CSEE4840 Project Final Report

Battle City

May 13, 2011

Group members:

Tian Chu (tc2531)

Liuxun Zhu (lz2275)

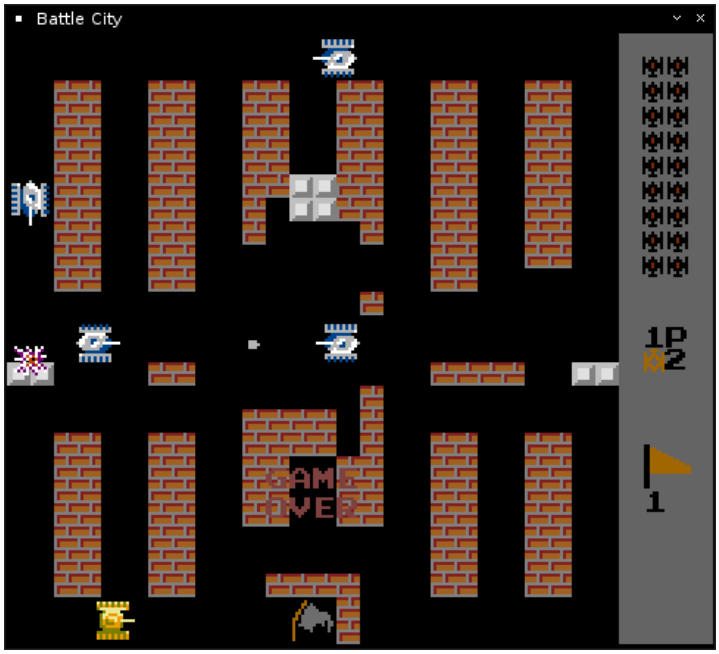
Tianchen Li (tl2445)

Quan Yuan (qy2129)

Yuanzhao Huangfu (yh2453)

Introduction:

Our project is to design a video game **Battle City** which was originally developed by Namco in 1985. The player, controlling a tank, must destroy enemy tanks in each level, which enter the playfield from the top of the screen. The enemy tanks attempt to destroy the player's base (represented on the map as a bird, eagle or Phoenix), as well as the human tank itself. A level is completed when the player destroys all enemy Tanks, but the game ends if the player's base is destroyed or the player loses all available lives. The general appearance of the game Battle City looks like this:



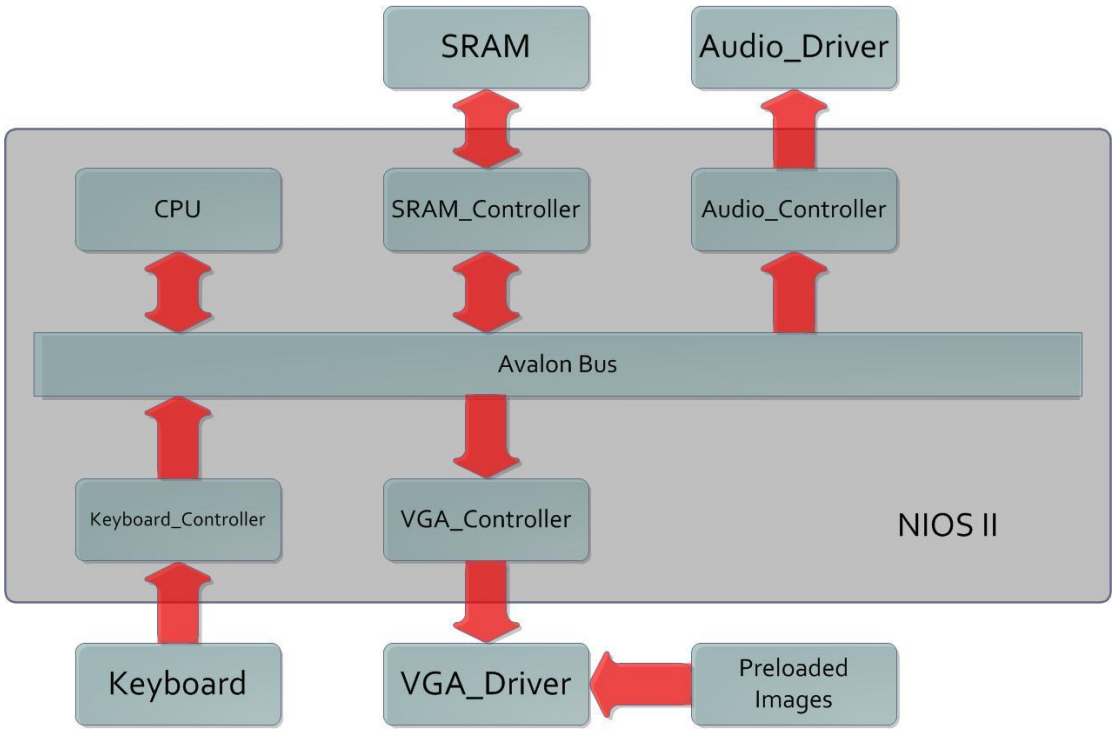
Challenges:

To implement this project on the DE2 board, we need to integrate the software algorithms with the hardware drivers. The hardware drives the keyboard, VGA monitor and audio decoder. The keyboard receives the user's inputs, like arrow keys to control the moving direction of the tank. The VGA monitor displays the scenario with user's tank, enemies and even the bullet. The audio decoder plays the sound appropriately, like bombing sound when an enemy is destroyed. The real-time video display will be the most challenging part, because the scenario changes all the time and all changes should be synchronized with the software.

The software receives the game player's inputs and translates them into actions of the tank, like moving and fire. At the start of the game, a scenario parser loads the predefined scenario setup and translates it into something that the hardware can understand and displayed properly. During the game playing, the algorithm should control the tanks' movement according to the current scenario setup, and detect the destroying of the enemies. Therefore the multi-tasking may be the most difficult thing the software should handle.

Architecture:

The following is the basic block diagrams of the whole design architecture, and it shows the connection between the different modules and the interaction between the CPU and hardware drivers.



A brief description of each module is given below:

- **CPU:** loads instructions stored in the SRAM and executes them one by one.
- **SRAM_Controller:** sends and receives data and instructions between the BUS and SRAM.
- **Audio_Controller:** receives audio commands from the BUS and translates it to Audio_Driver.
- **SRAM:** stores data and instructions of the NIOS II needs.
- **Audio_Driver:** drives the audio decoder with the preloaded sounds.
- **Keyboard_Controller:** receives keyboard inputs and puts them on the BUS.
- **VGA_Controller:** receives display data from the BUS and feeds them to the driver.
- **VGA_Driver:** drives the VGA monitor with the preloaded images.

In the next few sections, we will discuss the keyboard, VGA display, audio play and software algorithm in detail.

Keyboard:

The keyboard is one of the most important modules in this design, because it provides the only way for the DE2 board to get users' inputs. In this design, following keys are used and their functions are described below:

Keys	Functions
A, D, W, S	Player's tank moves left, right, up and down respectively
Space	Player's tank fires
Enter	Start game
E	Choose easy game mode
C	Choose crazy game mode

Note: all the functions will be performed repeatedly when the keys are pressed continuously.

Game play

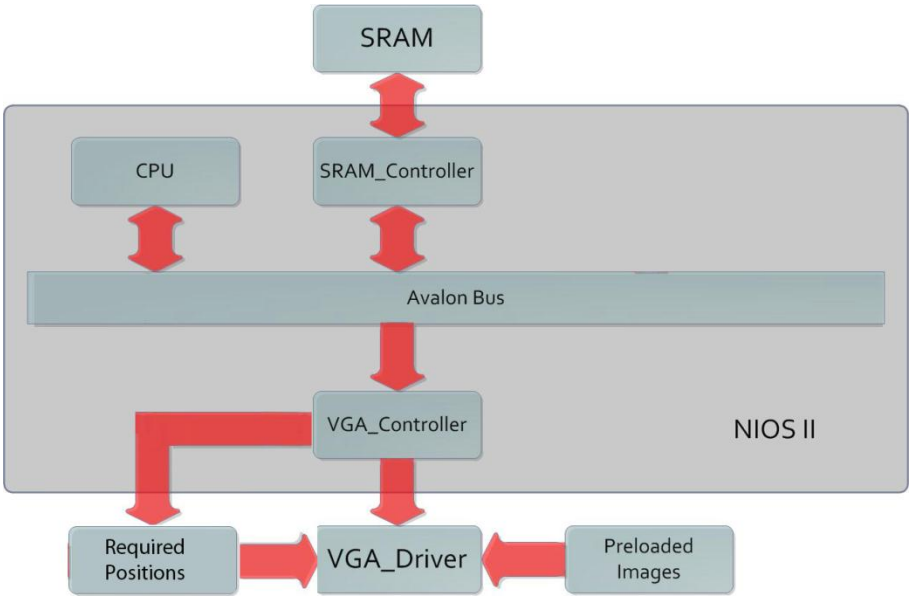
At the start of the game, two game modes can be selected. Use "E" to choose easy mode and "C" to choose crazy mode. Pressing "Enter" key will start the game. When each stage starts, the player will have three lives. When the number of lives becomes zero or the base is hit by a bullet, the game will restart. The bullets can be cancelled with each other. The "power up", which looks like a "star", can be eaten by the player or enemies. If the player gets it, the base may get steel protection, enemies may be frozen and the player may get power upgrade. If enemies get it, the corresponding bad effects may happen on the player.

VGA:

The VGA display is the most challenging part of this project, dues to the real-time changes of the scenario, like destruction of the walls and hostile tanks. The whole screen is separated into 13X13 squares and each of them is formed by 36X36 pixels. Since all the background items like bricks, river, and concrete are all symmetric in their shapes, we can just store 1/4 of the square's size and repeat them when displaying. The tanks and the bullet are considered as sprites and their images are stored individually in RAMs. In short, the VGA display part of this project should have following functions:

- Loads the scenario setup from the bus sequentially and put the correct images on the screen at proper positions.
- Adjust the tanks and bullets' positions on the screen when positions' update commands received from the bus.
- Handle the overlapping in the game video by using different layers.
- Display animations at proper time and positions when commands are received from the bus.

VGA Architecture:



VGA display is to draw the game scenario, user interfaces and sprites in real-time. The data and instruction communications are completed via the Avalon bus. The VGA display function mainly contains 4 modules: VGA_Controller, VGA_Driver, Preloaded Images and Required Positions.

- VGA_Controller: Receive the CPU instructions from the Avalon bus.
- VGA_Driver: Display scenario backgrounds, sprites and animation effects as required by CPU.
- Preloaded Images: ROM. Store all the images that may need to display in the game.
- Required Positions: RAM. Store the required position of each image in current scenario.

VGA_Controller gets the CPU instructions and then writes the required positions into the RAMs. VGA driver reads the memory and display preloaded images at the required positions in real-time.

VGA Interfaces:

PortMaps :

The portmaps are roughly defined below:

- VGA_Controller

```
clk          : in    std_logic;
reset_n      : in    std_logic;
write        : in    std_logic;
chipselect   : in    std_logic;
writedata    : in    unsigned(31 downto 0);
hcenter      : out unsigned(9 downto 0);
vcenter      : out unsigned(9 downto 0)
```

- VGA_Driver

```
reset          : in std_logic;
clk            : in std_logic;                    -- Should be 25.125 MHz
TANK_HCENTER : unsigned(9 downto 0);    -- Horizontal position (0-800)
TANK_VCENTER : unsigned(9 downto 0);    -- Vertical position (0-524)
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 unsigned(9 downto 0)        -- Blue[9:0]
```

- RAMs

  Created automatically by Quartus Wizard, and will be instantiated and utilized by VGA_Driver.

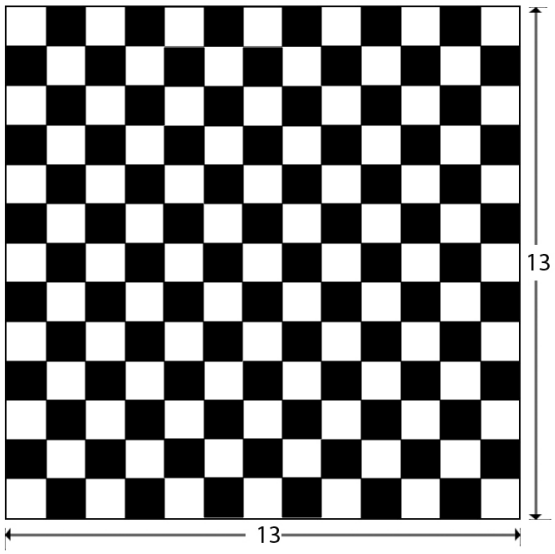
Instruction Formats

To achieve correct communications, instructions must be encoded by the software and decoded by the hardware. Instruction sets are defined in “data structure”. For detail, please refer to Software chapter.

Image Processing:

- User Screen:

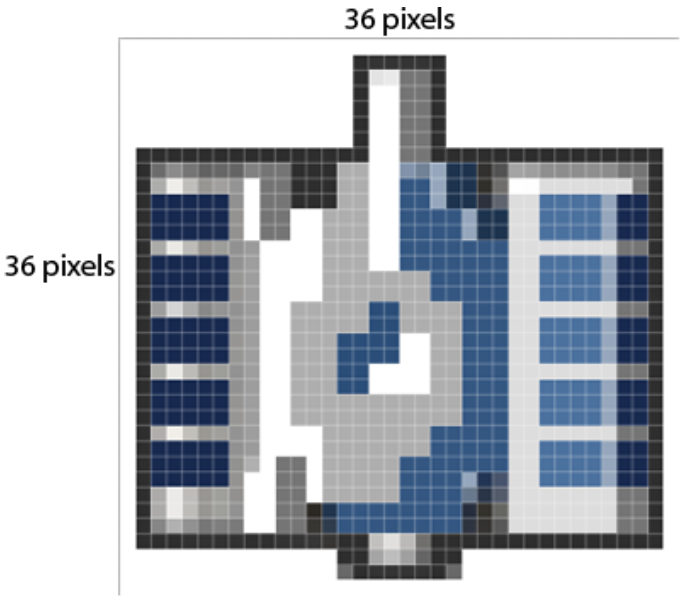
  The user screen is divided into 169 squares, each column or row contains 13 squares as indicated below:



The game scenario is constructed by 169 different images that loaded from RAM.

VGA_Controller will determine each image and its position.

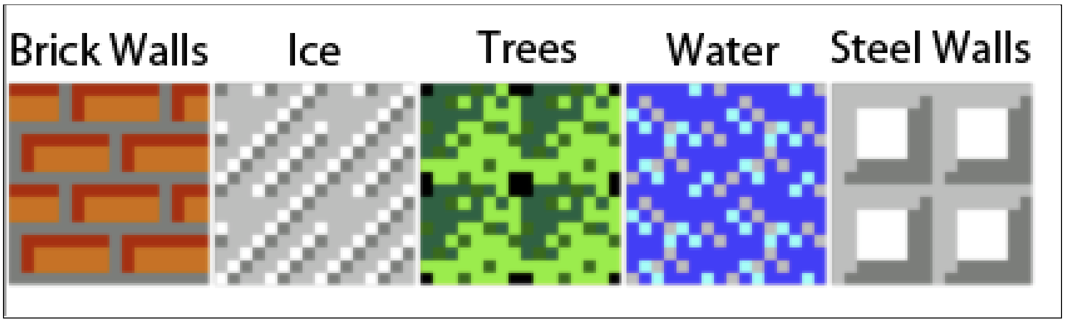
The sub-image is formed by 36X36 pixels. The image below is the tank image.



- Image type:

Two types of images need to be considered for real-time control: static scenario and sprites.

Static Scenario: They are background images that cannot be moved by player, but may disappear or burst according to events. There are five basic static scenario images:



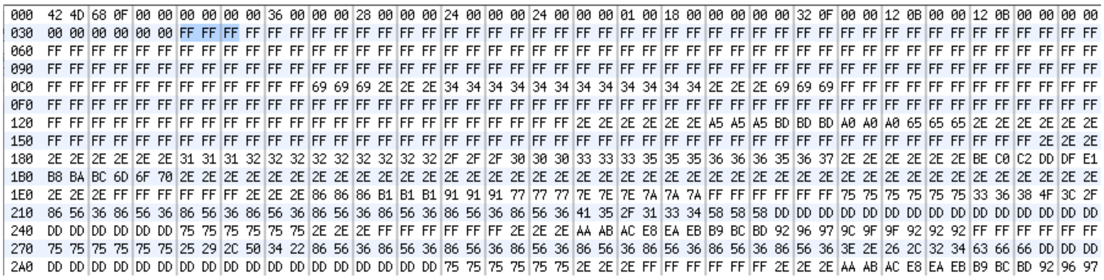
Static scenario could be constructed by simply replicating ¼ of itself in order to reduce memory size. The VGA_Driver only need the information of its position and type from RAM.

Sprites: Sprites are dynamic images that will change position with time on the screen. Sprites include tanks and bullets; they are fully stored in RAM. A different instruction will be defined for the sprites. VGA_Driver needs to get the position, sprite ID and color information to locate and change it over time.

- Image basics:

The images are stored in RAM as a pixel matrix.

First draw an image and save it as a 24-bit bmp. In bitmap image file on a disk or bitmap image in memory, the pixels can be defined by varying number of bits. The 24-bit pixel (24bpp) format stores each pixel value in 3 bytes. Each pixel value defines the Red, Green and Blue samples of the pixel. Specifically, they are stored in the order (Blue, Green and Red, 8-bits per each sample). Read the bmp file in Hex code, the pixel array (bitmap data) starts from the 0x36th byte. Then store pixel array into local disk. The RGB information represented by hex code is shown below:



```
000  42 4D 68 0F 00 00 00 00 00 00 36 00 00 00 28 00 00 00 24 00 00 00 24 00 00 00 01 00 18 00 00 00 00 00 32 0F 00 00 12 0B 00 00 12 0B 00 00 00 00
030  00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0C0  FF FF FF FF FF FF FF FF FF FF FF FF 69 69 69 2E 2E 2E 34 34 34 34 34 34 34 34 34 34 34 34 2E 2E 2E 69 69 69 FF FF FF FF FF FF FF FF FF FF FF FF
0F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
120  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 2E 2E 2E 2E 2E 2E A5 A5 A5 BD BD BD A0 A0 A0 65 65 65 2E 2E 2E 2E 2E 2E
150  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 2E 2E 2E
180  2E 2E 2E 2E 2E 2E 31 31 31 32 32 32 32 32 32 32 32 32 2F 2F 2F 30 30 30 33 33 33 35 35 35 36 36 36 35 36 37 2E 2E 2E 2E 2E 2E BE C0 C2 DD DF E1
1B0  B8 BA BC 6D 6F 70 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E
1E0  2E 2E 2E FF FF FF FF FF FF 2E 2E 2E 86 86 86 B1 B1 B1 91 91 91 77 77 77 7E 7E 7E 7A 7A 7A FF FF FF FF FF FF 75 75 75 75 75 75 33 36 38 4F 3C 2F
210  86 56 36 86 56 36 86 56 36 86 56 36 86 56 36 86 56 36 86 56 36 86 56 36 41 35 2F 31 33 34 58 58 58 DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD
240  DD DD DD DD DD DD 75 75 75 75 75 75 2E 2E 2E FF FF FF FF FF FF 2E 2E 2E AA AB AC E8 EA EB B9 BC BD 92 96 97 9C 9F 9F 92 92 92 FF FF FF FF FF FF
270  75 75 75 75 75 75 25 29 2C 50 34 22 86 56 36 86 56 36 86 56 36 86 56 36 86 56 36 86 56 36 86 56 36 3E 2E 26 2C 32 34 63 66 66 DD DD DD DD DD DD
2A0  DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD 75 75 75 75 75 75 2E 2E 2E FF FF FF FF FF FF 2E 2E 2E AA AB AC E8 EA EB B9 BC BD 92 96 97
```

- Image representations:

Due to the memory constrain, we classify the images by their color and represent them differently.

(i). 24-bits representation

In order to draw the tanks vividly, we store the tank pixel by 24-bits (8-bits for each RGB sample).

The tank images are composed of various colors. The 24-bits representation could fully store color information.

(ii). 1-bit representation

The image of phoenix only contains two kinds of color. The 24-bits representation for this image would cause a great waste of memory space. Thus, in the phoenix.mif file we only use 1 bit to distinct two colors. The VGA_driver reads in the mif file and assigns the 8-bit of each RGB channel to the image according to '0' or '1'. This method saves much memory space and it could also be applied to other 2-color images, such as letters and numbers.

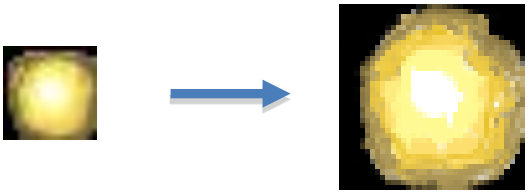
(iii). 2-bits representation

In most cases, the kinds of color are less than 4. Therefore, 2-bits representation is used for many images.

Images are encoded by 2-bits for each pixel and the RGB color assigned for each 2-bits code differs from image to image.

- Dynamic Effect:

  To achieve explosion effect, two explosion images of different sizes are used as frames.



  Simply display the two images in different clock periods and the dynamic effect will be shown.

- Overlapping:

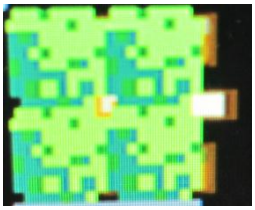
  There are total 5 scenarios in our game and each of them has a distinct functionality. Trees could shield tank but cannot fully cover it. Thus overlapping is needed in our design. The algorithm for overlapping is:

  (i) Set the overlapping area of upper layer to black.

  (ii) Judge the color of image, if it is black, write the data of lower layer to the non-overlapping

area.

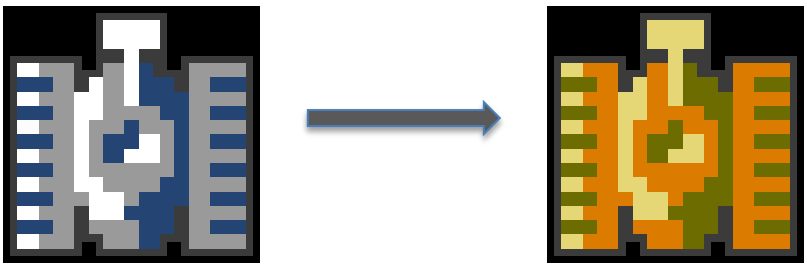
The effect is shown in image below:



It is used for explosion as well.

- Color conversion:

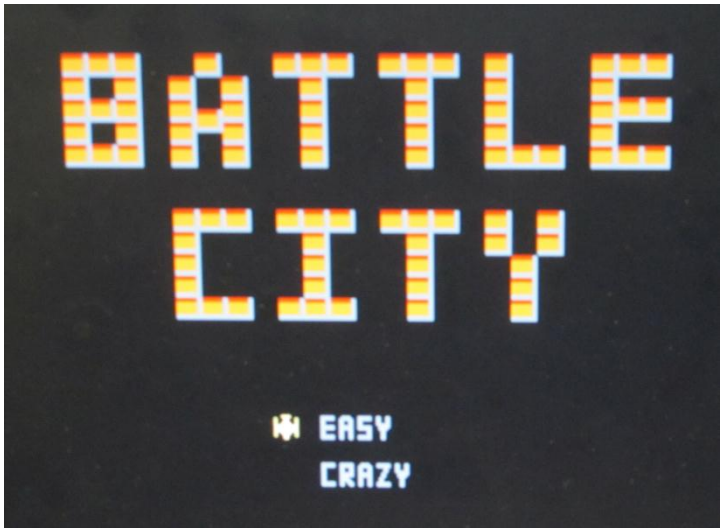
  As mentioned before, we store the tanks by 24 bits per pixel. This representation occupies very large memory space. There are total 6 kinds of tanks in our game and we encounter the memory constrains during the design. Thus, we consider reducing the number of tanks by converting the color. The appearances of enemy tanks are the same with those of players and the only difference is color. The enemy tanks are composed by grey and blue while players are yellow and green. Therefore we could only store the enemy tanks and judge the range of each color, then replace them with player tank colors.



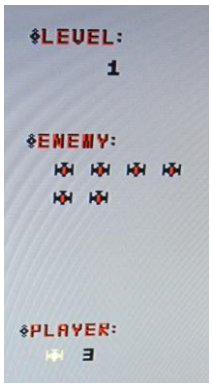
User Interfaces:

- Welcome page:

  In welcome page, user could choose the level of difficulty, named “easy” and “crazy”.



- Game Menu



The game menu on the left illustrates game information. “Level” refers to the stage playing currently. “Enemy” shows the number of enemies remained and similarly the “Player” means player’s lives.

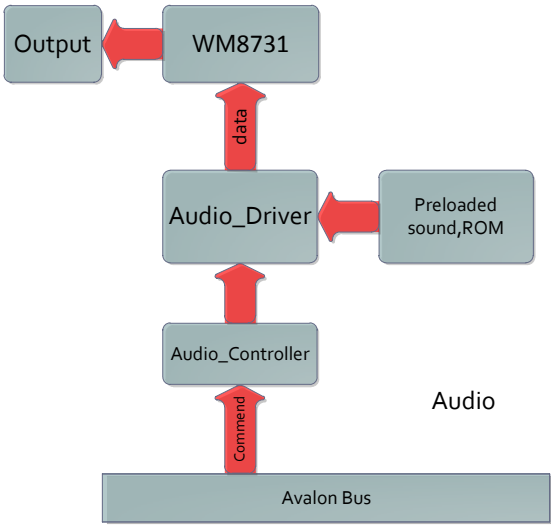
- Game Menu

“GAME OVER” is constructed by grass which could help reduce the space utilization.

Audio:

In our game, we want to play the sounds at proper times, like playing bombing sound when a tank is destroyed or and welcome music. There are two kind of sound. One is some sound effect which is preloaded in the individual ROMs. The other is welcome music which is composed of different frequency sin wave. Sound effect will play when the CPU gives the audio controller commands.

Audio Architecture:



Audio module plays welcome music and sound effect when user is playing game. The instruction communication is completed via Avlon bus. And data is directly sent to the DAC, WM8731. The audio function mainly contains 4 modules: Audio_controller, Audio_driver, preload sound and wm8731.

- Audio_controller: Receive the CPU instructions into the newly created RAM in FPGA.
- Audio_Driver: generate the music or sound effect required by CPU and sequentially output to wm8731.
- Preload Sound: ROM to store the sound data which is transformed by .wav file.
- Wm8731: DAC chip to convert digital data to analog signal.

Audio_controller get the cpu instructions and select which sound stored in ROM should be played and output the data to wm8731. Then wm8731 will play the sound.

Interfaces:

The port maps are showed as following.

- Audio_Controller:
- Audio_driver:

```
clock_50 : in std_logic;
clock_18 : in std_logic;
cpu_cmd : in std_logic_vector(31 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
```

- de2_wm8731_audio (this module has been declared as a component in audio_driver module)

```
clk : in std_logic;              --  Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
reset_n : in std_logic;
clk_50 : in std_logic;
disable : in std_logic; --when '1', no output from wm8731
sin_sel : in std_logic; --    when '1' play sin wave
play_finish1 : out std_logic;-- exp
play_finish2 : out std_logic;-- fire
reset_exp : in std_logic; -- when '1' reset
reset_fire : in std_logic; --when '1' reset
reset_sin : in std_logic; -- when '1' reset
tune : in unsigned(11 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
```

Audio_driver is the module to receive CPU instructions and use state machine for controlling the de2_wm8731_audio module selecting correct sound, playing it well, generating the correct clock and outputing to wm8731 chip.

de2_wm8731_audio is the module to generate the clock for wm8731 chip, read sound data from ROM and output to wm8731.

Audio Process:

- Sound type

There are two primary types of sound. One is *welcome music* which is played by composing several sin waves that have different frequency together. The other type is *sound effect* which includes two sound effects, fire and explosion. And the way to play sound effect is that read data that has been stored in ROM and sequentially output to wm8731, then the chip will play the correct sound effect.

- Welcome music

We design to play a starting music when the game is loading, and the music is composed of several sin waves that have different frequency. We take advantage of the idea of Lab3, audio_driver decide the sin wave of which frequency should be played. We use a counter to decide how long every tone in the music will play. The sin wave file is written in de2_wm8731_audio and each period consists of 48 points. And audio_driver actually save the music, control which frequency and how long every tone played, and the welcome music

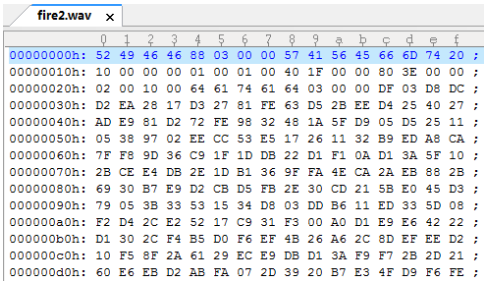
will be played well.

- Sound effect

When player's tank fires or tank explodes, we need some sound effect to make the game perform more interesting. Hence, we put some sound effect in our game. There are two main steps to play the sound effect: first, convert the wav format file into mif format file which can be read and stored in ROM on DE2. Second, if audio_driver receive a command from CPU to play sound effect, the driver will read preloaded data from ROM and output it to WM8731.

➢ Convert wave file into mif file

The wav format file is showed as following figure:



The file is composed of file head, which includes 43 bytes, and data that includes hundreds or thousands bytes. The wave file is little endian while our WM8731 reads big endian so that we need to swap lower byte's and higher byte's position. We write a small program in C++ to read the data in wave file and output them in a mif file that is like the following figure:

fire.mif

Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	03DF	DCD8	EAD2	1728	27D3	FE81	D563	EE2B
008	25D4	2740	E9AD	D281	FE72	3298	1A48	D95F
010	D505	1125	3805	0297	CCEE	E553	2617	3211
018	EDB9	CAA8	F87F	369D	1FC9	DB1D	D122	0AF1
020	3AD1	105F	CE2B	DBE4	1D2E	36B1	FA9F	CA4E
028	EB2A	2B88	3069	E9B7	CBD2	FBD5	302E	21CD
030	E05B	D345	0579	333B	1553	D834	DD03	11B6
038	33ED	085D	D4F2	E22C	1752	31C9	00F3	D1A0
040	E6E9	2242	30D1	F42C	D0B5	EFF6	264B	2CA6
048	EF8D	D2EE	F510	2A8F	2961	E9EC	D1DB	F93A
050	2BF7	212D	E660	D2EB	FAAB	2D07	2039	E3B7
058	D94F	FEF6	292F	1BA6	E535	DB6B	FEC3	2A68
060	196B	E5AE	DA3C	FDFC	2D08	1B5F	E4F5	D84D
068	FC47	2BEC	1EFA	E4DD	D86D	FC60	29E7	1D9D
070	E501	D67E	FA45	2DD6	1F0A	E5A2	D59A	FB52
078	2F32	20BF	E717	D201	F9FF	2E98	22B6	E87D
080	D23C	F66A	2914	269C	F0BF	D108	ED73	23AB
088	2D45	F5FF	D0F5	E557	2004	33C7	FFC9	D110
090	DF61	1A08	3705	04D3	D1F5	D8F8	13F8	3904
098	0CD9	D314	D5D1	0E01	36CE	12A1	D78C	D569
0a0	0881	33D9	1BBA	E34F	D4F8	FA95	2B1D	22B3
0a8	F3E9	D741	EA3D	189D	2867	0897	E37B	DB29
0b0	00FC	235A	1B96	F9B7	E038	EA1F	0DBB	2129
0b8	0F9B	F408	E2F7	F06F	0AC1	183E	1311	FA98
0c0	E6D0	E8A0	03D4	2154	1F8B	F719	D9B8	E076
0c8	0C17	31F6	1F59	E728	CCFF	E939	2162	371F
0d0	0FEB	D810	D013	F925	2BEF	3246	FD6B	CE0B
0d8	D59E	0D1C	3BA0	2928	E890	C4C7	E256	22E2

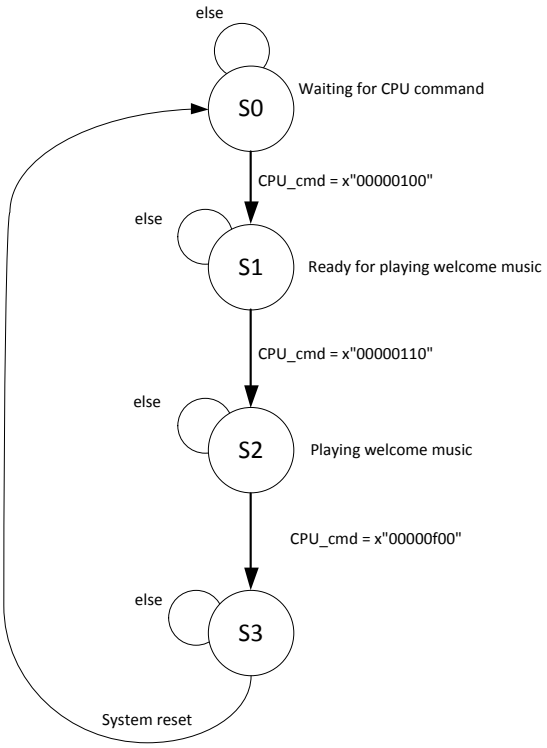
Such file can be read by Quartus and store data into a ROM.

➢ Play sound effect from ROM

When the game runs, two incidents happens will trigger playing sound effect, which are player's tank fire and tank's explosion. At that time, CPU will send Audio driver module two different instructions to tell the module which sound effect should be played then, audio driver will read data from ROM and send it to WM8731. In this game, we are using the wave file whose sample rate is 8khz and 4khz to save the

memory space. The key points of sound effect playing are two aspects: first, after received one instruction, the sound effect would be completely played only once. Second, if two different instructions received almost same time, which means fire and explosion happened meanwhile, we should solve the problem that two kinds of sound effect can be played simultaneously. To solve first problem, we use a feedback signal (say play_finish). Every time the counter read every address of ROM, the feedback signal will set 1 to tell audio driver that the sound effect finish playing, then, audio drive will reset the counter but won't start it until new instruction is received. To solve the second problem, we are using an adder at the de2_wm8731_audio module output. When two kinds of sound effect need to be played simultaneously, the output equals the sum of two data that from two different ROM.

- State Machine



This is the state machine graph of the audio system. After system started, the audio_driver module will in state 1 until it received a specific instruction. When it received instruction that is x"00000100", audio_driver become state 2 which means ready to playing music, however, in this state, no music will be played because signal 'disable' still equals '1'. Then if audio_driver received the instruction, x"00000110", it goes to state 2 and welcome music will be played only once, after that, no sound will play although audio_driver stay in state 2 until instruction x"00000f00" received. In state 3, which means we only need fire explosion and fire sound effect in our game. We define x"00000001" is fire signal and x"00000002" is explosion signal, so every time audio_driver receive fire or explosion signal from cpu, it will play the corresponding sound effect once. If overlap happens in playing sound music, the sound effect will sound like happening simultaneously due to we use the adder at data output. It will stay in state 2 through all game until the system reset again.

Software:

Software is definitely the most significant part of this design, since all hardware components are functioning according to the commands received from the software program. To make the game work, the software should have following functions:

- Handles the keyboard input interrupt and translates the make codes and break codes into the corresponding actions of the players' tanks.
- Loads the scenario setup from the SRAM and put them on the bus sequentially for the VGA display module.
- Adjusts and records all tanks' positions and their directions. Due to the existence of the obstacles, some movements should be prevented.
- Adjusts and records all bullets' positions and their directions. When the bullet hits the wall, the background scenario should be updated both in the program and VGA display.
- Generates the enemies at a certain time rate and controls their actions with a random algorithm. The difficulty of the game can be set to different level, and the enemies may be "smarter" in the higher level.

Data structure:

The whole game includes two kinds of objects—obstacles and sprites. Brick and concrete walls are typical obstacles while the tanks and bullets are sprites. Sprites' positions change with time while the obstacles' don't. To store the obstacles, we use simple integer variables. A 13X13 integer (16-bit) array will be used to store these obstacles, and they will have following format:

X				Y				Blocks				Type			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

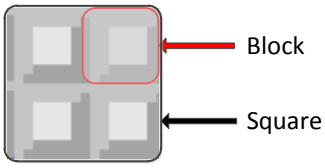
X: this four-bit field is used to store the obstacle's horizontal position (0-12 in decimal)

Y: this four-bit field is used to store the obstacle's vertical position (0-12 in decimal)

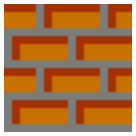
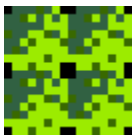
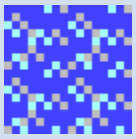
Blocks: each obstacle (or square in the screen) is formed by four blocks, and this four-bit field is used to indicate these blocks' existence

Type: this four-bit field is used to store the type of the obstacle

This figure illustrates the relationship between an obstacle and its blocks:



The following table illustrates the **Type** value of each kind of obstacles:

	**Brick Walls**	**Type = 0001,** Bricks stop tanks and bullets, but they can be slowly chipped away by shooting at them
	**Steel Walls**	**Type = 0010,** Steel walls completely stop tanks and bullets. They cannot be destroyed, unless player's tank has collected three stars.
	**Trees**	**Type = 0011,** Trees allow tanks and bullets to pass through unchecked. But they partially obscure the view beneath the tree tops.
	**Water**	**Type = 0100,** Tanks cannot traverse water, but bullets can safely fly across.

Sprites of the game are specified by their positions, types and colors. We will use a 32-bit integer to store a sprite in the Nios program, and it has following format:

X	Y	Types	Color
23-15	14-6	5-2	1-0

X: specify the horizontal position of this sprite (0-467 in pixels)

Y: specify the vertical position of this sprite (0-467 in pixels)

Types: specify the types of the sprites; details are given in the following table

Color: specify the color of the sprite, grey = 00, yellow = 01, red = 10 and green = 11

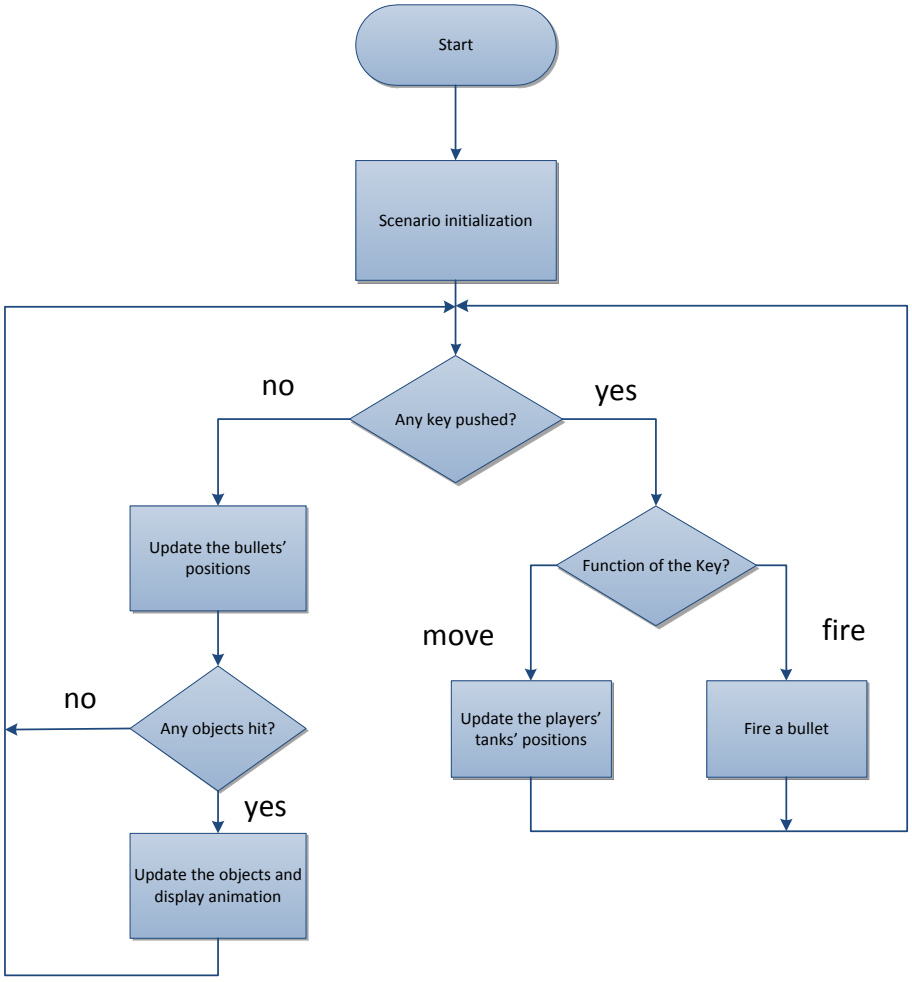
	**Player1's tank**	**Type = 0001**
	**Basic Tank**	**Type = 0011**
	**Power Tank**	**Type = 0101**

	**Armor Tank**	**Type = 0110**
	**Star**	**Type = 1010**

Algorithm:

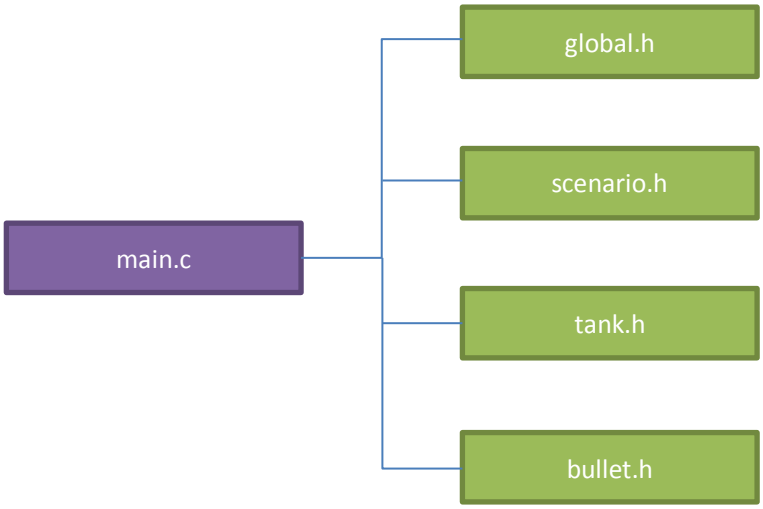
The algorithm of this game is mainly formed by three parts: scenario initialization, bullets trace and keyboard actions. The scenario initialization is executed at the start of the game. The bullets trace is always executed in the main loop, since it has to trace the positions' of bullets and detect the destruction of obstacles, like brick walls, when the bullet hits them. The keyboard action part is executed when a key is pushed, then the player's tank will move in a certain direction for a small step or the player's tank will fire a bullet.

The basic flow-char of the algorithm is plotted in the following figure:



Files:

In this project, we use five separate files to build our software system. Their structures and relationships are described in the figure below:



- main.c implements the main loop of the program.
- global.h defines all the global macros and variables.
- scenario.h defines all the scenarios of different stages and related functions.
- tank.h defines all the functions related to tanks, like updating positions and firing.
- bullet.h defines all the functions related to bullets, like updating positions, detecting collisions with tanks, obstacles and even other bullets. The functions handling explosions are also defined in this file.

main.c

In the main.c, the main loop of the program is defined. Before the loop starts, we need to initialize the program first:

- system_init: function initializing the system variables and flags.
- scenario_init: function loading the scenario to the hardware for display.
- player1_init: function setting the initial position of the player1.

Then inside the main loop, we have two branches: one is executed when the keyboard gets hit while the other is executed when the keyboard gets no hit. When a key is pressed, the function updatePlayer1() will be executed. In the other cases, the functions updateExplosion(), updateBullets() and updateTanks() will executed. Their functions will be briefly described below:

- updatePlayer1: update the position and direction of the player1 according to the key pressed. When the new position does not collide with other tanks and obstacles, it will be sent to the VGA driver for display.
- updateExplosion: update the explosions appeared on the screen. Big explosion has an animation effect and should be updated regularly. The small explosion only needs to be displayed on the screen for a certain time. Counters are used to update explosions regularly.
- updateBullets: update the bullets' positions and check whether they hit other tanks, bullets and obstacles. When they hit other bullets, they both need to be destroyed from the screen. When they hit obstacles, the scenario needs to be updated.
- updateTanks: update the enemy tanks. New tanks will be generated randomly; Tanks were hit by player1's bullet need to be destroyed; Tanks will randomly choose to turn left, turn right and fire.

The complete code is given below:

```
int main()
{
    system_init();
    scenario_init();
    player1_init();
    while (1)
    {
        while (!KEYPRESSED)
        {
            updateExplosion();
            updateBullets();
            updateTanks();
        }
        updatePlayer1();
    }
    return 0;
}
```

scenario.h

In this file, the scenario of different stages are defined in the two dimensional arrays. The scenario is defined for a 13 by 13 blocks. Each block is defined by a byte. The left-most four bits are used to indicate whether four sub-blocks exist or not. The right-most four bits are used to indicate the type of the block. For example, 0xf1 means a block formed by four sub-blocks of brick wall while 0xf2 represents a block of steel. The scenario will be loaded to the hardware registers when the game starts. And they will be updated when they are hit by bullets and the registers will also be updated according to the software.

global.h

The system_init function is defined in this file. And all the global variables and macros are also defined here. The type of the obstacles and tanks are defined by macros, which are more meaningful than the binary codes.

bullet.h

In this file, the key function is updateBullets(), which update all bullets. When a bullet is updated, first we will check whether it will hit other bullets or not. If two bullets hit each other, they will both destroyed and removed from the screen. Second, we need to check whether the bullet hit tanks or not. If the bullet hits a tank, the tank needs to be destroyed if the bullet is fired by the tanks of the other side. Third, we have to check whether the bullet hits an obstacle or not. If the bullet hits an obstacle and the obstacle is something can be destroyed, the scenario need to be updated both in the software and hardware. Finally, if the bullet hits nothing, its position will be updated by a step of 2 pixels.

tank.h

In this file, the key function is updateTanks(), which updates all the enemy tanks' positions and directions. First, we need to flip a coin and decided whether an enemy tank will turn left, right or even back. After that, if there is no obstacle and no tank in front of it, the tank moves by a step of 4 pixels. The bullet will also be fired randomly.

Contribution

Tian Chu is in charge of design partition and software design, such as building the Nios system in SOPC builder and C programming.

Quan Yuan and Yuanzhao Huangfu work together on the VGA display hardware design. Specifically, Quan Yuan is in charge of the UI layout, game graphic design and generating corresponding .mif files. Yuanzhao Huangfu is in charge of VHDL coding and debugging, which makes wonderful graphics displayed correctly and move smoothly.

Liuxun Zhu and Tianchen Li work together on the Audio and Music hardware design. Specifically, Tianchen Li is in charge of translating sound file in .wav format into .mif format which can be used by the ROM generated by Quartus. In addition, he makes the keyboard work and does a series of useful game tests. Liuxun Zhu is in charge of VHDL coding and debugging, which adds wonderful sounds and music into the game.

Lessons Learned

In this project, we had learned a lot of lessons on how to work as a team and how to make a good and clean design. First of all, appropriate design partition is a key for us to work as a team. If the design is not partitioned properly, some team members may be idling while the others are busy working. Working in parallel is important for a design team to make progress. Second, good software data structure is important for implementing complex functions. Poor data structure may be easy to quickly make the program run, but will definitely cause a lot of extra work when implementing complex functions. Third, backup source files regularly with a modification date associated will greatly help when they are needed. Fourth, online files sharing tools, like “Dropbox”, can help us to organize and share the latest work easily.

Source Files

main.c

```
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <alt_types.h>
#include <unistd.h>
#include "scenario.h"
#include "bullet.h"
#include "tank.h"
#include "global.h"

int main()
{
    system_init();
    scenario_init();
    player1_init();

    while (1)
    {
        while (!KEYPRESSED)
        {
            bulletCounterCheck();
            updateExplosion();
            updateBullets();
            tankCounterCheck();
            updateTanks();
            updatePlayer1();
        }
        updateKeyboard();
    }

    return 0;
}
```

scenario.h

```
#ifndef SCENARIO_H_
```

```
#define SCENARIO_H_

#include "global.h"

//                    Y   X
unsigned char scenario[13][13] =
{
    // 0    1    2    3    4    5    6    7    8    9    10   11   12

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
0

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
1

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
2

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0xf2,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
3

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
4

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
5

{0x00,0x00,0x00,0x00,0x00,0xf1,0x00,0xf1,0x00,0x00,0x00,0x00,0x00},//
6

{0xf2,0x00,0xf1,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0xf1,0x00,0xf2},//
7

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
8

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
9

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
10

{0x00,0xf1,0x00,0xf1,0x00,0x81,0xc1,0x41,0x00,0xf1,0x00,0xf1,0x00},//
11
```

```
    {0x00,0x00,0x00,0x00,0x00,0xa1,0xf6,0x51,0x00,0x00,0x00,0x00,0x00}
//12
};

//                    Y   X
unsigned char level1[13][13] =
{
    // 0    1    2    3    4    5    6    7    8    9    10   11   12

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
0

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
1

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
2

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0xf2,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
3

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
4

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
5

{0x00,0x00,0x00,0x00,0x00,0xf1,0x00,0xf1,0x00,0x00,0x00,0x00,0x00},//
6

{0xf2,0x00,0xf1,0xf1,0x00,0xf1,0xf2,0xf1,0x00,0xf1,0xf1,0x00,0xf2},//
7

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
8

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
9

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
10

{0x00,0xf1,0x00,0xf1,0x00,0x81,0xc1,0x41,0x00,0xf1,0x00,0xf1,0x00},//
11
```

```
    {0x00,0x00,0x00,0x00,0x00,0xa1,0xf6,0x51,0x00,0x00,0x00,0x00,0x00}
//12
};

//                    Y   X
unsigned char level2[13][13] =
{
    // 0    1    2    3    4    5    6    7    8    9    10   11   12

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
0

{0x00,0xf2,0xf2,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0xf2,0xf2,0x00},//
1

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
2

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0xf2,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
3

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
4

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
5

{0x00,0x00,0xf4,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf4,0x00,0x00},//
6

{0xf4,0x00,0xf4,0xf4,0xf3,0xf3,0xf3,0xf3,0xf3,0xf4,0xf4,0x00,0xf4},//
7

{0x00,0x00,0xf4,0x00,0xf3,0xf3,0xf3,0xf3,0xf3,0x00,0xf4,0x00,0x00},//
8

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0xf2,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
9

{0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00,0xf1,0x00},//
10

{0x00,0xf1,0x00,0xf1,0x00,0x81,0xc1,0x41,0x00,0xf1,0x00,0xf1,0x00},//
11
```

```
    {0x00,0x00,0x00,0x00,0x00,0xa1,0xf6,0x51,0x00,0x00,0x00,0x00,0x00}
//12
};

//                    Y   X
unsigned char level3[13][13] =
{
    // 0    1    2    3    4    5    6    7    8    9    10   11   12

{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},//
0

{0x00,0xf3,0xf3,0xf3,0x00,0x00,0x00,0xf1,0x00,0xf2,0x00,0xf1,0x00},//
1

{0x00,0xf3,0xf3,0xf3,0xf3,0x00,0x00,0xf1,0x00,0xf2,0x00,0xf1,0x00},//
2

{0x00,0xf3,0xf3,0xf3,0xf3,0x00,0xf2,0xf1,0x00,0xf2,0x00,0xf1,0x00},//
3

{0x00,0xf3,0xf3,0xf3,0xf3,0x00,0xf2,0xf1,0x00,0xf2,0x00,0xf1,0x00},//
4

{0x00,0x00,0xf3,0xf3,0x00,0x00,0x00,0x00,0x00,0xf2,0x00,0x00,0x00},//
5

{0xf2,0x00,0x00,0x00,0x00,0xf3,0x00,0xf3,0x00,0x00,0x00,0x00,0x00},//
6

{0xf2,0x00,0xf1,0xf2,0x00,0xf3,0xf2,0xf3,0xf3,0xf3,0xf1,0x00,0xf2},//
7

{0x00,0x00,0x00,0xf2,0x00,0x00,0xf2,0xf3,0xf3,0xf3,0xf3,0x00,0x00},//
8

{0x00,0xf2,0x00,0xf2,0x00,0xf3,0x00,0xf3,0xf3,0xf3,0xf3,0x00,0x00},//
9

{0x00,0xf2,0x00,0xf2,0x00,0xf3,0x00,0xf3,0xf3,0xf3,0xf3,0x00,0x00},//
10

{0x00,0xf2,0x00,0xf2,0x00,0x81,0xc1,0x41,0x00,0xf3,0xf3,0x00,0x00},//
11
```

```
    {0x00,0x00,0x00,0x00,0x00,0xa1,0xf6,0x51,0x00,0x00,0x00,0x00,0x00}
//12
};

void playSoundFire()
{
    AUDIO = FIRE;
}

void playInit()
{
    AUDIO = 0x00000f00;
}

void playSoundExplosion()
{
    AUDIO = EXPLOSION;
}

void playStartMusic()
{
    AUDIO = INIT;
    usleep(100);
    AUDIO = START;
}

void playSoundStop()
{
    AUDIO = 0x00000000;
}

void writeScenario(unsigned int y, unsigned int x)
{
    IOWR_32DIRECT(VGA_BASE,0,(scenario[y][x] + (y<<8) + (x<<12) +
(1<<31)));
    usleep(VGA_DELAY);
}

void writeSprite(unsigned int sprite, unsigned int type, unsigned int
color, unsigned int index)
{
    IOWR_32DIRECT(VGA_BASE,0,(color + (type<<2) + (sprite<<6) +
(index<<26) + (0<<31)));
}
```

```
void scenario_init()
{
    unsigned int i,j;
    for (i=0;i<MAXEXPLO;i++)
    {
        writeSprite((unsigned int)tankInit, SMALLEXPLO, GREY, i);
        writeSprite((unsigned int)tankInit, SMALLEXPLO, GREY, i);
    }
    VGA = 0xffffffff;
    if (level==1)
    {
        for (i=640; i>180; i--)
        {
            VGA = 0x00fff000 + i;
            usleep(5000);
        }
        while (!KEYPRESSED){};
        VGA = 0x00ffffff;
    }
    playStartMusic();
    VGA = 0x0fffffff + (level<<28);
    for (i=0;i<numOfEnemy;i++)
        numOfEnemyTemp += (1<<(19-i));
    VGA = 0xff000000 + numOfEnemyTemp;
    usleep(VGA_DELAY);
    if (level==1)
    {
        for (i=0; i<13; i++)
            for (j=0; j<13; j++)
                scenario[i][j]=level1[i][j];
    }
    else if (level==2)
    {
        for (i=0; i<13; i++)
            for (j=0; j<13; j++)
                scenario[i][j]=level2[i][j];
    }
    else if (level==3)
    {
        for (i=0; i<13; i++)
            for (j=0; j<13; j++)
                scenario[i][j]=level3[i][j];
    }
```

```
    for (i=0; i<13; i++)
        for (j=0; j<13; j++)
        {
            writeScenario(i,j);
            writeScenario(i,j);
        }
    VGA = 0x00000000;
    usleep(VGA_DELAY);
    tankCounterSet();
    bulletCounterSet();
}

void setPlayer1Life(unsigned int life)
{
    VGA = (0xfffff0) + ((life)<<24);
}

//initialize the player1
void player1_init()
{
    player1 = setPos(144,432,player1);
    player1 = setDir(UP,player1);
    setPlayer1Life(player1Life);
    currentDir=STOP;
    fireFlag=0;
    writeSprite(player1,PLAYER1,GREY,15);
    usleep(5000);
    writeSprite(player1,PLAYER1,GREY,15);
}

void gameover()
{
    system_init();
    level = 1;
    scenario_init();
    player1_init();
}

void gameWin()
{
    if(++level<=3)
    {
        system_init();
        scenario_init();
```

```
        player1_init();
    }
    else
    {
        level=1;
        system_init();
        scenario_init();
        player1_init();
    }
}

//set the position of the sprite
unsigned int setPos(unsigned int x, unsigned int y, unsigned int sprite)
{
    return (x<<11) + (y<<2) + (sprite&0x3);
}

//increase the X position of the sprite
unsigned int incX(unsigned int x, unsigned int sprite)
{
    return setPos((X(sprite))+x,Y(sprite),sprite);
}

//increase the Y position of the sprite
unsigned int incY(unsigned int y, unsigned int sprite)
{
    return setPos(X(sprite),(Y(sprite))+y,sprite);
}

//decrease the X position of the sprite
unsigned int decX(unsigned int x, unsigned int sprite)
{
    return setPos((X(sprite))-x,Y(sprite),sprite);
}

//decrease the Y position of the sprite
unsigned int decY(unsigned int y, unsigned int sprite)
{
    return setPos(X(sprite),(Y(sprite))-y,sprite);
}

//set the direction of the sprite
unsigned int setDir(unsigned int dir, unsigned int sprite)
{
```

```
    return dir + (sprite&0xffffc);
}

//tank can not move over these types
unsigned int notPenetrable(unsigned int type)
{
    if (type==BRICK || type==STEEL || type==WATER || type==BASE)
        return 1;
    else
        return 0;
}

//test whether this position is occupied by an obstacle or other tanks
unsigned int occupied(unsigned int x, unsigned int y)
{
    unsigned int X_SQR = x/36;
    unsigned int Y_SQR = y/36;

    if (x<=(X_SQR*36+18) && y<=(Y_SQR*36+18))
        if (BLOCKS(scenario[Y_SQR][X_SQR])&0x1 &&
notPenetrable(TYPES(scenario[Y_SQR][X_SQR])))
            return 1;
        else
            return 0;
    else if (y<=(Y_SQR*36+18))
        if (BLOCKS(scenario[Y_SQR][X_SQR])&0x2 &&
notPenetrable(TYPES(scenario[Y_SQR][X_SQR])))
            return 1;
        else
            return 0;
    else if (x<=(X_SQR*36+18))
        if (BLOCKS(scenario[Y_SQR][X_SQR])&0x4 &&
notPenetrable(TYPES(scenario[Y_SQR][X_SQR])))
            return 1;
        else
            return 0;
    else
        if (BLOCKS(scenario[Y_SQR][X_SQR])&0x8 &&
notPenetrable(TYPES(scenario[Y_SQR][X_SQR])))
            return 1;
        else
            return 0;
}
```

```
unsigned int isTankHere(unsigned int x, unsigned int y)
{
    int i;
    for (i=0;i<MAXTANK;i++)
        if (tanks[i]!=-1)
            if (x>=(X(tanks[i])) && x<=(X(tanks[i]))+35 && y>=(Y(tanks[i]))
&& y<=(Y(tanks[i]))+35)
                return 1;
    if (x>=(X(player1)) && x<=(X(player1))+35 && y>=(Y(player1)) &&
y<=(Y(player1))+35)
                return 1;
//    if (x>=(X(player2)) && x<=(X(player2))+35 && y>=(Y(player2)) &&
y<=(Y(player2))+35)
//                return 1;
    return 0;
}

//check whether the sprite can move left
unsigned int canMoveLeft(unsigned int sprite)
{
    unsigned int flag = 1;
    if ((X(sprite))<step)
        flag = 0;
    if (occupied((X(sprite))-step,(Y(sprite))+2))
        flag = 0;
    else if (occupied((X(sprite))-step,(Y(sprite))+18))
        flag = 0;
    else if (occupied((X(sprite))-step,(Y(sprite))+33))
        flag = 0;
    else if (isTankHere((X(sprite))-step,Y(sprite)))
        flag = 0;
    else if (isTankHere((X(sprite))-step,(Y(sprite))+35))
        flag = 0;
    return flag;
}

//check whether the sprite can move right
unsigned int canMoveRight(unsigned int sprite)
{
    unsigned int flag = 1;
    if ((X(sprite))+35+step>468)
        flag = 0;
    if (occupied((X(sprite))+35+step,(Y(sprite))+2))
        flag = 0;
```

```
    else if (occupied((X(sprite))+35+step,(Y(sprite))+18))
        flag = 0;
    else if (occupied((X(sprite))+35+step,(Y(sprite))+33))
        flag = 0;
    else if (isTankHere((X(sprite))+35+step,Y(sprite)))
        flag = 0;
    else if (isTankHere((X(sprite))+35+step,(Y(sprite))+35))
        flag = 0;
    return flag;
}

//check whether the sprite can move up
unsigned int canMoveUp(unsigned int sprite)
{
    unsigned int flag = 1;
    if ((Y(sprite))<step)
        flag = 0;
    if (occupied((X(sprite))+2,(Y(sprite))-step))
        flag = 0;
    else if (occupied((X(sprite))+18,(Y(sprite))-step))
        flag = 0;
    else if (occupied((X(sprite))+33,(Y(sprite))-step))
        flag = 0;
    else if (isTankHere(X(sprite),(Y(sprite))-step))
        flag = 0;
    else if (isTankHere((X(sprite))+35,(Y(sprite))-step))
        flag = 0;
    return flag;
}

//check whether the sprite can move down
unsigned int canMoveDown(unsigned int sprite)
{
    unsigned int flag = 1;
    if ((Y(sprite))+35+step>468)
        flag = 0;
    if (occupied((X(sprite))+2,(Y(sprite))+35+step))
        flag = 0;
    else if (occupied((X(sprite))+18,(Y(sprite))+35+step))
        flag = 0;
    else if (occupied((X(sprite))+33,(Y(sprite))+35+step))
        flag = 0;
    else if (isTankHere(X(sprite),(Y(sprite))+35+step))
        flag = 0;
```

```
    else if (isTankHere((X(sprite))+35,(Y(sprite))+35+step))
        flag = 0;
    return flag;
}

//the sprite fires the bullet
void fireBullet(unsigned int sprite, unsigned int spriteIndex)
{
    if (spriteIndex==MAXBULLET-1)
    {
        if (fireCounter<10)
            return;
        counterEnable |= (1<<30);
        bulletCounterSet();
        fireCounter=0;
    }

    if (bullets[spriteIndex])
        return;
    unsigned int bullet = 0;
    if ((DIR(sprite))==UP && (Y(sprite))>8)
    {
        bullet = setPos((X(sprite))+15,(Y(sprite))-8,bullet);
        bullet = setDir(UP,bullet);
    }
    else if ((DIR(sprite))==DN && (Y(sprite))+35+8<468)
    {
        bullet = setPos((X(sprite))+15,(Y(sprite))+35+8,bullet);
        bullet = setDir(DN,bullet);
    }
    else if ((DIR(sprite))==LF && (X(sprite))>8)
    {
        bullet = setPos((X(sprite))-8,(Y(sprite))+15,bullet);
        bullet = setDir(LF,bullet);
    }
    else if ((DIR(sprite))==RT && (X(sprite))+35+8<468)
    {
        bullet = setPos((X(sprite))+35+8,(Y(sprite))+15,bullet);
        bullet = setDir(RT,bullet);
    }
    if (bullet)//bullet is not empty
    {
        bullets[spriteIndex]=bullet;
        writeSprite((unsigned int)bullet, BULLET,
```

```
GREY,spriteIndex+MAXTANK);
        counterEnable |= (1<<spriteIndex);
        bulletCounterSet();
    }
    if (spriteIndex==MAXBULLET-1)
    {
        playSoundFire();
        playSoundStop();
    }
    //printf("fire a bullet at (%d,%d)\n",X(bullet),Y(bullet));
}

//update the position of the player1 and fire the bullet if space key is
pressed
void updateKeyboard()
{
    switch (KEY)
    {
        case 0x1c:      //'a', move left
            if (breakFlag)
            {
                breakFlag=0;
                currentDir=STOP;
            }
            else
                currentDir=LF;
            break;
        case 0x1d:      //'w', move up
            if (breakFlag)
            {
                breakFlag=0;
                currentDir=STOP;
            }
            else
                currentDir=UP;
            break;
        case 0x23:      //'d', move right
            if (breakFlag)
            {
                breakFlag=0;
                currentDir=STOP;
            }
            else
                currentDir=RT;
```

```
            break;
        case 0x1b:      //'s', move down
            if (breakFlag)
            {
                breakFlag=0;
                currentDir=STOP;
            }
            else
                currentDir=DN;
            break;
        case 0x29:      //"space", fire a bullet
            if (breakFlag)
            {
                breakFlag = 0;
                fireFlag = 0;
            }
            else
                fireFlag = 1;
            break;
        case 0xf0:      //break code, setup a flag
            breakFlag = 1;
            break;
    }
}

void updateExplosion()
{
    int i;
    for (i=0;i<MAXEXPLO;i++)
    {
        if (explosion[i]==BIGEXPLO && exploFlag[i]==1)
        {
            writeSprite((unsigned int)(exploPos[i]+(9<<11)+(9<<2)),
SMALLEXPLO, GREY, i);//small explosion
            exploFlag[i]=2;
        }
        else if (explosion[i]==BIGEXPLO && exploFlag[i]==2)
        {
            if (counterEnable&(1<<(MAXBULLET+i)))
                continue;
            counterEnable |= (1<<(MAXBULLET+i));
            bulletCounterSet();
            if (exploCounter[i]++>15)
            {
```

```
                writeSprite((unsigned int)exploPos[i], BIGEXPLO, GREY,
i);//big explosion
                exploFlag[i]=3;
                exploCounter[i]=0;
            }
        }
        else if (explosion[i]==BIGEXPLO && exploFlag[i]==3)
        {
            if (counterEnable&(1<<(MAXBULLET+i)))
                continue;
            counterEnable |= (1<<(MAXBULLET+i));
            bulletCounterSet();
            if (exploCounter[i]++>20)
            {
                writeSprite((unsigned int)tankInit, SMALLEXPLO, GREY, i);
                explosion[i]=0;
                exploFlag[i]=0;
                exploCounter[i]=0;
            }
        }
        else if (explosion[i]==SMALLEXPLO && exploFlag[i]==1)
        {
            writeSprite((unsigned int)(exploPos[i]-(6<<11)-(6<<2)),
SMALLEXPLO, GREY, i);//small explosion
            exploFlag[i]=2;
        }
        else if (explosion[i]==SMALLEXPLO && exploFlag[i]==2)
        {
            if (counterEnable&(1<<(MAXBULLET+i)))
                continue;
            counterEnable |= (1<<(MAXBULLET+i));
            bulletCounterSet();
            if (exploCounter[i]++>20)
            {
                writeSprite((unsigned int)tankInit, SMALLEXPLO, GREY, i);
                explosion[i]=0;
                exploFlag[i]=0;
                exploCounter[i]=0;
            }
        }
    }
}

#endif /*SCENARIO_H_*/
```

tank.h

```
#ifndef TANK_H_
#define TANK_H_

unsigned int indexOfTank = 0;

unsigned int turnLeft(unsigned int tank)
{
    if ((DIR(tank))==UP)
        return setDir(LF,tank);
    else if ((DIR(tank))==DN)
        return setDir(RT,tank);
    else if ((DIR(tank))==LF)
        return setDir(DN,tank);
    else if ((DIR(tank))==RT)
        return setDir(UP,tank);
}

unsigned int turnRight(unsigned int tank)
{
    if ((DIR(tank))==UP)
        return setDir(RT,tank);
    else if ((DIR(tank))==DN)
        return setDir(LF,tank);
    else if ((DIR(tank))==LF)
        return setDir(UP,tank);
    else if ((DIR(tank))==RT)
        return setDir(DN,tank);
}

unsigned int turnAround(unsigned int tank)
{
    if ((DIR(tank))==UP)
        return setDir(DN,tank);
    else if ((DIR(tank))==DN)
        return setDir(UP,tank);
    else if ((DIR(tank))==LF)
        return setDir(RT,tank);
    else if ((DIR(tank))==RT)
        return setDir(LF,tank);
}

void generateTank()
```

```
{
    //used to control the generating rate
    if (counterEnable&(1<<31))
        return;
    else if (generateTankCounter<500)
    {
        generateTankCounter++;
        counterEnable |= (1<<31);
        bulletCounterSet();
        return;
    }
    counterEnable |= (1<<31);
    bulletCounterSet();
    generateTankCounter=0;

    unsigned int tank = 0;
    unsigned int random = rand()%100;
    //unsigned int random = 20;
    if (random<30 && existTank(0,0)==-3)//generate the tank at the left
born point
    {
        tank = setPos(0,0,tank);
        //printf("A tank is generated at left born point!\n");
    }
    else if (random<60 && existTank(432,0)==-3)//generate the tank at the
right born point
    {
        tank = setPos(432,0,tank);
        //printf("A tank is generated at right born point!\n");
    }
    else if (existTank(216,0)==-3)//generate the tank at the middle born
point
    {
        tank = setPos(216,0,tank);
        //printf("A tank is generated at middle born point!\n");
    }
    else
        return;

    //let the tank face downward
    tank = setDir(DN,tank);

    int i;
    for (i=0;i<MAXTANK;i++)//insert this tank into the list
```

```
    {
        if (tanks[i]==tankInit)//tanks[i] is empty
        {
            tanks[i]=tank;
            //printf("tankIndex=%d\n",i);
            numOfTank++;
            tankNum++;
            int randTemp = rand()%100;
            if (randTemp<40)
            {
                writeSprite(tank, BASIC_TANK, GREY, i);
                tankLevel[i]=BASIC_TANK;
            }
            else if (randTemp<80)
            {
                writeSprite(tank, POWER_TANK, GREY, i);
                tankLevel[i]=POWER_TANK;
            }
            else
            {
                writeSprite(tank, SUPER_TANK, GREY, i);
                tankLevel[i]=SUPER_TANK;
            }
            break;
        }
    }
}

int eatPowerup(unsigned int sprite)
{
    if(insideTank(powerup, X(sprite), Y(sprite)) || insideTank(powerup,
(X(sprite))+35, Y(sprite))
    || insideTank(powerup, X(sprite), (Y(sprite))+35) ||
insideTank(powerup, (X(sprite))+35, (Y(sprite))+35))
        return 1;
    else
        return 0;
}

void powerUp(unsigned int sprite,unsigned int tankIndex)
{
    powerup = tankInit;
    writeSprite(powerup,POWERUP,GREY,14);
    int random = rand()%100;
```

```
if (random<30)//change the base
{
   if (sprite==player1)
   {
      scenario[12][5]=0xa2;
      scenario[11][5]=0x82;
      scenario[11][6]=0xc2;
      scenario[11][7]=0x42;
      scenario[12][7]=0x52;
   }
   else
   {
      scenario[12][5]=0x00;
      scenario[11][5]=0x00;
      scenario[11][6]=0x00;
      scenario[11][7]=0x00;
      scenario[12][7]=0x00;
   }
   writeScenario(12,5);
   writeScenario(12,5);
   writeScenario(11,5);
   writeScenario(11,5);
   writeScenario(11,6);
   writeScenario(11,6);
   writeScenario(11,7);
   writeScenario(11,7);
   writeScenario(12,7);
   writeScenario(12,7);
}
else if (random<60)//level up
{
    if (sprite==player1)
    {
       if (player1Level==PLAYER1)
          player1Level = PLY1;
       else if (player1Level==PLY1)
          player1Level = PLY2;
    }
    else if(tankLevel[tankIndex]==BASIC_TANK)
    {
       tankLevel[tankIndex]=POWER_TANK;
    }
    else if(tankLevel[tankIndex]==POWER_TANK)
    {
```

```
            tankLevel[tankIndex]=SUPER_TANK;
        }
    }
    else//freeze the tank
    {
        if (sprite==player1)
        {
            tankStop=50000;
        }
        else
        {
            player1Stop=80;
        }
    }
}

void updatePlayer1()
{
    if (tankEnable&(1<<MAXTANK))
        return;
    tankEnable |= (1<<MAXTANK);
    tankCounterSet();

    if (player1Stop>0)
    {
        player1Stop--;
        return;
    }

    switch(currentDir)
    {
        case LF:
            player1 = setDir(LF,player1);
            if (canMoveLeft(player1))
            {
                player1 = decX(step,player1);
            }
            writeSprite(player1, player1Level, GREY,15);
            break;
        case UP:
            player1 = setDir(UP,player1);
            if (canMoveUp(player1))
            {
                player1 = decY(step,player1);
```

```
            }
            writeSprite(player1, player1Level, GREY,15);
            break;
        case RT:
            player1 = setDir(RT,player1);
            if (canMoveRight(player1))
            {
                player1 = incX(step,player1);
            }
            writeSprite(player1, player1Level, GREY,15);
            break;
        case DN:
            player1 = setDir(DN,player1);
            if (canMoveDown(player1))
            {
                player1 = incY(step,player1);
            }
            writeSprite(player1, player1Level, GREY,15);
            break;
    }
    if (fireFlag)
        fireBullet(player1,MAXBULLET-1);
    if (eatPowerup(player1))
        powerUp(player1,15);
}

void generatePowerup()
{
    if (counterEnable&(1<<29))
        return;
    else if (generatePowerupCounter<500)
    {
        generatePowerupCounter++;
        counterEnable |= (1<<29);
        bulletCounterSet();
        return;
    }
    counterEnable |= (1<<29);
    bulletCounterSet();
    generatePowerupCounter=0;

    powerup = setPos(rand()%400,rand()%400,powerup);
    powerup = setDir(DN, powerup);
    writeSprite(powerup,POWERUP,GREY,14);
```

```
}

//update all enemy tanks, change directions and fire bullets
//generate enemy tanks under proper condition
void updateTanks()
{
    unsigned int i;
    if (numOfTank<MAXENEMY && tankNum<MAXTANK)
        generateTank();

    if (rand()%100<2 && powerup==tankInit)
        generatePowerup();

    if (tankStop>0)
    {
        tankStop--;
        return;
    }

    if (++musicCounter>5000)
    {
        playInit();
        musicCounter = 0;
    }

    for (i=0;i<MAXTANK;i++)//loop for each tank
    {
        if (tanks[i]!=tankInit)//tank[i] is not empty, then process it
        {
            if (tankEnable&(1<<i))
                continue;
            tankEnable |= (1<<i);
            tankCounterSet();

            int mod = 100;
            int upFlag = canMoveUp(tanks[i]);
            int downFlag = canMoveDown(tanks[i]);
            int leftFlag = canMoveLeft(tanks[i]);
            int rightFlag = canMoveRight(tanks[i]);

            if (upFlag+downFlag+leftFlag+rightFlag>3)//there are four
ways, decrease possibility of turning
                mod = 2000;
            int random = rand()%mod;
```

```
            if (upFlag+downFlag+leftFlag+rightFlag==1)//only one way,
turn around
            {
                if (rand()%100<50)
                    tanks[i]=turnAround(tanks[i]);
            }
            else if (random<30)//turn left
            {
                int flag = 0;
                switch(DIR(tanks[i]))
                {
                    case LF:  flag = downFlag;    break;
                    case RT: flag = upFlag;       break;
                    case DN: flag = rightFlag;    break;
                    case UP: flag = leftFlag;      break;
                }
                if (flag)
                    tanks[i]=turnLeft(tanks[i]);
                else if (rand()%100<5)
                    tanks[i]=turnLeft(tanks[i]);
            }
            else if (random<60)//turn right
            {
                int flag = 0;
                switch(DIR(tanks[i]))
                {
                    case LF:  flag = upFlag;    break;
                    case RT: flag = downFlag; break;
                    case DN: flag = leftFlag;  break;
                    case UP: flag = rightFlag;  break;
                }
                if (flag)
                    tanks[i]=turnRight(tanks[i]);
                else if (rand()%100<5)
                    tanks[i]=turnRight(tanks[i]);
            }
            else if (random<80 && tankLevel[i]!=SUPER_TANK)
                fireBullet(tanks[i],i);
            if (tankLevel[i]==SUPER_TANK)
                fireBullet(tanks[i],i);
            switch (DIR(tanks[i]))
            {
                case LF:     //'a', move left
                    if (leftFlag)
```

```
                {
                    tanks[i] = decX(step,tanks[i]);
                }
                break;
            case UP:     //'w', move up
                if (upFlag)
                {
                    tanks[i] = decY(step,tanks[i]);
                }
                break;
            case RT:     //'d', move right
                if (rightFlag)
                {
                    tanks[i] = incX(step,tanks[i]);
                }
                break;
            case DN:     //'s', move down
                if (downFlag)
                {
                    tanks[i] = incY(step,tanks[i]);
                }
                break;
        }
        if (eatPowerup(tanks[i]))
            powerUp(tanks[i],i);
        //printf("tank[%d]=(%d,%d)\n",i,X(tanks[i]),Y(tanks[i]));
        writeSprite((unsigned int)tanks[i], tankLevel[i], GREY, i);
    }
  }
}

#endif /*TANK_H_*/
```

bullet.h

```
#ifndef BULLET_H_
#define BULLET_H_

#include "global.h"

unsigned int hitable(unsigned int type)//bullet can not fly over
{
    if (type==BRICK || type==STEEL || type==BASE)
        return 1;
```

```
    else
        return 0;
}

//check whether a coordinate is inside a tank or not
unsigned int insideTank(unsigned tank, unsigned int x, unsigned int y)
{
    if (x>=(X(tank)) && x<=(X(tank))+35 && y>=(Y(tank)) &&
y<=(Y(tank))+35)
        return 1;
    else
        return 0;
}

//return the index of the tank hit by the bullet, -1=player1, -2=player2
unsigned int hitTank(unsigned int bullet)
{
    if (insideTank(player1,X(bullet),Y(bullet)))
        return -1;
//    else if (player2,X(bullet),Y(bullet))
//        return -2;
    int i;
    for (i=0;i<MAXTANK;i++)
        if (insideTank(tanks[i],X(bullet),Y(bullet)))
            return i;
    return -3;
}

//return the index of the tank hit by the bullet, -1=player1, -2=player2
unsigned int existTank(unsigned int x, unsigned int y)
{
    if (insideTank(player1,x,y) || insideTank(player1,x+35,y) ||
insideTank(player1,x,y+35) || insideTank(player1,x+35,y+35))
        return -1;
    int i;
    for (i=0;i<MAXTANK;i++)
        if (insideTank(tanks[i],x,y) || insideTank(tanks[i],x+35,y) ||
insideTank(tanks[i],x,y+35) || insideTank(tanks[i],x+35,y+35))
            return i;
    return -3;
}

//update the tank and bullet list, when the bullet hit the tank
void tankBomb(unsigned int x, unsigned int y, unsigned int tankIndex,
```

```
unsigned int bulletIndex)
{
    bullets[bulletIndex]=0;//remove this bullet

    if (tankIndex==-1)//player1 was hit
    {
        if (player1Level==PLAYER1)
        {
            explosion[MAXEXPLO-1]=BIGEXPLO;
            exploFlag[MAXEXPLO-1]=1;

exploPos[MAXEXPLO-1]=setPos(X(player1),Y(player1),exploPos[MAXEXPLO-1
]);
            player1Life--;
            setPlayer1Life(player1Life);
            if (player1Life<=0)
                gameover();
            player1_init();
        }
        else if (player1Level==PLY1)
        {
            player1Level=PLAYER1;
            writeSprite(player1, player1Level, GREY,15);
        }
        else if (player1Level==PLY2)
        {
            player1Level=PLY1;
            writeSprite(player1, player1Level, GREY,15);
        }
    }
    else if (bulletIndex==(MAXBULLET-1))//enemy was hit
    {
        if (tankLevel[tankIndex]==SUPER_TANK)
        {
            playSoundExplosion();
            //usleep(10000);
            playSoundStop();
            tankLevel[tankIndex]=POWER_TANK;
            writeSprite(tanks[tankIndex], POWER_TANK, GREY, tankIndex);
            explosion[tankIndex]=SMALLEXPLO;
            exploFlag[tankIndex]=1;
            exploPos[tankIndex]=setPos(x,y,exploPos[tankIndex]);
        }
        else if (tankLevel[tankIndex]==POWER_TANK)
```

```
        {
            playSoundExplosion();
            //usleep(10000);
            playSoundStop();
            tankLevel[tankIndex]=BASIC_TANK;
            writeSprite(tanks[tankIndex], BASIC_TANK, GREY, tankIndex);
            explosion[tankIndex]=SMALLEXPLO;
            exploFlag[tankIndex]=1;
            exploPos[tankIndex]=setPos(x,y,exploPos[tankIndex]);
        }
        else
        {
            playSoundExplosion();
            //usleep(10000);
            playSoundStop();
            tanks[tankIndex]=tankInit;//remove this tank
            tankNum = tankNum-1;
            numOfEnemy--;
            numOfEnemyTemp = (numOfEnemyTemp<<1)&0xfffff;
            VGA = 0xff000000 + numOfEnemyTemp;
            writeSprite((unsigned int)tankInit, BASIC_TANK, GREY,
tankIndex);
            explosion[tankIndex]=BIGEXPLO;
            exploFlag[tankIndex]=1;
            exploPos[tankIndex]=setPos(x,y,exploPos[tankIndex]);
            if (numOfEnemy<=0)
                gameWin();
        }
    }
}

unsigned int canDestroy(unsigned int X_SQR, unsigned int Y_SQR, unsigned
int bulletIndex)
{
    if (bulletIndex==(MAXBULLET-1))
    {
        if (player1Level==PLY1 || player1Level==PLY2)
        {
            if ((TYPES(scenario[Y_SQR][X_SQR]))==BRICK ||
(TYPES(scenario[Y_SQR][X_SQR]))==STEEL)
                return 1;
            else
                return 0;
        }
```

```
        else
        {
            if ((TYPES(scenario[Y_SQR][X_SQR]))==BRICK)
                return 1;
            else
                return 0;
        }
    }
    else
    {
        if (tankLevel[bulletIndex]==POWER_TANK ||
tankLevel[bulletIndex]==SUPER_TANK)
        {
            if ((TYPES(scenario[Y_SQR][X_SQR]))==BRICK ||
(TYPES(scenario[Y_SQR][X_SQR]))==STEEL)
                return 1;
            else
                return 0;
        }
        else
        {
            if ((TYPES(scenario[Y_SQR][X_SQR]))==BRICK)
                return 1;
            else
                return 0;
        }
    }
}

unsigned int canBeHit(unsigned int x, unsigned int y)
{
    unsigned int X_SQR = x/36;
    unsigned int Y_SQR = y/36;

    if (x<=(X_SQR*36+18) && y<=(Y_SQR*36+18))
        if (BLOCKS(scenario[Y_SQR][X_SQR])&0x1 &&
hitable(TYPES(scenario[Y_SQR][X_SQR])))
            return 1;
        else
            return 0;
    else if (y<=(Y_SQR*36+18))
        if (BLOCKS(scenario[Y_SQR][X_SQR])&0x2 &&
hitable(TYPES(scenario[Y_SQR][X_SQR])))
            return 1;
```

```
        else
            return 0;
    else if (x<=(X_SQR*36+18))
        if (BLOCKS(scenario[Y_SQR][X_SQR])&0x4 &&
hitable(TYPES(scenario[Y_SQR][X_SQR])))
            return 1;
        else
            return 0;
    else
        if (BLOCKS(scenario[Y_SQR][X_SQR])&0x8 &&
hitable(TYPES(scenario[Y_SQR][X_SQR])))
            return 1;
        else
            return 0;
}

unsigned int bulletHit(unsigned int bullet, unsigned int dir)
{
    unsigned int flag = 0;
    if (dir==UP)
    {
        if (canBeHit(X(bullet),(Y(bullet))))
            flag = 1;
        else if (canBeHit((X(bullet))+5,(Y(bullet))))
            flag = 1;
    }
    else if (dir==DN)
    {
        if (canBeHit(X(bullet),(Y(bullet))))
            flag = 1;
        else if (canBeHit((X(bullet))+5,(Y(bullet))))
            flag = 1;
    }
    else if (dir==RT)
    {
        if (canBeHit((X(bullet)),Y(bullet)))
            flag = 1;
        else if (canBeHit((X(bullet)),(Y(bullet))+5))
            flag = 1;
    }
    else if (dir==LF)
    {
        if (canBeHit((X(bullet)),Y(bullet)))
            flag = 1;
```

```
        else if (canBeHit((X(bullet)),(Y(bullet))+5))
            flag = 1;
    }
    return flag;
}

void updateScenario(unsigned int x, unsigned int y, unsigned int dir)
{
    unsigned int SQR_X=x/36;
    unsigned int SQR_Y=y/36;
    if (dir==UP)
    {
        if (x<=(SQR_X*36+9))
        {
            if (BLOCKS(scenario[SQR_Y][SQR_X])&0x4)//hit the blocks 2
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xbf;
            else//hit the blocks 0
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xef;
        }
        else if (x>=(SQR_X*36+27))
        {
            if (BLOCKS(scenario[SQR_Y][SQR_X])&0x8)//hit the blocks 3
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0x7f;
            else//hit the blocks 1
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xdf;
        }
        else
        {
            if (BLOCKS(scenario[SQR_Y][SQR_X])&0xc)//hit the blocks 2,3
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0x3f;
            else//hit the blocks 0,1
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xcf;
        }
    }
    else if (dir==DN)
    {
        if (x<=(SQR_X*36+9))
        {
            if (BLOCKS(scenario[SQR_Y][SQR_X])&0x1)//hit the blocks 0
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xef;
            else//hit the blocks 2
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xbf;
        }
        else if (x>=(SQR_X*36+27))
```

```
    {
        if (BLOCKS(scenario[SQR_Y][SQR_X])&0x2)//hit the blocks 1
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xdf;
        else//hit the blocks 3
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0x7f;
    }
    else
    {
        if (BLOCKS(scenario[SQR_Y][SQR_X])&0x3)//hit the blocks 0,1
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xcf;
        else//hit the blocks 2,3
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0x3f;
    }
}
else if (dir==LF)
{
    if (y<=(SQR_Y*36+9))
    {
        if (BLOCKS(scenario[SQR_Y][SQR_X])&0x2)//hit the blocks 1
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xdf;
        else//hit the blocks 0
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xef;
    }
    else if (y>=(SQR_Y*36+27))
    {
        if (BLOCKS(scenario[SQR_Y][SQR_X])&0x8)//hit the blocks 3
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0x7f;
        else//hit the blocks 2
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xbf;
    }
    else
    {
        if (BLOCKS(scenario[SQR_Y][SQR_X])&0xa)//hit the blocks 1,3
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0x5f;
        else//hit the blocks 0,1
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xaf;
    }
}
else if (dir==RT)
{
    if (y<=(SQR_Y*36+9))
    {
        if (BLOCKS(scenario[SQR_Y][SQR_X])&0x1)//hit the blocks 0
            scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xef;
```

```
            else//hit the blocks 1
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xdf;
        }
        else if (y>=(SQR_Y*36+27))
        {
            if (BLOCKS(scenario[SQR_Y][SQR_X])&0x4)//hit the blocks 2
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xbf;
            else//hit the blocks 3
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0x7f;
        }
        else
        {
            if (BLOCKS(scenario[SQR_Y][SQR_X])&0x5)//hit the blocks 0,2
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0xaf;
            else//hit the blocks 0,1
                scenario[SQR_Y][SQR_X] = scenario[SQR_Y][SQR_X]&0x5f;
        }
    }
    writeScenario(SQR_Y,SQR_X);
    writeScenario(SQR_Y,SQR_X);
}

void smallExplosion(unsigned int x, unsigned int y)
{
    int i;
    for (i=0;i<MAXEXPLO;i++)
    {
        if (!explosion[i])
        {
            explosion[i]=SMALLEXPLO;
            exploFlag[i]=1;
            exploPos[i]=setPos(x,y,exploPos[i]);
            break;
        }
    }
}

unsigned int dif(unsigned int x1, unsigned int x2)
{
    if (x1>x2)
        return x1-x2;
    else
        return x2-x1;
}
```

```
void bulletMeet(unsigned int bullet, unsigned int bulletIndex)
{
    int i;
    for (i=0;i<MAXBULLET;i++)
    {
        if (dif(X(bullet),X(bullets[i]))<6 &&
dif(Y(bullet),Y(bullets[i]))<6 && bulletIndex!=i)
        {
            bullets[bulletIndex] = 0;
            bullets[i] = 0;
            writeSprite((unsigned int)bullets[i], BULLET, GREY,
i+MAXTANK);
            break;
        }
    }
}

//update the bullets
void updateBullets()
{
    if (!(counterEnable&(1<<30)))
    {
        fireCounter++;
        counterEnable |= (1<<30);
        bulletCounterSet();
    }
    int i;
    for (i=0; i<MAXBULLET; i++)//loop for each bullet
    {
        if (bullets[i])//bullet exists
        {
            //update the bullet at the defined rate
            if (counterEnable&(1<<i))
                continue;
            counterEnable |= (1<<i);
            bulletCounterSet();

            bulletMeet(bullets[i],i);


//printf("bullets[%d]=(%d,%d,%d)\n",i,X(bullets[i]),Y(bullets[i]),DIR
(bullets[i]));
            if ((DIR(bullets[i]))==UP)//bullet is moving up
```

```
            {
                //update the position of the bullet
                if ((Y(bullets[i]))>bulletStep)//not hit boundary
                    bullets[i]=decY(bulletStep,bullets[i]);
                else//hit the boundary
                    bullets[i]=0;

                //check whether the bullet hits a tank or obstacle
                int tankIndex=hitTank(bullets[i]);
                if (tankIndex!=-3)//hit the tank

tankBomb(X(tanks[tankIndex]),Y(tanks[tankIndex]),tankIndex,i);
                else if (bulletHit(bullets[i],UP))//hit the obstacle
                {
                    smallExplosion(X(bullets[i]),(Y(bullets[i]))-9);
                    if
(canDestroy((X(bullets[i]))/36,(Y(bullets[i]))/36,i))

updateScenario((X(bullets[i])),(Y(bullets[i])),UP);
                    else if
(canDestroy(((X(bullets[i]))+5)/36,(Y(bullets[i]))/36,i))

updateScenario((X(bullets[i]))+5,(Y(bullets[i])),UP);
                    bullets[i]=0;
                }
            }
            else if ((DIR(bullets[i]))==DN)//bullet is moving down
            {
                //update the position of the bullet
                if ((Y(bullets[i]))+bulletStep<468)//not hit boundary
                    bullets[i]=incY(bulletStep,bullets[i]);
                else//hit the boundary
                    bullets[i]=0;

                //hit the base
                if ((X(bullets[i]))>=6*36 && (X(bullets[i]))<=7*36 &&
(Y(bullets[i]))>=12*36 && (Y(bullets[i]))<=13*36)
                {
                    gameover();
                    return;
                }

                //check whether the bullet hits a tank or obstacle
                int tankIndex=hitTank(bullets[i]);
```

```
            if (tankIndex!=-3)//hit the tank

tankBomb(X(tanks[tankIndex]),Y(tanks[tankIndex]),tankIndex,i);
            else if (bulletHit(bullets[i],DN))//hit the obstacle
            {
                smallExplosion(X(bullets[i]),(Y(bullets[i]))+9);
                if
(canDestroy((X(bullets[i]))/36,(Y(bullets[i]))/36,i))

updateScenario((X(bullets[i])),(Y(bullets[i])),DN);
                else if
(canDestroy(((X(bullets[i]))+5)/36,(Y(bullets[i]))/36,i))

updateScenario((X(bullets[i]))+5,(Y(bullets[i])),DN);
                bullets[i]=0;
            }
        }
        else if ((DIR(bullets[i]))==LF)//bullet is moving left
        {
            //update the position of the bullet
            if ((X(bullets[i]))>bulletStep)//not hit boundary
                bullets[i]=decX(bulletStep,bullets[i]);
            else//hit the boundary
                bullets[i]=0;

            //hit the base
            if ((X(bullets[i]))>=6*36 && (X(bullets[i]))<=7*36 &&
(Y(bullets[i]))>=12*36 && (Y(bullets[i]))<=13*36)
            {
                gameover();
                return;
            }

            //check whether the bullet hits a tank or obstacle
            int tankIndex=hitTank(bullets[i]);
            if (tankIndex!=-3)//hit the tank

tankBomb(X(tanks[tankIndex]),Y(tanks[tankIndex]),tankIndex,i);
            else if (bulletHit(bullets[i],LF))//hit the obstacle
            {
                smallExplosion((X(bullets[i]))-9,Y(bullets[i]));
                if
(canDestroy((X(bullets[i]))/36,(Y(bullets[i]))/36,i))
```

```
updateScenario((X(bullets[i])),(Y(bullets[i])),LF);
                else if
(canDestroy((X(bullets[i]))/36,((Y(bullets[i]))+5)/36,i))

updateScenario((X(bullets[i])),(Y(bullets[i]))+5,LF);
                bullets[i]=0;
            }
        }
        else if ((DIR(bullets[i]))==RT)//bullet is moving left
        {
            //update the position of the bullet
            if ((X(bullets[i]))+bulletStep<468)//not hit boundary
              bullets[i]=incX(bulletStep,bullets[i]);
            else//hit the boundary
                bullets[i]=0;

            //hit the base
            if ((X(bullets[i]))>=6*36 && (X(bullets[i]))<=7*36 &&
(Y(bullets[i]))>=12*36 && (Y(bullets[i]))<=13*36)
            {
                gameover();
                return;
            }

            //check whether the bullet hits a tank or obstacle
            int tankIndex=hitTank(bullets[i]);
            if (tankIndex!=-3)//hit the tank

tankBomb(X(tanks[tankIndex]),Y(tanks[tankIndex]),tankIndex,i);
            else if (bulletHit(bullets[i],RT))//hit the obstacle
            {
                smallExplosion((X(bullets[i]))+9,Y(bullets[i]));
                if
(canDestroy((X(bullets[i]))/36,(Y(bullets[i]))/36,i))

updateScenario((X(bullets[i])),(Y(bullets[i])),RT);
                else if
(canDestroy((X(bullets[i]))/36,((Y(bullets[i]))+5)/36,i))

updateScenario((X(bullets[i])),(Y(bullets[i]))+5,RT);
                bullets[i]=0;
            }
        }
```

```
//printf("bullet[%d]=(%d,%d)\n",i,X(bullets[i]),Y(bullets[i]));
          writeSprite((unsigned int)bullets[i], BULLET, GREY,
i+MAXTANK);
       }
    }
}

#endif /*BULLET_H_*/
```

global.h

```
#ifndef GLOBAL_H_
#define GLOBAL_H_

#define VGA (*(volatile unsigned long int*) VGA_BASE)
#define AUDIO (*(volatile unsigned long int*) AUDIO_BASE)
#define KEYPRESSED IORD_8DIRECT(PS2_BASE,0)
#define KEY IORD_8DIRECT(PS2_BASE,4)
#define UPDATE_FREQ 400000
#define XOFFSET 86
#define YOFFSET 6

#define FIRE 0x1
#define EXPLOSION 0x2
#define INIT 0x00000100
#define START 0x00000110
#define VGA_DELAY 6000

#define step 4
#define bulletStep 2
#define UP 0
#define DN 1
#define LF 3
#define RT 2
#define STOP -1
#define DIR(x) (x)&0x3
#define X(x) ((x)&0xff800)>>11
#define Y(x) ((x)&0x007fc)>>2

#define BRICK 1
#define STEEL 2
#define TREE  3
#define WATER 4
#define ICE   5
```

```
#define BASE  6
#define BULLET 7
#define SMALLEXPLO 8
#define BIGEXPLO 9

#define PLAYER1 1
#define PLAYER2 2
#define BASIC_TANK 3
#define POWERUP 4
#define POWER_TANK 5
#define SUPER_TANK 6
#define PLY1 10
#define PLY2 11

#define GREY 1

#define BLOCKS(x) ((x)&0xf0)>>4
#define TYPES(x) (x)&0xf

#define MAXBULLET (MAXTANK+2)
#define MAXTANK 4
#define MAXENEMY 6
#define MAXEXPLO 8

#define tankInit 0xfffff
#define bulletInit 0xffffff

unsigned int player1 = 0;

unsigned int player1Level = PLAYER1;

int bullets[MAXBULLET];
int tanks[MAXTANK];
int tankLevel[MAXTANK];
int explosion[MAXEXPLO];
int exploFlag[MAXEXPLO];
int exploPos[MAXEXPLO];
int exploCounter[MAXEXPLO];

int level = 1;
int player1Life = 3;
int numOfEnemy = MAXENEMY;
int numOfEnemyTemp = 0;
int currentDir = STOP;
```

```
int fireFlag = 0;
int breakFlag = 0;

unsigned int numOfTank = 0;
unsigned int tankNum = 0;

unsigned int powerup = tankInit;

unsigned int counterEnable = 0;

unsigned int tankEnable = 0;

unsigned int musicCounter = 0;

unsigned int player1Stop = 0;

unsigned int tankStop = 0;

unsigned int setPos(unsigned int x, unsigned int y, unsigned int sprite);

unsigned int setDir(unsigned int dir, unsigned int sprite);

unsigned int generateTankCounter = 0;
unsigned int generatePowerupCounter = 0;
unsigned int fireCounter = 0;

void tankCounterSet()
{
    IOWR_32DIRECT(TANK_COUNTER_BASE,0,tankEnable);
}

void tankCounterCheck()
{
    tankEnable = IORD_32DIRECT(TANK_COUNTER_BASE,0);
}

void bulletCounterSet()
{
    IOWR_32DIRECT(BULLET_COUNTER_BASE,0,counterEnable);
}

void bulletCounterCheck()
{
    counterEnable = IORD_32DIRECT(BULLET_COUNTER_BASE,0);
```

```
}

void system_init()
{
    int i;
    for (i=0;i<MAXTANK;i++)
        tanks[i]=tankInit;
    for (i=0;i<MAXBULLET;i++)
        bullets[i]=0;
    for (i=0;i<MAXEXPLO;i++)
        explosion[i]=0;
    for (i=0;i<MAXEXPLO;i++)
        exploFlag[i]=0;
    for (i=0;i<MAXEXPLO;i++)
        exploCounter[i]=0;
    player1Life = 3;
    generateTankCounter = 0;
    generatePowerupCounter = 0;
    fireCounter = 0;
    numOfEnemy = MAXENEMY;
    numOfEnemyTemp = 0;
    numOfTank = 0;
    tankNum = 0;
    currentDir = STOP;
    breakFlag = 0;
    fireFlag = 0;
    player1Stop = 0;
    tankStop = 0;
    counterEnable = 0;
    tankEnable = 0;
    musicCounter = 0;
    player1Level = PLAYER1;
    powerup = tankInit;
    tankCounterSet();
    bulletCounterSet();
}

#endif /*GLOBAL_H_*/
```

vga_driver.vhd

```
--------------------------------------------------------------------------------
-- Display a tank on the VGA screen
```

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

entity vga_driver is

port (
	reset : in std_logic;
	clk	: in std_logic;				-- Should be 25.125 MHz
	clk50	: in std_logic;
	command : in unsigned(31 downto 0);

	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 unsigned(9 downto 0) -- Blue[9:0]
);

end vga_driver;

architecture rtl of vga_driver is

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

constant VTOTAL		: integer := 525;
constant VSYNC		: integer := 2;
constant VBACK_PORCH	: integer := 33;
constant VACTIVE		: integer := 480;
constant VFRONT_PORCH : integer := 10;

constant HOFFSET : integer := 16;
constant VOFFSET : integer := 6;
```

```
constant TOTLE_BLOCKS : integer := 169;
constant BLOCKS_WIDTH : integer := 13;
constant SQUAR_PIXELS : integer := 18;
constant TOTAL_PIXELS : integer := 468;
constant SPIRITE_PIXELS : integer := 1295;
constant BULLET_HOFFSET : integer := 6;
constant BULLET_VOFFSET : integer := 12;
constant BLOCK_PIXELS : integer := 36;

constant PHOENIX_X : integer := 6;
constant PHOENIX_Y : integer := 12;
-------------------------------------------diff
constant OFFSET : integer := 120;
constant DIFF_Y : integer := 290;
constant DIFF_LENGTH :integer := 40;
constant DIFF_DEPTH : integer := 10;
signal      diff_exist    :std_logic:= '0';
signal      diff_address:unsigned (8 DOWNTO 0);
signal      diff_rgb        :STD_LOGIC_VECTOR (0 DOWNTO 0);
-------------------------------------------diff
-------------------------------------------normal
constant NORMAL_Y : integer := 270;
constant NORMAL_LENGTH : integer := 30;
constant NORMAL_DEPTH : integer := 10;
signal      normal_exist    :std_logic:= '0';
signal      normal_address:unsigned (8 DOWNTO 0);
signal      normal_rgb        :STD_LOGIC_VECTOR (0 DOWNTO 0);
-------------------------------------------normal

-------------------------------------------choose
constant CHOOSE_X             : integer := 100;
signal      choose1_exist    : std_logic:= '0';
signal      choose2_exist    : std_logic:= '0';
signal      choose1_address: unsigned (6 DOWNTO 0);
signal      choose2_address: unsigned (6 DOWNTO 0);
signal      choose_address : unsigned (6 DOWNTO 0);
signal      choose_diff      : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal      choose_rgb             :STD_LOGIC_VECTOR (1 DOWNTO 0);
-------------------------------------------choose

-------------------------------------------start
constant START_Y : integer := 120;
constant START_LENGTH : integer := 280;
constant START_DEPTH : integer := 120;
```

```
signal      start_x       : STD_LOGIC_VECTOR (9 DOWNTO 0);
signal      start_exist    :std_logic:= '0';
signal      start_address:unsigned (15 DOWNTO 0);
signal      start_rgb       :STD_LOGIC_VECTOR (1 DOWNTO 0);
signal      start_end    :std_logic:= '0';
-----------------------------------------start

-----------------------------------------end
--constant END_Y : integer := 180;
--constant END_LENGTH : integer := 300;
--constant END_DEPTH : integer := 160;
--signal      end_x       : STD_LOGIC_VECTOR (9 DOWNTO 0);
--signal      end_exist    :std_logic:= '0';
--signal      end_address:unsigned (15 DOWNTO 0);
--signal      end_rgb       :STD_LOGIC_VECTOR (1 DOWNTO 0);
--signal      over_end    :std_logic:= '0';
-----------------------------------------end

-----------------------------------------stage
constant STAGE_X : integer := 210;
constant STAGE_Y : integer := 220;
constant STAGE_LENGTH : integer := 144;
constant STAGE_NUMX : integer := 354;
constant STAGE_NUMY : integer := 220;
signal stage_exist    :std_logic:= '0';
signal stagenum_exist    :std_logic:= '0';
-----------------------------------------stage

-----------------------------------------UI LEVEL 50*10
constant LEVEL_X : integer := 504;
constant LEVEL_Y : integer := 46;
constant LEVEL_LENGTH : integer := 48;
constant LEVEL_DEPTH    : integer := 10;
signal level_exist       :std_logic:= '0';
signal level_rgb            : STD_LOGIC_VECTOR (1 DOWNTO 0);
signal level_address       : unsigned (8 DOWNTO 0);
-----------------------------------------UI LEVEL

-----------------------------------------UI NUM 10*10
constant LEVEL_NUMX : integer := 544;
constant LEVEL_NUMY : integer := 66;

constant LIFE_NUMX1 : integer := 540;
constant LIFE_NUMY1 : integer := 276;
```

```
constant LIFE_NUMX2 : integer := 540;
constant LIFE_NUMY2 : integer := 296;

signal levelnum_exist    :std_logic:= '0';

signal lifenum1_exist     :std_logic:= '0';
signal lifenum2_exist     :std_logic:= '0';

signal levelnum_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);

signal lifenum1_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal lifenum2_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);

--signal num_rgb          : STD_LOGIC_VECTOR (1 DOWNTO 0);
signal num0_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal num1_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal num2_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal num3_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal num4_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal num5_rgb         : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal num_address          : unsigned (6 DOWNTO 0);
signal levelnum_address: unsigned (6 DOWNTO 0);

signal lifenum1_address : unsigned (6 DOWNTO 0);
signal lifenum2_address : unsigned (6 DOWNTO 0);
signal lifenum_address    : unsigned (6 DOWNTO 0);

-------------------------------------------UI NUM

-------------------------------------------UI LIFE
constant LIFE_X : integer := 504;
constant LIFE_Y : integer := 256;
constant LIFE_LENGTH : integer := 56;
constant LIFE_DEPTH : integer := 10;
signal life_exist    :std_logic:= '0';
signal life_rgb                : STD_LOGIC_VECTOR (1 DOWNTO 0);
signal life_address         : unsigned (9 DOWNTO 0);

constant PLAYER1_X : integer := 520;
constant PLAYER1_Y : integer := 276;
constant PLAYER2_Y : integer := 296;

signal player_exist      : STD_LOGIC_VECTOR(1 DOWNTO 0) :="00";
```

```
signal playernum_y            : unsigned (9 DOWNTO 0);
signal signal_address    : unsigned (6 DOWNTO 0);
signal signal1_address : unsigned (6 DOWNTO 0);
signal signal2_address : unsigned (6 DOWNTO 0);
---------------------------------------------UI LIFE

---------------------------------------------UI ENEMY
type enemy_type is array(19 downto 0) of integer;
constant ENEMY_NUMX : enemy_type := (520,540,560,580,520,540,560,580,520,540,560,580,520,540,560,580,520,540,560,580);
constant ENEMY_NUMY : enemy_type := (136,136,136,136,156,156,156,156,176,176,176,176,196,196,196,196,216,216,216,216);
constant ENEMY_NUMDEPTH : integer := 10;
signal enemy_numexist    :STD_LOGIC_VECTOR(19 DOWNTO 0):= (others=>'0');
signal enemy_shown    :STD_LOGIC_VECTOR(19 DOWNTO 0):= (others=>'0');
signal enemynum_x             : unsigned (9 DOWNTO 0);
signal enemynum_y             : unsigned (9 DOWNTO 0);
signal enemynum_address : unsigned (6 DOWNTO 0);
signal enemynum_rgb           : STD_LOGIC_VECTOR (1 DOWNTO 0);

constant ENEMY_LENGTH : integer := 48;
constant ENEMY_DEPTH : integer := 10;
constant ENEMY_X : integer := 504;
constant ENEMY_Y : integer := 116;
signal enemy_exist    :std_logic:= '0';
signal enemy_address        : unsigned (8 DOWNTO 0);
signal enemy_rgb           : STD_LOGIC_VECTOR (1 DOWNTO 0);


--signal enemy_sel                :STD_LOGIC_VECTOR( downto 0);
---------------------------------------------UI ENEMY

--Signals for the video controller
type register_type is array(15 downto 0) of unsigned(25 downto 0);
signal RAM : register_type;
type register_type2 is array(7 downto 0) of unsigned(25 downto 0);
signal expRAM : register_type2;

type block_type is array(31 downto 0) of unsigned(5 downto 0);
signal Hpixel : block_type;    -- Horizontal position (36-6)
signal Vpixel : block_type;    -- Vorizontal position (36-12)
signal Hpixel2 : unsigned(5 downto 0);
signal Vpixel2 : unsigned(5 downto 0);
```

```
signal Hpixelexp : block_type;    -- Horizontal position (36-6)
signal Vpixelexp : block_type;    -- Vorizontal position (36-12)
signal Hpixel2exp : unsigned(5 downto 0);
signal Vpixel2exp : unsigned(5 downto 0);

signal select_reg : STD_LOGIC_VECTOR(31 DOWNTO 0);
signal select_exp : STD_LOGIC_VECTOR(31 DOWNTO 0);

signal cover : STD_LOGIC_VECTOR(31 DOWNTO 0);
signal expcover : STD_LOGIC_VECTOR(31 DOWNTO 0);

signal Hcount : unsigned(9 downto 0);    -- Horizontal position (0-800)
signal Vcount : unsigned(9 downto 0);    -- Vertical position (0-524)
signal EndOfLine, EndOfField : std_logic;
signal vga_hblank, vga_hsync, vga_vblank, vga_vsync : std_logic;    -- Sync. signals

signal background      :std_logic;

signal HBlockCount :unsigned(5 downto 0);
signal VBlockCount :unsigned(5 downto 0);
signal EndOfFourBlockLine, EndOfFourBlockField : std_logic;

signal HPixelCount :unsigned(8 downto 0);
signal VPixelCount :unsigned(8 downto 0);
signal EndOfBlockLine, EndOfBlockField : std_logic;

signal HSubPixelCount :unsigned(4 downto 0);
signal VSubPixelCount :unsigned(4 downto 0);
signal EndOfSubBlockLine, EndOfSubBlockField : std_logic;

signal BlockExist    :std_logic;

signal Scenario_type : unsigned(3 downto 0);

signal vposition    : unsigned(7 downto 0);
signal position      : unsigned(7 downto 0);

signal tank : std_logic;    -- tank area
signal tank_address :STD_LOGIC_VECTOR (10 DOWNTO 0);
signal Scenario_counter         : unsigned(7 downto 0);
signal pixel_hcounter      : unsigned(5 downto 0);
signal pixel_vcounter      : unsigned(5 downto 0);
signal block_hcounter     : unsigned(4 downto 0);
```

```
signal block_vcounter     : unsigned(4 downto 0);
signal tank_counter       : unsigned(7 downto 0);
signal tank_rgb           : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal hcenter                  : unsigned(9 downto 0);    -- Horizontal position (0-800)
signal vcenter                  : unsigned(9 downto 0);    -- Vertical position (0-524)

signal Brick_address    :STD_LOGIC_VECTOR (8 DOWNTO 0);
signal Wall_address     :STD_LOGIC_VECTOR (8 DOWNTO 0);
signal Trees_address    :STD_LOGIC_VECTOR (8 DOWNTO 0);
signal Water_address    :STD_LOGIC_VECTOR (8 DOWNTO 0);
signal Ice_address      :STD_LOGIC_VECTOR (8 DOWNTO 0);

signal Scenario_rgb       : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal Brick_rgb          : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal Wall_rgb           : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal Trees_rgb          : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal Water_rgb          : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal Ice_rgb            : STD_LOGIC_VECTOR (23 DOWNTO 0);

signal player1_address     :STD_LOGIC_VECTOR (10 DOWNTO 0);
signal player2_address     :STD_LOGIC_VECTOR (10 DOWNTO 0);
signal tank1_address       :STD_LOGIC_VECTOR (10 DOWNTO 0);
signal fasttank_address    :STD_LOGIC_VECTOR (10 DOWNTO 0);
signal armortank_address :STD_LOGIC_VECTOR (10 DOWNTO 0);
signal powertank_address :STD_LOGIC_VECTOR (10 DOWNTO 0);
signal bullet_address      :STD_LOGIC_VECTOR (5 DOWNTO 0);
signal smallburst_address:STD_LOGIC_VECTOR (8 DOWNTO 0);
signal bigburst_address    :STD_LOGIC_VECTOR (10 DOWNTO 0);

signal spirites_rgb       : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal player1_rgb        : STD_LOGIC_VECTOR (23 DOWNTO 0);
--signal player2_rgb            : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal tank1_rgb          : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal fasttank_rgb       : STD_LOGIC_VECTOR (1 DOWNTO 0);
signal powertank_rgb      : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal armortank_rgb      : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal bullet_rgb         : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal smallburst_rgb     : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal bigburst_rgb       : STD_LOGIC_VECTOR (23 DOWNTO 0);
signal explsion_rgb       : STD_LOGIC_VECTOR (23 DOWNTO 0);

-----------------------------------------------------------------
signal stage_rgb          : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal stage_address      : unsigned (12 DOWNTO 0);
```

```
signal stagenum_rgb           : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal stagenum1_rgb          : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal stagenum2_rgb          : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal stagenum3_rgb          : STD_LOGIC_VECTOR (0 DOWNTO 0);
signal stagenum_address          : unsigned (10 DOWNTO 0);
-------------------------------------------------------------

signal scenario_drawen    : STD_LOGIC := '0';

signal PhoenixExist    :         std_logic;    -- Phoenix
signal Phoenix_address        :STD_LOGIC_VECTOR (10 DOWNTO 0);
signal phoenix_rgb            :STD_LOGIC_VECTOR (0 DOWNTO 0);

signal Scenario_address : STD_LOGIC_VECTOR (8 DOWNTO 0);
signal commandUpdate        : std_logic;
signal temp_command          : STD_LOGIC_VECTOR(31 downto 0);
signal Scenario_command : STD_LOGIC_VECTOR(7 downto 0);
signal scenario_data        : STD_LOGIC_VECTOR(7 downto 0);
signal scenario_wren        : STD_LOGIC ;
signal Command_address    : STD_LOGIC_VECTOR(7 downto 0);
signal delete_bullet        : STD_LOGIC_VECTOR(31 downto 0);
--signal pre_command            : STD_LOGIC_VECTOR(31 downto 0);

signal num_sel                    :STD_LOGIC_VECTOR(1 downto 0);
signal life_sel1              :STD_LOGIC_VECTOR(2 downto 0);
signal life_sel2              :STD_LOGIC_VECTOR(2 downto 0);


signal spirites_data        :STD_LOGIC_VECTOR(30 downto 0);
signal explosion_data      :STD_LOGIC_VECTOR(30 downto 0);

signal explosion_command:unsigned(25 downto 0);
signal spirites_command :unsigned(25 downto 0);
signal spirites_address :unsigned (10 DOWNTO 0);
signal spirites_dir          :unsigned(1 downto 0);
signal spirites_col          :unsigned(1 downto 0);
signal spirites_type        :unsigned(3 downto 0);
signal spirites_tempdata:STD_LOGIC_VECTOR(25 downto 0);
signal explosion_address :unsigned (10 DOWNTO 0);

signal pixel : unsigned(10 downto 0);    -- Horizontal position (36-6)

component level IS
      PORT
```

```
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
	);
end component level;

component life IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (9 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
	);
end component life;

component enemy IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
	);
end component enemy;

component enemynum IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
	);
end component enemynum;

component choose IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
	);
end component choose;

component num0 IS
```

```
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
	);
end component num0;

component num1 IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
	);
end component num1;

component num2 IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
	);
end component num2;

component num3 IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
	);
end component num3;

component num4 IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
	);
end component num4;
```

```
component num5 IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
	);
end component num5;

component smallburst IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component smallburst;

component bigburst IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (10 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component bigburst;

component bullet IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component bullet;

component player1 IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (10 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component player1;
```

```
--component player2 IS
--	PORT
--	(
--		address		: IN STD_LOGIC_VECTOR (10 DOWNTO 0);
--		clock		: IN STD_LOGIC ;
--		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
--	);
--end component player2;

component armorTank IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (10 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component armorTank;

component powerTank IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (10 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component powerTank;

component fastTank IS		--stands for power
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (10 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
	);
end component fastTank;

component tank1 IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (10 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
```

```
end component tank1;

component phoenix IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (10 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
	);
end component phoenix;

component brick IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component brick;

component water IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component water;

component wall IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
	);
end component wall;

component trees IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
```

```
	);
end component trees;

--component ice IS
--	PORT
--	(
--		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
--		clock		: IN STD_LOGIC ;
--		q		: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
--	);
--end component ice;

component Scenario IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		data		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		wren		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
	);
END component Scenario;

component register_26 is
port (clock : in std_logic;
	D	: in unsigned(25 downto 0);
	Q	: out unsigned(25 downto 0);
	enable : in std_logic;
	rst	: in std_logic);
END component register_26;

component start IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (15 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
	);
end component start;

component easy IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
```

```
        clock           : IN STD_LOGIC ;
        q           : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
    );
end component easy;

component crazy IS
    PORT
    (
        address         : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
        clock           : IN STD_LOGIC ;
        q           : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
    );
end component crazy;

--component over IS
--  PORT
--  (
--      address         : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
--      clock           : IN STD_LOGIC ;
--      q           : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
--  );
--end component over;

component stage IS
    PORT
    (
        address         : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
        clock           : IN STD_LOGIC ;
        q           : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
    );
end component stage;

component stage_num1 IS
    PORT
    (
        address         : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
        clock           : IN STD_LOGIC ;
        q           : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
    );
end component stage_num1;

component stage_num2 IS
    PORT
    (
```

```
        address         : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
        clock           : IN STD_LOGIC ;
        q           : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
    );
end component stage_num2;

component stage_num3 IS
    PORT
    (
        address         : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
        clock           : IN STD_LOGIC ;
        q           : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
    );
end component stage_num3;

begin

the_num0 : num0
    port map(
        clock => clk50,
        address => STD_LOGIC_VECTOR(num_address),
        q => num0_rgb);

the_num1 : num1
    port map(
        clock => clk50,
        address => STD_LOGIC_VECTOR(num_address),
        q => num1_rgb);

the_num2 : num2
    port map(
        clock => clk50,
        address => STD_LOGIC_VECTOR(num_address),
        q => num2_rgb);

the_num3 : num3
    port map(
        clock => clk50,
        address => STD_LOGIC_VECTOR(num_address),
        q => num3_rgb);

the_num4 : num4
    port map(
        clock => clk50,
```

```
            address => STD_LOGIC_VECTOR(num_address),
            q => num4_rgb);

the_num5 : num5
        port map(
            clock => clk50,
            address => STD_LOGIC_VECTOR(num_address),
            q => num5_rgb);

the_level : level
        port map(
            clock => clk50,
            address => STD_LOGIC_VECTOR(level_address),
            q => level_rgb);

the_life : life
        port map(
            clock => clk50,
            address => STD_LOGIC_VECTOR(life_address),
            q => life_rgb);

the_enemy : enemy
        port map(
            clock => clk50,
            address => STD_LOGIC_VECTOR(enemy_address),
            q => enemy_rgb);

the_enemynum : enemynum
        port map(
            clock => clk50,
            address => STD_LOGIC_VECTOR(signal_address),
            q => enemynum_rgb);

the_choose : choose
        port map(
            clock => clk50,
            address => STD_LOGIC_VECTOR(choose_address),
            q => choose_rgb);

the_stage_num1 : stage_num1
        port map(
            clock => clk50,
            address => STD_LOGIC_VECTOR(stagenum_address),
            q => stagenum1_rgb);
```

```
the_stage_num2 : stage_num2
	port map(
		clock => clk50,
		address => STD_LOGIC_VECTOR(stagenum_address),
		q => stagenum2_rgb);

the_stage_num3 : stage_num3
	port map(
		clock => clk50,
		address => STD_LOGIC_VECTOR(stagenum_address),
		q => stagenum3_rgb);

the_start : start
	port map(
		clock => clk50,
		address => STD_LOGIC_VECTOR(start_address),
		q => start_rgb);

the_easy : easy
	port map(
		clock => clk50,
		address => STD_LOGIC_VECTOR(normal_address),
		q => normal_rgb);

the_crazy : crazy
	port map(
		clock => clk50,
		address => STD_LOGIC_VECTOR(diff_address),
		q => diff_rgb);

--the_over : over
--	port map(
--		clock => clk50,
--		address => STD_LOGIC_VECTOR(end_address),
--		q => end_rgb);

the_stage : stage
	port map(
		clock => clk50,
		address => STD_LOGIC_VECTOR(stage_address),
		q => stage_rgb);

the_Scenario :Scenario
```

```
        port map(
        clock => clk50,
        data    => scenario_data,
        wren    => scenario_wren,
        address => command_address,
        q => Scenario_command );

the_bullet : bullet
        port map(
                clock => clk50,
                address => bullet_address,
                q => bullet_rgb);

the_smallburst : smallburst
        port map(
                clock => clk50,
                address => smallburst_address,
                q => smallburst_rgb);

the_bigburst : bigburst
        port map(
                clock => clk50,
                address => bigburst_address,
                q => bigburst_rgb);

the_tank1 : tank1
        port map(
                clock => clk50,
                address => tank1_address,
                q => tank1_rgb);

the_player1 : player1
        port map(
                clock => clk50,
                address => player1_address,
                q => player1_rgb);

--the_player2 : player2
--        port map(
--                clock => clk50,
--                address => player2_address,
--                q => player2_rgb);

the_powerTank : powerTank
```

```
        port map(
                clock => clk50,
                address => powertank_address,
                q => powertank_rgb);

the_fastTank : fastTank
        port map(
                clock => clk50,
                address => fasttank_address,
                q => fasttank_rgb);

the_armorTank : armorTank
        port map(
                clock => clk50,
                address => armortank_address,
                q => armortank_rgb);

the_phoenix : phoenix
        port map(
                clock => clk50,
                address => phoenix_address,
                q => phoenix_rgb);

the_brick : brick
        port map(
                clock => clk50,
                address => Brick_address,
                q => Brick_rgb);

the_wall : wall
        port map(
                clock => clk50,
                address => Wall_address,
                q => Wall_rgb);

the_water : water
        port map(
                clock => clk50,
                address => Water_address,
                q => Water_rgb);

--the_ice : ice
--      port map(
--              clock => clk50,
```

```
--          address => Ice_address,
--          q => Ice_rgb);

the_trees : trees
    port map(
        clock => clk50,
        address => trees_address,
        q => trees_rgb);

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

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

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

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

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

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

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

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

VBlankGen : process (clk)
begin
if rising_edge(clk) then
	if reset = '1' then
		vga_vblank <= '1';
```

```
        elsif EndOfLine = '1' then
            if Vcount = VSYNC + VBACK_PORCH - 1 then
                vga_vblank <= '0';
            elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
                vga_vblank <= '1';
            end if;
        end if;
end if;
end process VBlankGen;

tempCommand : process (clk50)
begin
if rising_edge(clk50) then
        if reset = '1' then
        temp_command <= (others => '0');
        else
        temp_command <= std_logic_vector(command(31 downto 0));
        end if;
end if;
end process tempCommand;

scenario_wren <= '1' when (temp_command(31) = '1') and (commandUpdate = '1') and
(temp_command(23 downto 4) /= X"FFFFF") and (temp_command(31 downto 20) /= X"FF0") else
                        '0';

with scenario_wren select Command_address <=
        std_logic_vector(unsigned("0000"&(temp_command(15 downto 12))) +
(unsigned("0000"&(temp_command(11 downto 8))) sll 3) +
(unsigned("0000"&(temp_command(11 downto 8))) sll 2) +
unsigned("0000"&(temp_command(11 downto 8)))) when '1',
        std_logic_vector(position + (vposition sll 3) + (vposition sll 2) + (vposition)) when others;

with temp_command(31) select scenario_data <=
            temp_command(7 downto 0) when '1',
            (others => '0') when others;


spirites_data <= temp_command(30 downto 0) when (temp_command(31) = '0')
                                                and (temp_command /= 0)
                                                and (temp_command(5 downto 2) /=
"1000")
                                                and (temp_command(5 downto 2) /=
"1001") else
                                                (others => '0');
```

```
explosion_data <= temp_command(30 downto 0) when (temp_command(31) = '0')
                                                    and ((temp_command(5 downto 2) =
"1000")
                                                    or (temp_command(5 downto 2) =
"1001")) else
                                                    (others => '0');

--start_x <= temp_command(9 downto 0) when (temp_command(31 downto 12) = X"0FFFF" and
temp_command(11 downto 10) = "00")
--                                                          else "0000000000";

cordUpdate : process (clk50)
begin
if rising_edge(clk50) then
      if reset = '1' then
            scenario_drawen <= '0';
      elsif   (temp_command = X"FFFFFFFF") then
            scenario_drawen <= '0';
      elsif   (temp_command(31 downto 12) = X"00FFF" and temp_command(11 downto 10) =
"00") then
            start_x <= temp_command(9 downto 0);
            start_end <='0';
      elsif   (temp_command = X"00FFFFFF") then
            start_end <= '1';
      elsif   (temp_command(31 downto 4) = X"0F0FFFF")    then
            choose_diff <= temp_command(0 downto 0);
--    elsif   (temp_command = X"0F0FFFFF") then
--          over_end <= '1';
      elsif   (temp_command(27 downto 0) = X"FFFFFFF") then
            num_sel <= temp_command(29 downto 28);
      elsif   (temp_command(23 downto 0) = X"FFFFF0") then
            life_sel1<= temp_command(26 downto 24);
            life_sel2<= temp_command(30 downto 28);
      elsif   (temp_command(31 downto 20) = X"FF0") then
            enemy_shown <= temp_command(19 downto 0);
      elsif   (temp_command = X"00000000") then
            scenario_drawen <= '1';
      end if;
end if;
end process cordUpdate;
commandUpdate <= '1' when vga_hsync='1' and vga_vsync='1' else '0';

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

```
 background <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= HOFFSET) and
                          ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (HOFFSET +
TOTAL_PIXELS)) and
                          ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= VOFFSET) and
                          ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (VACTIVE -
VOFFSET))) else '0';

 start_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= start_x) and
                          ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (start_x +
START_LENGTH)) and
                          ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= START_Y) and
                          ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (START_Y +
START_DEPTH)) and
                           (start_end = '0')) else '0';

 diff_exist <= '1'  when (((to_integer(Hcount)-HBACK_PORCH-HSYNC-OFFSET) >= start_x) and
                          ((to_integer(Hcount)-HBACK_PORCH-HSYNC-OFFSET) < (start_x +
DIFF_LENGTH)) and
                          ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= DIFF_Y) and
                          ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (DIFF_Y +
DIFF_DEPTH))and
                           (start_end = '0')) else '0';

 normal_exist <= '1'when (((to_integer(Hcount)-HBACK_PORCH-HSYNC-OFFSET) >= start_x) and
                          ((to_integer(Hcount)-HBACK_PORCH-HSYNC-OFFSET) < (start_x +
NORMAL_LENGTH)) and
                          ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= NORMAL_Y) and
                          ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (NORMAL_Y +
NORMAL_DEPTH))and
                          (start_end = '0')) else '0';

 choose1_exist <= '1'when (((to_integer(Hcount)-HBACK_PORCH-HSYNC-CHOOSE_X ) >= start_x)
and
                           ((to_integer(Hcount)-HBACK_PORCH-HSYNC-CHOOSE_X ) <
(start_x + 10)) and
                           ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= NORMAL_Y) and
                           ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (NORMAL_Y +
10))and
                           (start_end = '0') and choose_diff = "0") else '0';

 choose2_exist <= '1'when (((to_integer(Hcount)-HBACK_PORCH-HSYNC-CHOOSE_X ) >= start_x)
and
                           ((to_integer(Hcount)-HBACK_PORCH-HSYNC-CHOOSE_X ) <
(start_x + 10)) and
```

```
                                ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= DIFF_Y) and
                                ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (DIFF_Y + 10))and
                                (start_end = '0') and choose_diff = "1") else '0';
--index_exist    <= '1' when

-- end_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= end_x) and
--                              ((to_integer(Hcount)-HBACK_PORCH-HSYNC)      <      (end_x      +
END_LENGTH)) and
--                              ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= END_Y) and
--                              ((to_integer(Vcount)-VBACK_PORCH-VSYNC)      <      (END_Y      +
END_DEPTH)) and
--                               (over_end = '0')) else '0';

  stage_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= STAGE_X) and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (STAGE_X + STAGE_LENGTH)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= STAGE_Y) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (STAGE_Y + BLOCK_PIXELS)) and (start_end = '1' ))
else '0';

  stagenum_exist  <=  '1'  when  (((to_integer(Hcount)-HBACK_PORCH-HSYNC)  >=  STAGE_NUMX)
and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (STAGE_NUMX + BLOCK_PIXELS)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= STAGE_NUMY) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (STAGE_NUMY + BLOCK_PIXELS))) else '0';
  ----------------------------------------------------------------------------------new
  level_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= LEVEL_X) and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (LEVEL_X + LEVEL_LENGTH)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= LEVEL_Y) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (LEVEL_Y + LEVEL_DEPTH))) else '0';

  levelnum_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= LEVEL_NUMX) and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (LEVEL_NUMX + LEVEL_DEPTH)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= LEVEL_NUMY) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (LEVEL_NUMY + LEVEL_DEPTH))) else '0';

    life_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= LIFE_X) and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (LIFE_X + LIFE_LENGTH)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= LIFE_Y) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (LIFE_Y + LIFE_DEPTH))) else '0';

    lifenum1_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= LIFE_NUMX1) and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (LIFE_NUMX1 + LIFE_DEPTH)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= LIFE_NUMY1) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (LIFE_NUMY1 + LIFE_DEPTH))) else '0';
```

```
   lifenum2_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= LIFE_NUMX2) and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (LIFE_NUMX2 + LIFE_DEPTH)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= LIFE_NUMY2) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (LIFE_NUMY2 + LIFE_DEPTH))) else '0';

   enemy_exist <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= ENEMY_X) and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (ENEMY_X + ENEMY_LENGTH)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= ENEMY_Y) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (ENEMY_Y + ENEMY_DEPTH))) else '0';

   generate_enemy:
      for i in 0 to 19 generate
   enemy_numexist(i) <= '1' when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >=
ENEMY_NUMX(i)) and
  ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (ENEMY_NUMX(i) + ENEMY_NUMDEPTH)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= ENEMY_NUMY(i)) and
  ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (ENEMY_NUMY(i) + ENEMY_NUMDEPTH)) and
enemy_shown(i) = '1') else '0';
  end generate;


   player_exist <= "01" when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= PLAYER1_X) and
                              ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (PLAYER1_X +
ENEMY_DEPTH)) and
                              ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= PLAYER1_Y) and
                              ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (PLAYER1_Y +
ENEMY_DEPTH))) else
                         "10" when (((to_integer(Hcount)-HBACK_PORCH-HSYNC) >= PLAYER1_X)
and
                              ((to_integer(Hcount)-HBACK_PORCH-HSYNC) < (PLAYER1_X +
ENEMY_DEPTH)) and
                              ((to_integer(Vcount)-VBACK_PORCH-VSYNC) >= PLAYER2_Y) and
                              ((to_integer(Vcount)-VBACK_PORCH-VSYNC) < (PLAYER2_Y +
ENEMY_DEPTH))) else "00";


  ----------------------------------------------------------------------------------------------new
   PhoenixExist <='1' when (position = PHOENIX_X) and (vposition = PHOENIX_Y) else '0';
  ----------------------------------------------------------------------------------------------

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

```
            if reset = '1' then
              HPixelCount <= (others => '0');
            elsif vga_hblank = '0' and vga_vblank ='0' and background = '1' then
            if EndOfBlockLine = '1' then
                  HPixelCount <= (others => '0');
            else
                  HPixelCount <= HPixelCount + 1;
            end if;
            else
                  HPixelCount <= (others => '0');
          end if;
       end if;
end process HPixelCounter;
EndOfBlockLine <= '1' when HPixelCount = TOTAL_PIXELS - 1 else '0';

VPixelCounter : process (clk)
begin
       if rising_edge(clk) then
          if reset = '1' then
                VPixelCount <= (others => '0');
         elsif vga_hblank = '0' and vga_vblank ='0' and background = '1' then
            if EndOfBlockLine = '1' then
               if EndOfBlockField = '1' then
                  VPixelCount <= (others => '0');
            else
                  VPixelCount <= VPixelCount + 1;
            end if;
          end if;
        end if;
end if;
end process VPixelCounter;
EndOfBlockField <= '1' when VPixelCount = TOTAL_PIXELS - 1 else '0';

HSubPixelCounter : process (clk)
begin
       if rising_edge(clk) then
          if reset = '1' then
            HSubPixelCount <= (others => '0');
          elsif vga_hblank = '0' and vga_vblank ='0' and background = '1' then
              if EndOfSubBlockLine = '1' then
                  HSubPixelCount <= (others => '0');
              else
                  HSubPixelCount <= HSubPixelCount + 1;
              end if;
```

```
            else
               HSubPixelCount <= (others => '0');
            end if;
        end if;
end process HSubPixelCounter;
EndOfSubBlockLine <= '1' when HSubPixelCount = SQUAR_PIXELS - 1 else '0';

VSubPixelCounter : process (clk)
begin
        if rising_edge(clk) then
            if reset = '1' then
               VSubPixelCount <= (others => '0');
        elsif vga_hblank = '0' and vga_vblank ='0' and background = '1' then
            if EndOfBlockLine = '1' then
               if EndOfSubBlockField = '1' then
                     VSubPixelCount <= (others => '0');
               else
                     VSubPixelCount <= VSubPixelCount + 1;
               end if;
            end if;
        end if;
end if;
end process VSubPixelCounter;
EndOfSubBlockField <= '1' when VSubPixelCount = SQUAR_PIXELS - 1 else '0';

HBlockCounter : process (clk)
begin
        if rising_edge(clk) then
            if reset = '1' then
               HBlockCount <= (others => '0');
            elsif background = '1' then
               if EndOfFourBlockLine = '1' then
               HBlockCount <= (others => '0');
               else
               HBlockCount <= HBlockCount + 1;
               end if;
               else
               HBlockCount <= (others => '0');
        end if;
end if;
end process HBlockCounter;
EndOfFourBlockLine <= '1' when HBlockCount = BLOCK_PIXELS - 1 else '0';

VBlockCounter : process (clk)
```

```
begin
      if rising_edge(clk) then
         if reset = '1' then
            VBlockCount <= (others => '0');
      elsif background = '1' then
        if EndOfBlockLine = '1' then
            if EndOfFourBlockField = '1' then
                  VBlockCount <= (others => '0');
            else
                  VBlockCount <= VBlockCount + 1;
            end if;
      end if;
end if;
end if;
end process VBlockCounter;
EndOfFourBlockField <= '1' when VBlockCount = BLOCK_PIXELS - 1 else '0';

positionCounter : process (clk)
begin
      if rising_edge(clk) then
            if reset = '1' then
                  position <= (others => '0');
            elsif background = '1' then
              if EndOfBlockField = '1' then
                  position <= (others => '0');
              elsif EndOfFourBlockLine = '1' then
                  position <= position + 1 ;
              end if;
            else
            position <= (others => '0');
          end if;
      end if;
end process positionCounter;

VpositionCounter : process (clk)
begin
      if rising_edge(clk) then
            if reset = '1' then
                  vposition <= (others => '0');
            elsif background = '1' then
              if EndOfBlockField = '1' then
                  vposition <= (others => '0');
              elsif EndOfFourBlockField = '1' and EndOfBlockLine = '1' then
                  vposition <= vposition + 1 ;
```

```
            end if;
        end if;
    end if;
end process VpositionCounter;

start_address <= to_unsigned(((to_integer(Hcount) - HBACK_PORCH - HSYNC -
to_integer(unsigned(start_x))) + START_LENGTH*(to_integer(Vcount)
-VBACK_PORCH-VSYNC-START_Y)),16);

normal_address <= to_unsigned(((to_integer(Hcount) - HBACK_PORCH - HSYNC - OFFSET -
to_integer(unsigned(start_x))) + NORMAL_LENGTH*(to_integer(Vcount)
-VBACK_PORCH-VSYNC-NORMAL_Y)),9);

diff_address <= to_unsigned(((to_integer(Hcount) - HBACK_PORCH - HSYNC - OFFSET -
to_integer(unsigned(start_x))) + DIFF_LENGTH*(to_integer(Vcount)
-VBACK_PORCH-VSYNC-DIFF_Y)),9);

choose1_address<= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC - CHOOSE_X -
to_integer(unsigned(start_x)) ) +
ENEMY_DEPTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-NORMAL_Y))),7);

choose2_address<= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC - CHOOSE_X -
to_integer(unsigned(start_x)) ) +
ENEMY_DEPTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-DIFF_Y))),7);

with choose_diff(0 downto 0) select choose_address <= choose1_address when "0",
                                                      choose2_address when "1",
                                                      (others => '0') when others;

--end_address <= to_unsigned(((to_integer(Hcount) - HBACK_PORCH - HSYNC -
to_integer(unsigned(end_x))) + END_LENGTH*(to_integer(Vcount)
-VBACK_PORCH-VSYNC-START_Y)),16);

Phoenix_address <= std_logic_vector("10100001111" - (("00000"&HBlockCount) +
(("00000"&VBlockCount) sll 5) + (("00000"&VBlockCount) sll 2)));

stage_address <= to_unsigned((5183 - ((to_integer(Hcount) - HBACK_PORCH - HSYNC -
STAGE_X ) + STAGE_LENGTH*(to_integer(Vcount) -VBACK_PORCH-VSYNC-STAGE_Y))),13);

stagenum_address<= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC -
STAGE_NUMX ) +
BLOCK_PIXELS*(to_integer(Vcount)-VBACK_PORCH-VSYNC-STAGE_NUMY))),11);

level_address<= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC - LEVEL_X ) +
```

```
LEVEL_LENGTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-LEVEL_Y))),9);

enemy_address<= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC - ENEMY_X ) +
ENEMY_LENGTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-ENEMY_Y))),9);

with enemy_numexist(19 downto 0) select enemynum_x <= to_unsigned(ENEMY_NUMX(0),10)
when X"00001",

to_unsigned(ENEMY_NUMX(1),10) when X"00002",

to_unsigned(ENEMY_NUMX(2),10) when X"00004",

to_unsigned(ENEMY_NUMX(3),10) when X"00008",

to_unsigned(ENEMY_NUMX(4),10) when X"00010",

to_unsigned(ENEMY_NUMX(5),10) when X"00020",

to_unsigned(ENEMY_NUMX(6),10) when X"00040",

to_unsigned(ENEMY_NUMX(7),10) when X"00080",

to_unsigned(ENEMY_NUMX(8),10) when X"00100",

to_unsigned(ENEMY_NUMX(9),10) when X"00200",

to_unsigned(ENEMY_NUMX(10),10) when X"00400",

to_unsigned(ENEMY_NUMX(11),10) when X"00800",

to_unsigned(ENEMY_NUMX(12),10)    when X"01000",

to_unsigned(ENEMY_NUMX(13),10) when X"02000",

to_unsigned(ENEMY_NUMX(14),10) when X"04000",

to_unsigned(ENEMY_NUMX(15),10) when X"08000",

to_unsigned(ENEMY_NUMX(16),10) when X"10000",

to_unsigned(ENEMY_NUMX(17),10) when X"20000",

to_unsigned(ENEMY_NUMX(18),10) when X"40000",
```

```
to_unsigned(ENEMY_NUMX(19),10) when X"80000",
                                                    (others => '0') when others;

with enemy_numexist(19 downto 0) select enemynum_y <= to_unsigned(ENEMY_NUMY(0),10)
when X"00001",

to_unsigned(ENEMY_NUMY(1),10) when X"00002",

to_unsigned(ENEMY_NUMY(2),10) when X"00004",

to_unsigned(ENEMY_NUMY(3),10) when X"00008",

to_unsigned(ENEMY_NUMY(4),10) when X"00010",

to_unsigned(ENEMY_NUMY(5),10) when X"00020",

to_unsigned(ENEMY_NUMY(6),10) when X"00040",

to_unsigned(ENEMY_NUMY(7),10) when X"00080",

to_unsigned(ENEMY_NUMY(8),10) when X"00100",

to_unsigned(ENEMY_NUMY(9),10) when X"00200",

to_unsigned(ENEMY_NUMY(10),10) when X"00400",

to_unsigned(ENEMY_NUMY(11),10) when X"00800",

to_unsigned(ENEMY_NUMY(12),10) when X"01000",

to_unsigned(ENEMY_NUMY(13),10) when X"02000",

to_unsigned(ENEMY_NUMY(14),10) when X"04000",

to_unsigned(ENEMY_NUMY(15),10) when X"08000",

to_unsigned(ENEMY_NUMY(16),10) when X"10000",

to_unsigned(ENEMY_NUMY(17),10) when X"20000",

to_unsigned(ENEMY_NUMY(18),10) when X"40000",

to_unsigned(ENEMY_NUMY(19),10) when X"80000",
                                                    (others => '0') when others;
```

```
with player_exist select signal_address <= signal1_address when "01",
                                                    signal2_address when "10",
                                                    enemynum_address when others;

enemynum_address<= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC -
to_integer(enemynum_x)) +
ENEMY_DEPTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-to_integer(enemynum_y)))),7);

signal1_address <= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC - PLAYER1_X) +
ENEMY_DEPTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-PLAYER1_Y))),7);

signal2_address <= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC - PLAYER1_X) +
ENEMY_DEPTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-PLAYER2_Y))),7);

levelnum_address <= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC -
LEVEL_NUMX ) + LEVEL_DEPTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-LEVEL_NUMY))),7);

life_address<= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC - LIFE_X ) +
LIFE_LENGTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-LIFE_Y))),10);

lifenum1_address <= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC -
LIFE_NUMX1 ) + LIFE_DEPTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-LIFE_NUMY1))),7);

lifenum2_address <= to_unsigned((((to_integer(Hcount) - HBACK_PORCH - HSYNC -
LIFE_NUMX2 ) + LIFE_DEPTH*(to_integer(Vcount)-VBACK_PORCH-VSYNC-LIFE_NUMY2))),7);

with num_sel(1 downto 0) select stagenum_rgb <= stagenum1_rgb when "01",
                                                    stagenum2_rgb when "10",
                                                    stagenum3_rgb when "11",
                                                    (others => '0') when others;

with num_sel(1 downto 0) select levelnum_rgb <= num1_rgb when "01",
                                                    num2_rgb when "10",
                                                    num3_rgb when "11",
                                                    (others => '0') when others;

with life_sel1(2 downto 0) select lifenum1_rgb <= num0_rgb when "000",
                                                    num1_rgb when "001",
                                                    num2_rgb when "010",
                                                    num3_rgb when "011",
                                                    num4_rgb when "100",
                                                    num5_rgb when "101",
```

```
                                                    (others => '0') when others;

with life_sel2(2 downto 0) select lifenum2_rgb <= num0_rgb when "000",
                                                    num1_rgb when "001",
                                                    num2_rgb when "010",
                                                    num3_rgb when "011",
                                                    num4_rgb when "100",
                                                    num5_rgb when "101",
                                                   (others => '0') when others;


with levelnum_exist select num_address <=    levelnum_address when '1',
                                                lifenum_address when others;

with lifenum1_exist select lifenum_address <= lifenum1_address when '1',
                                                lifenum2_address when others;

-----------------------------------------------------------------------------------------scenario
with Scenario_command(3 downto 0) select Scenario_rgb <=
      Brick_rgb when      "0001",
      Wall_rgb   when     "0010",
      Trees_rgb when      "0011",
      Water_rgb when      "0100",
      Ice_rgb     when    "0101",
      (others => '0') when others;

      BlockExist <= '1' when
                        (((HBlockCount >= "000000") and (HBlockCount <"010010") and
(VBlockCount >= "000000")   and (VBlockCount<"010010") and (Scenario_command(4) = '1'))
                    or ((HBlockCount >= "010010") and (HBlockCount <"100100") and
(VBlockCount >= "000000")   and (VBlockCount<"010010") and (Scenario_command(5) = '1'))
                    or ((HBlockCount >= "000000") and (HBlockCount <"010010") and
(VBlockCount >= "010010")   and (VBlockCount<"100100") and (Scenario_command(6) = '1'))
                    or ((HBlockCount >= "010010") and (HBlockCount <"100100") and
(VBlockCount >= "010010")   and (VBlockCount<"100100") and (Scenario_command(7) = '1')))
                    else '0';

      Scenario_address          <=          std_logic_vector((("0000"&HSubPixelCount)          +
((("0000"&VSubPixelCount) sll 4) + (("0000"&VSubPixelCount) sll 1)));
      Brick_address <= Scenario_address;
      Wall_address   <= Scenario_address;
      Trees_address <= Scenario_address;
      Water_address <= Scenario_address;
      Ice_address     <= Scenario_address;
```

```
-----------------------------------------------------------------explosion
explosion_select: select_exp <=
	X"ffffffff" when scenario_drawen = '0' else
	X"00000000" when explosion_data = 0 else
	X"00000001" when explosion_data(30 downto 26) = "00000" else
	X"00000002" when explosion_data(30 downto 26) = "00001" else
	X"00000004" when explosion_data(30 downto 26) = "00010" else
	X"00000008" when explosion_data(30 downto 26) = "00011" else
	X"00000010" when explosion_data(30 downto 26) = "00100" else
	X"00000020" when explosion_data(30 downto 26) = "00101" else
	X"00000040" when explosion_data(30 downto 26) = "00110" else
	X"00000080" when explosion_data(30 downto 26) = "00111" else
--	X"00000100" when explosion_data(30 downto 26) = "01000" else
--	X"00000200" when explosion_data(30 downto 26) = "01001" else
--	X"00000400" when explosion_data(30 downto 26) = "01010" else
--	X"00000800" when explosion_data(30 downto 26) = "01011" else
--	X"00001000" when explosion_data(30 downto 26) = "01100" else
--	X"00002000" when explosion_data(30 downto 26) = "01101" else
--	X"00004000" when explosion_data(30 downto 26) = "01110" else
--	X"00008000" when explosion_data(30 downto 26) = "01111" else
--	X"00010000" when explosion_data(30 downto 26) = "10000" else
--	X"00020000" when explosion_data(30 downto 26) = "10001" else
--	X"00040000" when explosion_data(30 downto 26) = "10010" else
--	X"00080000" when explosion_data(30 downto 26) = "10011" else
--	X"00100000" when explosion_data(30 downto 26) = "10100" else
--	X"00200000" when explosion_data(30 downto 26) = "10101" else
--	X"00400000" when explosion_data(30 downto 26) = "10110" else
--	X"00800000" when explosion_data(30 downto 26) = "10111" else
--	X"01000000" when explosion_data(30 downto 26) = "11000" else
--	X"02000000" when explosion_data(30 downto 26) = "11001" else
--	X"04000000" when explosion_data(30 downto 26) = "11010" else
--	X"08000000" when explosion_data(30 downto 26) = "11011" else
--	X"10000000" when explosion_data(30 downto 26) = "11100" else
--	X"20000000" when explosion_data(30 downto 26) = "11101" else
--	X"40000000" when explosion_data(30 downto 26) = "11110" else
--	X"80000000" when explosion_data(30 downto 26) = "11111" else
	X"00000000";


generate_labe3:
	for i in 0 to 7 generate
		stage3 : register_26 port map (clk50,unsigned (explosion_data(25 downto 0)),expRAM(i),select_exp(i),scenario_drawen);
```

```
        end generate;

generate_label4:
        for i in 0 to 7 generate

        Hexpselect: Hpixelexp(i) <=
        "010010" when (expRAM(i)(5 downto 2) = "1000") else
        "100100" ;

        Vexpselect: Vpixelexp(i) <=
        "010010" when (expRAM(i)(5 downto 2) = "1000") else
        "100100" ;

        stage4: expcover(i) <= '1' when ((expRAM(i) > 0) and
((to_integer(Hcount)-HBACK_PORCH-HSYNC - HOFFSET ) >= to_integer((expRAM(i)(25 downto
17)))) and
                                        ((to_integer(Hcount)-HBACK_PORCH-HSYNC -
HOFFSET ) < ( to_integer(Hpixelexp(i)) + (to_integer(expRAM(i)(25 downto 17))))) and
                                        ((to_integer(Vcount)-VBACK_PORCH-VSYNC -
VOFFSET ) >= to_integer((expRAM(i)(16 downto 8)))) and
                                        ((to_integer(Vcount)-VBACK_PORCH-VSYNC -
VOFFSET ) < ( to_integer(Vpixelexp(i)) + (to_integer(expRAM(i)(16 downto 8)))))) else '0';
        end generate;

        explosion_coder: explosion_command <=
          expRAM(0)   when  expcover(31 downto 0) =  X"00000001" else
          expRAM(1)   when  expcover(31 downto 0) =  X"00000002" else
          expRAM(2)   when  expcover(31 downto 0) =  X"00000004" else
          expRAM(3)   when  expcover(31 downto 0) =  X"00000008" else
          expRAM(4)   when  expcover(31 downto 0) =  X"00000010" else
          expRAM(5)   when  expcover(31 downto 0) =  X"00000020" else
          expRAM(6)   when  expcover(31 downto 0) =  X"00000040" else
          expRAM(7)   when  expcover(31 downto 0) =  X"00000080" else
--        expRAM(8)   when  expcover(31 downto 0) =  X"00000100" else
--        expRAM(9)   when  expcover(31 downto 0) =  X"00000200" else
--        expRAM(10)  when  expcover(31 downto 0) =  X"00000400" else
--        expRAM(11)  when  expcover(31 downto 0) =  X"00000800" else
--        expRAM(12)  when  expcover(31 downto 0) =  X"00001000" else
--        expRAM(13)  when  expcover(31 downto 0) =  X"00002000" else
--        expRAM(14)  when  expcover(31 downto 0) =  X"00004000" else
--        expRAM(15)  when  expcover(31 downto 0) =  X"00008000" else
--        expRAM(16)  when  expcover(31 downto 0) =  X"00010000" else
--        expRAM(17)  when  expcover(31 downto 0) =  X"00020000" else
--        expRAM(18)  when  expcover(31 downto 0) =  X"00040000" else
```

```
--      expRAM(19)   when   expcover(31 downto 0) =   X"00080000" else
--      expRAM(20)   when   expcover(31 downto 0) =   X"00100000" else
--      expRAM(21)   when   expcover(31 downto 0) =   X"00200000" else
--      expRAM(22)   when   expcover(31 downto 0) =   X"00400000" else
--      expRAM(23)   when   expcover(31 downto 0) =   X"00800000" else
--      expRAM(24)   when   expcover(31 downto 0) =   X"01000000" else
--      expRAM(25)   when   expcover(31 downto 0) =   X"02000000" else
--      expRAM(26)   when   expcover(31 downto 0) =   X"04000000" else
--      expRAM(27)   when   expcover(31 downto 0) =   X"08000000" else
--      expRAM(28)   when   expcover(31 downto 0) =   X"10000000" else
--      expRAM(29)   when   expcover(31 downto 0) =   X"20000000" else
--      expRAM(30)   when   expcover(31 downto 0) =   X"40000000" else
--      expRAM(31)   when   expcover(31 downto 0) =   X"80000000" else
        (others=>'0');


      with explosion_command(5 downto 2) select explsion_rgb <=
      smallburst_rgb    when "1000",
      bigburst_rgb      when "1001",
      (others => '0') when others;

      Hexpselect: Hpixel2exp <=
      "010010" when (explosion_command(5 downto 2) = "1000") else
      "100100" ;

      explosion_address <=to_unsigned(((to_integer(Hcount) - HBACK_PORCH - HSYNC - HOFFSET
- to_integer(explosion_command(25 downto 17))) + to_integer(Hpixel2exp)*(to_integer(Vcount)-
VOFFSET -VBACK_PORCH-VSYNC-to_integer(explosion_command(16 downto 8)))),11);

      smallburst_address<= std_logic_vector(explosion_address(8 DOWNTO 0));
      bigburst_address    <= std_logic_vector(explosion_address);

------------------------------------------------------------explosion

-------------------------------------------------------------------------------spirite
spirites_select: select_reg <=
      X"ffffffff" when scenario_drawen = '0' else
      X"00000000" when spirites_data = 0 else
      X"00000001" when spirites_data(30 downto 26) = "00000" else
      X"00000002" when spirites_data(30 downto 26) = "00001" else
      X"00000004" when spirites_data(30 downto 26) = "00010" else
      X"00000008" when spirites_data(30 downto 26) = "00011" else
      X"00000010" when spirites_data(30 downto 26) = "00100" else
      X"00000020" when spirites_data(30 downto 26) = "00101" else
```

```
        X"00000040" when spirites_data(30 downto 26) = "00110" else
        X"00000080" when spirites_data(30 downto 26) = "00111" else
        X"00000100" when spirites_data(30 downto 26) = "01000" else
        X"00000200" when spirites_data(30 downto 26) = "01001" else
        X"00000400" when spirites_data(30 downto 26) = "01010" else
        X"00000800" when spirites_data(30 downto 26) = "01011" else
        X"00001000" when spirites_data(30 downto 26) = "01100" else
        X"00002000" when spirites_data(30 downto 26) = "01101" else
        X"00004000" when spirites_data(30 downto 26) = "01110" else
        X"00008000" when spirites_data(30 downto 26) = "01111" else
--      X"00010000" when spirites_data(30 downto 26) = "10000" else
--      X"00020000" when spirites_data(30 downto 26) = "10001" else
--      X"00040000" when spirites_data(30 downto 26) = "10010" else
--      X"00080000" when spirites_data(30 downto 26) = "10011" else
--      X"00100000" when spirites_data(30 downto 26) = "10100" else
--      X"00200000" when spirites_data(30 downto 26) = "10101" else
--      X"00400000" when spirites_data(30 downto 26) = "10110" else
--      X"00800000" when spirites_data(30 downto 26) = "10111" else
--      X"01000000" when spirites_data(30 downto 26) = "11000" else
--      X"02000000" when spirites_data(30 downto 26) = "11001" else
--      X"04000000" when spirites_data(30 downto 26) = "11010" else
--      X"08000000" when spirites_data(30 downto 26) = "11011" else
--      X"10000000" when spirites_data(30 downto 26) = "11100" else
--      X"20000000" when spirites_data(30 downto 26) = "11101" else
--      X"40000000" when spirites_data(30 downto 26) = "11110" else
--      X"80000000" when spirites_data(30 downto 26) = "11111" else
        X"00000000";

generate_label:
        for i in 0 to 15 generate
                stage1 : register_26 port map (clk50,unsigned (spirites_data(25 downto
0)),RAM(i),select_reg(i),scenario_drawen);
        end generate;


generate_label2:
        for i in 0 to 15 generate

        Hselect: Hpixel(i) <=
        "000110" when (RAM(i)(5 downto 2) = "0111") else
        "100100" ;

        Vselect: Vpixel(i) <=
        "000110" when (RAM(i)(5 downto 2) = "0111") else
```

```
        "100100" ;

        delete_bullet(i) <= '1' when (RAM(i)(5 downto 2) = "0111") and (RAM(i)(25 downto 17) =
"000000000") and ((RAM(i)(16 downto 8) = "000000000")) else
                                        '0';

        stage2: cover(i) <= '1' when ((RAM(i) > 0) and (delete_bullet(i) = '0') and
                                                ((to_integer(Hcount)-HBACK_PORCH-HSYNC                  -
HOFFSET ) >= to_integer((RAM(i)(25 downto 17)))) and
                                                ((to_integer(Hcount)-HBACK_PORCH-HSYNC - HOFFSET )
< ( to_integer(Hpixel(i)) + (to_integer(RAM(i)(25 downto 17))))) and
                                                ((to_integer(Vcount)-VBACK_PORCH-VSYNC                  -
VOFFSET ) >= to_integer((RAM(i)(16 downto 8)))) and
                                                ((to_integer(Vcount)-VBACK_PORCH-VSYNC - VOFFSET )
< ( to_integer(Vpixel(i)) + (to_integer(RAM(i)(16 downto 8))))) else '0';
        end generate;


spirites_coder: spirites_command <=
        RAM(0)    when    cover(31 downto 0) =    X"00000001" else
        RAM(1)    when    cover(31 downto 0) =    X"00000002" else
        RAM(2)    when    cover(31 downto 0) =    X"00000004" else
        RAM(3)    when    cover(31 downto 0) =    X"00000008" else
        RAM(4)    when    cover(31 downto 0) =    X"00000010" else
        RAM(5)    when    cover(31 downto 0) =    X"00000020" else
        RAM(6)    when    cover(31 downto 0) =    X"00000040" else
        RAM(7)    when    cover(31 downto 0) =    X"00000080" else
        RAM(8)    when    cover(31 downto 0) =    X"00000100" else
        RAM(9)    when    cover(31 downto 0) =    X"00000200" else
        RAM(10)   when    cover(31 downto 0) =    X"00000400" else
        RAM(11)   when    cover(31 downto 0) =    X"00000800" else
        RAM(12)   when    cover(31 downto 0) =    X"00001000" else
        RAM(13)   when    cover(31 downto 0) =    X"00002000" else
        RAM(14)   when    cover(31 downto 0) =    X"00004000" else
        RAM(15)   when    cover(31 downto 0) =    X"00008000" else
--      RAM(16)   when    cover(31 downto 0) =    X"00010000" else
--      RAM(17)   when    cover(31 downto 0) =    X"00020000" else
--      RAM(18)   when    cover(31 downto 0) =    X"00040000" else
--      RAM(19)   when    cover(31 downto 0) =    X"00080000" else
--      RAM(20)   when    cover(31 downto 0) =    X"00100000" else
--      RAM(21)   when    cover(31 downto 0) =    X"00200000" else
--      RAM(22)   when    cover(31 downto 0) =    X"00400000" else
--      RAM(23)   when    cover(31 downto 0) =    X"00800000" else
--      RAM(24)   when    cover(31 downto 0) =    X"01000000" else
```

```
--      RAM(25)   when   cover(31 downto 0) =   X"02000000" else
--      RAM(26)   when   cover(31 downto 0) =   X"04000000" else
--      RAM(27)   when   cover(31 downto 0) =   X"08000000" else
--      RAM(28)   when   cover(31 downto 0) =   X"10000000" else
--      RAM(29)   when   cover(31 downto 0) =   X"20000000" else
--      RAM(30)   when   cover(31 downto 0) =   X"40000000" else
--      RAM(31)   when   cover(31 downto 0) =   X"80000000" else
        (others=>'0');

spirites_dir        <= spirites_command(7 downto 6);
spirites_type       <= spirites_command(5 downto 2);
spirites_col        <= spirites_command(1 downto 0);

Area_select:    pixel <=
      "00000100011" when (spirites_type = "0111") else
      "10100001111" ;

Hselect: Hpixel2 <=
      "000110" when (spirites_type = "0111") else
      "100100" ;

Vselect: Vpixel2 <=
      "000110" when (spirites_type = "0111") else
      "100100" ;

with spirites_dir select spirites_address <=
      to_unsigned(((to_integer(Hcount) - HBACK_PORCH - HSYNC - HOFFSET -
to_integer(spirites_command(25 downto 17))) + to_integer(Hpixel2)*(to_integer(Vcount)-
VOFFSET -VBACK_PORCH-VSYNC-to_integer(spirites_command(16 downto 8)))),11) when   "01",
      to_unsigned((to_integer(pixel) - (to_integer(Hcount) - HBACK_PORCH - HSYNC - HOFFSET-
to_integer(spirites_command(25 downto 17))) -
to_integer(Hpixel2)*(to_integer(Vcount)-VBACK_PORCH-VSYNC -
VOFFSET-to_integer(spirites_command(16 downto 8)))),11)      when   "00",
      to_unsigned(((to_integer(Hcount) - HBACK_PORCH - HSYNC - HOFFSET -
to_integer(spirites_command(25 downto 17)))*to_integer(Vpixel2) +
(to_integer(Vcount)-VBACK_PORCH-VSYNC- VOFFSET - to_integer(spirites_command(16 downto
8)))),11) when   "10",
      to_unsigned((to_integer(pixel) - (to_integer(Hcount) - HOFFSET - HBACK_PORCH - HSYNC -
to_integer(spirites_command(25 downto 17)))*to_integer(Vpixel2) -
(to_integer(Vcount)-VBACK_PORCH -VOFFSET- VSYNC-to_integer(spirites_command(16 downto
8)))),11)   when   "11",
      (others => '0') when others;

with spirites_type select spirites_rgb <=
```

```
        player1_rgb      when "0001",
        player1_rgb      when "0010",
        tank1_rgb         when "0011",
--      fasttank_rgb    when "0100",
        powertank_rgb when "0101",
        armortank_rgb when "0110",
        bullet_rgb       when "0111",
        powertank_rgb when "1010",
        armortank_rgb when "1011",
        (others => '0') when others;

        player1_address      <= std_logic_vector(spirites_address);
        tank1_address         <= std_logic_vector(spirites_address);
        fasttank_address    <= std_logic_vector(spirites_address);
        powertank_address <= std_logic_vector(spirites_address);
        armortank_address <= std_logic_vector(spirites_address);
        bullet_address       <= std_logic_vector(spirites_address(5 downto 0));

-- Registered video signals going to the video DAC
VideoOut: process (clk, reset)
begin
        if reset = '1' then
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
        elsif clk'event and clk = '1' then
        ------------------------------------------------------------------------stage
                if scenario_drawen = '0' then
                        if start_end ='0' then
                                if start_exist = '1' then
                                        if start_rgb = "00" then
                                                VGA_R <= "0000000000";
                                                VGA_G <= "0000000000";
                                                VGA_B <= "0000000000";
                                        elsif start_rgb = "01" then
                                                VGA_R <= "0111101100";
                                                VGA_G <= "0111110100";
                                                VGA_B <= "0111101100";
                                        elsif start_rgb = "10" then
                                                VGA_R <= "1100011000";
                                                VGA_G <= "0111000100";
                                                VGA_B <= "0000000000";
                                        elsif start_rgb = "11" then
                                                VGA_R <= "1010010100";
```

```
            VGA_G <= "0011000000";
            VGA_B <= "0000000000";
        end if;
elsif normal_exist = '1' then
         if normal_rgb = "1" then
            VGA_R <= "0000000000";
            VGA_G <= "0000000000";
            VGA_B <= "0000000000";
        else
            VGA_R <= "1000010100";
            VGA_G <= "1000010100";
            VGA_B <= "1000010100";
        end if;
elsif diff_exist = '1' then
         if diff_rgb = "1" then
            VGA_R <= "0000000000";
            VGA_G <= "0000000000";
            VGA_B <= "0000000000";
        else
            VGA_R <= "1000010100";
            VGA_G <= "1000010100";
            VGA_B <= "1000010100";
        end if;

elsif choose1_exist = '1' then
if choose_rgb = "01" then
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
elsif choose_rgb = "10" then
        VGA_R <= "1111111111";
        VGA_G <= "1111111111";
        VGA_B <= "1111111111";
else
        VGA_R <= "1110010100";
        VGA_G <= "1101011000";
        VGA_B <= "0111011000";
end if;

elsif choose2_exist = '1' then
if choose_rgb = "01" then
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
```

```
                elsif choose_rgb = "10" then
                        VGA_R <= "1111111111";
                        VGA_G <= "1111111111";
                        VGA_B <= "1111111111";
                else
                        VGA_R <= "1110010100";
                        VGA_G <= "1101011000";
                        VGA_B <= "0111011000";
                end if;

                elsif vga_hblank = '0' and vga_vblank ='0' then
                                VGA_R <= "0000000000";
                                VGA_G <= "0000000000";
                                VGA_B <= "0000000000";
                else
                                VGA_R <= "0000000000";
                                VGA_G <= "0000000000";
                                VGA_B <= "0000000000";
                end if;

--              elsif over_end ='0' then
--                      if end_exist = '1' then
--                               if end_rgb = "00" then
--                                      VGA_R <= "0000000000";
--                                      VGA_G <= "0000000000";
--                                      VGA_B <= "0000000000";
--                              elsif end_rgb = "01" then
--                                      VGA_R <= "1111111111";
--                                      VGA_G <= "1111111111";
--                                      VGA_B <= "1111111111";
--                              elsif end_rgb = "10" then
--                                      VGA_R <= "1010110100";
--                                      VGA_G <= "1010111000";
--                                      VGA_B <= "1010110100";
--                              end if;
--                      elsif vga_hblank = '0' and vga_vblank ='0' then
--                                      VGA_R <= "0000000000";
--                                      VGA_G <= "0000000000";
--                                      VGA_B <= "0000000000";
--                      else
--                                      VGA_R <= "0000000000";
--                                      VGA_G <= "0000000000";
--                                      VGA_B <= "0000000000";
--                      end if;
```

```
                elsif (to_integer(Vcount)-VBACK_PORCH-VSYNC) >= VOFFSET and
(to_integer(Vcount)-VBACK_PORCH-VSYNC) <= VACTIVE - VOFFSET and
                    (to_integer(Hcount) - HBACK_PORCH - HSYNC)>= 0    and (to_integer(Hcount)
- HBACK_PORCH - HSYNC)<= HACTIVE then
                    if stage_exist = '1' then
                          if stage_rgb = 1 then
                              VGA_R <= "0000000000";
                              VGA_G <= "0000000000";
                              VGA_B <= "0000000000";
                        else
                              VGA_R <= "1000010100";
                              VGA_G <= "1000010100";
                              VGA_B <= "1000010100";
                        end if;
                    elsif stagenum_exist = '1' then
                        if stagenum_rgb = 1 then
                              VGA_R <= "0000000000";
                              VGA_G <= "0000000000";
                              VGA_B <= "0000000000";
                        else
                              VGA_R <= "1000010100";
                              VGA_G <= "1000010100";
                              VGA_B <= "1000010100";
                        end if;
                    else
                        VGA_R <= "1000010100";
                        VGA_G <= "1000010100";
                        VGA_B <= "1000010100";
                    end if;
                elsif vga_hblank = '0' and vga_vblank ='0' then
                    VGA_R <= "0000000000";
                    VGA_G <= "0000000000";
                    VGA_B <= "0000000000";
                else
                    VGA_R <= "0000000000";
                    VGA_G <= "0000000000";
                    VGA_B <= "0000000000";
                end if;
-------------------------------------------------------------------------scenario
                elsif background = '1' then
                if expcover > 0 and (explsion_rgb /= X"000000") then
                VGA_R <= unsigned((explsion_rgb(7 downto 0))&"00");
                VGA_G <= unsigned((explsion_rgb(15 downto 8))&"00");
```

```
                VGA_B <= unsigned((explsion_rgb(23 downto 16))&"00");
                elsif BlockExist = '1' and PhoenixExist = '0' and cover = 0 then

                 VGA_R <= unsigned((Scenario_rgb(7 downto 0))&"00");
                 VGA_G <= unsigned((Scenario_rgb(15 downto 8))&"00");
                 VGA_B <= unsigned((Scenario_rgb(23 downto 16))&"00");
                elsif PhoenixExist    = '1' and cover = 0 then
                    if phoenix_rgb = 1 then
                       VGA_R <= "0000000000";
                       VGA_G <= "0000000000";
                       VGA_B <= "0000000000";
                    else
                       VGA_R <= "1000010100";
                       VGA_G <= "1000010100";
                       VGA_B <= "1000010100";
                     end if;
----------------------------------------------------------------------------------------------ice and tree
                 elsif cover >0 and Scenario_command(3 downto 0) = "0011" and BlockExist = '1'
and Scenario_rgb(15 downto 8) /= 0 then
                       VGA_R <= unsigned((Scenario_rgb(7 downto 0))&"00");
                       VGA_G <= unsigned((Scenario_rgb(15 downto 8))&"00");
                       VGA_B <= unsigned((Scenario_rgb(23 downto 16))&"00");
                 elsif cover >0 and Scenario_command(3 downto 0) = "0101" and BlockExist = '1'
and spirites_rgb(15 downto 8) = 0 then
                       VGA_R <= unsigned((Scenario_rgb(7 downto 0))&"00");
                       VGA_G <= unsigned((Scenario_rgb(15 downto 8))&"00");
                       VGA_B <= unsigned((Scenario_rgb(23 downto 16))&"00");
----------------------------------------------------------------------------------------------spirite
                 elsif cover > 0 then
                      if spirites_type = "0010" then
                           if spirites_rgb(7 downto 0) >100 then
                           VGA_R <= unsigned((spirites_rgb(7 downto 0)-100)&"00");
                           else
                           VGA_R <= unsigned((spirites_rgb(7 downto 0))&"00");
                           end if;
                           VGA_G <= unsigned((spirites_rgb(15 downto 8))&"00");
                           VGA_B <= unsigned((spirites_rgb(23 downto 16))&"00");
                      elsif spirites_type = "0100" then
                           if fasttank_rgb = "00" then
                                VGA_R <= "0000000000";
                                VGA_G <= "0000000000";
                                VGA_B <= "0000000000";
                           elsif fasttank_rgb = "01" then
                                VGA_R <= "1111111111";
```

```
                VGA_G <= "1111111111";
                VGA_B <= "1111111111";
            elsif fasttank_rgb = "10" then
                VGA_R <= "1101000000";
                VGA_G <= "1101000100";
                VGA_B <= "1101000000";
            else
                VGA_R <= "0011000100";
                VGA_G <= "0101000100";
                VGA_B <= "1000010000";
            end if;
        elsif spirites_type = "1010" then----------------------------need modifications
            if ((spirites_rgb(7 downto 0) = X"24") and spirites_rgb(15 downto 8)
=X"44" and spirites_rgb(23 downto 16) = X"74") then
            VGA_R <= "0110110000";
            VGA_G <= "0110110000";
            VGA_B <= "0000000000";
            elsif ((spirites_rgb(7 downto 0) = X"9a") and spirites_rgb(15 downto 8)
=X"9a" and spirites_rgb(23 downto 16) = X"9a") then
            VGA_R <= "1110001100";
            VGA_G <= "0111110000";
            VGA_B <= "0000000000";
            elsif ((spirites_rgb(7 downto 0) = X"FF") and spirites_rgb(15 downto 8)
=X"FF" and spirites_rgb(23 downto 16) = X"FF") then
            VGA_R <= "1111100000";
            VGA_G <= "1110100000";
            VGA_B <= "1000100100";
            else
            VGA_R <= unsigned((spirites_rgb(7 downto 0))&"00");
            VGA_G <= unsigned((spirites_rgb(15 downto 8))&"00");
            VGA_B <= unsigned((spirites_rgb(23 downto 16))&"00");
            end if;
        elsif spirites_type = "1011" then----------------------------need modifications

            if ((spirites_rgb(7 downto 0) = X"24") and spirites_rgb(15 downto 8)
=X"44" and spirites_rgb(23 downto 16) = X"74") then
            VGA_R <= "0110110000";
            VGA_G <= "0110110000";
            VGA_B <= "0000000000";
            elsif ((spirites_rgb(7 downto 0) = X"9a") and spirites_rgb(15 downto 8)
=X"9a" and spirites_rgb(23 downto 16) = X"9a") then
            VGA_R <= "1110001100";
            VGA_G <= "0111110000";
            VGA_B <= "0000000000";
```

```
                        elsif ((spirites_rgb(7 downto 0) = X"FF") and spirites_rgb(15 downto 8)
=X"FF" and spirites_rgb(23 downto 16) = X"FF") then
                        VGA_R <= "1111100000";
                        VGA_G <= "1110100000";
                        VGA_B <= "1000100100";
                        elsif ((spirites_rgb(7 downto 0) = X"3B") and spirites_rgb(15 downto 8)
=X"3B" and spirites_rgb(23 downto 16) = X"3B") then
                        VGA_R <= "1010101000";
                        VGA_G <= "0101111100";
                        VGA_B <= "0000010000";
                        else
                        VGA_R <= unsigned((spirites_rgb(7 downto 0))&"00");
                        VGA_G <= unsigned((spirites_rgb(15 downto 8))&"00");
                        VGA_B <= unsigned((spirites_rgb(23 downto 16))&"00");
                        end if;
                    else
                    VGA_R <= unsigned((spirites_rgb(7 downto 0))&"00");
                    VGA_G <= unsigned((spirites_rgb(15 downto 8))&"00");
                    VGA_B <= unsigned((spirites_rgb(23 downto 16))&"00");
                    end if;
                else
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
               end if;
----------------------------------------------------------------------------------------------------UI
            elsif level_exist = '1' then
                if level_rgb = "01" then
                        VGA_R <= "1000010100";
                        VGA_G <= "1000010100";
                        VGA_B <= "1000010100";
                elsif level_rgb = "10" then
                        VGA_R <= "1010010100";
                        VGA_G <= "0011000000";
                        VGA_B <= "0000000000";
                else
                        VGA_R <= "0000000000";
                        VGA_G <= "0000000000";
                        VGA_B <= "0000000000";
                    end if;
            elsif levelnum_exist = '1' then
                    if levelnum_rgb = 0 then
                        VGA_R <= "0000000000";
                        VGA_G <= "0000000000";
```

```
                VGA_B <= "0000000000";
            elsif levelnum_rgb = 1 then
                VGA_R <= "1000010100";
                VGA_G <= "1000010100";
                VGA_B <= "1000010100";
            end if;

elsif enemy_exist = '1' then
    if enemy_rgb = "01" then
                VGA_R <= "1000010100";
                VGA_G <= "1000010100";
                VGA_B <= "1000010100";
            elsif enemy_rgb = "10" then
                VGA_R <= "1010010100";
                VGA_G <= "0011000000";
                VGA_B <= "0000000000";
            else
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
            end if;

  elsif enemy_numexist >0 then
    if enemynum_rgb = "01" then
                VGA_R <= "1000010100";
                VGA_G <= "1000010100";
                VGA_B <= "1000010100";
            elsif enemynum_rgb = "10" then
                VGA_R <= "1010010100";
                VGA_G <= "0011000000";
                VGA_B <= "0000000000";
            else
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
            end if;

    elsif life_exist = '1' then
    if life_rgb = "01" then
                VGA_R <= "1000010100";
                VGA_G <= "1000010100";
                VGA_B <= "1000010100";
            elsif life_rgb = "10" then
                VGA_R <= "1010010100";
```

```
                    VGA_G <= "0011000000";
                    VGA_B <= "0000000000";
                else
                    VGA_R <= "0000000000";
                    VGA_G <= "0000000000";
                    VGA_B <= "0000000000";
                end if;
            elsif    lifenum1_exist = '1' then
                if lifenum1_rgb = 0 then
                    VGA_R <= "0000000000";
                    VGA_G <= "0000000000";
                    VGA_B <= "0000000000";
                elsif lifenum1_rgb = 1 then
                    VGA_R <= "1000010100";
                    VGA_G <= "1000010100";
                    VGA_B <= "1000010100";
                end if;

--              elsif    lifenum2_exist = '1' then
--                  if lifenum2_rgb = 0 then
--                      VGA_R <= "0000000000";
--                      VGA_G <= "0000000000";
--                      VGA_B <= "0000000000";
--                  elsif lifenum2_rgb = 1 then
--                      VGA_R <= "1000010100";
--                      VGA_G <= "1000010100";
--                      VGA_B <= "1000010100";
--                  end if;

            elsif player_exist ="01" then
                if enemynum_rgb = "01" then
                    VGA_R <= "1000010100";
                    VGA_G <= "1000010100";
                    VGA_B <= "1000010100";
                elsif enemynum_rgb = "10" then
                    VGA_R <= "1111111111";
                    VGA_G <= "1111111111";
                    VGA_B <= "1111111111";
                else
                    VGA_R <= "1110010100";
                    VGA_G <= "1101011000";
                    VGA_B <= "0111011000";
                end if;
```

```
--                  elsif player_exist ="10" then
--                  if enemynum_rgb = "01" then
--                              VGA_R <= "1100011000";
--                              VGA_G <= "0111000100";
--                              VGA_B <= "0000000000";
--                      elsif enemynum_rgb = "10" then
--                              VGA_R <= "1010010100";
--                              VGA_G <= "0011000000";
--                              VGA_B <= "0000000000";
--                      else
--                              VGA_R <= "0000000000";
--                              VGA_G <= "0000000000";
--                              VGA_B <= "0000000000";
--                      end if;
-------------------------------------------------------------------------------------------------UI
            elsif vga_hblank = '0' and vga_vblank ='0' then
                VGA_R <= "1000010100";
                VGA_G <= "1000010100";
                VGA_B <= "1000010100";
            else
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
            end if;
end if;
end process VideoOut;

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

end rtl;
```

Battle_City.vhd

```
--
-- 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 Battle_City 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,                         -- Data
PS2_CLK : in 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 unsigned(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 Battle_City;

architecture datapath of Battle_City is

    signal clk25 : std_logic := '0';
    signal audio_clock_18 : std_logic;
    signal vga_command : std_logic_vector(31 downto 0);
    signal audio_command : std_logic_vector(31 downto 0);

    signal counter : unsigned(31 downto 0);
```

```
    signal temp : std_logic_vector(31 downto 0);

begin

    process (CLOCK_50)
    begin
        if rising_edge(CLOCK_50) then
            clk25 <= not clk25;
        end if;
    end process;

V1: entity work.vga_driver port map (
        reset => '0',
        clk => clk25,
        clk50 => CLOCK_50,
        command => unsigned(vga_command),
        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
        );

AUD_XCK <= audio_clock_18;

PLL : entity work.audio_pll port map (
                inclk0      => CLOCK_50,
                c0          => audio_clock_18
        );

AudioDriver : entity work.audio_driver port map (
        clock_50            => CLOCK_50,
        clock_18            => audio_clock_18,
        cpu_cmd             => audio_command,

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

```
);

nios : entity work.nios_system port map (
        clk                                     => CLOCK_50,
        reset_n                         => '1',

        -- the_de2_vga_bus_inst
        command_from_the_vga            => vga_command,
        command_from_the_audio          => audio_command,

        -- the_ps2
        PS2_Clk_to_the_ps2                      => PS2_CLK,
        PS2_Data_to_the_ps2             => PS2_DAT,

        SRAM_ADDR_from_the_sram          => SRAM_ADDR,
        SRAM_CE_N_from_the_sram          => SRAM_CE_N,
        SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
        SRAM_LB_N_from_the_sram          => SRAM_LB_N,
        SRAM_OE_N_from_the_sram          => SRAM_OE_N,
        SRAM_UB_N_from_the_sram          => SRAM_UB_N,
        SRAM_WE_N_from_the_sram          => SRAM_WE_N
        );

    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';
```

```
UART_TXD <= '0';
DRAM_ADDR <= (others => '0');
DRAM_LDQM <= '0';
DRAM_UDQM <= '0';
DRAM_WE_N <= '1';
DRAM_CAS_N <= '1';
DRAM_RAS_N <= '1';
DRAM_CS_N <= '1';
DRAM_BA_0 <= '0';
DRAM_BA_1 <= '0';
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';

-- 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');
    GPIO_0          <= (others => 'Z');
    GPIO_1          <= (others => 'Z');

end datapath;
```

audio_driver.vhd

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

entity audio_driver is
port(
        clock_50 : in std_logic;
        clock_18 : in std_logic;
        cpu_cmd : in std_logic_vector(31 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 audio_driver;

architecture behavior of audio_driver is

signal disable : std_logic; -- when '1' disable audio module
signal reset_n : std_logic; -- when '0' reset audio module
signal sin_sel : std_logic; -- when "0001", play "fire", when "0010", play "explosion"
signal play_finish1 : std_logic;-- exp
signal play_finish2 : std_logic;-- fire
signal reset_sm : std_logic; -- when '1' reset state machine
signal reset_exp : std_logic := '1';
```

```
signal reset_fire : std_logic := '1';
signal tune : unsigned(11 downto 0);
signal counter : unsigned(31 downto 0);
signal enable_ct : std_logic;

component de2_wm8731_audio is
port (
      clk : in std_logic;              --    Audio CODEC Chip Clock AUD_XCK (18.43 MHz)
      reset_n : in std_logic;
      clk_50 : in std_logic;
      disable : in std_logic;
      sin_sel : in std_logic; -- select which sound will be played
      play_finish1 : out std_logic;-- exp
      play_finish2 : out std_logic;-- fire
      reset_exp : in std_logic; -- when '1' reset
      reset_fire : in std_logic; --when '1' reset
      tune : in unsigned(11 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;

signal audio_request : std_logic;

type state is (s0, s1, s2, s3);
signal pr_state : state;

begin

reset_sm <= cpu_cmd(28);

process(clock_50, enable_ct)
begin
      if rising_edge(clock_50) then
            if enable_ct = '1' then
                  if counter = x"FFFFFFFF" then
                        counter <= x"00000000";
                  else
                        counter <= counter + 1;
```

```
                end if;
            else
                counter <= x"00000000";
            end if;
        end if;
end process;

process(clock_50,reset_sm, cpu_cmd,pr_state)
begin
        if rising_edge(clock_50) then
            if cpu_cmd = x"00000100" then --play welcome music
                pr_state<=s1;
            elsif cpu_cmd = x"00000f00" then -- play sound: fire, exp, move
                pr_state<=s2;
            elsif pr_state = s1 and cpu_cmd = x"00000110" then
                pr_state <= s3;
            else
                if pr_state = s1 then
                    pr_state<=s1;
                elsif pr_state = s2 then
                    pr_state<=s2;
                elsif pr_state = s3 then
                    pr_state<=s3;
                else
                    pr_state<=s0; -- waiting for the cpu cmd;
                end if;
            end if;
        end if;
end process;

process(clock_50,reset_sm, play_finish1, play_finish2, reset_fire, reset_exp, cpu_cmd,pr_state)
begin
        if rising_edge(clock_50) then
            if pr_state = s1 then
                enable_ct <= '0';
                reset_n <= '0';
                disable <= '1';
                sin_sel <= '1';

            elsif pr_state = s3 then
                enable_ct <= '1';
                reset_n <= '1';
                sin_sel <= '1';
                if counter = x"00000000" then
```

```
        disable <= '0';
        tune <= x"169";
elsif counter = x"007270e0" then
        tune <= x"141";
elsif counter = x"00e4e1c0" then
        tune <= x"12f";
elsif counter = x"015752a0" then
        tune <= x"169";
elsif counter = x"01c9c380" then
        tune <= x"141";
elsif counter = x"023c3460" then
        tune <= x"12f";
elsif counter = x"02aea540" then
        disable <= '1';
elsif counter = x"02d4cae0" then
        disable <= '0';
        tune <= x"12f";
elsif counter = x"03473bc0" then
        tune <= x"10e";
elsif counter = x"03b9aca0" then
        tune <= x"0ff";
elsif counter = x"042c1d80" then
        tune <= x"12f";
elsif counter = x"049e8e60" then
        tune <= x"10e";
elsif counter = x"0510ff40" then
        tune <= x"0ff";
elsif counter = x"05837020" then
        tune <= x"10e";
elsif counter = x"05f5e100" then
        tune <= x"0ff";
elsif counter = x"066851e0" then
        tune <= x"0f1";
elsif counter = x"06dac2c0" then
        tune <= x"10e";
elsif counter = x"074d33a0" then
        tune <= x"0ff";
elsif counter = x"07bfa480" then
        tune <= x"0f1";
elsif counter = x"08321560" then
        disable <= '1';
elsif counter = x"08583b00" then
        disable <= '0';
        tune <= x"0f1";
```

```
        elsif counter = x"08caabe0" then
            tune <= x"0D6";
        elsif counter = x"093d1cc0" then
            tune <= x"0bf";
        elsif counter = x"09af8da0" then
            tune <= x"0f1";
        elsif counter = x"0a21fe80" then
            tune <= x"0d6";
        elsif counter = x"0a946f60" then
            tune <=x"0bf";
        elsif counter = x"0b06e040" then
            disable <= '1';
        elsif counter = x"0b162280" then
            disable <= '0';
            tune <= x"0bf";
        elsif counter = x"0b889360" then
            disable <= '1';
        elsif counter = x"0c6d7520" then
            disable <= '0';
            tune <= x"0bf";
        elsif counter = x"0cb9c060" then
            disable <= '1';
        elsif counter = x"0cdfe600" then
            disable <= '0';
            tune <= x"0bf";
        elsif counter = x"0d2c3140" then
            disable <= '1';
        elsif counter = x"0d5256e0" then
            disable <= '0';
            tune <= x"0bf";
        elsif counter = x"0d9ea220" then
            disable <='1';
        elsif counter = x"0dc4c7c0" then
            disable <= '0';
            tune <= x"0bf";
        elsif counter = x"0e111300" then
            disable <= '1';
        end if;
    elsif pr_state = s2 then
        tune <= x"0C0";
        disable <='0';
        reset_n <='1';
        enable_ct <= '0';
        sin_sel <='0';
```

```
                if play_finish2 = '1' then
                    reset_fire<='1';
                elsif play_finish1 = '1' then
                    reset_exp<='1';
                else
                    if cpu_cmd = x"00000001" then-- fire
                        reset_fire<='0';
                    elsif cpu_cmd = x"00000002" then-- exp
                        reset_exp <= '0';
                    end if;
                end if;


            else
                disable <= '1';
                reset_n <= '0';
                tune<=X"0C0";
                reset_exp <='1';
                reset_fire <='1';
                enable_ct <='0';
                sin_sel <='0';
            end if;
        end if;
--          if pr_state = s1 then
--              tune <= unsigned(cpu_cmd(27 downto 16));
----            disable <= '0';
----            reset_n <= '1';
--          elsif pr_state = s2 then
--              tune <= x"0C0";
----            disable <= '0';
----            reset_n <= '1';
--              if cpu_cmd = x"00000001" then-- fire
--                  reset_fire<='0';
--                  reset_exp <='0';
----                if play_finish2 = '1' then
----                    reset_fire<='1';
----                else
----                    reset_fire<='0';
----                end if;
--              elsif cpu_cmd = x"00000002" then--exp
--                  if play_finish1 = '1' then
--                      reset_exp<='1';
--                  else
--                      reset_exp<='0';
```

```
--                          end if;
--                  else
--                          reset_fire <= '1';
--                          reset_exp <= '1';
--                  end if;
--          end if;
----   end if;
end process;

--port map to the wm8731 module
audio: de2_wm8731_audio port map(

        clk => clock_18,
        reset_n => reset_n,
        clk_50 =>clock_50,
        disable => disable,
        sin_sel => sin_sel,
        play_finish1 => play_finish1,--exp
        play_finish2 => play_finish2,--fire
        reset_exp => reset_exp,
        reset_fire => reset_fire,
        tune =>tune,

        -- 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;
```