)
begin
    if de='1' then
        ps2_data<=ps2_data_out;
    else
        ps2_data<='Z';
    end if;
end process;
--synchronize
process
begin
    wait until clk='1';
    ps2_clk_syn1<=ps2_clk_syn0;
    ps2_dat_syn1<=ps2_dat_syn0;
    ps2_clk_in<=ps2_clk_syn1;
    ps2_data_in<=ps2_dat_syn1;
end process;
--tranceive 0xF4 to device when you press the rst button,additional debouncing work
process(clk)
begin
   if clk'event and clk='1' then
    case state is
    when listen =>    if rst='0' and cnt="11111111" then state<=pullclk;
                          end if;
                          ce<='0';de<='0';
    when pullclk => if delay="1100" then state<=pulldata;
                          end if;
                          ce<='1';de<='0';
    when pulldata=> state<=trans;
                          ce<='1';de<='1';
    when trans =>      if byte_cnt="1010" then state<=listen;
                         end if;
                         ce<='0';de<='1';
    when others    => state<=listen;
    end case;
  end if;
end process;
--pull clk low for about 100us
process(clk,state)
begin
    if state=pullclk then
        if clk'event and clk='1' then
            delay<=delay+1;
        end if;
    else
        delay<=(others=>'0');
    end if;
end process;
--tranceive data;adds up byte
process(ps2_clk_in,state)
begin
  if state=trans then
      if ps2_clk_in'event and ps2_clk_in='0' then
          dout_reg<='0'&dout_reg(9 downto 1);
      end if;
  else
      dout_reg<=enable_data;
  end if;
end process;
process(ps2_clk_in,state)
begin
if state=trans then
      if ps2_clk_in'event and ps2_clk_in='0' then
          byte_cnt<=byte_cnt+1;
      end if;
  else
      byte_cnt<=(others=>'0');
  end if;
end process;
--output assignment;
ps2_data_out<=dout_reg(0);
ps2_clk_out<='0';
--
```

```
end behav;
```

Timer vhdl code

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

entity timer 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 timer;

architecture rtl of timer is
    signal data        : std_logic_vector(15 downto 0);
    signal counter : unsigned(19 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
                    --if counter = "1111110000000000000000" then
                    --      counter <= (others => '0');
                    --else
                            counter <= counter +1 ;
                    --end if;
                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';
                end if;
            end if;
        end if;
    end process;

end rtl;
```

C code

helloword.c

```
#include <io.h>
#include <system.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/alt_irq.h> // the irq functions
#include <alt_types.h>
#include "start_interface.h"
#include "map1.h"
#include "map2.h"
#include "monster1.h"
#include "boss1.h"
#include "monster2.h"
#include "boss2.h"
#include "tower_glue.h"
#include "tower_green.h"
#include "tower_dart.h"
#include "tower_bomb.h"
#include "bullet.h"
#include "tower_glue_up.h"
#include "tower_green_up.h"
#include "tower_dart_up.h"
#include "tower_bomb_up.h"
#include "bullet_up.h"
#include "glue_effect.h"
#include "health_bar.h"
#include "select.h"
#include "button.h"
#include "tower_upgrade.h"
#include "bullet_effect.h"
#include "number.h"
#include "quit_restart.h"
#include "tower_select.h"
#include "tower_upgrade2.h"
#include "green_sound.h"
#include "glue_sound.h"
#include "dart_sound.h"
#include "bomb_sound.h"
#include "monster_sound.h"
#include "scream_sound.h"
```

```
#define IOWR_WRITE_DATA(base, offset, data) \
	IOWR_16DIRECT(base, (offset*2), data)
#define IORD_READ_DATA(base, offset) \
	IORD_16DIRECT(base, (offset*2))
#define IOWR_WRITE_SPEED(base, data) \
	IOWR_16DIRECT(base + 32, 0, data)
#define IOWR_SET_SPRITE(x, y, rom, sprite, phase) \
	IOWR_32DIRECT(SPRITE_CTRL_BASE, 4*((rom<<7) + sprite), (phase<<20) + (y<<10)
+ (x<<0))
#define IOWR_SET_SOUND(rom, length, sound, en) \
	IOWR_16DIRECT(SOUND_CTRL_BASE, 0, (length<<10) + (rom<<6) + (sound<<1) +
(en<<0))
#define SPRITE_IRQ_CLEAR() \
	IOWR_32DIRECT(SPRITE_CTRL_BASE, 0, (1<<31) + 1023 + (1023<<10) )
#define MOUSE_IRQ_DISABLE() \
	IOWR_32DIRECT(MOUSE_CTRL_BASE, 0, 1)
#define MOUSE_IRQ_ENABLE() \
	IOWR_32DIRECT(MOUSE_CTRL_BASE, 0, 0)
#define TIMER_IRQ_CLEAR() \
	IOWR_16DIRECT(TIMER_CTRL_BASE, 0, 0)

/******ram information******/
#define SLOW_MON	0
#define NORM_MON	3
#define FAST_MON	6
#define GLUE_TOW	9
#define GREEN_TOW	36
#define DART_TOW	63
#define BOMB_TOW	66
#define GLUE_BUL69
#define GREEN_BUL	70
#define DART_BUL72
#define BOMB_BUL	78
#define GLUE_IMA	159
#define SEL		162
#define FORBID		163
#define MOUSE		164
#define HEALTH		165
#define PAUSE		173
#define SPEED1		175
#define UNMUTE		177
#define UPGRADE		179
#define MAX		180
```

```
#define SELL          181
#define RANGE               182
#define GLUE_EFF1       183
#define GREEN_EFF1     185
#define BOMB_EFF1      187
#define GLUE_EFF2       183
#define GREEN_EFF2     186
#define BOMB_EFF2      188
#define BOSS_SLOW      189
#define BOSS_FAST       192
#define NUMBER_0       195
#define NUMBER_1       196
#define NUMBER_2       197
#define NUMBER_3       198
#define NUMBER_4       199
#define NUMBER_5       200
#define NUMBER_6       201
#define NUMBER_7       202
#define NUMBER_8       203
#define NUMBER_9       204
#define QUIT                205
#define UPGRADE2       208
#define SELL2        209
#define SELL3        210
#define TOW_SELECT1         211
#define TOW_SELECT2         212
#define TOW_SELECT3         213
#define TOW_SELECT4         214

/******game information******/
#define GENERATE_TIME1          15
#define GENERATE_TIME2          20
#define GENERATE_TIME3          25
#define GENERATE_TIME4          30
#define GAME_START                  0
#define GAME_MAP_READY                1
#define GAME_PLAY                   2
#define GAME_QUIT                   3
#define GAME_DONE                   4

int generate_time = 30;
int interface_state = GAME_START;
int tow_run = 0;
int tow_en = 0;
int mons_run = 0;
int run_count = 0;
int new_map = 0;
int current_map = 0;

/******path information******/
#define BG1_PATH       135
#define BG2_PATH       125

int background_path1[BG1_PATH][2] =
{{0,13},{0,14},
{1,5},{1,7},{1,8},{1,9},{1,13},{1,14},
{2,7},{2,8},{2,9},{2,10},{2,11},{2,12},
{3,2},{3,3},{3,4},{3,7},{3,8},{3,9},{3,10},{3,11},{3,12},
{4,2},{4,3},{4,4},{4,11},{4,12},
{5,2},{5,3},{5,4},{5,9},{5,11},{5,12},
{6,5},{6,6},{6,7},{6,8},{6,9},{6,10},{6,11},{6,12},
{7,5},{7,6},{7,7},{7,8},{7,9},{7,10},{7,11},{7,12},
{8,5},{8,6},{8,9},
{9,5},{9,6},{9,7},{9,8},{9,9},{9,10},{9,11},{9,12},
{10,2},{10,3},{10,4},{10,5},{10,6},{10,7},{10,8},{10,9},{10,10},{10,11},{10,12},
{11,2},{11,3},{11,4},{11,7},{11,11},{11,12},
{12,2},{12,3},{12,4},{12,11},{12,12},
{13,3},{13,4},{13,5},{13,6},{13,7},{13,8},{13,11},{13,12},{13,13},
{14,3},{14,4},{14,5},{14,6},{14,7},{14,8},{14,11},{14,12},
{15,3},{15,4},{15,7},{15,8},{15,11},{15,12},
{16,3},{16,4},{16,9},{16,11},{16,12},
{17,3},{17,4},{17,5},{17,6},{17,7},{17,8},{17,9},{17,10},{17,11},{17,12},
{18,3},{18,4},{18,5},{18,6},{18,7},{18,8},{18,9},{18,10},{18,11},{18,12},{18,13},{18,14},
{19,13},{19,14}
};

int background_path2[BG2_PATH][2] =
{{0,9},{0,10},{0,12},{0,13},{0,14},
{1,2},{1,3},{1,5},{1,6},
{2,2},{2,3},{2,5},{2,6},{2,7},{2,8},{2,9},{2,10},{2,11},{2,12},
{3,5},{3,6},{3,7},{3,8},{3,9},{3,10},{3,11},{3,12},
{3,7},{3,8},{3,9},{3,10},{3,11},{3,12},
{4,2},{4,3},{4,11},{4,12},
{5,2},{5,3},{5,7},{5,8},{5,9},{5,11},{5,12},
{6,7},{6,8},{6,9},{6,11},{6,12},
{7,2},{7,3},{7,7},{7,8},{7,9},{7,11},{7,12},
{8,2},{8,3},{8,4},{8,5},{8,6},{8,7},{8,8},{8,9},{8,10},{8,11},{8,12},
{9,3},{9,4},{9,5},{9,6},{9,7},{9,8},{9,9},{9,10},{9,11},{9,12},
{10,3},{10,4},
{11,3},{11,4},{11,5},{11,6},{11,7},{11,8},{11,9},{11,10},{11,11},{11,12},
{12,3},{12,4},{12,5},{12,6},{12,7},{12,8},{12,9},{12,10},{12,11},{12,12},
{13,11},{13,12},
{14,11},{14,12},
{15,11},{15,12},
{16,3},{16,4},{16,5},{16,6},{16,7},{16,8},{16,9},{16,10},{16,11},{16,12},
{17,3},{17,4},{17,5},{17,6},{17,7},{17,8},{17,9},{17,10},{17,11},{17,12}
};

/******info_bar information******/
int pause_sel = PAUSE;
int speed_sel = SPEED1;
int sound_sel = UNMUTE;
int pause_sel_x = 1023;
int speed_sel_x = 1023;
int sound_sel_x = 1023;
int button_sel_y = 1023;
int pause_en = 0;
int speed_level = 0;
int mute_en = 0;

/******tower information******/
#define GLUE_RANGE          68
#define GREEN_RANGE         68
#define DART_RANGE          88
#define BOMB_RANGE          88

#define TOW_SLOW            80
#define TOW_FAST            40

#define ATTACK_STATE0       0
#define ATTACK_STATE1       1
#define ATTACK_STATE2       2
#define ATTACK_STATE3       3
#define ATTACK_STATE4       4
#define ATTACK_STATE5       5
#define ATTACK_STATE6       6
#define ATTACK_STATE7       7
#define ATTACK_STATE8       8
#define ATTACK_STATE9       9
#define ATTACK_STATE10      10
#define ATTACK_STATE11      11
```

```
#define SOUND_GREEN      0
#define SOUND_GLUE       1
#define SOUND_DART       2
#define SOUND_BOMB       3

int tower_num = 0;
int current_tower = 0;
int attack_state = 0;

typedef struct
{
        int x;
        int y;
        int range;
        int cent_x;
        int cent_y;
        int step_x;
        int step_y;
        int speed;
        int image;
        alt_u8 level;
        alt_u8 busy;
        alt_u8 target;
        alt_u8 type;
        alt_u8 phase;
        alt_u8 sound_en;
        alt_u8 sound_type;
        alt_u8 sound_l;
}tower;

tower tow[20];

/******monster information******/
#define SLOW_SPEED1             3
#define NORM_SPEED1             5
#define FAST_SPEED1      7


#define HEALTH_LEVEL1    2
#define HEALTH_LEVEL2    4
#define HEALTH_LEVEL3    8
#define HEALTH_LEVEL4    12
#define HEALTH_LEVEL5    16
#define HEALTH_LEVEL6    20
```

```
#define HEALTH_LEVEL7      24
#define HEALTH_LEVEL8      28
#define HEALTH_LEVEL9      32
#define HEALTH_LEVEL10     50

#define SOUND_KILLED4
#define SOUND_SCREAM       5

int monster_num[20] = {5,5,5,8,8,8,10,10,10,13,13,13,13,10,10,10,8,8,5,5};
int current_monster = 0;

typedef struct
{
	int x;
	int y;
	int cent_x;
	int cent_y;
	int l;
	int image;
	int speed;
	int phase;
	alt_u8 type;
	int health;
	alt_u8 hight;
	alt_u8 kill;
	alt_u8 sound_en;
	alt_u8 sound_type;
	alt_u8 sound_l;
	alt_u8 sound_scream;
	alt_u8 sound_killed;
}monster;

monster mons[20];

/******bullet information******/
typedef struct
{
	int x;
	int y;
	int type;
	int image;
	alt_u8 phase;
}bullet;
```

```
bullet bul[20];

/******status information******/
#define GLUE_TIME           90

typedef struct
{
	int x;
	int y;
	alt_u8 time;
	int image;
}status;

status sta[20];

/******health bar information******/
#define HEALTH_TIME         90
typedef struct
{
	int x;
	int y;
	int image;
	alt_u8 en;
	alt_u8 busy;
	alt_u8 time;
	int health;
	alt_u8 damage;
}health;

health heal[20];

/******number information******/
#define WRITE           8
#define GREEN           4
#define YELLOW          0

int life = 9;
int wave = 0;
int money = 500;
int score = 0;
int color_life = YELLOW;
int color_wave = YELLOW;
int color_score = YELLOW;
int color_money = YELLOW;
int life_x[2] = {1023,1023};
int wave_x[2] = {1023,1023};
int money_x[4] = {1023,1023,1023,1023};
int score_x[4] = {1023,1023,1023,1023};
int life_y[2] = {1023,1023};
int wave_y[2] = {1023,1023};
int money_y[4] = {1023,1023,1023,1023};
int score_y[4] = {1023,1023,1023,1023};

int life_disp[2] = {0,0};
int wave_disp[2] = {0,0};
int score_disp[4] = {0,0,0,0};
int money_disp[4] = {0,0,0,0};

/******mouse information******/
#define INIT_STATE          0
#define GAME_STATE          1
#define FORBID_STATE 2
#define BUILD_STATE         3
#define UPGRADE_STATE       4
#define DONE_STATE          5

int mouse = 0;
int mouse_state = 0;
int left_button = 0;
int right_button = 0;
int x_position = 0;
int y_position = 0;
int mouse_sel_x = 1023;
int mouse_sel_y = 1023;
int tow_sel_x = 1023;
int tow_sel_y = 1023;
int upgrade_sel_x = 1023;
int upgrade_sel_y = 1023;
int sell_sel_x = 1023;
int sell_sel_y = 1023;
int range_sel_x1 = 1023;
int range_sel_y1 = 1023;
int range_sel_x2 = 1023;
int range_sel_y2 = 1023;
int range_sel_x3 = 1023;
int range_sel_y3 = 1023;
int range_sel_x4 = 1023;
int range_sel_y4 = 1023;
int mouse_sel = MOUSE;
int upgrade_sel = UPGRADE;
int sell_sel = SELL;
int tower_sel = TOW_SELECT1;

/******wave generation******/
int wave_gen[20][20][3] =
{
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL2},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL2},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL2},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL2},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL2}},
//WAVE 1
{{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL2},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL2},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL2},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL2},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL2}},
//WAVE 2
{{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL2},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL2},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL2},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL2},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL2}},
//WAVE 3
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL3},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL3},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL3},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL3},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL3},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL3},
{NORM_SPEED1, FAST_MON, HEALTH_LEVEL3},{NORM_SPEED1, FAST_MON,
HEALTH_LEVEL3},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL3},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL3},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3}},
//WAVE 4
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},
{NORM_SPEED1, FAST_MON, HEALTH_LEVEL8},{NORM_SPEED1, FAST_MON,
HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8}},
//WAVE 5
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL5},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL5},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},
{NORM_SPEED1, FAST_MON, HEALTH_LEVEL5},{NORM_SPEED1, FAST_MON,
HEALTH_LEVEL5},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5}},
//WAVE 6
{{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL3},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL3},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL3},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL3},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL3},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL3},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL3},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL3}},
//WAVE 7
{{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8}},
//WAVE 8
{{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5}},
//WAVE 9
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8}},
//WAVE 10
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL5},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL5},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL5}},
//WAVE 11
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL6},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL6},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL6},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL6},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL6},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL6},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL6},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL6},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL6},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL6},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL6},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL6},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL6}},
//WAVE 12
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL7},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL7},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL7},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL7},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL7},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL7},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL7},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL7},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL7},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL7},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL7},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL7},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL7}},
//WAVE 13
{{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8}},
//WAVE 14
{{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL5},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8}},
//WAVE 15
{{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL8},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL8},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL10},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL8},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL8}},
//WAVE 16
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL9},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL9},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL9},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL9},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL9},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL9},
{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL5},{SLOW_SPEED1, BOSS_SLOW,
HEALTH_LEVEL5},{FAST_SPEED1, BOSS_FAST, HEALTH_LEVEL5},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL9},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL9},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL9},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL9}},
//WAVE 17
{{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL9},{SLOW_SPEED1, SLOW_MON,
HEALTH_LEVEL9},{SLOW_SPEED1, SLOW_MON, HEALTH_LEVEL9},
{NORM_SPEED1, NORM_MON, HEALTH_LEVEL9},{NORM_SPEED1, NORM_MON,
HEALTH_LEVEL9},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL9},
{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL5},{SLOW_SPEED1, BOSS_SLOW,
HEALTH_LEVEL5},{FAST_SPEED1, BOSS_FAST, HEALTH_LEVEL5},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL9},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL9},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL9},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL9}},
//WAVE 18
{{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL10},{SLOW_SPEED1, BOSS_SLOW,
HEALTH_LEVEL10},{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL10},
{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL10},{SLOW_SPEED1, BOSS_SLOW,
HEALTH_LEVEL10},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL10},
{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL10},{SLOW_SPEED1, BOSS_SLOW,
HEALTH_LEVEL10},{FAST_SPEED1, BOSS_FAST, HEALTH_LEVEL10},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL10},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL10},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL10},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL10}},
//WAVE 19
{{FAST_SPEED1, BOSS_FAST, HEALTH_LEVEL10},{FAST_SPEED1, BOSS_FAST,
HEALTH_LEVEL10},{FAST_SPEED1, BOSS_FAST, HEALTH_LEVEL10},
{FAST_SPEED1, BOSS_FAST, HEALTH_LEVEL10}, {FAST_SPEED1, BOSS_FAST,
HEALTH_LEVEL10},{NORM_SPEED1, NORM_MON, HEALTH_LEVEL10},
{SLOW_SPEED1, BOSS_SLOW, HEALTH_LEVEL10},{SLOW_SPEED1, BOSS_SLOW,
HEALTH_LEVEL10},{FAST_SPEED1, BOSS_FAST, HEALTH_LEVEL10},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL10},{FAST_SPEED1, FAST_MON,
HEALTH_LEVEL10},{FAST_SPEED1, FAST_MON, HEALTH_LEVEL10},
{FAST_SPEED1, FAST_MON, HEALTH_LEVEL10}},
//WAVE 20
};




void sprite_clear(int *x, int *y)
{
	*x = 1023;
	*y = 1023;
}

void game_init()
```

```
{
	int i = 0;
	wave = 0;
	life = 9;
	pause_en = 0;
	speed_level = 0;
	mute_en = 0;
	tower_num = 0;
	pause_sel = PAUSE;
	speed_sel = SPEED1;
	sound_sel = UNMUTE;

	sprite_clear(&tow_sel_x, &tow_sel_y);
	sprite_clear(&mouse_sel_x, &mouse_sel_y);
	sprite_clear(&upgrade_sel_x, &upgrade_sel_y);
	sprite_clear(&sell_sel_x, &sell_sel_y);
	sprite_clear(&range_sel_x1, &range_sel_y1);
	sprite_clear(&range_sel_x2, &range_sel_y2);
	sprite_clear(&range_sel_x3, &range_sel_y3);
	sprite_clear(&range_sel_x4, &range_sel_y4);

	sprite_clear(&pause_sel_x, &button_sel_y);
	sprite_clear(&speed_sel_x, &button_sel_y);
	sprite_clear(&sound_sel_x, &button_sel_y);

	sprite_clear(&life_x[0], &life_y[0]);
	sprite_clear(&life_x[1], &life_y[1]);
	sprite_clear(&wave_x[0], &wave_y[0]);
	sprite_clear(&wave_x[1], &wave_y[1]);
	sprite_clear(&score_x[0], &score_y[0]);
	sprite_clear(&score_x[1], &score_y[1]);
	sprite_clear(&score_x[2], &score_y[2]);
	sprite_clear(&score_x[3], &score_y[3]);
	sprite_clear(&money_x[0], &money_y[0]);
	sprite_clear(&money_x[1], &money_y[1]);
	sprite_clear(&money_x[2], &money_y[2]);
	sprite_clear(&money_x[3], &money_y[3]);

	for(i = 0; i < 13; i++)
	{
		sprite_clear(&mons[i].x, &mons[i].y);
		sprite_clear(&tow[i].x, &tow[i].y);
		sprite_clear(&bul[i].x, &bul[i].y);
		sprite_clear(&sta[i].x, &sta[i].y);
```

```
                sprite_clear(&heal[i].x, &heal[i].y);
                mons[i].kill = 1;
        }
}

void monster_new(monster *m, health *h, status *s, int speed, int image, int health_level)
{
        m->l = 0;
        m->kill = 0;
        m->speed = speed;
        m->type = image;
        m->image = image;
        if(image <= 100)
                m->hight = 0;
        else
                m->hight = 8;
        m->health = health_level;
        m->sound_type = SOUND_KILLED;
        m->sound_l = 15;
        m->sound_scream = 0;
        m->sound_killed = 0;
        h->health = health_level;
        h->en = 0;
        h->busy = 0;
        h->time = 0;
        s->time = 0;
}

void tower_new(int x, int y, int range, int speed, int tow_image, int bul_image, int sound)
{
        score += 10;
        tow[tower_num].x = x;
        tow[tower_num].y = y;
        tow[tower_num].cent_x = x + 16;
        tow[tower_num].cent_y = y + 16;
        tow[tower_num].range = range;
        tow[tower_num].speed = speed/(speed_level+1);
        tow[tower_num].type = tow_image;
        tow[tower_num].image = tow_image;
        tow[tower_num].level = 1;
        tow[tower_num].busy = 0;
        tow[tower_num].target = 0;
        tow[tower_num].phase = 0;
        tow[tower_num].sound_type = sound;
```

```
	tow[tower_num].sound_l = sound;
	tow[tower_num].step_x = 0;
	tow[tower_num].step_y = 0;
	bul[tower_num].type = bul_image;
	bul[tower_num].image = bul_image;
	bul[tower_num].phase = 0;
	tower_num ++;
}

void tower_sell(int current)
{
	score += 5;
	int i;
	if (tow[current].type == GREEN_TOW && tow[current].level == 1)
		money += 90;
	else if (tow[current].type == GREEN_TOW    && tow[current].level == 2)
		money += 200;
	else if (tow[current].type == GLUE_TOW    && tow[current].level == 1)
		money += 90;
	else if (tow[current].type == GLUE_TOW    && tow[current].level == 2)
		money += 200;
	else if (tow[current].type == BOMB_TOW && tow[current].level == 1)
		money += 200;
	else if (tow[current].type == BOMB_TOW && tow[current].level == 2)
		money += 480;
	else if (tow[current].type == DART_TOW && tow[current].level == 1)
		money += 200;
	else if (tow[current].type == DART_TOW && tow[current].level == 2)
		money += 480;

	for(i = current; i < tower_num - 1; i++)
	{
		tow[i].x = tow[i+1].x;
		tow[i].y = tow[i+1].y;
		tow[i].cent_x = tow[i+1].cent_x;
		tow[i].cent_y = tow[i+1].cent_y;
		tow[i].range = tow[i+1].range;
		tow[i].speed = tow[i+1].speed;
		tow[i].type = tow[i+1].type;
		tow[i].image = tow[i+1].image;
		tow[i].level = tow[i+1].level;
		tow[i].busy = tow[i+1].busy;
		tow[i].target = tow[i+1].target;
		tow[i].phase = tow[i+1].phase;
		tow[i].sound_type = tow[i+1].sound_type;
		tow[i].sound_l = tow[i+1].sound_l;
		tow[i].step_x = tow[i+1].step_x;
		tow[i].step_y = tow[i+1].step_y;
		bul[i].type = bul[i+1].type;
		bul[i].image = bul[i+1].image;
		bul[i].phase = bul[i+1].phase;
	}
	sprite_clear(&tow[tower_num - 1].x, &tow[tower_num - 1].y);
	sprite_clear(&bul[tower_num - 1].x, &bul[tower_num - 1].y);
	tower_num --;
}

void monster_update(monster *m, status *s)
{
	if(m -> kill == 0)
	{
		if (m->image == m->type+2)
		{
			m->image = m->type;
			s->image = GLUE_IMA;
		}
		else
		{
			m->image ++;
			s->image ++;
		}
	}
}

void bullet_update(bullet *b)
{
	if(b->image == b->type + 5)
		b->image = b->type;
	else if(b->image < b->type + 5)
		b->image ++;
}

void health_update(health *h, monster *m)
{
	if(h->time != 0)
	{
		h->x = m->x;
		h->y = m->y - 4;
		if(h->time == HEALTH_TIME/(speed_level+1))
		{
			h->health --;

			float health_ratio = (float)h->health/(float)m->health;

			if(health_ratio > 0.875)
				h->image = HEALTH + 1;
			else if(health_ratio > 0.75)
				h->image = HEALTH + 1;
			else if(health_ratio > 0.625)
				h->image = HEALTH + 2;
			else if(health_ratio > 0.5)
				h->image = HEALTH + 3;
			else if(health_ratio > 0.375)
				h->image = HEALTH + 4;
			else if(health_ratio > 0.25)
				h->image = HEALTH + 5;
			else if(health_ratio > 0.125)
				h->image = HEALTH + 6;
			else if(health_ratio > 0.0)
				h->image = HEALTH + 7;

			else if(h->health == 0)
			{
				h->time = 0;
				m->l = 1490;
				m->kill = 1;
				m->sound_killed = 1;
				sprite_clear(&m->x, &m->y);
				sprite_clear(&h->x, &h->y);
			}
		}
		h->time --;
	}
	else
		sprite_clear(&h->x, &h->y);
}

void status_update(status *s, monster *m)
{
	if(s->time != 0)
	{
		if(m->x != 1023 && m->y != 1023)
```

```
        {
            s->x = m->x;
            s->y = m->y + m->hight;
        }
        else
            sprite_clear(&s->x, &s->y);
        s->time --;
        if(m->type == SLOW_MON)
            m->speed = SLOW_SPEED1 - 2;
        else if(m->type == NORM_MON)
            m->speed = NORM_SPEED1 - 2;
        else if(m->type == FAST_MON)
            m->speed = FAST_SPEED1 - 3;
        else if(m->type == BOSS_SLOW)
            m->speed = SLOW_SPEED1 - 2;
        else if(m->type == BOSS_FAST)
            m->speed = FAST_SPEED1 - 3;
    }
    else
    {
        sprite_clear(&s->x, &s->y);
        if(m->type == SLOW_MON)
            m->speed = SLOW_SPEED1;
        else if(m->type == NORM_MON)
            m->speed = NORM_SPEED1;
        else if(m->type == FAST_MON)
            m->speed = FAST_SPEED1;
        else if(m->type == BOSS_SLOW)
            m->speed = SLOW_SPEED1;
        else if(m->type == BOSS_FAST)
            m->speed = FAST_SPEED1;
    }
}

void monster_path1_update(monster *m, status *s, health *h)
{

    if(m->kill == 0)

    {
        m->l = m->l + m->speed * (speed_level + 1);

        if(m->l >= 0 && m->l <= 112)
        {
```

```
        m->kill = 0;
        m->phase = 0;
        m->cent_x = 96;
        m->cent_y = m->l + 262 - m->hight;
    }
    else if(m->l > 112 && m->l <= 240)
    {
        m->cent_x = m->l - 112 + 96;
        m->cent_y = 374 - m->hight;
    }
    else if(m->l > 240 && m->l <= 432)
    {
        m->cent_x = 224;
        m->cent_y = 432 - m->l + 182 - m->hight;
    }
    else if(m->l > 432 && m->l <= 528)
    {
        m->cent_x = m->l - 432 + 224;
        m->cent_y = 182 - m->hight;
    }
    else if(m->l > 528 && m->l <= 720)
    {
        m->cent_x = 320;
        m->cent_y = m->l -528 + 182 - m->hight;
    }
    else if(m->l > 720 && m->l <= 976)
    {
        m->cent_x = m->l -720 + 320;
        m->cent_y = 374 - m->hight;
    }
    else if(m->l > 976 && m->l <= 1232)
    {
        m->cent_x = 576;
        m->cent_y = 1232 - m->l + 118 - m->hight;
    }
    else if(m->l > 1232 && m->l <= 1360)
    {
        m->cent_x = 1360 - m->l + 448;
        m->cent_y = 118 - m->hight;
        m->phase = 1;
    }
    else if(m->l > 1360 && m->l <= 1475)
    {
        m->cent_x = 448;
```

```
            m->cent_y = m->l - 1360 + 118 - m->hight;
        }
        else if(m->l > 1475)
        {
            m->kill = 1;
            if (m->type > 100)
                life = life - 2;
            else
                life --;
            m->sound_scream = 1;
            m->cent_x = 1039;
            m->cent_y = 1039;
        }
        m->x = m->cent_x - 16;
        m->y = m->cent_y - 16;
    }
    else
    {
        m->x = 1023;
        m->y = 1023;
    }
}


void monster_path2_update(monster *m, status *s, health *h)
{

    if(m->kill == 0)

    {
        m->l = m->l + m->speed * (speed_level + 1);

        if(m->l >= 0 && m->l <= 192)
        {
            m->kill = 0;
            m->phase = 0;
            m->cent_x = 96;
            m->cent_y = m->l + 182 - m->hight;
        }
        else if(m->l > 192 && m->l <= 384)
        {
            m->cent_x = m->l - 192 + 96;
            m->cent_y = 374 - m->hight;
        }
        else if(m->l > 384 && m->l <= 640)
        {
                m->cent_x = 288;
                m->cent_y = 640 - m->l + 118 - m->hight;
        }
        else if(m->l > 640 && m->l <= 736)
        {
                m->cent_x = m->l - 640 + 288;
                m->cent_y = 118 - m->hight;
        }
        else if(m->l > 736 && m->l <= 992)
        {
                m->cent_x = 384;
                m->cent_y = m->l - 736 + 118 - m->hight;
        }
        else if(m->l > 992 && m->l <= 1152)
        {
                m->cent_x = m->l - 992 + 384;
                m->cent_y = 374 - m->hight;
        }
        else if(m->l > 1152 && m->l <= 1376)
        {
                m->cent_x = 544;
                m->cent_y = 1376 - m->l + 150 - m->hight;
        }
        else if(m->l > 1376)
        {
                m->kill = 1;
                m->sound_scream = 1;
                if (m->type > 100)
                        life = life - 2;
                else
                        life --;
                m->cent_x = 1039;
                m->cent_y = 1039;
        }
        m->x = m->cent_x - 16;
        m->y = m->cent_y - 16;
}
else
{
        m->x = 1023;
        m->y = 1023;
}
```

```
}

int main()
{
int i,j,k;
IOWR_WRITE_DATA(MUX_CTRL_BASE, 0, 0x0000);
printf("Welcome to save edward\n");
IOWR_WRITE_SPEED(SRAM_CTRL_BASE, 0x0040);

int offset = 0;

/******************background**********************/
for(i = offset; i < 153600+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, START_INTERFACE[i-offset]);
offset = i; printf("writing %x\n", offset);

/******************** monster *************************/
for(i = offset; i < 4608+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, MONSTER1[i-offset]);
offset = i; printf("writing %x\n", offset);

/**********************tower glue**********************/
for(i = offset; i < 13824+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_GLUE[i-offset]);
offset = i; printf("writing %x\n", offset);

/**********************tower green*********************/
for(i = offset; i < 13824+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_GREEN[i-offset]);
offset = i; printf("writing %x\n", offset);

/**********************tower dart **********************/
for(i = offset; i < 1536+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_DART[i-offset]);
offset = i; printf("writing %x\n", offset);

/**********************tower bomb **********************/
for(i = offset; i < 1536+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_BOMB[i-offset]);
offset = i; printf("writing %x\n", offset);

/**********************bullet************************/
for(i = offset; i < 7680+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, BULLET[i-offset]);
```

```
offset = i; printf("writing %x\n", offset);

/***********************tower glue up***********************/
for(i = offset; i < 13824+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_GLUE_UP[i-offset]);
offset = i; printf("writing %x\n", offset);

/***********************tower green up**********************/
for(i = offset; i < 13824+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_GREEN_UP[i-offset]);
offset = i; printf("writing %x\n", offset);

/***********************tower dart up***********************/
for(i = offset; i < 1536+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_DART_UP[i-offset]);
offset = i; printf("writing %x\n", offset);

/***********************tower bomb up**********************/
for(i = offset; i < 1536+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_BOMB_UP[i-offset]);
offset = i; printf("writing %x\n", offset);

/************************bullet up*************************/
for(i = offset; i < 7680+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, BULLET_UP[i-offset]);
offset = i; printf("writing %x\n", offset);

/*****************glue effect 1,2,3********************/
for(i = offset; i < 1536+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, GLUE_EFFECT[i-offset]);
offset = i; printf("writing %x\n", offset);

/********************select 1,2,3*********************/
for(i = offset; i < 1536+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, SELECT[i-offset]);
offset = i; printf("writing %x\n", offset);

/******************health bar 1-8********************/
for(i = offset; i < 512+offset; i++)
      IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, HEALTH_BAR[i-offset]);
offset = i; printf("writing %x\n", offset);

/*********************pause***********************/
for(i = offset; i < 3072+offset; i++)
```

```
	IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, BUTTON[i-offset]);
offset = i; printf("writing %x\n", offset);

/**********************tower upgrade*************************/
for(i = offset; i < 2048+offset; i++)
	IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_UPGRADE[i-offset]);
offset = i; printf("writing %x\n", offset);

/**********************bullet effect*************************/
for(i = offset; i < 3072+offset; i++)
	IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, BULLET_EFFECT[i-offset]);
offset = i; printf("writing %x\n", offset);

/************************* boss ***************************/
for(i = offset; i < 3840+offset; i++)
	IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, BOSS1[i-offset]);
offset = i; printf("writing %x\n", offset);

/*********************** number *************************/
for(i = offset; i < 640+offset; i++)
	IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, NUMBER[i-offset]);
offset = i; printf("writing %x\n", offset);

/********************* quit_restart *********************/
for(i = offset; i < 3072+offset; i++)
	IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, QUIT_RESTART[i-offset]);
offset = i; printf("writing %x\n", offset);

/**********************tower upgrade2*************************/
for(i = offset; i < 1536+offset; i++)
	IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_UPGRADE2[i-offset]);
offset = i; printf("writing %x\n", offset);

/******************tower select *********************/
for(i = 257952; i < 2048+257952; i++)
	IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, TOWER_SELECT[i-257952]);
offset = i; printf("writing %x\n", offset);

/******************tower sound *********************/
offset = 0;
for(i = offset; i < 1536 + offset; i++)
	IOWR_WRITE_DATA(RAM_CTRL_BASE, i, GREEN_SOUND[i-offset]);
offset = i;
```

```
for(i = offset; i < 1024 + offset; i++)
        IOWR_WRITE_DATA(RAM_CTRL_BASE, i, GLUE_SOUND[i-offset]);
offset = i;

for(i = offset; i < 7168 + offset; i++)
        IOWR_WRITE_DATA(RAM_CTRL_BASE, i, DART_SOUND[i-offset]);
offset = i;

for(i = offset; i < 5120 + offset; i++)
        IOWR_WRITE_DATA(RAM_CTRL_BASE, i, BOMB_SOUND[i-offset]);
offset = i;

for(i = offset; i < 4864 + offset; i++)
        IOWR_WRITE_DATA(RAM_CTRL_BASE, i, MONSTER_SOUND[i-offset]);
offset = i;

for(i = offset; i < 2560 + offset; i++)
        IOWR_WRITE_DATA(RAM_CTRL_BASE, i, SCREAM_SOUND[i-offset]);
offset = i;

IOWR_WRITE_DATA(MUX_CTRL_BASE, 0, 0x000c);

void sprite_interrupt_handler ();
void mouse_interrupt_handler ();
void audio_interrupt_handler ();
void timer_interrupt_handler ();

alt_ic_isr_register(SPRITE_CTRL_IRQ_INTERRUPT_CONTROLLER_ID,          SPRITE_CTRL_IRQ,
(void*)sprite_interrupt_handler, NULL, NULL);
alt_ic_isr_register(MOUSE_CTRL_IRQ_INTERRUPT_CONTROLLER_ID,        MOUSE_CTRL_IRQ,
(void*)mouse_interrupt_handler, NULL, NULL);
alt_ic_isr_register(TIMER_CTRL_IRQ_INTERRUPT_CONTROLLER_ID,           TIMER_CTRL_IRQ,
(void*)timer_interrupt_handler, NULL, NULL);
alt_ic_isr_register(SOUND_CTRL_IRQ_INTERRUPT_CONTROLLER_ID,         SOUND_CTRL_IRQ,
(void*)audio_interrupt_handler, NULL, NULL);

for(;;)
{

        switch (interface_state)
        {
                case GAME_START:
                        game_init();
                break;
                case GAME_MAP_READY:
                        game_init();
                        color_life = YELLOW;
                        color_score = YELLOW;

                        IOWR_WRITE_DATA(MUX_CTRL_BASE, 0, 0x0000);
                        if(current_map == 1)
                        {
                            for(i = 0; i < 153600; i++)
                                IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, MAP1[i]);

                            for(i = 153600; i < 4608+153600; i++)
                                IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, MONSTER1[i-153600]);

                            for(i = 246784; i < 3840+246784; i++)
                                IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, BOSS1[i-246784]);

                            money = 500;
                            score = 0;
                            life_x[0] = 460;
                            life_y[0] = 285;
                        }
                        else if(current_map == 2)
                        {
                            for(i = 0; i < 153600; i++)
                                IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, MAP2[i]);

                            for(i = 153600; i < 4608+153600; i++)
                                IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, MONSTER2[i-153600]);

                            for(i = 246784; i < 3840+246784; i++)
                                IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, BOSS2[i-246784]);

                            money = 500;
                            score = 0;
                            life_x[0] = 505;
                            life_y[0] = 160;
                        }
                        IOWR_WRITE_DATA(MUX_CTRL_BASE, 0, 0x000c);

                        for(i = 0; i < 13; i++)
                        {
                            sprite_clear(&tow[i].x, &tow[i].y);
                            sprite_clear(&bul[i].x, &bul[i].y);
```

```
                        sprite_clear(&sta[i].x, &sta[i].y);
                        sprite_clear(&heal[i].x, &heal[i].y);
                        mons[i].kill = 1;
                    }
                    wave_x[1] = 340;
                    wave_y[1] = 28;
                    money_x[1] = 67;
                    money_y[1] = 16;
                    money_x[2] = 77;
                    money_y[2] = 16;
                    money_x[3] = 87;
                    money_y[3] = 16;
                    score_x[3] = 360;
                    score_y[3] = 5;
                    color_life = GREEN;
                    color_score = WRITE;
                    interface_state = GAME_PLAY;
                    mouse_state = GAME_STATE;
                break;
                case GAME_PLAY:

                    //display button
                    pause_sel_x = 600;
                    speed_sel_x = 510;
                    sound_sel_x = 420;
                    button_sel_y = 8;

                    /**************************generate                          new
wave**************************/
                    if(mons_run == 1)
                    {
                        mons_run = 0;

                        //dynamic display the monster status
                        for(i = 0; i < monster_num[wave-1]; i++)
                            monster_update(&mons[i], &sta[i]);

                        if(pause_en == 0)
                        {
                            if(wave < 20){
                                int die = 0;
                                for(i = 0; i < monster_num[wave-1]; i++)
                                {
                                    if(mons[i].kill == 1)
```

```
                                        die ++;
                                }
                                if(die == monster_num[wave-1])
                                {
                                        wave ++;
                                        generate_time = GENERATE_TIME1/(speed_level + 1);
                                        monster_new(&mons[0],          &heal[0],          &sta[0],
wave_gen[wave-1][0][0], wave_gen[wave-1][0][1], wave_gen[wave-1][0][2]);
                                        current_monster = 1;
                                }

                                if(generate_time == 0)
                                {
                                        if(current_monster < monster_num[wave-1])
                                                monster_new(&mons[current_monster],
&heal[current_monster], &sta[current_monster],
                                                        wave_gen[wave-1][current_monster][0],
wave_gen[wave-1][current_monster][1], wave_gen[wave-1][current_monster][2]);
                                        current_monster ++;
                                        generate_time = GENERATE_TIME1/(speed_level + 1);
                                }
                                else
                                        generate_time --;

                                if(current_map == 1)
                                {
                                        for(i = 0; i < monster_num[wave-1]; i++)
                                                monster_path1_update(&mons[i], &sta[i], &heal[i]);
                                }
                                else if(current_map == 2)
                                {
                                        for(i = 0; i < monster_num[wave-1]; i++)
                                                monster_path2_update(&mons[i], &sta[i], &heal[i]);
                                }
                        }
                        else if(wave == 20){
                                int die = 0;
                                for(i = 0; i < monster_num[wave-1]; i++)
                                {
                                        if(mons[i].kill == 1)
                                                die ++;
                                }
                                if(die == monster_num[wave-1])
                                {
                                        interface_state = GAME_QUIT;
                              }

                              if(generate_time == 0)
                              {
                                        if(current_monster < monster_num[wave-1])
                                                  monster_new(&mons[current_monster],
&heal[current_monster], &sta[current_monster],
                                                                      wave_gen[wave-1][current_monster][0],
wave_gen[wave-1][current_monster][1], wave_gen[wave-1][current_monster][2]);
                                        current_monster ++;
                                        generate_time = GENERATE_TIME1/(speed_level + 1);
                              }
                              else
                                        generate_time --;

                              if(current_map == 1)
                              {
                                        for(i = 0; i < monster_num[wave-1]; i++)
                                                  monster_path1_update(&mons[i], &sta[i], &heal[i]);
                              }
                              else if(current_map == 2)
                              {
                                        for(i = 0; i < monster_num[wave-1]; i++)
                                                  monster_path2_update(&mons[i], &sta[i], &heal[i]);
                              }
                    }
          }
     }

     /**************************dissipate                                    the
wave**************************/

     if(tow_run == 1)
     {
          tow_run = 0;
          if(tow_en == 1)
                tow_en = 0;
          else
                tow_en ++;

          //dynamic display the bullet status
          for(i = 0; i < tower_num; i++)
          {
```

```
                    if((tow[i].type == DART_TOW || tow[i].type == BOMB_TOW) &&
tow_en == 1)
                        bullet_update(&bul[i]);
                }

                for(i = 0; i < monster_num[wave-1]; i++)
                {
                    status_update(&sta[i], &mons[i]);
                    health_update(&heal[i], &mons[i]);

                    if(mons[i].sound_scream == 1)
                    {
                        mons[i].sound_scream = 0;
                        mons[i].sound_type = SOUND_SCREAM;
                        if(mute_en == 0)
                            mons[i].sound_en = 1 - mons[i].sound_en;
                        mons[i].sound_l = 15;
                    }
                    if(mons[i].sound_killed == 1)
                    {
                        mons[i].sound_killed = 0;
                        mons[i].sound_type = SOUND_KILLED;
                        if(mute_en == 0)
                            mons[i].sound_en = 1 - mons[i].sound_en;
                        mons[i].sound_l = 12;
                        if (mons[i].type > 100)
                        {
                            money += 48;
                            score += 18;
                        }
                        else
                        {
                            money += 24;
                            score += 6;
                        }
                    }
                }

                if(pause_en == 0)
                {
                    for(j = 0; j < tower_num; j++)
                    {
                        int upgrade = (tow[j].level-1)*75;
                        int attack_speed = 0;
                        int attack_interval = 0;
                        int sound_interval = 0;
                        int game_speed = speed_level + 1;

                        if(tow[j].type != DART_TOW)
                            attack_speed = TOW_FAST/tow[j].level;
                        else
                            attack_speed = TOW_SLOW/tow[j].level;

                        attack_interval = 4/(game_speed*tow[j].level);
                        sound_interval = tow[j].level + speed_level - 1;

                        //find the foremost monster that enter the attack range(when
the tower is not busy)
                        if(tow[j].busy == 0)
                        {
                            int foremost = 0;
                            int foremost_l = 0;
                            int attack_true = 0;

                            //clear the bullet if it is not busy
                            if(tow[j].type    ==    GLUE_TOW    ||    tow[j].type    ==
GREEN_TOW)
                                tow[j].image = tow[j].type + (tow[j].image - tow[j].type
- upgrade)%9 + upgrade;
                            else
                                tow[j].image = tow[j].type + upgrade;

                            sprite_clear(&bul[j].x, &bul[j].y);

                            for(i = 0; i < monster_num[wave-1]; i++)
                            {
                                int  x_length  =  abs((mons[i].cent_x-tow[j].cent_x))  *
abs((mons[i].cent_x-tow[j].cent_x));
                                int  y_length  =  abs((mons[i].cent_y-tow[j].cent_y))  *
abs((mons[i].cent_y-tow[j].cent_y));
                                if(x_length + y_length <= tow[j].range*tow[j].range)
                                {
                                    if(mons[i].l > foremost_l)
                                    {
                                        if((tow[j].type  ==  GLUE_TOW  ||  tow[j].type
== GREEN_TOW || tow[j].type == DART_TOW || tow[j].type == BOMB_TOW) && mons[i].kill
== 0)
                                        {
                                            foremost_l = mons[i].l;
                                            foremost = i;
                                            attack_true = 1;
                                        }
                                    }
                                }
                            }
                            if(attack_true == 1)
                            {
                                tow[j].target = foremost;
                                if(tow[j].type == DART_TOW)
                                {
                                    float                       x_diff                     =
mons[foremost].cent_x-tow[j].cent_x;
                                    float                       y_diff                     =
mons[foremost].cent_y-tow[j].cent_y;
                                    tow[j].step_x                                          =
(int)((5*x_diff)/sqrt(((x_diff*x_diff)+(y_diff*y_diff))) );
                                    tow[j].step_y                                          =
(int)((5*y_diff)/sqrt(((x_diff*x_diff)+(y_diff*y_diff))) );
                                }
                                tow[j].busy = 1;
                            }
                        }

                        else
                        {
                            i = tow[j].target;
                            int state = 0;

                            //judge the phase of the tower
                            if(tow[j].type     ==     GLUE_TOW     ||     tow[j].type     ==
GREEN_TOW)
                            {
                                if(mons[i].cent_x       <       tow[j].cent_x              &&
mons[i].cent_y > tow[j].cent_y)
                                    tow[j].phase = 0;
                                else     if(mons[i].cent_x     <     tow[j].cent_x         &&
mons[i].cent_y < tow[j].cent_y)
                                    tow[j].phase = 2;
                                else     if(mons[i].cent_x     >     tow[j].cent_x         &&
mons[i].cent_y < tow[j].cent_y)
                                    tow[j].phase = 3;
                                else     if(mons[i].cent_x     >     tow[j].cent_x         &&
mons[i].cent_y > tow[j].cent_y)
                                        tow[j].phase = 1;
                              }
                              else if(tow[j].type == DART_TOW || tow[j].type ==
BOMB_TOW)
                                   tow[j].phase = 0;

                              float x_diff = abs(mons[i].cent_x-tow[j].cent_x);
                              float y_diff = abs(mons[i].cent_y-tow[j].cent_y);

                              if(tow[j].type != DART_TOW)
                              {
                                   if(tow[j].level == 1 && game_speed == 1)
                                   {
                                        if(tow[j].speed/attack_interval > 7)
                                             attack_state = ATTACK_STATE0;
                                        else if(tow[j].speed/attack_interval > 6)
                                             attack_state = ATTACK_STATE1;
                                        else if(tow[j].speed/attack_interval > 5)
                                             attack_state = ATTACK_STATE4;
                                        else if(tow[j].speed/attack_interval > 4)
                                             attack_state = ATTACK_STATE5;
                                        else if(tow[j].speed/attack_interval > 3)
                                             attack_state = ATTACK_STATE6;
                                        else if(tow[j].speed/attack_interval > 2)
                                             attack_state = ATTACK_STATE7;
                                        else if(tow[j].speed/attack_interval == 2)
                                             attack_state = ATTACK_STATE8;
                                        else if(tow[j].speed/attack_interval > 1)
                                             attack_state = ATTACK_STATE9;
                                        else if(tow[j].speed/attack_interval > 0)
                                             attack_state = ATTACK_STATE10;
                                        else if(tow[j].speed/attack_interval == 0)
                                             attack_state = ATTACK_STATE11;
                                   }
                                   else if((tow[j].level == 2 && game_speed == 1) ||
(tow[j].level == 1 && game_speed == 2))
                                   {
                                        if(tow[j].speed > 14)
                                             attack_state = ATTACK_STATE0;
                                        else if(tow[j].speed > 12)
                                             attack_state = ATTACK_STATE1;
                                        else if(tow[j].speed > 10)
                                             attack_state = ATTACK_STATE4;
```

```
                    else if(tow[j].speed > 8)
                        attack_state = ATTACK_STATE5;
                    else if(tow[j].speed > 6)
                        attack_state = ATTACK_STATE6;
                    else if(tow[j].speed > 4)
                        attack_state = ATTACK_STATE7;
                    else if(tow[j].speed == 4)
                        attack_state = ATTACK_STATE8;
                    else if(tow[j].speed > 2)
                        attack_state = ATTACK_STATE9;
                    else if(tow[j].speed > 0)
                        attack_state = ATTACK_STATE10;
                    else if(tow[j].speed == 0)
                        attack_state = ATTACK_STATE11;
                }
                else if(tow[j].level == 2 && game_speed == 2)
                {
                    if(tow[j].speed > 7)
                        attack_state = ATTACK_STATE0;
                    else if(tow[j].speed > 6)
                        attack_state = ATTACK_STATE1;
                    else if(tow[j].speed > 5)
                        attack_state = ATTACK_STATE4;
                    else if(tow[j].speed > 4)
                        attack_state = ATTACK_STATE5;
                    else if(tow[j].speed > 3)
                        attack_state = ATTACK_STATE6;
                    else if(tow[j].speed > 2)
                        attack_state = ATTACK_STATE7;
                    else if(tow[j].speed == 2)
                        attack_state = ATTACK_STATE8;
                    else if(tow[j].speed > 1)
                        attack_state = ATTACK_STATE9;
                    else if(tow[j].speed > 0)
                        attack_state = ATTACK_STATE10;
                    else if(tow[j].speed == 0)
                        attack_state = ATTACK_STATE11;
                }

            }
            if(tow[j].type == DART_TOW)
            {
                if(tow[j].level == 1 && game_speed == 1)
                {
```

```
        if(tow[j].speed/attack_interval > 19)
            attack_state = ATTACK_STATE0;
        else if(tow[j].speed/attack_interval > 18)
            attack_state = ATTACK_STATE1;
        else if(tow[j].speed/attack_interval == 18)
            attack_state = ATTACK_STATE2;
        else if(tow[j].speed/attack_interval > 0)
            attack_state = ATTACK_STATE3;
        else if(tow[j].speed/attack_interval == 0)
            attack_state = ATTACK_STATE11;
    }
    else if((tow[j].level == 2 && game_speed == 1) || (tow[j].level == 1 && game_speed == 2))
    {
        if(tow[j].speed > 38)
            attack_state = ATTACK_STATE0;
        else if(tow[j].speed > 36)
            attack_state = ATTACK_STATE1;
        else if(tow[j].speed == 36)
            attack_state = ATTACK_STATE2;
        else if(tow[j].speed/2 > 0)
            attack_state = ATTACK_STATE3;
        else if(tow[j].speed/2 == 0)
            attack_state = ATTACK_STATE11;
    }
    else if(tow[j].level == 2 && game_speed == 2)
    {
        if(tow[j].speed > 19)
            attack_state = ATTACK_STATE0;
        else if(tow[j].speed > 18)
            attack_state = ATTACK_STATE1;
        else if(tow[j].speed == 18)
            attack_state = ATTACK_STATE2;
        else if(tow[j].speed > 0)
            attack_state = ATTACK_STATE3;
        else if(tow[j].speed == 0)
            attack_state = ATTACK_STATE11;
    }
}

switch (attack_state)
{
    case ATTACK_STATE0:
        state = 9;
```

```
                                        break;
                                        case ATTACK_STATE1:
                                            state = 18;
                                        break;
                                        case ATTACK_STATE2:
                                            state = 0;
                                            if(mute_en == 0)
                                                tow[j].sound_en = 1 - tow[j].sound_en;
                                            tow[j].sound_l      =      tow[j].sound_type      +
sound_interval*4;
                                            bul[j].x = tow[j].x;
                                            bul[j].y = tow[j].y;
                                        break;
                                        case ATTACK_STATE3:
                                            bul[j].x                                      +=
tow[j].step_x*(game_speed*tow[j].level);
                                            bul[j].y                                      +=
tow[j].step_y*(game_speed*tow[j].level);
                                            if(bul[j].x <= 8 || bul[j].x >= 632 || bul[j].y <= 48
|| bul[j].y >= 472)
                                                tow[j].speed = 1;
                                            else
                                            {
                                                for(k = 0; k < monster_num[wave-1]; k++)
                                                {
                                                    if(bul[j].x >= mons[k].x - 32 && bul[j].x <=
mons[k].x + 32
                                                    &&   bul[j].y   <=   mons[k].y   +   32   &&
bul[j].y >= mons[k].y - 32)
                                                        heal[k].time                          =
HEALTH_TIME/(speed_level + 1);
                                                }
                                            }
                                        break;
                                        case ATTACK_STATE4:
                                            state = 0;
                                            bul[j].x                                          =
(int)(((mons[i].cent_x-tow[j].cent_x)*2/6) + tow[j].cent_x) - 16;
                                            bul[j].y                                          =
(int)(((mons[i].cent_y-tow[j].cent_y)*2/6) + tow[j].cent_y) - 16;
                                        break;
                                        case ATTACK_STATE5:
                                            bul[j].x                                          =
(int)(((mons[i].cent_x-tow[j].cent_x)*3/6) + tow[j].cent_x) - 16;
```

```
                                    bul[j].y                                    =
(int)(((mons[i].cent_y-tow[j].cent_y)*3/6) + tow[j].cent_y) - 16;
                                break;
                                case ATTACK_STATE6:
                                    bul[j].x                                    =
(int)(((mons[i].cent_x-tow[j].cent_x)*4/6) + tow[j].cent_x) - 16;
                                    bul[j].y                                    =
(int)(((mons[i].cent_y-tow[j].cent_y)*4/6) + tow[j].cent_y) - 16;
                                break;
                                case ATTACK_STATE7:
                                    bul[j].x                                    =
(int)(((mons[i].cent_x-tow[j].cent_x)*5/6) + tow[j].cent_x) - 16;
                                    bul[j].y                                    =
(int)(((mons[i].cent_y-tow[j].cent_y)*5/6) + tow[j].cent_y) - 16;
                                break;
                                case ATTACK_STATE8:
                                    bul[j].x = mons[tow[j].target].x;
                                    bul[j].y = mons[tow[j].target].y;
                                    if(tow[j].type == GLUE_TOW)
                                        bul[j].image =      GLUE_EFF1;
                                    else if(tow[j].type == GREEN_TOW)
                                        bul[j].image =      GREEN_EFF1;
                                    else if(tow[j].type == BOMB_TOW)
                                        bul[j].image =      BOMB_EFF1;

                                    if(mute_en == 0 && tow[j].type != DART_TOW)
                                        tow[j].sound_en = 1 - tow[j].sound_en;
                                    tow[j].sound_l     =     tow[j].sound_type     +
sound_interval*4;

                                    if(tow[j].type  ==  GREEN_TOW  ||  tow[j].type  ==
DART_TOW)
                                        heal[tow[j].target].time                =
HEALTH_TIME/(speed_level + 1);
                                    else if(tow[j].type == BOMB_TOW)
                                    {
                                        for(k = 0; k < monster_num[wave-1]; k++)
                                        {
                                            if(abs(mons[tow[j].target].l  -  mons[k].l)
<= 48)
                                                heal[k].time                    =
HEALTH_TIME/(speed_level + 1);
                                        }
                                    }
```

```
                                        break;
                                        case ATTACK_STATE9:
                                            bul[j].x = mons[tow[j].target].x;
                                            bul[j].y = mons[tow[j].target].y;
                                            if(tow[j].type == GLUE_TOW)
                                                bul[j].image =      GLUE_EFF1;
                                            else if(tow[j].type == GREEN_TOW)
                                                bul[j].image =      GREEN_EFF1;
                                            else if(tow[j].type == BOMB_TOW)
                                                bul[j].image =      BOMB_EFF1;
                                        break;
                                        case ATTACK_STATE10:
                                            if(tow[j].type == GLUE_TOW)
                                                bul[j].image =      GLUE_EFF2;
                                            else if(tow[j].type == GREEN_TOW)
                                                bul[j].image =      GREEN_EFF2;
                                            else if(tow[j].type == BOMB_TOW)
                                                bul[j].image =      BOMB_EFF2;
                                        break;
                                        case ATTACK_STATE11:
                                            //if(mute_en == 0 && tow[j].type != DART_TOW)
                                            //    tow[j].sound_en = 1 - tow[j].sound_en;
                                            //tow[j].sound_l      =      tow[j].sound_type      +
sound_interval*4;

                                            tow[j].busy = 0;
                                            bul[j].image = bul[j].type;
                                            tow[j].speed    = attack_speed/game_speed+1;
                                            if(tow[j].type == GLUE_TOW)
                                                sta[tow[j].target].time = GLUE_TIME;
                                            sprite_clear(&bul[j].x, &bul[j].y);
                                        break;
                                    }
                                    tow[j].speed --;

                                    float angle1 = x_diff/y_diff;
                                    float angle2 = y_diff/x_diff;
                                    if(tow[j].type    ==    GLUE_TOW    ||    tow[j].type    ==
GREEN_TOW)
                                    {
                                        if(angle1 >= 0.0 && angle1 <= 0.0875)
                                            tow[j].image = tow[j].type + state + upgrade;
                                        else if(angle1 >= 0.0875 && angle1 <= 0.176)
                                            tow[j].image = tow[j].type + 1 + state + upgrade;
                                        else if(angle1 >= 0.176 && angle1 <= 0.466)
```

```
                                                tow[j].image = tow[j].type + 2 + state + upgrade;
                                            else if(angle1 >= 0.466 && angle1 <= 0.70)
                                                tow[j].image = tow[j].type + 3 + state + upgrade;
                                            else if(angle1 >= 0.70 && angle1 <= 1)
                                                tow[j].image = tow[j].type + 4 + state + upgrade;
                                            else if(angle2 >= 0.70 && angle2 <= 1)
                                                tow[j].image = tow[j].type + 4 + state + upgrade;
                                            else if(angle2 >= 0.466 && angle2 <= 0.70)
                                                tow[j].image = tow[j].type + 5 + state + upgrade;
                                            else if(angle2 >= 0.176 && angle2 <= 0.466)
                                                tow[j].image = tow[j].type + 6 + state + upgrade;
                                            else if(angle2 >= 0.0875 && angle2 <= 0.176)
                                                tow[j].image = tow[j].type + 7 + state + upgrade;
                                            else if(angle2 >= 0.0 && angle2 <= 0.0875)
                                                tow[j].image = tow[j].type + 8 + state + upgrade;
                                        }
                                        else    if(tow[j].type    ==    DART_TOW    ||    tow[j].type    ==
BOMB_TOW)
                                            tow[j].image = tow[j].type + state/9 + upgrade;

                                        if(mons[tow[j].target].kill    ==    1    &&    tow[j].type    !=
DART_TOW)
                                        {
                                            tow[j].busy = 0;
                                            bul[j].image = bul[j].type;
                                            tow[j].speed    = attack_speed/game_speed;
                                            heal[tow[j].target].en = 0;
                                            sprite_clear(&bul[j].x, &bul[j].y);
                                            sprite_clear(&sta[tow[j].target].x,
&sta[tow[j].target].y);
                                        }
                                    }
                                }
                            }
                        }

        wave_disp[0] = (int)wave/10 + NUMBER_0;
        wave_disp[1] = wave%10 + NUMBER_0;

        life_disp[0] = life + NUMBER_0;

        money_disp[0] = (int)(money/1000) + NUMBER_0;
        money_disp[1] = (int)((money - 1000 * (int)(money/1000))/100) + NUMBER_0;
        money_disp[2] = (int)((money - 100 * (int)(money/100))/10) + NUMBER_0;
```

```
money_disp[3] = money%10 + NUMBER_0;

score_disp[0] = (int)(score/1000) + NUMBER_0;
score_disp[1] = (int)((score - 1000 * (int)(score/1000))/100) + NUMBER_0;
score_disp[2] = (int)((score - 100 * (int)(score/100))/10) + NUMBER_0;
score_disp[3] = score%10 + NUMBER_0;

if(wave >= 10)
{
    wave_x[0] = 330;
    wave_y[0] = 28;
}
if(money >= 1000)
{
    money_x[0] = 57;
    money_y[0] = 16;
    money_x[1] = 67;
    money_y[1] = 16;
    money_x[2] = 77;
    money_y[2] = 16;
    money_x[3] = 87;
    money_y[3] = 16;
}
else if(money >= 100)
{
    money_x[0] = 1023;
    money_y[0] = 1023;
    money_x[1] = 57;
    money_y[1] = 16;
    money_x[2] = 67;
    money_y[2] = 16;
    money_x[3] = 77;
    money_y[3] = 16;
}
else if(money >= 10)
{
    money_x[0] = 1023;
    money_y[0] = 1023;
    money_x[1] = 1023;
    money_y[1] = 1023;
    money_x[2] = 57;
    money_y[2] = 16;
    money_x[3] = 67;
    money_y[3] = 16;
```

```
        }
        else
        {
            money_x[0] = 1023;
            money_y[0] = 1023;
            money_x[1] = 1023;
            money_y[1] = 1023;
            money_x[2] = 1023;
            money_y[2] = 1023;
            money_x[3] = 57;
            money_y[3] = 16;
        }
        if(score >= 1000)
        {
            score_x[0] = 330;
            score_y[0] = 5;
            score_x[1] = 340;
            score_y[1] = 5;
            score_x[2] = 350;
            score_y[2] = 5;
            score_x[3] = 360;
            score_y[3] = 5;
        }
        else if(score >= 100)
        {
            score_x[0] = 1023;
            score_y[0] = 1023;
            score_x[1] = 330;
            score_y[1] = 5;
            score_x[2] = 340;
            score_y[2] = 5;
            score_x[3] = 350;
            score_y[3] = 5;
        }
        else if(score >= 10)
        {
            score_x[0] = 1023;
            score_y[0] = 1023;
            score_x[1] = 1023;
            score_y[1] = 1023;
            score_x[2] = 330;
            score_y[2] = 5;
            score_x[3] = 340;
            score_y[3] = 5;
            }
            else
            {
                score_x[0] = 1023;
                score_y[0] = 1023;
                score_x[1] = 1023;
                score_y[1] = 1023;
                score_x[2] = 1023;
                score_y[2] = 1023;
                score_x[3] = 330;
                score_y[3] = 5;
            }
            if(life == 0)
                interface_state = GAME_QUIT;
            break;
            case GAME_QUIT:
                upgrade_sel_x = 272;
                upgrade_sel_y = 208;
                upgrade_sel = QUIT;
                sell_sel_x = 304;
                sell_sel_y = 208;
                sell_sel = QUIT + 1;
                mouse_sel_x = 336;
                mouse_sel_y = 208;
                mouse_sel = QUIT + 2;
                mouse_state = DONE_STATE;
            break;
            case GAME_DONE:
                color_life = YELLOW;
                color_score = YELLOW;
                IOWR_WRITE_DATA(MUX_CTRL_BASE, 0, 0x0000);
                for(i = 0; i < 153600; i++)
                    IOWR_WRITE_DATA(SRAM_CTRL_BASE, i, START_INTERFACE[i]);
                IOWR_WRITE_DATA(MUX_CTRL_BASE, 0, 0x000c);
                mouse_state = INIT_STATE;
                interface_state = GAME_START;
            break;
        }
}

printf("Goodbye\n");
return 0;
}
```

```
void sprite_interrupt_handler()
{
	SPRITE_IRQ_CLEAR();

	mouse = IORD_32DIRECT(MOUSE_CTRL_BASE, 0);
	x_position = (mouse >> 3) & 0x3ff;
	y_position = (mouse >> 13) & 0x3ff;

	/************************display mouse*****************************/
	IOWR_SET_SPRITE(x_position, y_position, MOUSE, 88, 0);

	/***********************display select****************************/
	IOWR_SET_SPRITE(mouse_sel_x, mouse_sel_y, mouse_sel, 87, 0);
	IOWR_SET_SPRITE(tow_sel_x, tow_sel_y, tower_sel, 86, 0);

	/************************display button****************************/
	IOWR_SET_SPRITE(pause_sel_x, button_sel_y, pause_sel, 85, 0);
	IOWR_SET_SPRITE(speed_sel_x, button_sel_y, speed_sel, 84, 0);
	IOWR_SET_SPRITE(sound_sel_x, button_sel_y, sound_sel, 83, 0);

	/************************display button****************************/
	IOWR_SET_SPRITE(upgrade_sel_x, upgrade_sel_y, upgrade_sel, 82, 0);
	IOWR_SET_SPRITE(sell_sel_x, sell_sel_y, sell_sel, 81, 0);
	IOWR_SET_SPRITE(range_sel_x1, range_sel_y1, RANGE, 80, 0);
	IOWR_SET_SPRITE(range_sel_x2, range_sel_y2, RANGE, 79, 2);
	IOWR_SET_SPRITE(range_sel_x3, range_sel_y3, RANGE, 78, 3);
	IOWR_SET_SPRITE(range_sel_x4, range_sel_y4, RANGE, 77, 1);

	/***********************display monsters***************************/
	IOWR_SET_SPRITE(mons[0].x, mons[0].y, mons[0].image, 12, mons[0].phase);
	IOWR_SET_SPRITE(mons[1].x, mons[1].y, mons[1].image, 15, mons[1].phase);
	IOWR_SET_SPRITE(mons[2].x, mons[2].y, mons[2].image, 18, mons[2].phase);
	IOWR_SET_SPRITE(mons[3].x, mons[3].y, mons[3].image, 21, mons[3].phase);
	IOWR_SET_SPRITE(mons[4].x, mons[4].y, mons[4].image, 24, mons[4].phase);
	IOWR_SET_SPRITE(mons[5].x, mons[5].y, mons[5].image, 27, mons[5].phase);
	IOWR_SET_SPRITE(mons[6].x, mons[6].y, mons[6].image, 30, mons[6].phase);
	IOWR_SET_SPRITE(mons[7].x, mons[7].y, mons[7].image, 33, mons[7].phase);
	IOWR_SET_SPRITE(mons[8].x, mons[8].y, mons[8].image, 36, mons[8].phase);
	IOWR_SET_SPRITE(mons[9].x, mons[9].y, mons[9].image, 39, mons[9].phase);
	IOWR_SET_SPRITE(mons[10].x, mons[10].y, mons[10].image, 42, mons[10].phase);
	IOWR_SET_SPRITE(mons[11].x, mons[11].y, mons[11].image, 45, mons[11].phase);
	IOWR_SET_SPRITE(mons[12].x, mons[12].y, mons[12].image, 48, mons[12].phase);

	/*********************display glue effects************************/
```

```
IOWR_SET_SPRITE(sta[0].x, sta[0].y, sta[0].image, 13, 0);
IOWR_SET_SPRITE(sta[1].x, sta[1].y, sta[1].image, 16, 0);
IOWR_SET_SPRITE(sta[2].x, sta[2].y, sta[2].image, 19, 0);
IOWR_SET_SPRITE(sta[3].x, sta[3].y, sta[3].image, 22, 0);
IOWR_SET_SPRITE(sta[4].x, sta[4].y, sta[4].image, 25, 0);
IOWR_SET_SPRITE(sta[5].x, sta[5].y, sta[5].image, 28, 0);
IOWR_SET_SPRITE(sta[6].x, sta[6].y, sta[6].image, 31, 0);
IOWR_SET_SPRITE(sta[7].x, sta[7].y, sta[7].image, 34, 0);
IOWR_SET_SPRITE(sta[8].x, sta[8].y, sta[8].image, 37, 0);
IOWR_SET_SPRITE(sta[9].x, sta[9].y, sta[9].image, 40, 0);
IOWR_SET_SPRITE(sta[10].x, sta[10].y, sta[10].image, 43, 0);
IOWR_SET_SPRITE(sta[11].x, sta[11].y, sta[11].image, 46, 0);
IOWR_SET_SPRITE(sta[12].x, sta[12].y, sta[12].image, 49, 0);

/************************display health****************************/
IOWR_SET_SPRITE(heal[0].x, heal[0].y, heal[0].image, 14, 0);
IOWR_SET_SPRITE(heal[1].x, heal[1].y, heal[1].image, 17, 0);
IOWR_SET_SPRITE(heal[2].x, heal[2].y, heal[2].image, 20, 0);
IOWR_SET_SPRITE(heal[3].x, heal[3].y, heal[3].image, 23, 0);
IOWR_SET_SPRITE(heal[4].x, heal[4].y, heal[4].image, 26, 0);
IOWR_SET_SPRITE(heal[5].x, heal[5].y, heal[5].image, 29, 0);
IOWR_SET_SPRITE(heal[6].x, heal[6].y, heal[6].image, 32, 0);
IOWR_SET_SPRITE(heal[7].x, heal[7].y, heal[7].image, 35, 0);
IOWR_SET_SPRITE(heal[8].x, heal[8].y, heal[8].image, 38, 0);
IOWR_SET_SPRITE(heal[9].x, heal[9].y, heal[9].image, 41, 0);
IOWR_SET_SPRITE(heal[10].x, heal[10].y, heal[10].image, 44, 0);
IOWR_SET_SPRITE(heal[11].x, heal[11].y, heal[11].image, 47, 0);
IOWR_SET_SPRITE(heal[12].x, heal[12].y, heal[12].image, 50, 0);

/************************display tower****************************/
IOWR_SET_SPRITE(tow[0].x, tow[0].y, tow[0].image, 51, tow[0].phase);
IOWR_SET_SPRITE(tow[1].x, tow[1].y, tow[1].image, 52, tow[1].phase);
IOWR_SET_SPRITE(tow[2].x, tow[2].y, tow[2].image, 53, tow[2].phase);
IOWR_SET_SPRITE(tow[3].x, tow[3].y, tow[3].image, 54, tow[3].phase);
IOWR_SET_SPRITE(tow[4].x, tow[4].y, tow[4].image, 55, tow[4].phase);
IOWR_SET_SPRITE(tow[5].x, tow[5].y, tow[5].image, 56, tow[5].phase);
IOWR_SET_SPRITE(tow[6].x, tow[6].y, tow[6].image, 57, tow[6].phase);
IOWR_SET_SPRITE(tow[7].x, tow[7].y, tow[7].image, 58, tow[7].phase);
IOWR_SET_SPRITE(tow[8].x, tow[8].y, tow[8].image, 59, tow[8].phase);
IOWR_SET_SPRITE(tow[9].x, tow[9].y, tow[9].image, 60, tow[9].phase);
IOWR_SET_SPRITE(tow[10].x, tow[10].y, tow[10].image, 61, tow[10].phase);
IOWR_SET_SPRITE(tow[11].x, tow[11].y, tow[11].image, 62, tow[11].phase);
IOWR_SET_SPRITE(tow[12].x, tow[12].y, tow[12].image, 63, tow[12].phase);
```

```
    /************************display bullet***************************/
    IOWR_SET_SPRITE(bul[0].x, bul[0].y, bul[0].image, 64, bul[0].phase);
    IOWR_SET_SPRITE(bul[1].x, bul[1].y, bul[1].image, 65, bul[1].phase);
    IOWR_SET_SPRITE(bul[2].x, bul[2].y, bul[2].image, 66, bul[2].phase);
    IOWR_SET_SPRITE(bul[3].x, bul[3].y, bul[3].image, 67, bul[3].phase);
    IOWR_SET_SPRITE(bul[4].x, bul[4].y, bul[4].image, 68, bul[4].phase);
    IOWR_SET_SPRITE(bul[5].x, bul[5].y, bul[5].image, 69, bul[5].phase);
    IOWR_SET_SPRITE(bul[6].x, bul[6].y, bul[6].image, 70, bul[6].phase);
    IOWR_SET_SPRITE(bul[7].x, bul[7].y, bul[7].image, 71, bul[7].phase);
    IOWR_SET_SPRITE(bul[8].x, bul[8].y, bul[8].image, 72, bul[8].phase);
    IOWR_SET_SPRITE(bul[9].x, bul[9].y, bul[9].image, 73, bul[9].phase);
    IOWR_SET_SPRITE(bul[10].x, bul[10].y, bul[10].image, 74, bul[10].phase);
    IOWR_SET_SPRITE(bul[11].x, bul[11].y, bul[11].image, 75, bul[11].phase);
    IOWR_SET_SPRITE(bul[12].x, bul[12].y, bul[12].image, 76, bul[12].phase);

    /************************display number***************************/
    IOWR_SET_SPRITE(wave_x[0], wave_y[0], wave_disp[0], 1, color_wave);
    IOWR_SET_SPRITE(wave_x[1], wave_y[1], wave_disp[1], 2, color_wave);

    IOWR_SET_SPRITE(life_x[0], life_y[0], life_disp[0], 3, color_life);

    IOWR_SET_SPRITE(money_x[0], money_y[0], money_disp[0], 4, color_money);
    IOWR_SET_SPRITE(money_x[1], money_y[1], money_disp[1], 5, color_money);
    IOWR_SET_SPRITE(money_x[2], money_y[2], money_disp[2], 6, color_money);
    IOWR_SET_SPRITE(money_x[3], money_y[3], money_disp[3], 7, color_money);

    IOWR_SET_SPRITE(score_x[0], score_y[0], score_disp[0], 8, color_score);
    IOWR_SET_SPRITE(score_x[1], score_y[1], score_disp[1], 9, color_score);
    IOWR_SET_SPRITE(score_x[2], score_y[2], score_disp[2], 10, color_score);
    IOWR_SET_SPRITE(score_x[3], score_y[3], score_disp[3], 11, color_score);

}

void audio_interrupt_handler ()
{
    /************************play tower sound***************************/
    IOWR_SET_SOUND(tow[0].sound_type, tow[0].sound_l, 0, tow[0].sound_en);
    IOWR_SET_SOUND(tow[1].sound_type, tow[1].sound_l, 1, tow[1].sound_en);
    IOWR_SET_SOUND(tow[2].sound_type, tow[2].sound_l, 2, tow[2].sound_en);
    IOWR_SET_SOUND(tow[3].sound_type, tow[3].sound_l, 3, tow[3].sound_en);
    IOWR_SET_SOUND(tow[4].sound_type, tow[4].sound_l, 4, tow[4].sound_en);
    IOWR_SET_SOUND(tow[5].sound_type, tow[5].sound_l, 5, tow[5].sound_en);
    IOWR_SET_SOUND(tow[6].sound_type, tow[6].sound_l, 6, tow[6].sound_en);
    IOWR_SET_SOUND(tow[7].sound_type, tow[7].sound_l, 7, tow[7].sound_en);
```

```
	IOWR_SET_SOUND(tow[8].sound_type, tow[8].sound_l, 8, tow[8].sound_en);
	IOWR_SET_SOUND(tow[9].sound_type, tow[9].sound_l, 9, tow[9].sound_en);
	IOWR_SET_SOUND(tow[10].sound_type, tow[10].sound_l, 10, tow[10].sound_en);
	IOWR_SET_SOUND(tow[11].sound_type, tow[11].sound_l, 11, tow[11].sound_en);
	IOWR_SET_SOUND(tow[12].sound_type, tow[12].sound_l, 12, tow[12].sound_en);

	/************************play monster sound**************************/
	IOWR_SET_SOUND(mons[0].sound_type, mons[0].sound_l, 13, mons[0].sound_en);
	IOWR_SET_SOUND(mons[0].sound_type, mons[1].sound_l, 14, mons[1].sound_en);
	IOWR_SET_SOUND(mons[0].sound_type, mons[2].sound_l, 15, mons[2].sound_en);
	IOWR_SET_SOUND(mons[0].sound_type, mons[3].sound_l, 16, mons[3].sound_en);
	IOWR_SET_SOUND(mons[4].sound_type, mons[4].sound_l, 17, mons[4].sound_en);
	IOWR_SET_SOUND(mons[5].sound_type, mons[5].sound_l, 18, mons[5].sound_en);
	IOWR_SET_SOUND(mons[6].sound_type, mons[6].sound_l, 19, mons[6].sound_en);
	IOWR_SET_SOUND(mons[7].sound_type, mons[7].sound_l, 20, mons[7].sound_en);
	IOWR_SET_SOUND(mons[8].sound_type, mons[8].sound_l, 21, mons[8].sound_en);
	IOWR_SET_SOUND(mons[9].sound_type, mons[9].sound_l, 22, mons[9].sound_en);
	IOWR_SET_SOUND(mons[10].sound_type, mons[10].sound_l, 23, mons[10].sound_en);
	IOWR_SET_SOUND(mons[11].sound_type, mons[11].sound_l, 24, mons[11].sound_en);
	IOWR_SET_SOUND(mons[12].sound_type, mons[12].sound_l, 25, mons[12].sound_en);
}

void mouse_interrupt_handler()
{
	int i;
	MOUSE_IRQ_DISABLE();

	mouse = IORD_32DIRECT(MOUSE_CTRL_BASE, 0);
	x_position = (mouse >> 3) & 0x3ff;
	y_position = (mouse >> 13) & 0x3ff;
	left_button = mouse & 0x1;
	right_button = (mouse>>1) & 0x1;

	switch (mouse_state)
	{
		case INIT_STATE:
			if(left_button == 1)
			{
				if(x_position >= 22 && x_position < 277 && y_position >= 160 &&
y_position < 351)
				{
					current_map = 1;
					interface_state = GAME_MAP_READY;
					mouse_state = GAME_STATE;
				}
				else if(x_position >= 361 && x_position < 616 && y_position >= 160 &&
y_position < 351)
				{
					current_map = 2;
					interface_state = GAME_MAP_READY;
					mouse_state = GAME_STATE;
				}
				else
					mouse_state = INIT_STATE;
			}
		break;
		case GAME_STATE:
			if(left_button == 1)
			{
				if(x_position >= 590 && x_position < 632 && y_position >= 8 &&
y_position < 40)
				{
					pause_en = 1 - pause_en;
					pause_sel = PAUSE + pause_en;
				}
				else if(x_position >= 560 && x_position < 580 && y_position >= 8 &&
y_position < 40)
				{
					interface_state = GAME_QUIT;
				}
				else if(x_position >= 470 && x_position < 542 && y_position >= 8 &&
y_position < 40)
				{
					speed_level = 1 - speed_level;
					generate_time = generate_time/(speed_level + 1);
					speed_sel = SPEED1 + speed_level;
				}
				else if(x_position >= 420 && x_position < 452 && y_position >= 8 &&
y_position < 40)
				{
					mute_en = 1 - mute_en;
					sound_sel = UNMUTE + mute_en;
				}
				else if(y_position > 64)
				{
					for(i = 0; i < BG1_PATH; i++)
					{
						if(current_map == 1 && x_position >=
background_path1[i][0]*32 && x_position < background_path1[i][0]*32 + 32 &&
                        y_position >= background_path1[i][1]*32 &&
y_position < background_path1[i][1]*32 + 32)
                  {
                        mouse_sel_x = x_position - 6;
                        mouse_sel_y = y_position - 6;
                        mouse_sel = FORBID;
                        mouse_state = FORBID_STATE;
                        break;
                  }
                  if(current_map == 2 && x_position >=
background_path2[i][0]*32 && x_position < background_path2[i][0]*32 + 32 &&
                        y_position >= background_path2[i][1]*32 &&
y_position < background_path2[i][1]*32 + 32)
                  {
                        mouse_sel_x = x_position - 6;
                        mouse_sel_y = y_position - 6;
                        mouse_sel = FORBID;
                        mouse_state = FORBID_STATE;
                        break;
                  }
            }
            if(i < BG1_PATH)
                  break;
            for(i = 0; i < tower_num; i++)
            {
                  if(x_position >= tow[i].x && x_position <= tow[i].x+32 &&
y_position >= tow[i].y && y_position <= tow[i].y + 32)
                  {
                        if((money >= 150 && (tow[i].type == GLUE_TOW ||
tow[i].type == GREEN_TOW))
                                    || (money >= 300 && (tow[i].type == DART_TOW
|| tow[i].type == BOMB_TOW)))
                        {
                              upgrade_sel_x = tow[i].x;
                              upgrade_sel_y = tow[i].y -32;
                        }
                        range_sel_x1 = tow[i].x - (tow[i].range - 16);
                        range_sel_y1 = tow[i].y + (tow[i].range - 20);
                        range_sel_x2 = tow[i].x - (tow[i].range - 16);
                        range_sel_y2 = tow[i].y - (tow[i].range - 20);
                        range_sel_x3 = tow[i].x + (tow[i].range - 16);
                        range_sel_y3 = tow[i].y - (tow[i].range - 20);
                        range_sel_x4 = tow[i].x + (tow[i].range - 16);
```

```
                range_sel_y4 = tow[i].y + (tow[i].range - 20);
                if(tow[i].y == 448 && tow[i].x < 608)
                {
                    sell_sel_x = tow[i].x + 32;
                    sell_sel_y = tow[i].y;
                }
                else if(tow[i].y == 448 && tow[i].x == 608)
                {
                    sell_sel_x = tow[i].x - 32;
                    sell_sel_y = tow[i].y;
                }
                else
                {
                    sell_sel_x = tow[i].x;
                    sell_sel_y = tow[i].y +32;
                }
                if(tow[i].level == 2)
                    upgrade_sel = MAX;
                else
                {
                    if(tow[i].type    ==    GLUE_TOW    ||    tow[i].type    ==
GREEN_TOW)
                    {
                        upgrade_sel = UPGRADE;
                        if(tow[i].level == 1)
                            sell_sel = SELL;
                        else if(tow[i].level == 2)
                            sell_sel = SELL2;
                    }
                    else
                    {
                        upgrade_sel = UPGRADE2;
                        if(tow[i].level == 1)
                            sell_sel = SELL3;
                        else if(tow[i].level == 2)
                            sell_sel = SELL3;
                    }
                }
                current_tower = i;
                mouse_state = UPGRADE_STATE;
                break;
            }
        }
        if(i < tower_num)
```

```
                break;
        if(tower_num < 13 && money >= 100)
        {
                mouse_sel = SEL;
                mouse_sel_x = x_position - x_position%32;
                mouse_sel_y = y_position - y_position%32;
                if(money < 120)
                {
                        if(x_position >= 608)
                                tow_sel_x = mouse_sel_x - 32;
                        else
                                tow_sel_x = mouse_sel_x + 32;
                        tow_sel_y = mouse_sel_y;
                        tower_sel = TOW_SELECT1;
                }
                else if(money < 210)
                {
                        if(x_position >= 608)
                                tow_sel_x = mouse_sel_x - 32;
                        else
                                tow_sel_x = mouse_sel_x + 32;
                        tow_sel_y = mouse_sel_y - 32;
                        tower_sel = TOW_SELECT2;
                }
                else if(money < 240)
                {
                        if(x_position >= 608)
                                tow_sel_x = mouse_sel_x - 32;
                        else
                                tow_sel_x = mouse_sel_x + 32;
                        if(y_position >= 448)
                                tow_sel_y = mouse_sel_y - 64;
                        else
                                tow_sel_y = mouse_sel_y - 32;
                        tow_sel_y = mouse_sel_y - 32;
                        tower_sel = TOW_SELECT3;
                }
                else
                {
                        if(x_position >= 608)
                                tow_sel_x = mouse_sel_x - 32;
                        else
                                tow_sel_x = mouse_sel_x + 32;
                        if(y_position <= 96)
```

```
                                tow_sel_y = mouse_sel_y - 32;
                            else if(y_position >= 448)
                                tow_sel_y = mouse_sel_y - 96;
                            else
                                tow_sel_y = mouse_sel_y - 64;
                            tower_sel = TOW_SELECT4;
                        }
                        mouse_state = BUILD_STATE;
                    }
                    else
                        mouse_state = GAME_STATE;
                }
            }
        break;
        case FORBID_STATE:
            sprite_clear(&mouse_sel_x, &mouse_sel_y);
            mouse_state = GAME_STATE;
        break;
        case BUILD_STATE:
            if(left_button == 1)
            {
                if(x_position >= tow_sel_x && x_position <= tow_sel_x + 32 &&
y_position >= tow_sel_y && y_position <= tow_sel_y + 32)
                {
                    tower_new(mouse_sel_x, mouse_sel_y, GREEN_RANGE,
TOW_FAST,GREEN_TOW, GREEN_BUL, SOUND_GREEN);
                    money -= 100;
                }
                else if(x_position >= tow_sel_x && x_position <= tow_sel_x + 32 &&
y_position >= tow_sel_y + 32 && y_position <= tow_sel_y + 64 && tower_sel >=212)
                {
                    tower_new(mouse_sel_x, mouse_sel_y, GLUE_RANGE, TOW_FAST,
GLUE_TOW, GLUE_BUL, SOUND_GLUE);
                    money -= 120;
                }
                else if(x_position >= tow_sel_x && x_position <= tow_sel_x + 32 &&
y_position >= tow_sel_y + 64 && y_position <= tow_sel_y + 96 && tower_sel >=213)
                {
                    tower_new(mouse_sel_x, mouse_sel_y, BOMB_RANGE, TOW_FAST,
BOMB_TOW, BOMB_BUL, SOUND_BOMB);
                    money -= 210;
                }
                else if(x_position >= tow_sel_x && x_position <= tow_sel_x + 32 &&
y_position >= tow_sel_y + 96 && y_position <= tow_sel_y + 128 && tower_sel ==214)
```

```
                        {
                              tower_new(mouse_sel_x, mouse_sel_y, DART_RANGE, TOW_SLOW,
DART_TOW, DART_BUL, SOUND_DART);
                              money -= 240;
                        }
                        sprite_clear(&tow_sel_x, &tow_sel_y);
                        sprite_clear(&mouse_sel_x, &mouse_sel_y);
                        mouse_state = GAME_STATE;
                  }
            break;
            case UPGRADE_STATE:
                  if(left_button == 1)
                  {
                        if(x_position >= upgrade_sel_x && x_position <= upgrade_sel_x + 32 &&
y_position >= upgrade_sel_y && y_position <= upgrade_sel_y + 32
                                    && tow[current_tower].level == 1)
                        {
                              if(tow[current_tower].type == GREEN_TOW ||
tow[current_tower].type == GLUE_TOW)
                                    money -= 150;
                              else
                                    money -= 300;
                              tow[current_tower].level = 2;
                              tow[current_tower].image += 75;
                              tow[current_tower].range += (tow[current_tower].level-1)*20;
                              bul[current_tower].type += 75;
                              bul[current_tower].image += 75;
                        }
                        else if(x_position >= sell_sel_x && x_position <= sell_sel_x + 32 &&
y_position >= sell_sel_y && y_position <= sell_sel_y + 32)
                              tower_sell(current_tower);
                        sprite_clear(&sell_sel_x, &sell_sel_y);
                        sprite_clear(&upgrade_sel_x, &upgrade_sel_y);
                        sprite_clear(&range_sel_x1, &range_sel_y1);
                        sprite_clear(&range_sel_x2, &range_sel_y2);
                        sprite_clear(&range_sel_x3, &range_sel_y3);
                        sprite_clear(&range_sel_x4, &range_sel_y4);
                        mouse_state = GAME_STATE;
                  }
            break;
            case DONE_STATE:
                  if(left_button == 1)
                  {
                        if(x_position >= 272 && x_position <= 368 && y_position >= 208 &&
y_position <= 240)
                        {
                              sprite_clear(&mouse_sel_x, &mouse_sel_y);
                              sprite_clear(&sell_sel_x, &sell_sel_y);
                              sprite_clear(&upgrade_sel_x, &upgrade_sel_y);
                              interface_state = GAME_MAP_READY;
                              mouse_state = GAME_STATE;
                        }
                        else if(x_position >= 288 && x_position <= 352 && y_position >= 240 &&
y_position <= 272)
                        {
                              sprite_clear(&mouse_sel_x, &mouse_sel_y);
                              sprite_clear(&sell_sel_x, &sell_sel_y);
                              sprite_clear(&upgrade_sel_x, &upgrade_sel_y);
                              interface_state = GAME_DONE;
                              mouse_state = INIT_STATE;
                        }
                  }
            break;
      }
      MOUSE_IRQ_ENABLE();
}

void timer_interrupt_handler ()
{
      TIMER_IRQ_CLEAR();
      tow_run = 1;
      if(run_count == 3)
      {
            run_count = 0;
            mons_run = 1;
      }
      else
            run_count ++;
}
```