CSEE 4840 Embedded System Project Report

Battle Tank- Inspired Multidirectional Shooter Video Game

Group Name: Tank

Lupeng Fan	**LF2447**
Yinshen Wang	**YW2561**
Di Yang	**DY2266**
Yichen Zhu	**YZ2582**

5/14/2014

Contents

List of Figures

List of Tables

1. Project Introduction

In this project, we intend to implement a battle tank game inspired by the classic tank game battle city. Basically, player needs to defend their homebase from being shot by the enemies. The rule to win the game is that player should defend its homebase successfully and wipe out all the enemy tanks. In this game, we have a map consisting of many different kinds of obstacles: obstacles such as water, steel, homebase cannot be moved through and destroyed; obstacles such as brick cannot be moved through but can be destroyed; for obstacles such as grass can be moved through but cannot be destroyed. If you wipe out all the enemy tanks before they damage the homebase, then you win this game. Additional signs are used to show the life left for both our tank and enemy tank on the right side of the screen. Flags are used to show the stage we are currently in.

There are some important points worthy to be noticed in our project. The first one is the part of VGA display. As discussed above, different images with different location, different direction and different display effects can appear on the screen simultaneously, we need to handle this dynamic display effectively. Thus we design a VGA controller to solve this problem. The second is about the map used in our project. Actually, three maps are needed in our project. One is for VGA display in hardware, one is for bullet in software and the last one is for tank in software. A difficult problem for us is that we need to keep these three maps updating synchronously. Another important point for the project is that we have one player tank controlled by player and 5 enemy tanks move randomly, we need to ensure that they cannot overlap each other and their own behavior will not conflicts with each other. The fourth one is about using game controller, in which keycodes for different keys on game controller should be identified and used correctly. The last one is on sound effect played by the FPGA. Different sound effects should be played in different situation.

2. A high-level hardware structure

The system architecture is below:

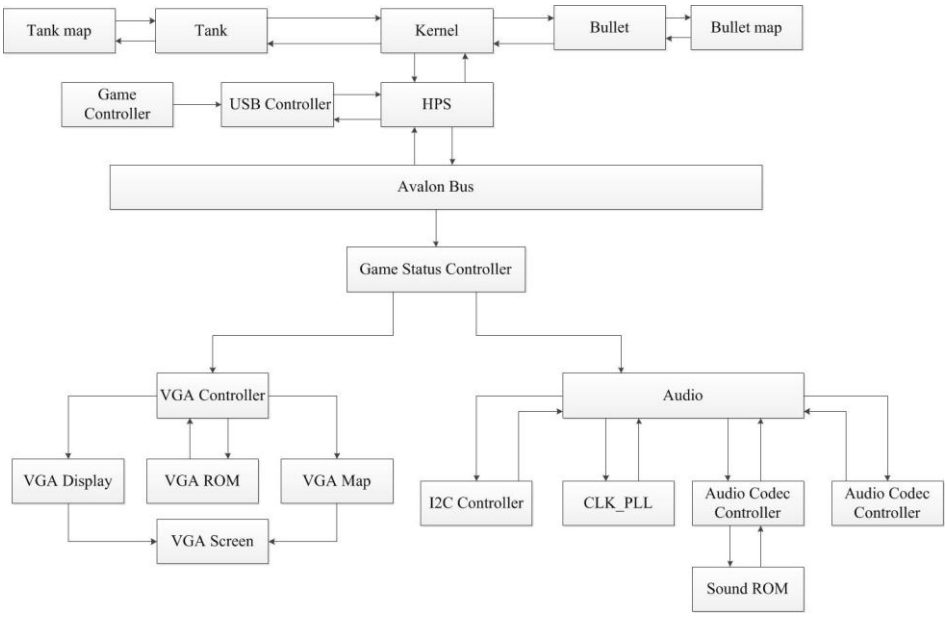


Figure 1: System architecture block diagram

As shown above, the software design part including Tank, Tank Map, Bullet and bullet map were compiled and stored in the server root folder. The lower part of the diagram includes the HPS and FPGA components. The kernel is responsible for managing the input/output requests from software and translates them into data processing instructions for the ARM processor in Hard Processor system (HPS). Once the software application request to send the messages, the kernel will manage to sent them through Ethernet and HPS store it in the SDRAM. Besides, there is a USB controller module can collect the data from USB device. When the USB devices interrupt caused by pressing a button, the kernel will also manage to collect the USB input data from hardware.

After HPS received messages from software and stored in the SDRAM, it will send them to the game controller module through Avalon Bus (Master to Slave data transfer). Similarly, there is a Avalon bus controller responsible for this.

Once the game controller received the messages, it will decode them and sent the corresponding signal to audio and VGA controllers. The detailed information about FPGA components processing will be explained in the following chapters.

2.1 Game Controller Module

We have created a game controller module to build up a FSM for game operation and Software message decoding. The Game Controller Module is connected to the HPS through Avalon bus in slave mode. As mentioned before, the HPS fetched the message from external server through Ethernet and store them in the SDRAM. Then, it sends the data to the game controller. There are totally 127 messages sending to the game controller.

The FSM is just a simple 3 stage machine which comprises welcome stage, gaming stage and game over stage. The FSM is controlled by the 127^{th} message from software.

In the second stage, the game controller will decode the remaining 126 messages which including information for game map display, tanks display, bullet display and audio play. Then, the game controller will send the relative vga and audio signals to VGA module and audio module.

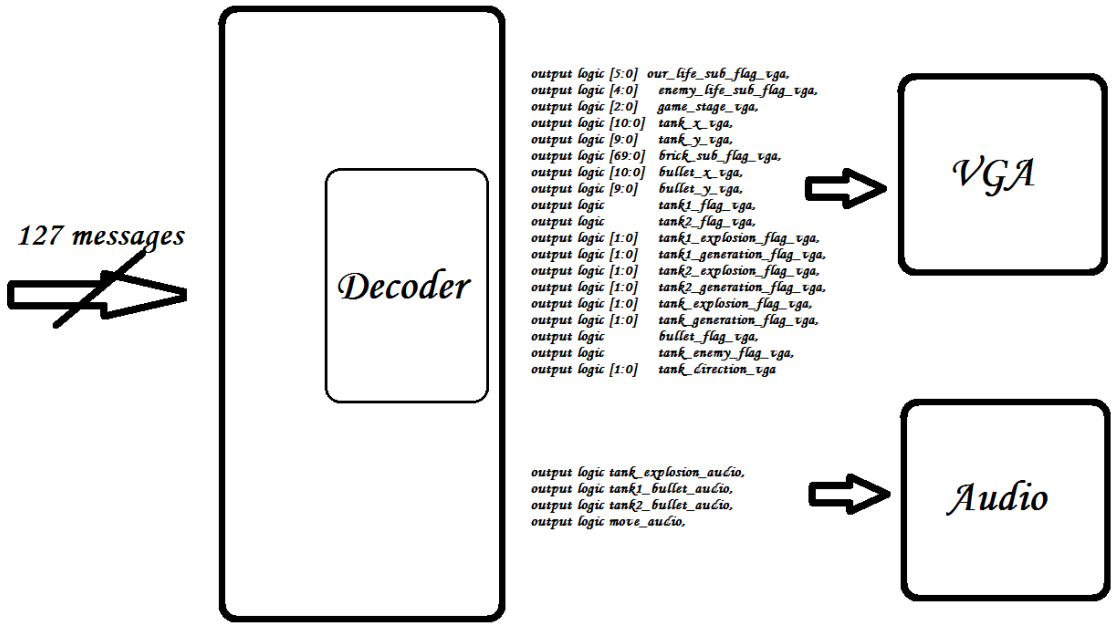


Figure 2: Decoding Message at 2nd Stage of Game Controller Module

2.1 VGA Block

2.1.1 Image preparation

All images used in our projects are extracted from the classic game – Battle City. In order to use them in our project, we need to process the ".png" file and convert them to ".mif" file so that they can be imported into the FPGA board.

For VGA screen, effective display area is divide into 300 square areas. Each row contains 20 squares and each column contains 15 squares. In these 300 squares, the area that tank can move arbitrarily contains 16*13=208 squares. Such division are shown as drawn below.

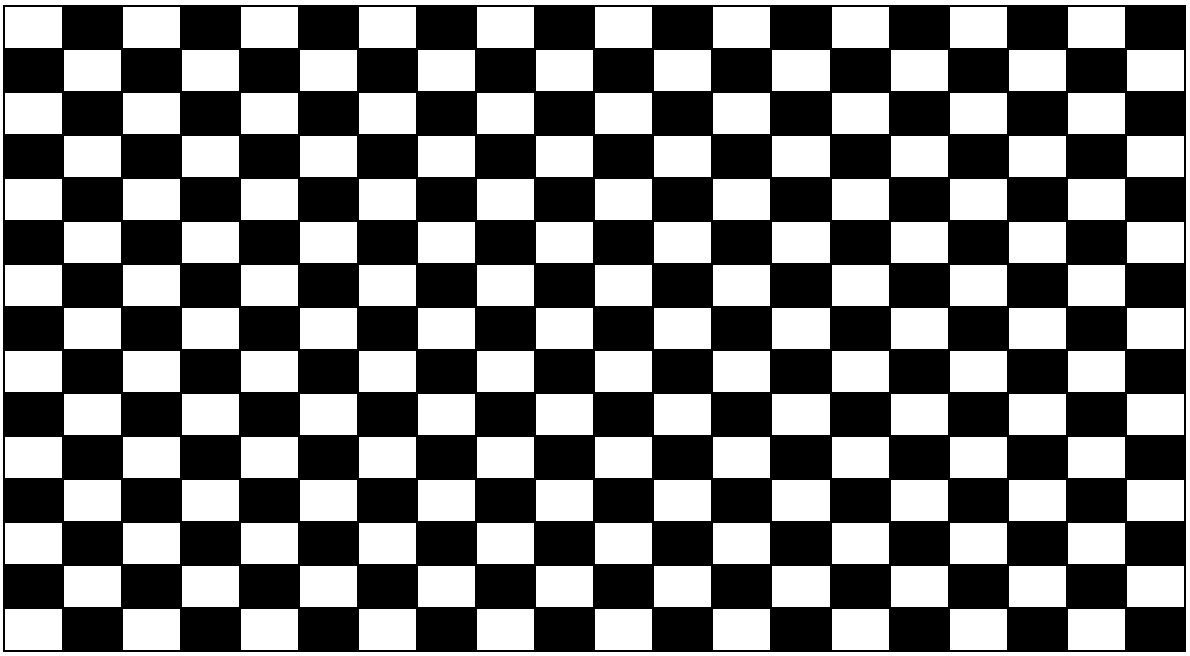


Figure 3: Division Map

2.1.1.1 Image classification and size

In this game, all images are classified in the following way: welcome screen, game over screen, our tank, enemy tank, bullet, sign for life of our tank and enemy life, flag for showing the stage, three kinds of obstacles: one of them can stop the tank but cannot be destroyed by the bullet, including water, steel and homebase; one of them can both stop the tank and be destroyed by the bullet, including brick; the third one cannot stop the tank and be destroyed by the bullet, including grass.

The welcome screen and game over screen are sized to 256*256 and place in the middle of the screen. Each element existing after the game starts, including tank, water, brick, grass, steel, homebase, is sized to 32*32. For the picture of tank, because each tank can move along one of the four directions, we implement this with loading pictures with different direction. For example, when the FPGA detects the direction of up, it loads the picture storing "up" information. So we have four pictures for each kind of tank. But brick is implemented in a different way because it can be destroyed by the bullet, so it is sized to 16*16. When the bullet meets one of the brick block, such block disappears. In this way, we can realize that part of the brick disappears.

For the rest of decorations, including life flag, our life and enemy life, are sized to 32*32. For the image of generation and explosion, each of them has 2 stages and is sized to be the same with tank, which is 32*32.

2.1.1. 2 Color implementation

In order to use these pictures with the board, Matlab is used to convert the original picture into "mif" file, in which each value is 24-bit, that can be used by the FPGA through making a ROM to store these data. VGA controller in this board uses 24 bits to represent RGB values, which is fetched from the "mif" file. The lower 0-7 bits stands for VGA_B, 8-15 bits stands for VGA_G and 16-23 bits stands for VGA_R. These three signals, along with VGA_CLK, VGA_SYNC_N and VGA_BLANK_N, are sent to analog device ADV 7123 to produce the analog data signals VGA_R, VGA_G and VGA_B. These three output from ADV 7123, along with VGA_VS and VGA_HS are sent to VGA monitor to display the picture we want. A VGA controller is also used to display the image whenever and wherever we want. Such controller will be discussed in the following part.

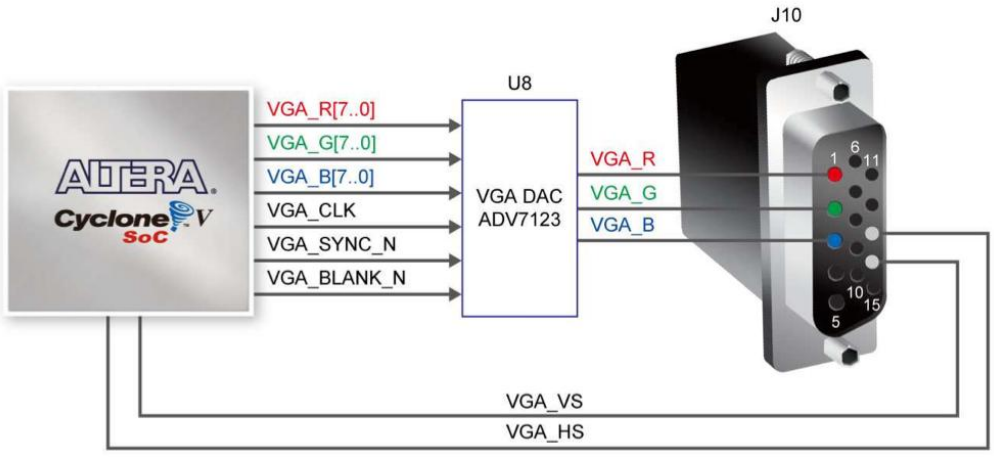


Figure 4: ADV7123 Pin Connection with FPGA

Image Classification	Module Name	Number of Images	Single Image Size	Total Size(KB)
Background	brick16	1	16*16	0.25
	water	1	32*32	1
	steel	1	32*32	1
	grass	1	32*32	1
	homebase	1	32*32	1
	life_flag	1	32*32	1
Welcome screen	begin1player	1	256*256	64
Game over screen	game_win	1	256*256	64
Tank	tank1	4	32*32	4
	tank_enemy	4	32*32	4
Display Effect	explosion	2	32*32	2
	generation	2	32*32	2
Total		20		145.25

Table 1: VGA data size table

2.1.2 VGA controller and VGA display

Our video game has up to 2 our tanks and 10 enemy tanks and each of them has its own characteristic and motion. A requirement for displaying tanks is that they can appear on the screen at the same time and cannot overlap each other. Another important features for our game is that brick can be destroyed by the bullet and tank can move over after that part of brick disappears. What is more, we also have another element shows rest lives of our tank and enemy tank. So we need to design a sprite controller to handle different sprites with different direction, location and color.

2.1.2.1 VGA display

The VGA display module mainly finishes the work of confining the area that certain dynamic sprites should appear. These dynamic sprites change its location and direction all the time according to the message from game controller. If the counter ‘hcount’ and ‘vcount’ enter the area that one of these dynamic sprites should appear, relative flag for this sprite showing display of it are set to ‘1’ to tell the VGA controller it should be drawn in this area. As we use different registers to store the information of sprite data and sprite address, there will be no conflicts in drawing these dynamic sprites.

2.1.2.2 VGA map

The VGA map module stores the information of static sprites in the background. This module helps to set all the obstacles in the map. In this map, if a bullet collide with a brick block, the relative flag for this brick block will be set to ‘0’, indicating that this part of brick has been destroyed.

2.1.2.3 VGA controller

All of the flag information indicating certain sprite appears at certain place is sent to this module. The main function of this module is to tell the VGA screen what and where certain sprites should be drawn. With this VGA controller, we can set the priority of display and

determine when to draw the sprites.

Figure below shows timing diagram for VGA horizontal timing specification:

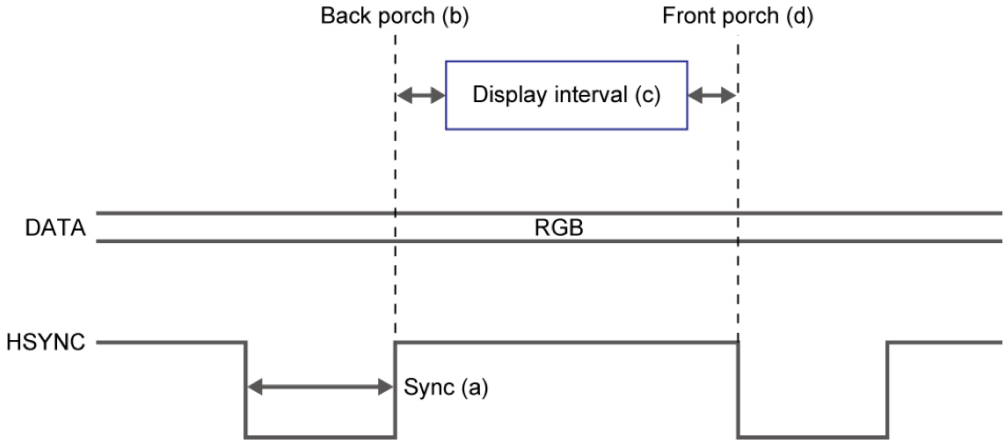


Figure 5:VGA Horizontal Timing Diagram

The RGB data should be read and sent to ADV7123 during the display interval. This timing mechanism can also be applied to vertical timing specification.
For ADV7123, which receives the digital signals from FPGA and convert them to analog signal, timing diagram for it is shown in the following figure.

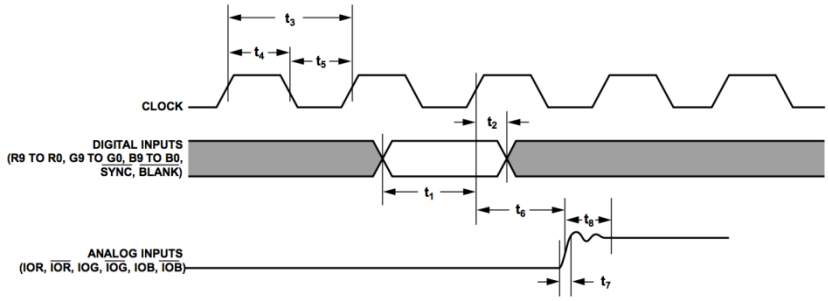


Figure 6: ADV7123 Timing Diagram

2.3 Audio Block Module

Our sound effects are extracted from the real classic battle tank game. The three two track sound effects including fire, move and explosion are extracted and analyzed by Matlab. The results show that the compressed .wav format sound file data is 16bit width at 44.1KHz sample rate. The DAC output analog signal diagram is shown below:

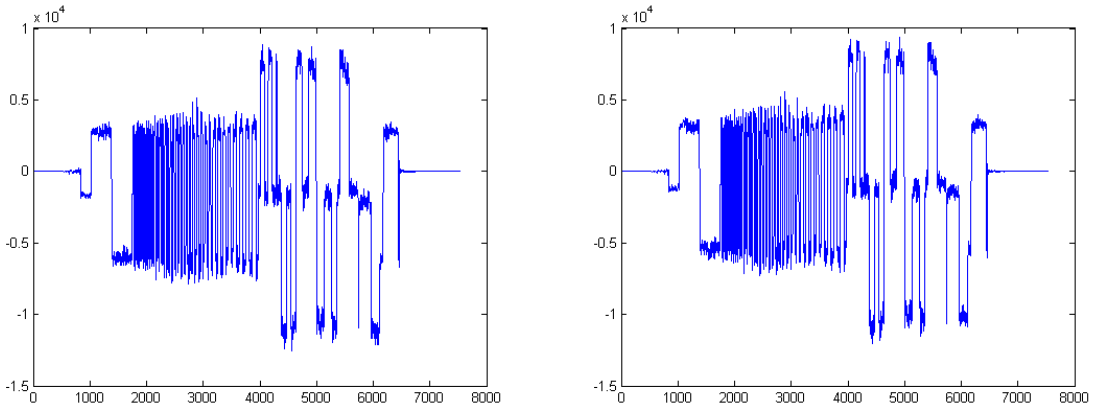


Figure 7: DAC analog output signal for left and right channel fire sound data

Therefore, the we have create relevant ROM portion to store the audio data below:

Sound Name	Width	Length	Size(KB)
Fire_L	16	7535	14.72
Fire_R	16	7535	14.72
Explosion_L	16	30575	59.72
Explosion_R	16	30575	59.72
Move	16	21359	41.72
Total	**16**	**97579**	**190.6**

Table 2: Table of audio effects size

From the data sheet, we choose the optimal Main Clock Frequency(MCLK) for the 44.1KHz as 11.2896MHz, this clock will be used for both audio codec controller and audio data controller while the I^2C bus controller use the main CPU clock frequency as 50MHz. Since the main clock frequency 50 MHz is hardly divided to obtain the 11.2896MHz audio clock. We have created a clock generator to generate the desired audio clock frequency by MegaWizard.

The software control interface provides access to the user-selectable control registers and can operate with a 2-wire (I2C) interface.

Within each control register is a control data-word consisting of 16 bits, MSB first. Bit B15 to Bit B9 are the register map address, and Bit B8 to Bit B0 are register data for the associated register map.

SDIN generates the serial control data-word, SCLK clocks the serial data, and CSB determines the I2C device address. If the CSB pin is set to 0, the address selected is 0011010; if 1, the address is 0011011.

The table below shows the relevant register we can use to configure the basic functions for our audio system such as volume and sampling rate.

Reg.	Address	Name	D8	D7	D6	D5	D4	D3	D2	D1	D0	Default
R0	0x00	Left-channel ADC input volume	LRINBOTH	LINMUTE	0	LINVOL[5:0]						010010111
R1	0x01	Right-channel ADC input volume	RLINBOTH	RINMUTE	0	RINVOL[5:0]						010010111
R2	0x02	Left-channel DAC volume	LRHPBOTH	0	LHPVOL[6:0]							001111001
R3	0x03	Right-channel DAC volume	RLHPBOTH	0	RHPVOL[6:0]							001111001
R4	0x04	Analog audio path	0	SIDETONE_ATT[1:0]		SIDETONE_EN	DACSEL	Bypass	INSEL	MUTEMIC	MICBOOST	000001010
R5	0x05	Digital audio path	0	0	0	0	HPOR	DACMU	DEEMPH[1:0]		ADCHPF	000001000
R6	0x06	Power management	0	PWROFF	CLKOUT	OSC	Out	DAC	ADC	MIC	LINEIN	010011111
R7	0x07	Digital audio I/F	0	BCLKINV	MS	LRSWAP	LRP	WL[1:0]		Format[1:0]		000001010
R8	0x08	Sampling rate	0	CLKODIV2	CLKDIV2	SR[3:0]				BOSR	USB	000000000
R9	0x09	Active	0	0	0	0	0	0	0	0	Active	000000000
R15	0x0F	Software reset	Reset[8:0]									000000000
R16	0x10	ALC Control 1	ALCSEL[1:0]		MAXGAIN[2:0]			ALCL[3:0]				001111011
R17	0x11	ALC Control 2	0	DCY[3:0]				ATK[3:0]				000110010
R18	0x12	Noise gate	0	NGTH[4:0]					NGG[1:0]		NGAT	000000000

Table 3: SSM2603 Register Map for I^2C Configuration

For example we write value 0x0479 and 0x0679 to register R2 and R3 to adjust the output volume for left and right channel as 0dB amplification of the original digital data. Furthermore, we give R8 the value of 0x1020 to set the sampling as 44.1KHz.

The digital audio input can support the following four digital audio communication protocols: right-justified mode, left-justified mode, I^2S mode, and digital signal processor (DSP) mode. For our project we choose to use Left-Justified mode.

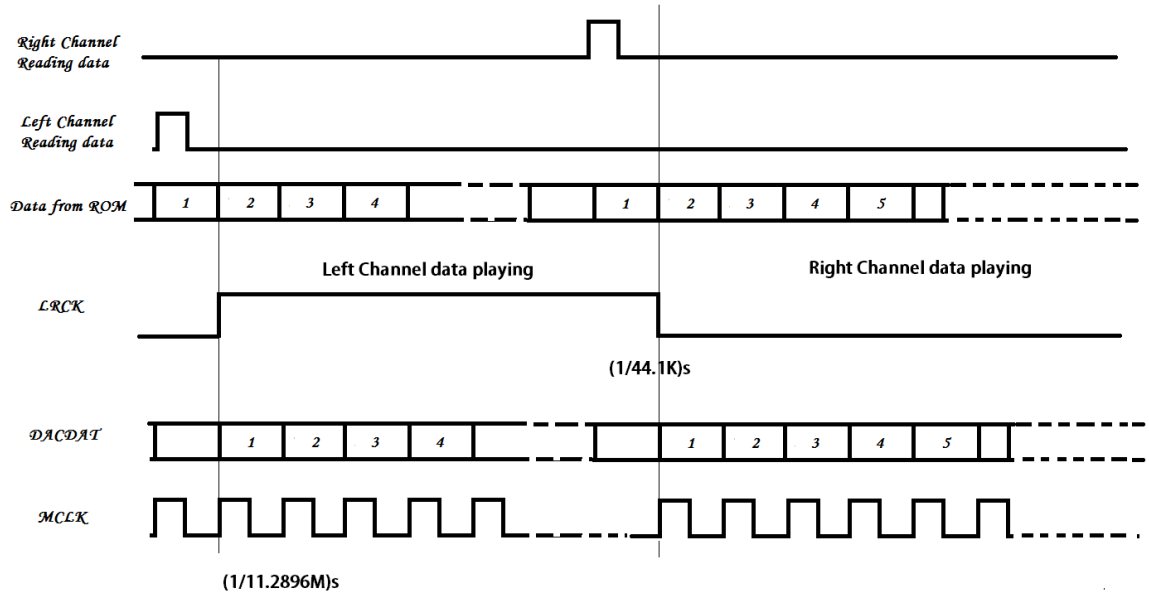


Figure 8: Left-Justified Mode audio codec configuration

As shown in Figure 7, the MCLK frequency is further divided by 256 to generate the 44.1KHz sampling frequency clock (LRCK). At each LRCK clock cycle, the first half high level clock cycle is used for codec decode the digital data for left channel while the second half clock cycle is for right channel output. There are two one hot signal before the left and right cannel data decoding period for the audio codec reading data from ROM. Then, at the rising edge and falling edge for the LRCK clock, the digital data will be sent to DAC to output analog signal for left and right channels respectively.

Besides, there is one more audio data controller to facilitate the audio controller to read digital data from different ROM portions.

2.4 Joystick USB controller

The game requires a PC joystick get the operation signals from the players. It will be connected directly to the SoCKit Board and communicate with the HPS through the USB port on the board. The button keys of the controller will be used to direct the tank to dodge the bullets from the enemies. And a shooting button will be used to shoot the enemy tanks to protect the homebase from being attacked. A command button to start this game is also needed. Like lab2, a USB controller needs to be built in System Verilog. And the interface for the joystick will be implemented in C program. The schematic diagram of the USB circuitry is shown below.

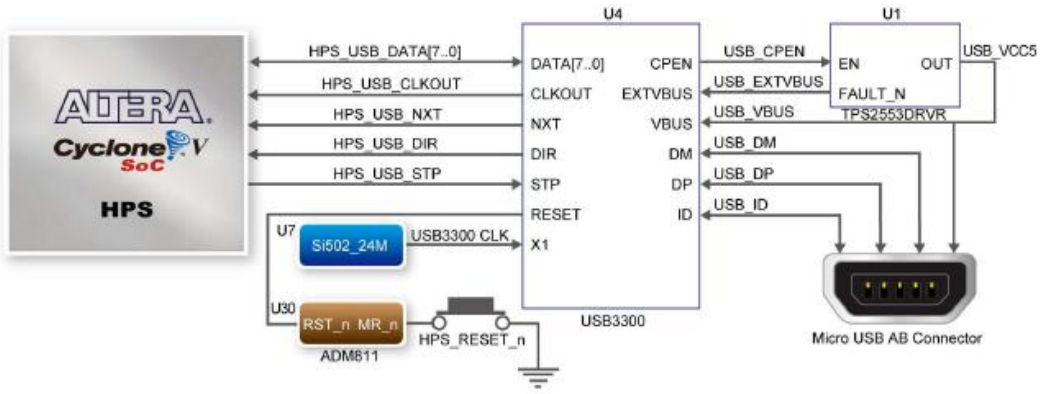


Figure 9: Connections between CycloneV SoC FPGA and USB OTG PHY

The Sockit board provides USB interfaces using the SMSC USB3300 controller. A SMSC USB3300 device in a 32-pin QFN package device is used to interface to a single Type AB Micro-USB connector. This device supports UTMI+ Low Pin Interface (ULPI) to communicate to USB 2.0controller in HPS. The ULPI interface consists of 12 interface pins; 8 bi-directional data pins, 3 control pins, and a 60 MHz clock. As defined by OTG mode, the PHY can operate in Host or Device modes. When operating in Host mode, the interface will supply the power to the device through the Micro-USB interface. In our project, we will make the PHY work in the device mode as what we need are just signals from the game controller. The pin assignments for the associated interface are listed in Tables below.

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
HPS_USB_CLKOUT	PIN_N16	60MHz Reference Clock Output	3.3V
HPS_USB_DATA[0]	PIN_E16	HPS USB_DATA[0]	3.3V
HPS_USB_DATA[1]	PIN_G16	HPS USB_DATA[1]	3.3V
HPS_USB_DATA[2]	PIN_D16	HPS USB_DATA[2]	3.3V
HPS_USB_DATA[3]	PIN_D14	HPS USB_DATA[3]	3.3V
HPS_USB_DATA[4]	PIN_A15	HPS USB_DATA[4]	3.3V
HPS_USB_DATA[5]	PIN_C14	HPS USB_DATA[5]	3.3V
HPS_USB_DATA[6]	PIN_D15	HPS USB_DATA[6]	3.3V
HPS_USB_DATA[7]	PIN_M17	HPS USB_DATA[7]	3.3V
HPS_USB_DIR	PIN_E14	Direction of the Data Bus	3.3V
HPS_USB_NXT	PIN_A14	Throttle the Data	3.3V
HPS_USB_RESET_PHY	PIN_G17	HPS USB PHY Reset	3.3V
HPS_USB_STP	PIN_C15	Stop Data Stream on theBus	3.3V

Table 4: Table for USB OTG PHY Pin Assignments

3.Software

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

1. The overall game logic control

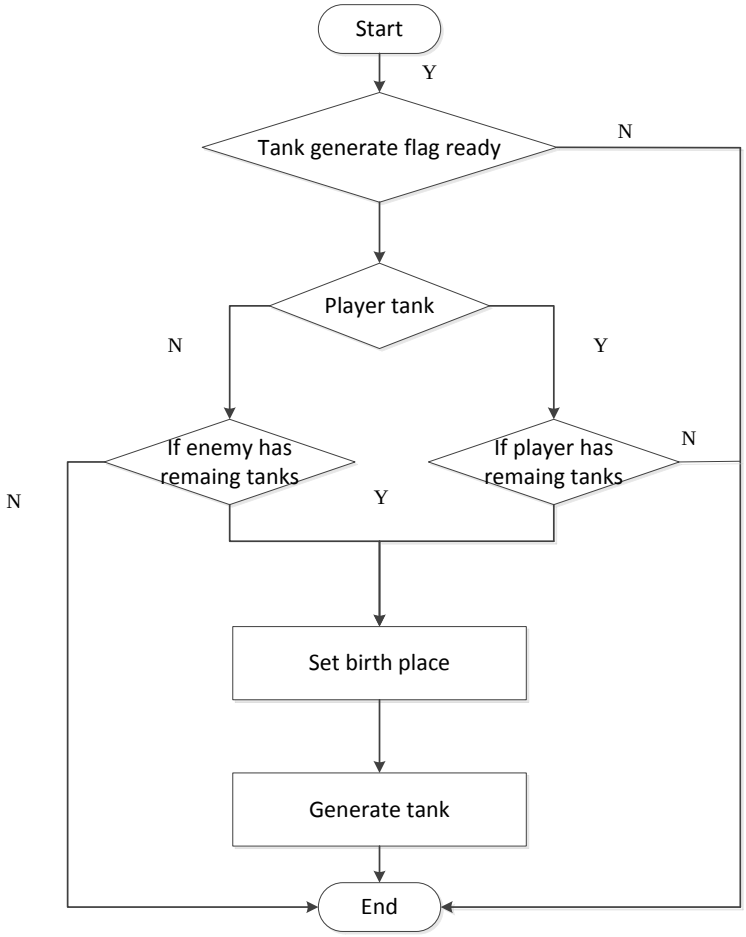
Software part controls the stages of the game. stage1: the welcoming at the beginning, stage2: the game progress, stage3: the game over. Use a FSM to achieve this. At the beginning, the game will stay at stage 1. Wait for the signal from the joystick to wake up into the game stage. At the game stage, there are three conditions to end up the game to stage 3. Such as the enemy are all gone, the own tanks are all gone and the homebase has been attacked.

2. The detection keycode module

The module called "gamecontroller" used to identify the joystick and detect the keycode to move the tank and shoot bullet.

3. The tank generate and movement logic

- Generating player tanks and moving them according to information from joysticker.
- Generating enemy tanks and giving the enemy an initial direction. Making them move until the enemy meets an obstacle ahead and stop. Then a new direction is given according to the random information.
- Obstacles are judged using function from tank map module which is introduced later.



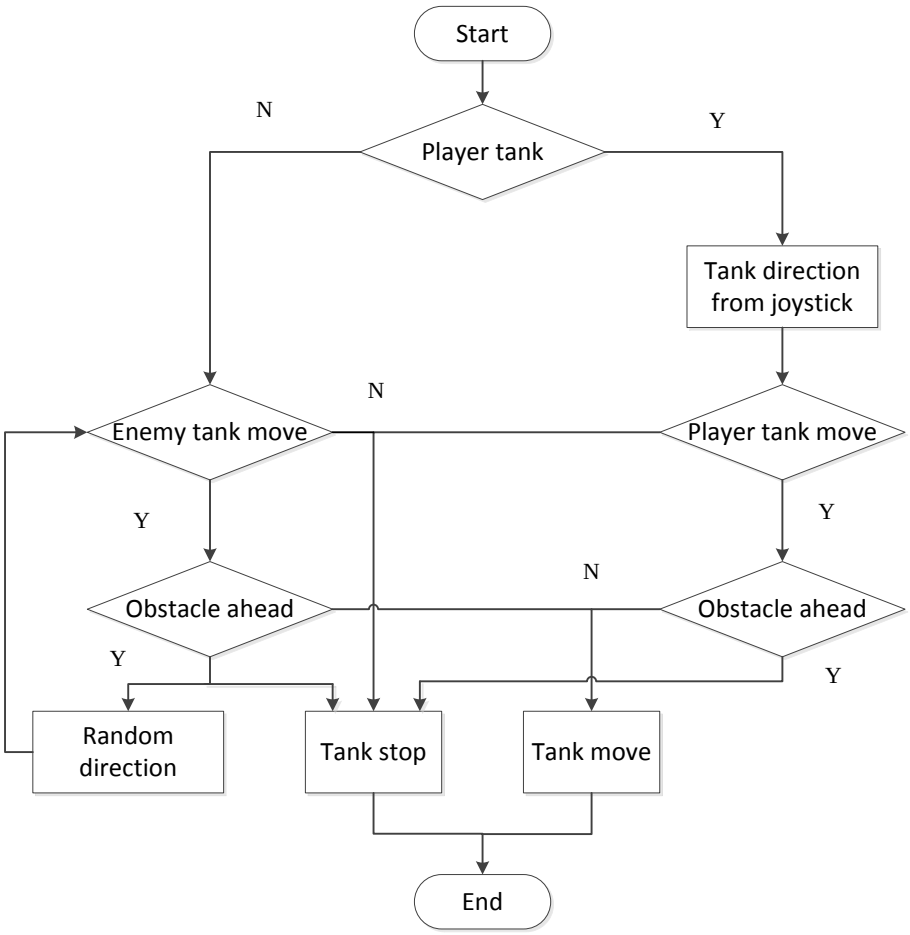


Figure 10: Tank generation and movement flowchart

4. The bullet control logic

- Generating the bullet according to the direction of tank and tank flag which shows whether the tank is exist.
- Judge whether the bullet meet a tank, brick and homebase or not. If the bullet meets the brick, it will disappear the bullet, the corresponding brick and update the map. If the bullet meets the tank, it will do an additional thing that changing the remaining tank number. If the bullet meets the homebase, it will disappears the homebase, and update the game to stage 3(game over)
- Judge whether two bullets meet or not, If the bullets meet, it will vanish all the bullets.
- Identify whether the bullet belongs to enemy or own tank, in order to avoid to injure own tank,accidentally.

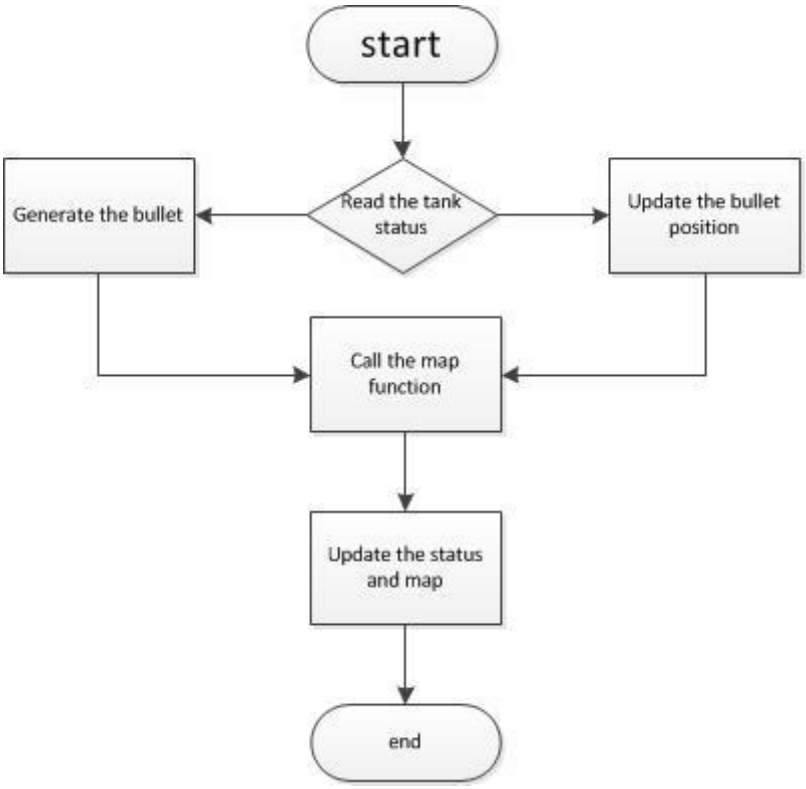


Figure 11: Bullet Generation flowchart

5. Explosion function

This function is used to connect the bullet function and tank function in some case. For instance, if the tank meet bullet, the tank will explode at the current position. After some time, the tank will regenerate in the initial position. The explosion function used a FSM to achieve these.

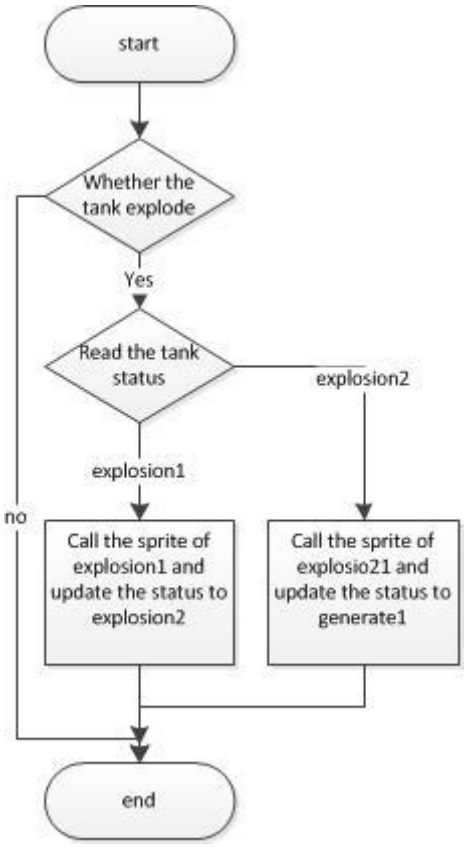


Figure 12: Explosion Flowchart

6. Global.h

All the game parameters are shows in the document of “global”. Such as game parameters, including the enemy number, the shoot rate of enemy, etc. And the command messages to Avalon bus, all the global variable, which is used between more functions.

4. Lesson learned

After a whole semester’s working with the FPGA board, we become more familiar with resources and architecture of FPGA. We definitely improve our skills in SystemVerilog coding and C program coding. In the whole process, we can feel the convenience provided by the SystemVerilog and learn deeply how to make software and hardware work well together.

Design the architecture of hardware and software to make them cooperates well.

Before we start to build the whole project, it is better for us to have a general idea of the whole architecture of both software and hardware. This will help us work on programming more efficiently. For example, we have the whole view of our design and build the block diagram of game architecture and implementation. When we start programming, we can have the ports connected correctly and message sent from software to hardware timely.

Use Qsys to build the component and have it connected to the hardware

Resources allocation is a key point in the whole project

During the project, we have met the problem of running out of ROM because of storing to much data into the FPGA. Thus synthesis cannot continue because of running out of resources. Then we redo the resources allocation and make the resources used more efficiently. This not only reduces the running time, but also makes our SystemVerilog looks more clear.

5. Contribution

Lupeng Fan (lf2447)：leading the hardware part including audio codec, VGA display module, game control state machine, configure all the software to hardware interface.

Yichen Zhu (yz2582): Hardware part: VGA display module, debugging all modules, Software part: game controller module, tank explosion module, tank bullet module

Di Yang (dy2266) : VGA display module, VGA map module, VGA controller module, part of image preparation, software part: build map for tank moving in software.

Yinshen Wang (yw2561): Hardware part: VGA display module, VGA image preparation, audio codec. Software part: Tank generate module, Tank move module.

```
// ============================================================================
// E4840 Design Project - Battle Tank Game Design
//
//        Team: Tank
//
// Columbia University, EE Dept
// ============================================================================

`define ENABLE_DDR3

module SoCKit_Top(

                ///////////AUD/////////////
                AUD_ADCDAT,
                AUD_ADCLRCK,
                AUD_BCLK,
                AUD_DACDAT,
                AUD_DACLRCK,
                AUD_I2C_SCLK,
                AUD_I2C_SDAT,
                AUD_MUTE,
                AUD_XCK,

`ifdef ENABLE_DDR3
                /////////DDR3/////////
                DDR3_A,
                DDR3_BA,
                DDR3_CAS_n,
                DDR3_CKE,
                DDR3_CK_n,
                DDR3_CK_p,
                DDR3_CS_n,
                DDR3_DM,
                DDR3_DQ,
                DDR3_DQS_n,
                DDR3_DQS_p,
                DDR3_ODT,
                DDR3_RAS_n,
                DDR3_RESET_n,
                DDR3_RZQ,
                DDR3_WE_n,
`endif /*ENABLE_DDR3*/

                /////////FAN/////////
                FAN_CTRL,

`ifdef ENABLE_HPS
                /////////HPS/////////
                HPS_CLOCK_25,
```

HPS_CLOCK_50,
HPS_CONV_USB_n,
HPS_DDR3_A,
HPS_DDR3_BA,
HPS_DDR3_CAS_n,
HPS_DDR3_CKE,
HPS_DDR3_CK_n,
HPS_DDR3_CK_p,
HPS_DDR3_CS_n,
HPS_DDR3_DM,
HPS_DDR3_DQ,
HPS_DDR3_DQS_n,
HPS_DDR3_DQS_p,
HPS_DDR3_ODT,
HPS_DDR3_RAS_n,
HPS_DDR3_RESET_n,
HPS_DDR3_RZQ,
HPS_DDR3_WE_n,
HPS_ENET_GTX_CLK,
HPS_ENET_INT_n,
HPS_ENET_MDC,
HPS_ENET_MDIO,
HPS_ENET_RESET_n,
HPS_ENET_RX_CLK,
HPS_ENET_RX_DATA,
HPS_ENET_RX_DV,
HPS_ENET_TX_DATA,
HPS_ENET_TX_EN,
HPS_FLASH_DATA,
HPS_FLASH_DCLK,
HPS_FLASH_NCSO,
HPS_GSENSOR_INT,
HPS_I2C_CLK,
HPS_I2C_SDA,
HPS_KEY,
HPS_LCM_D_C,
HPS_LCM_RST_N,
HPS_LCM_SPIM_CLK,
HPS_LCM_SPIM_MISO,
HPS_LCM_SPIM_MOSI,
HPS_LCM_SPIM_SS,
HPS_LED,
HPS_LTC_GPIO,
HPS_RESET_n,
HPS_SD_CLK,
HPS_SD_CMD,
HPS_SD_DATA,
HPS_SPIM_CLK,
HPS_SPIM_MISO,

```
                HPS_SPIM_MOSI,
                HPS_SPIM_SS,
                HPS_SW,
                HPS_UART_RX,
                HPS_UART_TX,
                HPS_USB_CLKOUT,
                HPS_USB_DATA,
                HPS_USB_DIR,
                HPS_USB_NXT,
                HPS_USB_RESET_PHY,
                HPS_USB_STP,
                HPS_WARM_RST_n,
`endif /*ENABLE_HPS*/

                /////////HSMC/////////
                HSMC_CLKIN_n,
                HSMC_CLKIN_p,
                HSMC_CLKOUT_n,
                HSMC_CLKOUT_p,
                HSMC_CLK_IN0,
                HSMC_CLK_OUT0,
                HSMC_D,

`ifdef ENABLE_HSMC_XCVR

                HSMC_GXB_RX_p,
                HSMC_GXB_TX_p,
                HSMC_REF_CLK_p,
`endif
                HSMC_RX_n,
                HSMC_RX_p,
                HSMC_SCL,
                HSMC_SDA,
                HSMC_TX_n,
                HSMC_TX_p,

                /////////IRDA/////////
                IRDA_RXD,

                /////////KEY/////////
                KEY,

                /////////LED/////////
                LED,

                /////////OSC/////////
                OSC_50_B3B,
                OSC_50_B4A,
                OSC_50_B5B,
```

OSC_50_B8A,

//////////PCIE//////////
PCIE_PERST_n,
PCIE_WAKE_n,

//////////RESET//////////
RESET_n,

//////////SI5338//////////
SI5338_SCL,
SI5338_SDA,

//////////SW//////////
SW,

//////////TEMP//////////
TEMP_CS_n,
TEMP_DIN,
TEMP_DOUT,
TEMP_SCLK,

//////////USB//////////
USB_B2_CLK,
USB_B2_DATA,
USB_EMPTY,
USB_FULL,
USB_OE_n,
USB_RD_n,
USB_RESET_n,
USB_SCL,
USB_SDA,
USB_WR_n,

//////////VGA//////////
VGA_B,
VGA_BLANK_n,
VGA_CLK,
VGA_G,
VGA_HS,
VGA_R,
VGA_SYNC_n,
VGA_VS,
////////////hps//////////
memory_mem_a,
memory_mem_ba,
memory_mem_ck,
memory_mem_ck_n,
memory_mem_cke,

memory_mem_cs_n,

memory_mem_ras_n,

memory_mem_cas_n,

memory_mem_we_n,

memory_mem_reset_n,

memory_mem_dq,

memory_mem_dqs,

memory_mem_dqs_n,

memory_mem_odt,

memory_mem_dm,

memory_oct_rzqin,

hps_io_hps_io_emac1_inst_TX_CLK,

hps_io_hps_io_emac1_inst_TXD0,

hps_io_hps_io_emac1_inst_TXD1,

hps_io_hps_io_emac1_inst_TXD2,

hps_io_hps_io_emac1_inst_TXD3,

hps_io_hps_io_emac1_inst_RXD0,

hps_io_hps_io_emac1_inst_MDIO,

hps_io_hps_io_emac1_inst_MDC,

hps_io_hps_io_emac1_inst_RX_CTL,

hps_io_hps_io_emac1_inst_TX_CTL,

hps_io_hps_io_emac1_inst_RX_CLK,

hps_io_hps_io_emac1_inst_RXD1,

hps_io_hps_io_emac1_inst_RXD2,

hps_io_hps_io_emac1_inst_RXD3,

hps_io_hps_io_qspi_inst_IO0,

hps_io_hps_io_qspi_inst_IO1,

hps_io_hps_io_qspi_inst_IO2,

hps_io_hps_io_qspi_inst_IO3,

hps_io_hps_io_qspi_inst_SS0,

hps_io_hps_io_qspi_inst_CLK,

hps_io_hps_io_sdio_inst_CMD,

hps_io_hps_io_sdio_inst_D0,

hps_io_hps_io_sdio_inst_D1,

hps_io_hps_io_sdio_inst_CLK,

hps_io_hps_io_sdio_inst_D2,

hps_io_hps_io_sdio_inst_D3,

hps_io_hps_io_usb1_inst_D0,

hps_io_hps_io_usb1_inst_D1,

hps_io_hps_io_usb1_inst_D2,

hps_io_hps_io_usb1_inst_D3,

hps_io_hps_io_usb1_inst_D4,

hps_io_hps_io_usb1_inst_D5,

hps_io_hps_io_usb1_inst_D6,

hps_io_hps_io_usb1_inst_D7,

hps_io_hps_io_usb1_inst_CLK,

hps_io_hps_io_usb1_inst_STP,

hps_io_hps_io_usb1_inst_DIR,

hps_io_hps_io_usb1_inst_NXT,

```
                hps_io_hps_io_spim0_inst_CLK,
                hps_io_hps_io_spim0_inst_MOSI,
                hps_io_hps_io_spim0_inst_MISO,
                hps_io_hps_io_spim0_inst_SS0,
                hps_io_hps_io_spim1_inst_CLK,
                hps_io_hps_io_spim1_inst_MOSI,
                hps_io_hps_io_spim1_inst_MISO,
                hps_io_hps_io_spim1_inst_SS0,
                hps_io_hps_io_uart0_inst_RX,
                hps_io_hps_io_uart0_inst_TX,
                hps_io_hps_io_i2c1_inst_SDA,
                hps_io_hps_io_i2c1_inst_SCL,
                hps_io_hps_io_gpio_inst_GPIO00
                );

    //=======================================================
    //  PORT declarations
    //=======================================================

    ///////// AUD /////////
    input                                   AUD_ADCDAT;
    inout                                   AUD_ADCLRCK;
    inout                                   AUD_BCLK;
    output                                  AUD_DACDAT;
    inout                                   AUD_DACLRCK;
    output                                  AUD_I2C_SCLK;
    inout                                   AUD_I2C_SDAT;
    output                                  AUD_MUTE;
    output                                  AUD_XCK;

`ifdef ENABLE_DDR3
    ///////// DDR3 /////////
    output [14:0]                                   DDR3_A;
    output [2:0]                                DDR3_BA;
    output                                  DDR3_CAS_n;
    output                                  DDR3_CKE;
    output                                  DDR3_CK_n;
    output                                  DDR3_CK_p;
    output                                  DDR3_CS_n;
    output [3:0]                                DDR3_DM;
    inout [31:0]                                DDR3_DQ;
    inout [3:0]                                 DDR3_DQS_n;
    inout [3:0]                                 DDR3_DQS_p;
    output                                  DDR3_ODT;
    output                                  DDR3_RAS_n;
    output                                  DDR3_RESET_n;
    input                                   DDR3_RZQ;
    output                                  DDR3_WE_n;
`endif /*ENABLE_DDR3*/
```

```
    ///////// FAN /////////
    output                                  FAN_CTRL;

`ifdef ENABLE_HPS
    ///////// HPS /////////
    input                                   HPS_CLOCK_25;
    input                                   HPS_CLOCK_50;
    input                                   HPS_CONV_USB_n;
    output [14:0]                                       HPS_DDR3_A;
    output [2:0]                                        HPS_DDR3_BA;
    output                                  HPS_DDR3_CAS_n;
    output                                  HPS_DDR3_CKE;
    output                                  HPS_DDR3_CK_n;
    output                                  HPS_DDR3_CK_p;
    output                                  HPS_DDR3_CS_n;
    output [3:0]                                        HPS_DDR3_DM;
    inout [31:0]                                         HPS_DDR3_DQ;
    inout [3:0]                                         HPS_DDR3_DQS_n;
    inout [3:0]                                         HPS_DDR3_DQS_p;
    output                                  HPS_DDR3_ODT;
    output                                  HPS_DDR3_RAS_n;
    output                                  HPS_DDR3_RESET_n;
    input                                   HPS_DDR3_RZQ;
    output                                  HPS_DDR3_WE_n;
    input                                   HPS_ENET_GTX_CLK;
    input                                   HPS_ENET_INT_n;
    output                                  HPS_ENET_MDC;
    inout                                   HPS_ENET_MDIO;
    output                                  HPS_ENET_RESET_n;
    input                                   HPS_ENET_RX_CLK;
    input [3:0]                                         HPS_ENET_RX_DATA;
    input                                   HPS_ENET_RX_DV;
    output [3:0]                                        HPS_ENET_TX_DATA;
    output                                  HPS_ENET_TX_EN;
    inout [3:0]                                         HPS_FLASH_DATA;
    output                                  HPS_FLASH_DCLK;
    output                                  HPS_FLASH_NCSO;
    input                                   HPS_GSENSOR_INT;
    inout                                   HPS_I2C_CLK;
    inout                                   HPS_I2C_SDA;
    inout [3:0]                                         HPS_KEY;
    output                                  HPS_LCM_D_C;
    output                                  HPS_LCM_RST_N;
    input                                   HPS_LCM_SPIM_CLK;
    inout                                   HPS_LCM_SPIM_MISO;
    output                                  HPS_LCM_SPIM_MOSI;
    output                                  HPS_LCM_SPIM_SS;
    output [3:0]                                        HPS_LED;
```

```
    inout                                   HPS_LTC_GPIO;
    input                                   HPS_RESET_n;
    output                                  HPS_SD_CLK;
    inout                                   HPS_SD_CMD;
    inout [3:0]                                         HPS_SD_DATA;
    output                                  HPS_SPIM_CLK;
    input                                   HPS_SPIM_MISO;
    output                                  HPS_SPIM_MOSI;
    output                                  HPS_SPIM_SS;
    input [3:0]                                         HPS_SW;
    input                                   HPS_UART_RX;
    output                                  HPS_UART_TX;
    input                                   HPS_USB_CLKOUT;
    inout [7:0]                                         HPS_USB_DATA;
    input                                   HPS_USB_DIR;
    input                                   HPS_USB_NXT;
    output                                  HPS_USB_RESET_PHY;
    output                                  HPS_USB_STP;
    input                                   HPS_WARM_RST_n;
`endif /*ENABLE_HPS*/

    ///////// HSMC /////////
    input [2:1]                                             HSMC_CLKIN_n;
    input [2:1]                                         HSMC_CLKIN_p;
    output [2:1]                                        HSMC_CLKOUT_n;
    output [2:1]                                        HSMC_CLKOUT_p;
    input                                   HSMC_CLK_IN0;
    output                                  HSMC_CLK_OUT0;
    inout [3:0]                                         HSMC_D;
`ifdef ENABLE_HSMC_XCVR
    input [7:0]                                         HSMC_GXB_RX_p;
    output [7:0]                                        HSMC_GXB_TX_p;
    input                                   HSMC_REF_CLK_p;
`endif
    inout [16:0]                                            HSMC_RX_n;
    inout [16:0]                                        HSMC_RX_p;
    output                                  HSMC_SCL;
    inout                                   HSMC_SDA;
    inout [16:0]                                        HSMC_TX_n;
    inout [16:0]                                        HSMC_TX_p;

    ///////// IRDA /////////
    input                                   IRDA_RXD;

    ///////// KEY /////////
    input [3:0]                                         KEY;

    ///////// LED /////////
    output [3:0]                                        LED;
```

```
//////// OSC ////////
input                                   OSC_50_B3B;
input                                   OSC_50_B4A;
input                                   OSC_50_B5B;
input                                   OSC_50_B8A;

//////// PCIE ////////
input                                   PCIE_PERST_n;
input                                   PCIE_WAKE_n;

//////// RESET ////////
input                                   RESET_n;

//////// SI5338 ////////
inout                                   SI5338_SCL;
inout                                   SI5338_SDA;

//////// SW ////////
input [3:0]                                             SW;

//////// TEMP ////////
output                                  TEMP_CS_n;
output                                  TEMP_DIN;
input                                   TEMP_DOUT;
output                                  TEMP_SCLK;

//////// USB ////////
input                                   USB_B2_CLK;
inout [7:0]                                             USB_B2_DATA;
output                                  USB_EMPTY;
output                                  USB_FULL;
input                                   USB_OE_n;
input                                   USB_RD_n;
input                                   USB_RESET_n;
inout                                   USB_SCL;
inout                                   USB_SDA;
input                                   USB_WR_n;

//////// VGA ////////
output [7:0]                                            VGA_B;
output                                  VGA_BLANK_n;
output                                  VGA_CLK;
output [7:0]                                            VGA_G;
output                                  VGA_HS;
output [7:0]                                            VGA_R;
output                                  VGA_SYNC_n;
output                                  VGA_VS;
```

```
////////hps pin///////
output wire [14:0]          memory_mem_a;
output wire [2:0]           memory_mem_ba;
output wire                 memory_mem_ck;
output wire                 memory_mem_ck_n;
output wire                 memory_mem_cke;
output wire                 memory_mem_cs_n;
output wire                 memory_mem_ras_n;
output wire                 memory_mem_cas_n;
output wire                 memory_mem_we_n;
output wire                 memory_mem_reset_n;
inout   wire [31:0]         memory_mem_dq;
inout   wire [3:0]          memory_mem_dqs;
inout   wire [3:0]          memory_mem_dqs_n;
output wire                 memory_mem_odt;
output wire [3:0]           memory_mem_dm;
input   wire                memory_oct_rzqin;
output wire                 hps_io_hps_io_emac1_inst_TX_CLK;
output wire                 hps_io_hps_io_emac1_inst_TXD0;
output wire                 hps_io_hps_io_emac1_inst_TXD1;
output wire                 hps_io_hps_io_emac1_inst_TXD2;
output wire                 hps_io_hps_io_emac1_inst_TXD3;
input   wire                hps_io_hps_io_emac1_inst_RXD0;
inout   wire                hps_io_hps_io_emac1_inst_MDIO;
output wire                 hps_io_hps_io_emac1_inst_MDC;
input   wire                hps_io_hps_io_emac1_inst_RX_CTL;
output wire                 hps_io_hps_io_emac1_inst_TX_CTL;
input   wire                hps_io_hps_io_emac1_inst_RX_CLK;
input   wire                hps_io_hps_io_emac1_inst_RXD1;
input   wire                hps_io_hps_io_emac1_inst_RXD2;
input   wire                hps_io_hps_io_emac1_inst_RXD3;
inout   wire                hps_io_hps_io_qspi_inst_IO0;
inout   wire                hps_io_hps_io_qspi_inst_IO1;
inout   wire                hps_io_hps_io_qspi_inst_IO2;
inout   wire                hps_io_hps_io_qspi_inst_IO3;
output wire                 hps_io_hps_io_qspi_inst_SS0;
output wire                 hps_io_hps_io_qspi_inst_CLK;
inout   wire                hps_io_hps_io_sdio_inst_CMD;
inout   wire                hps_io_hps_io_sdio_inst_D0;
inout   wire                hps_io_hps_io_sdio_inst_D1;
output wire                 hps_io_hps_io_sdio_inst_CLK;
inout   wire                hps_io_hps_io_sdio_inst_D2;
inout   wire                hps_io_hps_io_sdio_inst_D3;
inout   wire                hps_io_hps_io_usb1_inst_D0;
inout   wire                hps_io_hps_io_usb1_inst_D1;
inout   wire                hps_io_hps_io_usb1_inst_D2;
inout   wire                hps_io_hps_io_usb1_inst_D3;
inout   wire                hps_io_hps_io_usb1_inst_D4;
inout   wire                hps_io_hps_io_usb1_inst_D5;
```

```
inout   wire                    hps_io_hps_io_usb1_inst_D6;
inout   wire                    hps_io_hps_io_usb1_inst_D7;
input   wire                    hps_io_hps_io_usb1_inst_CLK;
output wire                     hps_io_hps_io_usb1_inst_STP;
input   wire                    hps_io_hps_io_usb1_inst_DIR;
input   wire                    hps_io_hps_io_usb1_inst_NXT;
output wire                     hps_io_hps_io_spim0_inst_CLK;
output wire                     hps_io_hps_io_spim0_inst_MOSI;
input   wire                    hps_io_hps_io_spim0_inst_MISO;
output wire                     hps_io_hps_io_spim0_inst_SS0;
output wire                     hps_io_hps_io_spim1_inst_CLK;
output wire                     hps_io_hps_io_spim1_inst_MOSI;
input   wire                    hps_io_hps_io_spim1_inst_MISO;
output wire                     hps_io_hps_io_spim1_inst_SS0;
input   wire                    hps_io_hps_io_uart0_inst_RX;
output wire                     hps_io_hps_io_uart0_inst_TX;
inout   wire                    hps_io_hps_io_i2c1_inst_SDA;
inout   wire                    hps_io_hps_io_i2c1_inst_SCL;
inout   wire                    hps_io_hps_io_gpio_inst_GPIO00;
//=======================================================
//  REG/WIRE declarations
//=======================================================

    // For Game Controller
    wire [31:0]         battle_tank_game_controller_writedata;  //          .writedata
    wire                battle_tank_game_controller_write;      //          .write
    wire                battle_tank_game_controller_chipselect;//          .chipselect
    wire [6:0]          battle_tank_game_controller_address;
    wire                battle_tank_game_controller_reset;
    wire                bullet_audio_signal;


//  For Audio CODEC
    wire                tank_explosion_audio;
    wire                tank1_bullet_audio;
    wire                tank2_bullet_audio;
    wire                move_audio;

//  For VGA Controller
    wire [10:0]         hcount_vga;
    wire [9:0]          vcount_vga;
    wire                vga_led_reset;
    wire [2:0]          game_stage_vga;
    wire [10:0]         tank_x_vga;
    wire [9:0]          tank_y_vga;
    wire [69:0]         brick_sub_flag_vga;
    wire [10:0]         bullet_x_vga;
    wire [9:0]          bullet_y_vga;
    wire                tank1_flag_vga;
```

```
    wire                tank2_flag_vga;
    wire [1:0]          tank_explosion_flag_vga;
    wire [1:0]          tank_generation_flag_vga;
    wire [1:0]          tank1_explosion_flag_vga;
    wire [1:0]          tank1_generation_flag_vga;
    wire [1:0]          tank2_explosion_flag_vga;
    wire [1:0]          tank2_generation_flag_vga;
    wire                bullet_flag_vga;
    wire                tank_enemy_flag_vga;
    wire [1:0]          tank_direction_vga;
    wire [5:0]          our_life_sub_flag_vga;
    wire [4:0]          enemy_life_sub_flag_vga;

//  For Down Sample
wire [3:0]                  Remain;
wire [9:0]                  Quotient;

wire                        AUD_MUTE;

// Drive the LEDs with the switches
assign LED = SW;

// Make the FPGA reset cause an HPS reset
reg [19:0]                  hps_reset_counter = 20'h0;
reg                         hps_fpga_reset_n = 0;

always @(posedge OSC_50_B4A) begin
    if (hps_reset_counter == 20'h fffffff) hps_fpga_reset_n <= 1;
    hps_reset_counter <= hps_reset_counter + 1;
end

battle_tank t0 (
        .clk_clk                (OSC_50_B4A),           //      clk.clk
        .reset_reset_n          (hps_fpga_reset_n),     //    reset.reset_n
        .memory_mem_a           (memory_mem_a),         //   memory.mem_a
        .memory_mem_ba          (memory_mem_ba),        //   .mem_ba
        .memory_mem_ck          (memory_mem_ck),        //   .mem_ck
        .memory_mem_ck_n        (memory_mem_ck_n),      //   .mem_ck_n
        .memory_mem_cke         (memory_mem_cke),       //   .mem_cke
        .memory_mem_cs_n        (memory_mem_cs_n),      //   .mem_cs_n
        .memory_mem_ras_n       (memory_mem_ras_n),     //   .mem_ras_n
        .memory_mem_cas_n       (memory_mem_cas_n),     //   .mem_cas_n
        .memory_mem_we_n        (memory_mem_we_n),      //   .mem_we_n
        .memory_mem_reset_n     (memory_mem_reset_n),   //   .mem_reset_n
        .memory_mem_dq          (memory_mem_dq),        //   .mem_dq
        .memory_mem_dqs         (memory_mem_dqs),       //   .mem_dqs
        .memory_mem_dqs_n       (memory_mem_dqs_n),     //   .mem_dqs_n
        .memory_mem_odt         (memory_mem_odt),       //   .mem_odt
        .memory_mem_dm          (memory_mem_dm),        //   .mem_dm
```

```
        .memory_oct_rzqin                       (memory_oct_rzqin),                 //                  .oct_rzqin
        .hps_io_hps_io_emac1_inst_TX_CLK        (hps_io_hps_io_emac1_inst_TX_CLK), //
    .hps_0_hps_io.hps_io_emac1_inst_TX_CLK
        .hps_io_hps_io_emac1_inst_TXD0          (hps_io_hps_io_emac1_inst_TXD0),
//              .hps_io_emac1_inst_TXD0
        .hps_io_hps_io_emac1_inst_TXD1          (hps_io_hps_io_emac1_inst_TXD1),
//              .hps_io_emac1_inst_TXD1
        .hps_io_hps_io_emac1_inst_TXD2          (hps_io_hps_io_emac1_inst_TXD2),
//              .hps_io_emac1_inst_TXD2
        .hps_io_hps_io_emac1_inst_TXD3          (hps_io_hps_io_emac1_inst_TXD3),
//              .hps_io_emac1_inst_TXD3
        .hps_io_hps_io_emac1_inst_RXD0          (hps_io_hps_io_emac1_inst_RXD0),
//              .hps_io_emac1_inst_RXD0
        .hps_io_hps_io_emac1_inst_MDIO          (hps_io_hps_io_emac1_inst_MDIO),
//              .hps_io_emac1_inst_MDIO
        .hps_io_hps_io_emac1_inst_MDC           (hps_io_hps_io_emac1_inst_MDC),
//              .hps_io_emac1_inst_MDC
        .hps_io_hps_io_emac1_inst_RX_CTL        (hps_io_hps_io_emac1_inst_RX_CTL),
//              .hps_io_emac1_inst_RX_CTL
        .hps_io_hps_io_emac1_inst_TX_CTL        (hps_io_hps_io_emac1_inst_TX_CTL),
//              .hps_io_emac1_inst_TX_CTL
        .hps_io_hps_io_emac1_inst_RX_CLK        (hps_io_hps_io_emac1_inst_RX_CLK),
//              .hps_io_emac1_inst_RX_CLK
        .hps_io_hps_io_emac1_inst_RXD1          (hps_io_hps_io_emac1_inst_RXD1),
//              .hps_io_emac1_inst_RXD1
        .hps_io_hps_io_emac1_inst_RXD2          (hps_io_hps_io_emac1_inst_RXD2),
//              .hps_io_emac1_inst_RXD2
        .hps_io_hps_io_emac1_inst_RXD3          (hps_io_hps_io_emac1_inst_RXD3),
//              .hps_io_emac1_inst_RXD3
        .hps_io_hps_io_qspi_inst_IO0            (hps_io_hps_io_qspi_inst_IO0),      //              .hps_io_qspi_inst_IO0
        .hps_io_hps_io_qspi_inst_IO1            (hps_io_hps_io_qspi_inst_IO1),      //              .hps_io_qspi_inst_IO1
        .hps_io_hps_io_qspi_inst_IO2            (hps_io_hps_io_qspi_inst_IO2),      //              .hps_io_qspi_inst_IO2
        .hps_io_hps_io_qspi_inst_IO3            (hps_io_hps_io_qspi_inst_IO3),      //              .hps_io_qspi_inst_IO3
        .hps_io_hps_io_qspi_inst_SS0            (hps_io_hps_io_qspi_inst_SS0),      //              .hps_io_qspi_inst_SS0
        .hps_io_hps_io_qspi_inst_CLK            (hps_io_hps_io_qspi_inst_CLK),      //              .hps_io_qspi_inst_CLK
        .hps_io_hps_io_sdio_inst_CMD            (hps_io_hps_io_sdio_inst_CMD),
//              .hps_io_sdio_inst_CMD
        .hps_io_hps_io_sdio_inst_D0             (hps_io_hps_io_sdio_inst_D0),       //              .hps_io_sdio_inst_D0
        .hps_io_hps_io_sdio_inst_D1             (hps_io_hps_io_sdio_inst_D1),       //              .hps_io_sdio_inst_D1
        .hps_io_hps_io_sdio_inst_CLK            (hps_io_hps_io_sdio_inst_CLK),      //              .hps_io_sdio_inst_CLK
        .hps_io_hps_io_sdio_inst_D2             (hps_io_hps_io_sdio_inst_D2),       //              .hps_io_sdio_inst_D2
        .hps_io_hps_io_sdio_inst_D3             (hps_io_hps_io_sdio_inst_D3),       //              .hps_io_sdio_inst_D3
        .hps_io_hps_io_usb1_inst_D0             (hps_io_hps_io_usb1_inst_D0),       //              .hps_io_usb1_inst_D0
        .hps_io_hps_io_usb1_inst_D1             (hps_io_hps_io_usb1_inst_D1),       //              .hps_io_usb1_inst_D1
        .hps_io_hps_io_usb1_inst_D2             (hps_io_hps_io_usb1_inst_D2),       //              .hps_io_usb1_inst_D2
        .hps_io_hps_io_usb1_inst_D3             (hps_io_hps_io_usb1_inst_D3),       //              .hps_io_usb1_inst_D3
        .hps_io_hps_io_usb1_inst_D4             (hps_io_hps_io_usb1_inst_D4),       //              .hps_io_usb1_inst_D4
        .hps_io_hps_io_usb1_inst_D5             (hps_io_hps_io_usb1_inst_D5),       //              .hps_io_usb1_inst_D5
        .hps_io_hps_io_usb1_inst_D6             (hps_io_hps_io_usb1_inst_D6),       //              .hps_io_usb1_inst_D6
```

```
        .hps_io_hps_io_usb1_inst_D7        (hps_io_hps_io_usb1_inst_D7),        //        .hps_io_usb1_inst_D7
        .hps_io_hps_io_usb1_inst_CLK       (hps_io_hps_io_usb1_inst_CLK),
//              .hps_io_usb1_inst_CLK
        .hps_io_hps_io_usb1_inst_STP       (hps_io_hps_io_usb1_inst_STP),       //        .hps_io_usb1_inst_STP
        .hps_io_hps_io_usb1_inst_DIR       (hps_io_hps_io_usb1_inst_DIR),       //        .hps_io_usb1_inst_DIR
        .hps_io_hps_io_usb1_inst_NXT       (hps_io_hps_io_usb1_inst_NXT),
//              .hps_io_usb1_inst_NXT
        .hps_io_hps_io_spim0_inst_CLK      (hps_io_hps_io_spim0_inst_CLK),
//              .hps_io_spim0_inst_CLK
        .hps_io_hps_io_spim0_inst_MOSI     (hps_io_hps_io_spim0_inst_MOSI),
//              .hps_io_spim0_inst_MOSI
        .hps_io_hps_io_spim0_inst_MISO     (hps_io_hps_io_spim0_inst_MISO),
//              .hps_io_spim0_inst_MISO
        .hps_io_hps_io_spim0_inst_SS0      (hps_io_hps_io_spim0_inst_SS0),
//              .hps_io_spim0_inst_SS0
        .hps_io_hps_io_spim1_inst_CLK      (hps_io_hps_io_spim1_inst_CLK),
//              .hps_io_spim1_inst_CLK
        .hps_io_hps_io_spim1_inst_MOSI     (hps_io_hps_io_spim1_inst_MOSI),
//              .hps_io_spim1_inst_MOSI
        .hps_io_hps_io_spim1_inst_MISO     (hps_io_hps_io_spim1_inst_MISO),
//              .hps_io_spim1_inst_MISO
        .hps_io_hps_io_spim1_inst_SS0      (hps_io_hps_io_spim1_inst_SS0),      //        .hps_io_spim1_inst_SS0
        .hps_io_hps_io_uart0_inst_RX       (hps_io_hps_io_uart0_inst_RX),       //        .hps_io_uart0_inst_RX
        .hps_io_hps_io_uart0_inst_TX       (hps_io_hps_io_uart0_inst_TX),       //        .hps_io_uart0_inst_TX
        .hps_io_hps_io_i2c1_inst_SDA       (hps_io_hps_io_i2c1_inst_SDA),       //        .hps_io_i2c1_inst_SDA
        .hps_io_hps_io_i2c1_inst_SCL       (hps_io_hps_io_i2c1_inst_SCL),       //        .hps_io_i2c1_inst_SCL
                .writedata                                      (battle_tank_game_controller_writedata),
//              .writedata
                .write                                          (battle_tank_game_controller_write),
//              .write
                        .chipselect                             (battle_tank_game_controller_chipselect),
//              .chipselect
                        .address                                        (battle_tank_game_controller_address),
                        .reset_out                              (battle_tank_game_controller_reset)
    );

    vga_led_controller vga_led_controller_0 (
        .clk            (OSC_50_B4A),                           //        clock.clk
        .reset          (battle_tank_game_controller_reset),    //  reset_sink.reset
        .VGA_R          (VGA_R),                                //  conduit_end.export
        .VGA_G          (VGA_G),                                //             .export
        .VGA_B          (VGA_B),                                //             .export
        .VGA_CLK        (VGA_CLK),                              //             .export
        .VGA_HS         (VGA_HS),                               //             .export
        .VGA_VS         (VGA_VS),                               //             .export
        .VGA_BLANK_n (VGA_BLANK_n),                             //             .export
        .VGA_SYNC_n   (VGA_SYNC_n),                             //             .export

          /* input signal from vga module */
```

```
        .vcount_vga                     (vcount_vga),
        .hcount_vga                     (hcount_vga),

        /* vga signal */
        .our_life_sub_flag_vga          (our_life_sub_flag_vga),
        .enemy_life_sub_flag_vga        (enemy_life_sub_flag_vga),
        .game_stage_vga                 (game_stage_vga),
        .tank_x_vga                     (tank_x_vga),
        .tank_y_vga                     (tank_y_vga),
        .brick_sub_flag_vga             (brick_sub_flag_vga),
        .bullet_x_vga                   (bullet_x_vga),
        .bullet_y_vga                   (bullet_y_vga),
        .tank1_flag_vga                 (tank1_flag_vga),
        .tank2_flag_vga                 (tank2_flag_vga),
        .tank_explosion_flag_vga        (tank_explosion_flag_vga),
        .tank_generation_flag_vga       (tank_generation_flag_vga),
        .tank1_explosion_flag_vga       (tank1_explosion_flag_vga),
        .tank1_generation_flag_vga      (tank1_generation_flag_vga),
        .tank2_explosion_flag_vga       (tank2_explosion_flag_vga),
        .tank2_generation_flag_vga      (tank2_generation_flag_vga),
        .bullet_flag_vga                (bullet_flag_vga),
        .tank_enemy_flag_vga            (tank_enemy_flag_vga),
        .tank_direction_vga             (tank_direction_vga) //0-up, 1-down, 2-left, 3-right
);

audio audio_0(
                .OSC_50_B4A                     (OSC_50_B4A),
                .AUD_ADCLRCK                    (AUD_ADCLRCK),
                .AUD_ADCDAT                     (AUD_ADCDAT),
                .AUD_DACLRCK                    (AUD_DACLRCK),
                .AUD_DACDAT                     (AUD_DACDAT),
                .AUD_XCK                        (AUD_XCK),
                .AUD_BCLK                       (AUD_BCLK),
                .AUD_I2C_SCLK                   (AUD_I2C_SCLK),
                .AUD_I2C_SDAT                   (AUD_I2C_SDAT),
                .AUD_MUTE                       (AUD_MUTE),
                .tank_explosion_audio           (tank_explosion_audio),
                .tank1_bullet_audio             (tank1_bullet_audio),
                .tank2_bullet_audio             (tank2_bullet_audio),
                .move_audio                     (move_audio),
                .KEY                            (KEY),
                .SW                             (SW)
        );

battle_tank_game_controller game_controller(
                .clk                    (OSC_50_B4A),
                .reset                  (battle_tank_game_controller_reset),
        /* input signal from vga module */
                .vcount_vga             (vcount_vga),
```

```
                .hcount_vga             (hcount_vga),

    /* autio signal */
                .tank_explosion_audio       (tank_explosion_audio),
                .tank1_bullet_audio             (tank1_bullet_audio),
                .tank2_bullet_audio             (tank2_bullet_audio),
                .move_audio                     (move_audio),
            /* vga signal */
                .our_life_sub_flag_vga      (our_life_sub_flag_vga),
                .enemy_life_sub_flag_vga    (enemy_life_sub_flag_vga),
                .game_stage_vga                 (game_stage_vga),
                .tank_x_vga                     (tank_x_vga),
                .tank_y_vga                     (tank_y_vga),
                .brick_sub_flag_vga             (brick_sub_flag_vga),
                .bullet_x_vga                   (bullet_x_vga),
                .bullet_y_vga                   (bullet_y_vga),
                .tank1_flag_vga                 (tank1_flag_vga),
                .tank2_flag_vga                 (tank2_flag_vga),
                .tank_explosion_flag_vga    (tank_explosion_flag_vga),
                .tank_generation_flag_vga   (tank_generation_flag_vga),
                .tank1_explosion_flag_vga   (tank1_explosion_flag_vga),
                .tank1_generation_flag_vga  (tank1_generation_flag_vga),
                .tank2_explosion_flag_vga   (tank2_explosion_flag_vga),
                .tank2_generation_flag_vga  (tank2_generation_flag_vga),
                .bullet_flag_vga                (bullet_flag_vga),
                .tank_enemy_flag_vga        (tank_enemy_flag_vga),
                .tank_direction_vga             (tank_direction_vga) //0-up, 1-down, 2-left, 3-right
            );

Endmodule
```

```
/*
 * Battle Tank HPS top module
 *
 * Team: Battle Tank
 * Columbia University
 */

`timescale 1 ps / 1 ps
module battle_tank (
		input	wire		clk_clk,				//	clk.clk
		input	wire		reset_reset_n,			//	reset.reset_n
		output	wire [14:0]	memory_mem_a,			// memory.mem_a
		output	wire [2:0]	memory_mem_ba,			//		.mem_ba
		output	wire		memory_mem_ck,			//		.mem_ck
		output	wire		memory_mem_ck_n,		//		.mem_ck_n
		output	wire		memory_mem_cke,			//		.mem_cke
		output	wire		memory_mem_cs_n,		//		.mem_cs_n
		output	wire		memory_mem_ras_n,		//		.mem_ras_n
		output	wire		memory_mem_cas_n,		//		.mem_cas_n
		output	wire		memory_mem_we_n,		//		.mem_we_n
		output	wire		memory_mem_reset_n,		//		.mem_reset_n
		inout	wire [31:0]	memory_mem_dq,			//		.mem_dq
		inout	wire [3:0]	memory_mem_dqs,			//		.mem_dqs
		inout	wire [3:0]	memory_mem_dqs_n,		//		.mem_dqs_n
		output	wire		memory_mem_odt,			//		.mem_odt
		output	wire [3:0]	memory_mem_dm,			//		.mem_dm
		input	wire		memory_oct_rzqin,		//		.oct_rzqin
		output	wire		hps_io_hps_io_emac1_inst_TX_CLK, // hps_io.hps_io_emac1_inst_TX_CLK
		output	wire		hps_io_hps_io_emac1_inst_TXD0,	//		.hps_io_emac1_inst_TXD0
		output	wire		hps_io_hps_io_emac1_inst_TXD1,	//		.hps_io_emac1_inst_TXD1
		output	wire		hps_io_hps_io_emac1_inst_TXD2,	//		.hps_io_emac1_inst_TXD2
		output	wire		hps_io_hps_io_emac1_inst_TXD3,	//		.hps_io_emac1_inst_TXD3
		input	wire		hps_io_hps_io_emac1_inst_RXD0,	//		.hps_io_emac1_inst_RXD0
		inout	wire		hps_io_hps_io_emac1_inst_MDIO,	//		.hps_io_emac1_inst_MDIO
		output	wire		hps_io_hps_io_emac1_inst_MDC,	//		.hps_io_emac1_inst_MDC
		input	wire		hps_io_hps_io_emac1_inst_RX_CTL, //		.hps_io_emac1_inst_RX_CTL
		output	wire		hps_io_hps_io_emac1_inst_TX_CTL, //		.hps_io_emac1_inst_TX_CTL
		input	wire		hps_io_hps_io_emac1_inst_RX_CLK, //		.hps_io_emac1_inst_RX_CLK
		input	wire		hps_io_hps_io_emac1_inst_RXD1,	//		.hps_io_emac1_inst_RXD1
		input	wire		hps_io_hps_io_emac1_inst_RXD2,	//		.hps_io_emac1_inst_RXD2
		input	wire		hps_io_hps_io_emac1_inst_RXD3,	//		.hps_io_emac1_inst_RXD3
		inout	wire		hps_io_hps_io_qspi_inst_IO0,	//		.hps_io_qspi_inst_IO0
		inout	wire		hps_io_hps_io_qspi_inst_IO1,	//		.hps_io_qspi_inst_IO1
		inout	wire		hps_io_hps_io_qspi_inst_IO2,	//		.hps_io_qspi_inst_IO2
		inout	wire		hps_io_hps_io_qspi_inst_IO3,	//		.hps_io_qspi_inst_IO3
		output	wire		hps_io_hps_io_qspi_inst_SS0,	//		.hps_io_qspi_inst_SS0
		output	wire		hps_io_hps_io_qspi_inst_CLK,	//		.hps_io_qspi_inst_CLK
		inout	wire		hps_io_hps_io_sdio_inst_CMD,	//		.hps_io_sdio_inst_CMD
		inout	wire		hps_io_hps_io_sdio_inst_D0,	//		.hps_io_sdio_inst_D0
```

```
        inout   wire        hps_io_hps_io_sdio_inst_D1,       //       .hps_io_sdio_inst_D1
        output wire         hps_io_hps_io_sdio_inst_CLK,      //       .hps_io_sdio_inst_CLK
        inout   wire        hps_io_hps_io_sdio_inst_D2,       //       .hps_io_sdio_inst_D2
        inout   wire        hps_io_hps_io_sdio_inst_D3,       //       .hps_io_sdio_inst_D3
        inout   wire        hps_io_hps_io_usb1_inst_D0,       //       .hps_io_usb1_inst_D0
        inout   wire        hps_io_hps_io_usb1_inst_D1,       //       .hps_io_usb1_inst_D1
        inout   wire        hps_io_hps_io_usb1_inst_D2,       //       .hps_io_usb1_inst_D2
        inout   wire        hps_io_hps_io_usb1_inst_D3,       //       .hps_io_usb1_inst_D3
        inout   wire        hps_io_hps_io_usb1_inst_D4,       //       .hps_io_usb1_inst_D4
        inout   wire        hps_io_hps_io_usb1_inst_D5,       //       .hps_io_usb1_inst_D5
        inout   wire        hps_io_hps_io_usb1_inst_D6,       //       .hps_io_usb1_inst_D6
        inout   wire        hps_io_hps_io_usb1_inst_D7,       //       .hps_io_usb1_inst_D7
        input   wire        hps_io_hps_io_usb1_inst_CLK,      //       .hps_io_usb1_inst_CLK
        output wire         hps_io_hps_io_usb1_inst_STP,      //       .hps_io_usb1_inst_STP
        input   wire        hps_io_hps_io_usb1_inst_DIR,      //       .hps_io_usb1_inst_DIR
        input   wire        hps_io_hps_io_usb1_inst_NXT,      //       .hps_io_usb1_inst_NXT
        output wire         hps_io_hps_io_spim0_inst_CLK,     //       .hps_io_spim0_inst_CLK
        output wire         hps_io_hps_io_spim0_inst_MOSI,    //       .hps_io_spim0_inst_MOSI
        input   wire        hps_io_hps_io_spim0_inst_MISO,    //       .hps_io_spim0_inst_MISO
        output wire         hps_io_hps_io_spim0_inst_SS0,     //       .hps_io_spim0_inst_SS0
        output wire         hps_io_hps_io_spim1_inst_CLK,     //       .hps_io_spim1_inst_CLK
        output wire         hps_io_hps_io_spim1_inst_MOSI,    //       .hps_io_spim1_inst_MOSI
        input   wire        hps_io_hps_io_spim1_inst_MISO,    //       .hps_io_spim1_inst_MISO
        output wire         hps_io_hps_io_spim1_inst_SS0,     //       .hps_io_spim1_inst_SS0
        input   wire        hps_io_hps_io_uart0_inst_RX,      //       .hps_io_uart0_inst_RX
        output wire         hps_io_hps_io_uart0_inst_TX,      //       .hps_io_uart0_inst_TX
        inout   wire        hps_io_hps_io_i2c1_inst_SDA,      //       .hps_io_i2c1_inst_SDA
        inout   wire        hps_io_hps_io_i2c1_inst_SCL,      //       .hps_io_i2c1_inst_SCL
        output wire [31:0] writedata,                         //                .writedata       // change avelon bus
#bits
        output wire             write,                        //                .write
        output wire           chipselect,                     //                .chipselect     //change avelon
bus #address
        output wire [6:0]     address,
        output wire                 reset_out
    );

    wire                    hps_0_h2f_lw_axi_master_awvalid;                                    //  hps_0:h2f_lw_AWVALID  ->
mm_interconnect_0:hps_0_h2f_lw_axi_master_awvalid
    wire        [2:0]   hps_0_h2f_lw_axi_master_arsize;                                     //  hps_0:h2f_lw_ARSIZE  ->
mm_interconnect_0:hps_0_h2f_lw_axi_master_arsize
    wire        [1:0]   hps_0_h2f_lw_axi_master_arlock;                                     //  hps_0:h2f_lw_ARLOCK  ->
mm_interconnect_0:hps_0_h2f_lw_axi_master_arlock
    wire        [3:0]   hps_0_h2f_lw_axi_master_awcache;                                    //  hps_0:h2f_lw_AWCACHE  ->
mm_interconnect_0:hps_0_h2f_lw_axi_master_awcache
    wire            hps_0_h2f_lw_axi_master_arready;                                        // mm_interconnect_0:hps_0_h2f_lw_axi_master_arready ->
hps_0:h2f_lw_ARREADY
    wire      [11:0]    hps_0_h2f_lw_axi_master_arid;                                       //  hps_0:h2f_lw_ARID  ->
mm_interconnect_0:hps_0_h2f_lw_axi_master_arid
```

```
        wire                    hps_0_h2f_lw_axi_master_rready;                                 //  hps_0:h2f_lw_RREADY   -> mm_interconnect_0:hps_0_h2f_lw_axi_master_rready
        wire                    hps_0_h2f_lw_axi_master_bready;                                 //  hps_0:h2f_lw_BREADY   -> mm_interconnect_0:hps_0_h2f_lw_axi_master_bready
        wire        [2:0]   hps_0_h2f_lw_axi_master_awsize;                                 //  hps_0:h2f_lw_AWSIZE   -> mm_interconnect_0:hps_0_h2f_lw_axi_master_awsize
        wire        [2:0]   hps_0_h2f_lw_axi_master_awprot;                                 //  hps_0:h2f_lw_AWPROT   -> mm_interconnect_0:hps_0_h2f_lw_axi_master_awprot
        wire                    hps_0_h2f_lw_axi_master_arvalid;                                //  hps_0:h2f_lw_ARVALID  -> mm_interconnect_0:hps_0_h2f_lw_axi_master_arvalid
        wire        [2:0]   hps_0_h2f_lw_axi_master_arprot;                                 //  hps_0:h2f_lw_ARPROT   -> mm_interconnect_0:hps_0_h2f_lw_axi_master_arprot
        wire   [11:0] hps_0_h2f_lw_axi_master_bid;                                  //  mm_interconnect_0:hps_0_h2f_lw_axi_master_bid -> hps_0:h2f_lw_BID
        wire        [3:0]   hps_0_h2f_lw_axi_master_arlen;                                  //  hps_0:h2f_lw_ARLEN    -> mm_interconnect_0:hps_0_h2f_lw_axi_master_arlen
      wire            hps_0_h2f_lw_axi_master_awready;                            //  mm_interconnect_0:hps_0_h2f_lw_axi_master_awready -> hps_0:h2f_lw_AWREADY
        wire      [11:0]    hps_0_h2f_lw_axi_master_awid;                                   //  hps_0:h2f_lw_AWID     -> mm_interconnect_0:hps_0_h2f_lw_axi_master_awid
        wire              hps_0_h2f_lw_axi_master_bvalid;                             //  mm_interconnect_0:hps_0_h2f_lw_axi_master_bvalid -> hps_0:h2f_lw_BVALID
        wire      [11:0]    hps_0_h2f_lw_axi_master_wid;                                    //  hps_0:h2f_lw_WID      -> mm_interconnect_0:hps_0_h2f_lw_axi_master_wid
        wire        [1:0]   hps_0_h2f_lw_axi_master_awlock;                                 //  hps_0:h2f_lw_AWLOCK   -> mm_interconnect_0:hps_0_h2f_lw_axi_master_awlock
        wire        [1:0]   hps_0_h2f_lw_axi_master_awburst;                                //  hps_0:h2f_lw_AWBURST  -> mm_interconnect_0:hps_0_h2f_lw_axi_master_awburst
        wire     [1:0] hps_0_h2f_lw_axi_master_bresp;                                //  mm_interconnect_0:hps_0_h2f_lw_axi_master_bresp -> hps_0:h2f_lw_BRESP
        wire        [3:0]   hps_0_h2f_lw_axi_master_wstrb;                                  //  hps_0:h2f_lw_WSTRB    -> mm_interconnect_0:hps_0_h2f_lw_axi_master_wstrb
        wire              hps_0_h2f_lw_axi_master_rvalid;                             //  mm_interconnect_0:hps_0_h2f_lw_axi_master_rvalid -> hps_0:h2f_lw_RVALID
        wire      [31:0]    hps_0_h2f_lw_axi_master_wdata;                                  //  hps_0:h2f_lw_WDATA    -> mm_interconnect_0:hps_0_h2f_lw_axi_master_wdata
        wire            hps_0_h2f_lw_axi_master_wready;                             //  mm_interconnect_0:hps_0_h2f_lw_axi_master_wready -> hps_0:h2f_lw_WREADY
        wire        [1:0]   hps_0_h2f_lw_axi_master_arburst;                                //  hps_0:h2f_lw_ARBURST  -> mm_interconnect_0:hps_0_h2f_lw_axi_master_arburst
        wire   [31:0] hps_0_h2f_lw_axi_master_rdata;                                //  mm_interconnect_0:hps_0_h2f_lw_axi_master_rdata -> hps_0:h2f_lw_RDATA
        wire      [20:0]    hps_0_h2f_lw_axi_master_araddr;                                 //  hps_0:h2f_lw_ARADDR   -> mm_interconnect_0:hps_0_h2f_lw_axi_master_araddr
        wire        [3:0]   hps_0_h2f_lw_axi_master_arcache;                                //  hps_0:h2f_lw_ARCACHE  -> mm_interconnect_0:hps_0_h2f_lw_axi_master_arcache
        wire        [3:0]   hps_0_h2f_lw_axi_master_awlen;                                  //  hps_0:h2f_lw_AWLEN    -> mm_interconnect_0:hps_0_h2f_lw_axi_master_awlen
        wire      [20:0]    hps_0_h2f_lw_axi_master_awaddr;                                 //  hps_0:h2f_lw_AWADDR   ->
```

```
mm_interconnect_0:hps_0_h2f_lw_axi_master_awaddr
        wire    [11:0]  hps_0_h2f_lw_axi_master_rid;                    // mm_interconnect_0:hps_0_h2f_lw_axi_master_rid -> hps_0:h2f_lw_RID
        wire            hps_0_h2f_lw_axi_master_wvalid;                 // hps_0:h2f_lw_WVALID -> mm_interconnect_0:hps_0_h2f_lw_axi_master_wvalid
        wire    [1:0]   hps_0_h2f_lw_axi_master_rresp;                  // mm_interconnect_0:hps_0_h2f_lw_axi_master_rresp -> hps_0:h2f_lw_RRESP
        wire            hps_0_h2f_lw_axi_master_wlast;                  // hps_0:h2f_lw_WLAST -> mm_interconnect_0:hps_0_h2f_lw_axi_master_wlast
        wire            hps_0_h2f_lw_axi_master_rlast;                  // mm_interconnect_0:hps_0_h2f_lw_axi_master_rlast -> hps_0:h2f_lw_RLAST
        wire            master_0_master_waitrequest;                    // mm_interconnect_0:master_0_master_waitrequest -> master_0:master_waitrequest
        wire    [31:0]  master_0_master_writedata;                      // master_0:master_writedata -> mm_interconnect_0:master_0_master_writedata
        wire    [31:0]  master_0_master_address;                        // master_0:master_address -> mm_interconnect_0:master_0_master_address
        wire            master_0_master_write;                          // master_0:master_write -> mm_interconnect_0:master_0_master_write
        wire            master_0_master_read;                           // master_0:master_read -> mm_interconnect_0:master_0_master_read
        wire    [31:0]  master_0_master_readdata;                       // mm_interconnect_0:master_0_master_readdata -> master_0:master_readdata
        wire    [3:0]   master_0_master_byteenable;                     // master_0:master_byteenable -> mm_interconnect_0:master_0_master_byteenable
        wire            master_0_master_readdatavalid;                  // mm_interconnect_0:master_0_master_readdatavalid -> master_0:master_readdatavalid
        wire    [31:0] hps_0_f2h_irq0_irq;                              // irq_mapper:sender_irq -> hps_0:f2h_irq_p0
        wire    [31:0] hps_0_f2h_irq1_irq;                              // irq_mapper_001:sender_irq -> hps_0:f2h_irq_p1
        wire            rst_controller_reset_out_reset;                 // rst_controller:reset_out -> [mm_interconnect_0:master_0_clk_reset_reset_bridge_in_reset_reset,
mm_interconnect_0:battle_tank_game_controller_reset_sink_reset_bridge_in_reset_reset, battle_tank_game_controller:reset]
        wire            rst_controller_001_reset_out_reset;             // rst_controller_001:reset_out -> mm_interconnect_0:hps_0_h2f_lw_axi_master_agent_clk_reset_reset_bridge_in_reset_reset
        wire            hps_0_h2f_reset_reset;                          // hps_0:h2f_rst_n -> rst_controller_001:reset_in0

        assign          reset_out = rst_controller_reset_out_reset;

        battle_tank_hps_0 #(
                .F2S_Width (2),
                .S2F_Width (2)
        ) hps_0 (
                .mem_a          (memory_mem_a),         //      memory.mem_a
                .mem_ba         (memory_mem_ba),        //            .mem_ba
                .mem_ck         (memory_mem_ck),        //            .mem_ck
                .mem_ck_n       (memory_mem_ck_n),      //            .mem_ck_n
                .mem_cke        (memory_mem_cke),       //            .mem_cke
                .mem_cs_n       (memory_mem_cs_n),      //            .mem_cs_n
                .mem_ras_n      (memory_mem_ras_n),     //            .mem_ras_n
```

```
.mem_cas_n                    (memory_mem_cas_n),                    //                    .mem_cas_n
.mem_we_n                     (memory_mem_we_n),                     //                    .mem_we_n
.mem_reset_n                  (memory_mem_reset_n),                  //                    .mem_reset_n
.mem_dq                       (memory_mem_dq),                       //                    .mem_dq
.mem_dqs                      (memory_mem_dqs),                      //                    .mem_dqs
.mem_dqs_n                    (memory_mem_dqs_n),                    //                    .mem_dqs_n
.mem_odt                      (memory_mem_odt),                      //                    .mem_odt
.mem_dm                       (memory_mem_dm),                       //                    .mem_dm
.oct_rzqin                    (memory_oct_rzqin),                    //                    .oct_rzqin
.hps_io_emac1_inst_TX_CLK (hps_io_hps_io_emac1_inst_TX_CLK), //          hps_io.hps_io_emac1_inst_TX_CLK
.hps_io_emac1_inst_TXD0     (hps_io_hps_io_emac1_inst_TXD0),     //              .hps_io_emac1_inst_TXD0
.hps_io_emac1_inst_TXD1     (hps_io_hps_io_emac1_inst_TXD1),     //              .hps_io_emac1_inst_TXD1
.hps_io_emac1_inst_TXD2     (hps_io_hps_io_emac1_inst_TXD2),     //              .hps_io_emac1_inst_TXD2
.hps_io_emac1_inst_TXD3     (hps_io_hps_io_emac1_inst_TXD3),     //              .hps_io_emac1_inst_TXD3
.hps_io_emac1_inst_RXD0     (hps_io_hps_io_emac1_inst_RXD0),     //              .hps_io_emac1_inst_RXD0
.hps_io_emac1_inst_MDIO     (hps_io_hps_io_emac1_inst_MDIO),     //              .hps_io_emac1_inst_MDIO
.hps_io_emac1_inst_MDC      (hps_io_hps_io_emac1_inst_MDC),      //              .hps_io_emac1_inst_MDC
.hps_io_emac1_inst_RX_CTL (hps_io_hps_io_emac1_inst_RX_CTL), //              .hps_io_emac1_inst_RX_CTL
.hps_io_emac1_inst_TX_CTL (hps_io_hps_io_emac1_inst_TX_CTL), //              .hps_io_emac1_inst_TX_CTL
.hps_io_emac1_inst_RX_CLK (hps_io_hps_io_emac1_inst_RX_CLK), //              .hps_io_emac1_inst_RX_CLK
.hps_io_emac1_inst_RXD1     (hps_io_hps_io_emac1_inst_RXD1),     //              .hps_io_emac1_inst_RXD1
.hps_io_emac1_inst_RXD2     (hps_io_hps_io_emac1_inst_RXD2),     //              .hps_io_emac1_inst_RXD2
.hps_io_emac1_inst_RXD3     (hps_io_hps_io_emac1_inst_RXD3),     //              .hps_io_emac1_inst_RXD3
.hps_io_qspi_inst_IO0       (hps_io_hps_io_qspi_inst_IO0),       //              .hps_io_qspi_inst_IO0
.hps_io_qspi_inst_IO1       (hps_io_hps_io_qspi_inst_IO1),       //              .hps_io_qspi_inst_IO1
.hps_io_qspi_inst_IO2       (hps_io_hps_io_qspi_inst_IO2),       //              .hps_io_qspi_inst_IO2
.hps_io_qspi_inst_IO3       (hps_io_hps_io_qspi_inst_IO3),       //              .hps_io_qspi_inst_IO3
.hps_io_qspi_inst_SS0       (hps_io_hps_io_qspi_inst_SS0),       //              .hps_io_qspi_inst_SS0
.hps_io_qspi_inst_CLK       (hps_io_hps_io_qspi_inst_CLK),       //              .hps_io_qspi_inst_CLK
.hps_io_sdio_inst_CMD       (hps_io_hps_io_sdio_inst_CMD),       //              .hps_io_sdio_inst_CMD
.hps_io_sdio_inst_D0        (hps_io_hps_io_sdio_inst_D0),        //              .hps_io_sdio_inst_D0
.hps_io_sdio_inst_D1        (hps_io_hps_io_sdio_inst_D1),        //              .hps_io_sdio_inst_D1
.hps_io_sdio_inst_CLK       (hps_io_hps_io_sdio_inst_CLK),       //              .hps_io_sdio_inst_CLK
.hps_io_sdio_inst_D2        (hps_io_hps_io_sdio_inst_D2),        //              .hps_io_sdio_inst_D2
.hps_io_sdio_inst_D3        (hps_io_hps_io_sdio_inst_D3),        //              .hps_io_sdio_inst_D3
.hps_io_usb1_inst_D0        (hps_io_hps_io_usb1_inst_D0),        //              .hps_io_usb1_inst_D0
.hps_io_usb1_inst_D1        (hps_io_hps_io_usb1_inst_D1),        //              .hps_io_usb1_inst_D1
.hps_io_usb1_inst_D2        (hps_io_hps_io_usb1_inst_D2),        //              .hps_io_usb1_inst_D2
.hps_io_usb1_inst_D3        (hps_io_hps_io_usb1_inst_D3),        //              .hps_io_usb1_inst_D3
.hps_io_usb1_inst_D4        (hps_io_hps_io_usb1_inst_D4),        //              .hps_io_usb1_inst_D4
.hps_io_usb1_inst_D5        (hps_io_hps_io_usb1_inst_D5),        //              .hps_io_usb1_inst_D5
.hps_io_usb1_inst_D6        (hps_io_hps_io_usb1_inst_D6),        //              .hps_io_usb1_inst_D6
.hps_io_usb1_inst_D7        (hps_io_hps_io_usb1_inst_D7),        //              .hps_io_usb1_inst_D7
.hps_io_usb1_inst_CLK       (hps_io_hps_io_usb1_inst_CLK),       //              .hps_io_usb1_inst_CLK
.hps_io_usb1_inst_STP       (hps_io_hps_io_usb1_inst_STP),       //              .hps_io_usb1_inst_STP
.hps_io_usb1_inst_DIR       (hps_io_hps_io_usb1_inst_DIR),       //              .hps_io_usb1_inst_DIR
.hps_io_usb1_inst_NXT       (hps_io_hps_io_usb1_inst_NXT),       //              .hps_io_usb1_inst_NXT
.hps_io_spim0_inst_CLK      (hps_io_hps_io_spim0_inst_CLK),      //              .hps_io_spim0_inst_CLK
.hps_io_spim0_inst_MOSI     (hps_io_hps_io_spim0_inst_MOSI),     //              .hps_io_spim0_inst_MOSI
```

```
.hps_io_spim0_inst_MISO  (hps_io_hps_io_spim0_inst_MISO),  //  .hps_io_spim0_inst_MISO
.hps_io_spim0_inst_SS0  (hps_io_hps_io_spim0_inst_SS0),  //  .hps_io_spim0_inst_SS0
.hps_io_spim1_inst_CLK  (hps_io_hps_io_spim1_inst_CLK),  //  .hps_io_spim1_inst_CLK
.hps_io_spim1_inst_MOSI  (hps_io_hps_io_spim1_inst_MOSI),  //  .hps_io_spim1_inst_MOSI
.hps_io_spim1_inst_MISO  (hps_io_hps_io_spim1_inst_MISO),  //  .hps_io_spim1_inst_MISO
.hps_io_spim1_inst_SS0  (hps_io_hps_io_spim1_inst_SS0),  //  .hps_io_spim1_inst_SS0
.hps_io_uart0_inst_RX  (hps_io_hps_io_uart0_inst_RX),  //  .hps_io_uart0_inst_RX
.hps_io_uart0_inst_TX  (hps_io_hps_io_uart0_inst_TX),  //  .hps_io_uart0_inst_TX
.hps_io_i2c1_inst_SDA  (hps_io_hps_io_i2c1_inst_SDA),  //  .hps_io_i2c1_inst_SDA
.hps_io_i2c1_inst_SCL  (hps_io_hps_io_i2c1_inst_SCL),  //  .hps_io_i2c1_inst_SCL
.h2f_rst_n  (hps_0_h2f_reset_reset),  //  h2f_reset.reset_n
.h2f_axi_clk  (clk_clk),  //  h2f_axi_clock.clk
.h2f_AWID  (),  //  h2f_axi_master.awid
.h2f_AWADDR  (),  //  .awaddr
.h2f_AWLEN  (),  //  .awlen
.h2f_AWSIZE  (),  //  .awsize
.h2f_AWBURST  (),  //  .awburst
.h2f_AWLOCK  (),  //  .awlock
.h2f_AWCACHE  (),  //  .awcache
.h2f_AWPROT  (),  //  .awprot
.h2f_AWVALID  (),  //  .awvalid
.h2f_AWREADY  (),  //  .awready
.h2f_WID  (),  //  .wid
.h2f_WDATA  (),  //  .wdata
.h2f_WSTRB  (),  //  .wstrb
.h2f_WLAST  (),  //  .wlast
.h2f_WVALID  (),  //  .wvalid
.h2f_WREADY  (),  //  .wready
.h2f_BID  (),  //  .bid
.h2f_BRESP  (),  //  .bresp
.h2f_BVALID  (),  //  .bvalid
.h2f_BREADY  (),  //  .bready
.h2f_ARID  (),  //  .arid
.h2f_ARADDR  (),  //  .araddr
.h2f_ARLEN  (),  //  .arlen
.h2f_ARSIZE  (),  //  .arsize
.h2f_ARBURST  (),  //  .arburst
.h2f_ARLOCK  (),  //  .arlock
.h2f_ARCACHE  (),  //  .arcache
.h2f_ARPROT  (),  //  .arprot
.h2f_ARVALID  (),  //  .arvalid
.h2f_ARREADY  (),  //  .arready
.h2f_RID  (),  //  .rid
.h2f_RDATA  (),  //  .rdata
.h2f_RRESP  (),  //  .rresp
.h2f_RLAST  (),  //  .rlast
.h2f_RVALID  (),  //  .rvalid
.h2f_RREADY  (),  //  .rready
.f2h_axi_clk  (clk_clk),  //  f2h_axi_clock.clk
```

```
.f2h_AWID        (),                          //  f2h_axi_slave.awid
.f2h_AWADDR      (),                          //               .awaddr
.f2h_AWLEN       (),                          //               .awlen
.f2h_AWSIZE      (),                          //               .awsize
.f2h_AWBURST     (),                          //               .awburst
.f2h_AWLOCK      (),                          //               .awlock
.f2h_AWCACHE     (),                          //               .awcache
.f2h_AWPROT      (),                          //               .awprot
.f2h_AWVALID     (),                          //               .awvalid
.f2h_AWREADY     (),                          //               .awready
.f2h_AWUSER      (),                          //               .awuser
.f2h_WID         (),                          //               .wid
.f2h_WDATA       (),                          //               .wdata
.f2h_WSTRB       (),                          //               .wstrb
.f2h_WLAST       (),                          //               .wlast
.f2h_WVALID      (),                          //               .wvalid
.f2h_WREADY      (),                          //               .wready
.f2h_BID         (),                          //               .bid
.f2h_BRESP       (),                          //               .bresp
.f2h_BVALID      (),                          //               .bvalid
.f2h_BREADY      (),                          //               .bready
.f2h_ARID        (),                          //               .arid
.f2h_ARADDR      (),                          //               .araddr
.f2h_ARLEN       (),                          //               .arlen
.f2h_ARSIZE      (),                          //               .arsize
.f2h_ARBURST     (),                          //               .arburst
.f2h_ARLOCK      (),                          //               .arlock
.f2h_ARCACHE     (),                          //               .arcache
.f2h_ARPROT      (),                          //               .arprot
.f2h_ARVALID     (),                          //               .arvalid
.f2h_ARREADY     (),                          //               .arready
.f2h_ARUSER      (),                          //               .aruser
.f2h_RID         (),                          //               .rid
.f2h_RDATA       (),                          //               .rdata
.f2h_RRESP       (),                          //               .rresp
.f2h_RLAST       (),                          //               .rlast
.f2h_RVALID      (),                          //               .rvalid
.f2h_RREADY      (),                          //               .rready
.h2f_lw_axi_clk  (clk_clk),                   //  h2f_lw_axi_clock.clk
.h2f_lw_AWID     (hps_0_h2f_lw_axi_master_awid),     // h2f_lw_axi_master.awid
.h2f_lw_AWADDR   (hps_0_h2f_lw_axi_master_awaddr),   //               .awaddr
.h2f_lw_AWLEN    (hps_0_h2f_lw_axi_master_awlen),    //               .awlen
.h2f_lw_AWSIZE   (hps_0_h2f_lw_axi_master_awsize),   //               .awsize
.h2f_lw_AWBURST  (hps_0_h2f_lw_axi_master_awburst),  //               .awburst
.h2f_lw_AWLOCK   (hps_0_h2f_lw_axi_master_awlock),   //               .awlock
.h2f_lw_AWCACHE  (hps_0_h2f_lw_axi_master_awcache),  //               .awcache
.h2f_lw_AWPROT   (hps_0_h2f_lw_axi_master_awprot),   //               .awprot
.h2f_lw_AWVALID  (hps_0_h2f_lw_axi_master_awvalid),  //               .awvalid
.h2f_lw_AWREADY  (hps_0_h2f_lw_axi_master_awready),  //               .awready
```

```
        .h2f_lw_WID                 (hps_0_h2f_lw_axi_master_wid),       //                  .wid
        .h2f_lw_WDATA               (hps_0_h2f_lw_axi_master_wdata),     //                  .wdata
        .h2f_lw_WSTRB               (hps_0_h2f_lw_axi_master_wstrb),     //                  .wstrb
        .h2f_lw_WLAST               (hps_0_h2f_lw_axi_master_wlast),     //                  .wlast
        .h2f_lw_WVALID              (hps_0_h2f_lw_axi_master_wvalid),    //                  .wvalid
        .h2f_lw_WREADY              (hps_0_h2f_lw_axi_master_wready),    //                  .wready
        .h2f_lw_BID                 (hps_0_h2f_lw_axi_master_bid),       //                  .bid
        .h2f_lw_BRESP               (hps_0_h2f_lw_axi_master_bresp),     //                  .bresp
        .h2f_lw_BVALID              (hps_0_h2f_lw_axi_master_bvalid),    //                  .bvalid
        .h2f_lw_BREADY              (hps_0_h2f_lw_axi_master_bready),    //                  .bready
        .h2f_lw_ARID                (hps_0_h2f_lw_axi_master_arid),      //                  .arid
        .h2f_lw_ARADDR              (hps_0_h2f_lw_axi_master_araddr),    //                  .araddr
        .h2f_lw_ARLEN               (hps_0_h2f_lw_axi_master_arlen),     //                  .arlen
        .h2f_lw_ARSIZE              (hps_0_h2f_lw_axi_master_arsize),    //                  .arsize
        .h2f_lw_ARBURST             (hps_0_h2f_lw_axi_master_arburst),   //                  .arburst
        .h2f_lw_ARLOCK              (hps_0_h2f_lw_axi_master_arlock),    //                  .arlock
        .h2f_lw_ARCACHE             (hps_0_h2f_lw_axi_master_arcache),   //                  .arcache
        .h2f_lw_ARPROT              (hps_0_h2f_lw_axi_master_arprot),    //                  .arprot
        .h2f_lw_ARVALID             (hps_0_h2f_lw_axi_master_arvalid),   //                  .arvalid
        .h2f_lw_ARREADY             (hps_0_h2f_lw_axi_master_arready),   //                  .arready
        .h2f_lw_RID                 (hps_0_h2f_lw_axi_master_rid),       //                  .rid
        .h2f_lw_RDATA               (hps_0_h2f_lw_axi_master_rdata),     //                  .rdata
        .h2f_lw_RRESP               (hps_0_h2f_lw_axi_master_rresp),     //                  .rresp
        .h2f_lw_RLAST               (hps_0_h2f_lw_axi_master_rlast),     //                  .rlast
        .h2f_lw_RVALID              (hps_0_h2f_lw_axi_master_rvalid),    //                  .rvalid
        .h2f_lw_RREADY              (hps_0_h2f_lw_axi_master_rready),    //                  .rready
        .f2h_irq_p0                 (hps_0_f2h_irq0_irq),                //          f2h_irq0.irq
        .f2h_irq_p1                 (hps_0_f2h_irq1_irq)                 //          f2h_irq1.irq
);

battle_tank_master_0 #(
        .USE_PLI        (0),
        .PLI_PORT       (50000),
        .FIFO_DEPTHS (2)
) master_0 (
        .clk_clk                (clk_clk),                          //              clk.clk
        .clk_reset_reset        (~reset_reset_n),                   //      clk_reset.reset
        .master_address         (master_0_master_address),          //          master.address
        .master_readdata        (master_0_master_readdata),         //                .readdata
        .master_read            (master_0_master_read),             //                .read
        .master_write           (master_0_master_write),            //                .write
        .master_writedata       (master_0_master_writedata),        //                .writedata
        .master_waitrequest     (master_0_master_waitrequest),      //                .waitrequest
        .master_readdatavalid (master_0_master_readdatavalid), //                .readdatavalid
        .master_byteenable      (master_0_master_byteenable),       //                .byteenable
        .master_reset_reset     ()                                  // master_reset.reset
);

battle_tank_mm_interconnect_0 mm_interconnect_0 (
```

```
        .hps_0_h2f_lw_axi_master_awid                    (hps_0_h2f_lw_axi_master_awid),
//                                  hps_0_h2f_lw_axi_master.awid
        .hps_0_h2f_lw_axi_master_awaddr                  (hps_0_h2f_lw_axi_master_awaddr),
//                                                          .awaddr
        .hps_0_h2f_lw_axi_master_awlen                   (hps_0_h2f_lw_axi_master_awlen),
//                                                          .awlen
        .hps_0_h2f_lw_axi_master_awsize                  (hps_0_h2f_lw_axi_master_awsize),
//                                                          .awsize
        .hps_0_h2f_lw_axi_master_awburst                 (hps_0_h2f_lw_axi_master_awburst),
//                                                          .awburst
        .hps_0_h2f_lw_axi_master_awlock                  (hps_0_h2f_lw_axi_master_awlock),
//                                                          .awlock
        .hps_0_h2f_lw_axi_master_awcache                 (hps_0_h2f_lw_axi_master_awcache),
//                                                          .awcache
        .hps_0_h2f_lw_axi_master_awprot                  (hps_0_h2f_lw_axi_master_awprot),
//                                                          .awprot
        .hps_0_h2f_lw_axi_master_awvalid                 (hps_0_h2f_lw_axi_master_awvalid),
//                                                          .awvalid
        .hps_0_h2f_lw_axi_master_awready                 (hps_0_h2f_lw_axi_master_awready),
//                                                          .awready
        .hps_0_h2f_lw_axi_master_wid                     (hps_0_h2f_lw_axi_master_wid),
//                                                          .wid
        .hps_0_h2f_lw_axi_master_wdata                   (hps_0_h2f_lw_axi_master_wdata),
//                                                          .wdata
        .hps_0_h2f_lw_axi_master_wstrb                   (hps_0_h2f_lw_axi_master_wstrb),
//                                                          .wstrb
        .hps_0_h2f_lw_axi_master_wlast                   (hps_0_h2f_lw_axi_master_wlast),
//                                                          .wlast
        .hps_0_h2f_lw_axi_master_wvalid                  (hps_0_h2f_lw_axi_master_wvalid),
//                                                          .wvalid
        .hps_0_h2f_lw_axi_master_wready                  (hps_0_h2f_lw_axi_master_wready),
//                                                          .wready
        .hps_0_h2f_lw_axi_master_bid                     (hps_0_h2f_lw_axi_master_bid),
//                                                          .bid
        .hps_0_h2f_lw_axi_master_bresp                   (hps_0_h2f_lw_axi_master_bresp),
//                                                          .bresp
        .hps_0_h2f_lw_axi_master_bvalid                  (hps_0_h2f_lw_axi_master_bvalid),
//                                                          .bvalid
        .hps_0_h2f_lw_axi_master_bready                  (hps_0_h2f_lw_axi_master_bready),
//                                                          .bready
        .hps_0_h2f_lw_axi_master_arid                    (hps_0_h2f_lw_axi_master_arid),
//                                                          .arid
        .hps_0_h2f_lw_axi_master_araddr                  (hps_0_h2f_lw_axi_master_araddr),
//                                                          .araddr
        .hps_0_h2f_lw_axi_master_arlen                   (hps_0_h2f_lw_axi_master_arlen),
//                                                          .arlen
        .hps_0_h2f_lw_axi_master_arsize                  (hps_0_h2f_lw_axi_master_arsize),
//                                                          .arsize
        .hps_0_h2f_lw_axi_master_arburst                 (hps_0_h2f_lw_axi_master_arburst),
```

```
//                                                                .arburst
        .hps_0_h2f_lw_axi_master_arlock                                 (hps_0_h2f_lw_axi_master_arlock),
//                                                                .arlock
        .hps_0_h2f_lw_axi_master_arcache                                (hps_0_h2f_lw_axi_master_arcache),
//                                                                .arcache
        .hps_0_h2f_lw_axi_master_arprot                                 (hps_0_h2f_lw_axi_master_arprot),
//                                                                .arprot
        .hps_0_h2f_lw_axi_master_arvalid                                (hps_0_h2f_lw_axi_master_arvalid),
//                                                                .arvalid
        .hps_0_h2f_lw_axi_master_arready                                (hps_0_h2f_lw_axi_master_arready),
//                                                                .arready
        .hps_0_h2f_lw_axi_master_rid                                    (hps_0_h2f_lw_axi_master_rid),
//                                                                .rid
        .hps_0_h2f_lw_axi_master_rdata                                  (hps_0_h2f_lw_axi_master_rdata),
//                                                                .rdata
        .hps_0_h2f_lw_axi_master_rresp                                  (hps_0_h2f_lw_axi_master_rresp),
//                                                                .rresp
        .hps_0_h2f_lw_axi_master_rlast                                  (hps_0_h2f_lw_axi_master_rlast),
//                                                                .rlast
        .hps_0_h2f_lw_axi_master_rvalid                                 (hps_0_h2f_lw_axi_master_rvalid),
//                                                                .rvalid
        .hps_0_h2f_lw_axi_master_rready                                 (hps_0_h2f_lw_axi_master_rready),
//                                                                .rready
        .clk_0_clk_clk                                                  (clk_clk),
//                                                       clk_0_clk.clk
        .hps_0_h2f_lw_axi_master_agent_clk_reset_reset_bridge_in_reset_reset    (rst_controller_001_reset_out_reset),
// hps_0_h2f_lw_axi_master_agent_clk_reset_reset_bridge_in_reset.reset
        .master_0_clk_reset_reset_bridge_in_reset_reset                 (rst_controller_reset_out_reset),
//                         master_0_clk_reset_reset_bridge_in_reset.reset
        .battle_tank_game_controller_reset_sink_reset_bridge_in_reset_reset    (rst_controller_reset_out_reset),
//                               battle_tank_game_controller_reset_sink_reset_bridge_in_reset.reset
        .master_0_master_address                                        (master_0_master_address),
//                                                        master_0_master.address
        .master_0_master_waitrequest                                    (master_0_master_waitrequest),
//                                                                .waitrequest
        .master_0_master_byteenable                                     (master_0_master_byteenable),
//                                                                .byteenable
        .master_0_master_read                                           (master_0_master_read),
//                                                                .read
        .master_0_master_readdata                                       (master_0_master_readdata),
//                                                                .readdata
        .master_0_master_readdatavalid                                  (master_0_master_readdatavalid),
//                                                                .readdatavalid
        .master_0_master_write                                          (master_0_master_write),
//                                                                .write
        .master_0_master_writedata                                      (master_0_master_writedata),
//                                                                .writedata
        .battle_tank_game_controller_avalon_slave_0_address             (address),      //
battle_tank_game_controller_avalon_slave_0.address
```

```
        .battle_tank_game_controller_avalon_slave_0_write                                   (write),
//                                                                                          .write
        .battle_tank_game_controller_avalon_slave_0_writedata                               (writedata),
//                                                                                          .writedata
        .battle_tank_game_controller_avalon_slave_0_chipselect                              (chipselect)
//                                                                                          .chipselect
    );

    battle_tank_irq_mapper irq_mapper (
        .clk           (),                          //       clk.clk
        .reset         (),                          // clk_reset.reset
        .sender_irq (hps_0_f2h_irq0_irq)   //    sender.irq
    );

    battle_tank_irq_mapper irq_mapper_001 (
        .clk           (),                          //       clk.clk
        .reset         (),                          // clk_reset.reset
        .sender_irq (hps_0_f2h_irq1_irq)   //    sender.irq
    );

    altera_reset_controller #(
        .NUM_RESET_INPUTS              (1),
        .OUTPUT_RESET_SYNC_EDGES    ("deassert"),
        .SYNC_DEPTH                    (2),
        .RESET_REQUEST_PRESENT      (0),
        .RESET_REQ_WAIT_TIME        (1),
        .MIN_RST_ASSERTION_TIME     (3),
        .RESET_REQ_EARLY_DSRT_TIME (1),
        .USE_RESET_REQUEST_IN0      (0),
        .USE_RESET_REQUEST_IN1      (0),
        .USE_RESET_REQUEST_IN2      (0),
        .USE_RESET_REQUEST_IN3      (0),
        .USE_RESET_REQUEST_IN4      (0),
        .USE_RESET_REQUEST_IN5      (0),
        .USE_RESET_REQUEST_IN6      (0),
        .USE_RESET_REQUEST_IN7      (0),
        .USE_RESET_REQUEST_IN8      (0),
        .USE_RESET_REQUEST_IN9      (0),
        .USE_RESET_REQUEST_IN10     (0),
        .USE_RESET_REQUEST_IN11     (0),
        .USE_RESET_REQUEST_IN12     (0),
        .USE_RESET_REQUEST_IN13     (0),
        .USE_RESET_REQUEST_IN14     (0),
        .USE_RESET_REQUEST_IN15     (0),
        .ADAPT_RESET_REQUEST        (0)
    ) rst_controller (
        .reset_in0      (~reset_reset_n),                   // reset_in0.reset
        .clk            (clk_clk),                          //       clk.clk
        .reset_out      (rst_controller_reset_out_reset), // reset_out.reset
```

```
        .reset_req         (),                          // (terminated)
        .reset_req_in0     (1'b0),                      // (terminated)
        .reset_in1         (1'b0),                      // (terminated)
        .reset_req_in1     (1'b0),                      // (terminated)
        .reset_in2         (1'b0),                      // (terminated)
        .reset_req_in2     (1'b0),                      // (terminated)
        .reset_in3         (1'b0),                      // (terminated)
        .reset_req_in3     (1'b0),                      // (terminated)
        .reset_in4         (1'b0),                      // (terminated)
        .reset_req_in4     (1'b0),                      // (terminated)
        .reset_in5         (1'b0),                      // (terminated)
        .reset_req_in5     (1'b0),                      // (terminated)
        .reset_in6         (1'b0),                      // (terminated)
        .reset_req_in6     (1'b0),                      // (terminated)
        .reset_in7         (1'b0),                      // (terminated)
        .reset_req_in7     (1'b0),                      // (terminated)
        .reset_in8         (1'b0),                      // (terminated)
        .reset_req_in8     (1'b0),                      // (terminated)
        .reset_in9         (1'b0),                      // (terminated)
        .reset_req_in9     (1'b0),                      // (terminated)
        .reset_in10        (1'b0),                      // (terminated)
        .reset_req_in10    (1'b0),                      // (terminated)
        .reset_in11        (1'b0),                      // (terminated)
        .reset_req_in11    (1'b0),                      // (terminated)
        .reset_in12        (1'b0),                      // (terminated)
        .reset_req_in12    (1'b0),                      // (terminated)
        .reset_in13        (1'b0),                      // (terminated)
        .reset_req_in13    (1'b0),                      // (terminated)
        .reset_in14        (1'b0),                      // (terminated)
        .reset_req_in14    (1'b0),                      // (terminated)
        .reset_in15        (1'b0),                      // (terminated)
        .reset_req_in15    (1'b0)                       // (terminated)
);

altera_reset_controller #(
        .NUM_RESET_INPUTS              (1),
        .OUTPUT_RESET_SYNC_EDGES       ("deassert"),
        .SYNC_DEPTH                    (2),
        .RESET_REQUEST_PRESENT         (0),
        .RESET_REQ_WAIT_TIME           (1),
        .MIN_RST_ASSERTION_TIME        (3),
        .RESET_REQ_EARLY_DSRT_TIME     (1),
        .USE_RESET_REQUEST_IN0         (0),
        .USE_RESET_REQUEST_IN1         (0),
        .USE_RESET_REQUEST_IN2         (0),
        .USE_RESET_REQUEST_IN3         (0),
        .USE_RESET_REQUEST_IN4         (0),
        .USE_RESET_REQUEST_IN5         (0),
        .USE_RESET_REQUEST_IN6         (0),
```

```
        .USE_RESET_REQUEST_IN7      (0),
        .USE_RESET_REQUEST_IN8      (0),
        .USE_RESET_REQUEST_IN9      (0),
        .USE_RESET_REQUEST_IN10     (0),
        .USE_RESET_REQUEST_IN11     (0),
        .USE_RESET_REQUEST_IN12     (0),
        .USE_RESET_REQUEST_IN13     (0),
        .USE_RESET_REQUEST_IN14     (0),
        .USE_RESET_REQUEST_IN15     (0),
        .ADAPT_RESET_REQUEST        (0)
    ) rst_controller_001 (
        .reset_in0      (~hps_0_h2f_reset_reset),                // reset_in0.reset
        .clk            (clk_clk),                               //       clk.clk
        .reset_out      (rst_controller_001_reset_out_reset), // reset_out.reset
        .reset_req      (),                                      // (terminated)
        .reset_req_in0  (1'b0),                                  // (terminated)
        .reset_in1      (1'b0),                                  // (terminated)
        .reset_req_in1  (1'b0),                                  // (terminated)
        .reset_in2      (1'b0),                                  // (terminated)
        .reset_req_in2  (1'b0),                                  // (terminated)
        .reset_in3      (1'b0),                                  // (terminated)
        .reset_req_in3  (1'b0),                                  // (terminated)
        .reset_in4      (1'b0),                                  // (terminated)
        .reset_req_in4  (1'b0),                                  // (terminated)
        .reset_in5      (1'b0),                                  // (terminated)
        .reset_req_in5  (1'b0),                                  // (terminated)
        .reset_in6      (1'b0),                                  // (terminated)
        .reset_req_in6  (1'b0),                                  // (terminated)
        .reset_in7      (1'b0),                                  // (terminated)
        .reset_req_in7  (1'b0),                                  // (terminated)
        .reset_in8      (1'b0),                                  // (terminated)
        .reset_req_in8  (1'b0),                                  // (terminated)
        .reset_in9      (1'b0),                                  // (terminated)
        .reset_req_in9  (1'b0),                                  // (terminated)
        .reset_in10     (1'b0),                                  // (terminated)
        .reset_req_in10 (1'b0),                                  // (terminated)
        .reset_in11     (1'b0),                                  // (terminated)
        .reset_req_in11 (1'b0),                                  // (terminated)
        .reset_in12     (1'b0),                                  // (terminated)
        .reset_req_in12 (1'b0),                                  // (terminated)
        .reset_in13     (1'b0),                                  // (terminated)
        .reset_req_in13 (1'b0),                                  // (terminated)
        .reset_in14     (1'b0),                                  // (terminated)
        .reset_req_in14 (1'b0),                                  // (terminated)
        .reset_in15     (1'b0),                                  // (terminated)
        .reset_req_in15 (1'b0)                                   // (terminated)
    );

Endmodule
```

```
/*
 * Battle Tank Game Controller
 *
 * Team: Battle Tank
 * Columbia University
 */

module battle_tank_game_controller(input logic            clk,
                input logic                  reset,
                input logic [31:0]   writedata,
                input logic                  write,
                input                                chipselect,
                input logic [6:0]    address,

                        /* input signal from vga module */
                        input logic[9:0]             vcount_vga,
                        input logic[10:0]   hcount_vga,

            /* autio signal */
                        output logic tank_explosion_audio,
                output logic tank1_bullet_audio,
                        output logic tank2_bullet_audio,
                        output logic move_audio,

                        /* vga signal */
                        output logic [5:0]   our_life_sub_flag_vga,
                        output logic [4:0]  enemy_life_sub_flag_vga,
                        output logic [2:0]  game_stage_vga,
                        output logic [10:0]         tank_x_vga,
                        output logic [9:0]  tank_y_vga,
                        output logic [69:0]         brick_sub_flag_vga,
                        output logic [10:0]         bullet_x_vga,
                        output logic [9:0]  bullet_y_vga,
                        output logic                tank1_flag_vga,
                        output logic                tank2_flag_vga,
                        output logic [1:0]  tank1_explosion_flag_vga,
                        output logic [1:0]  tank1_generation_flag_vga,
                        output logic [1:0]  tank2_explosion_flag_vga,
                        output logic [1:0]  tank2_generation_flag_vga,
                        output logic [1:0]  tank_explosion_flag_vga,
                        output logic [1:0]  tank_generation_flag_vga,
                        output logic            bullet_flag_vga,
                        output logic            tank_enemy_flag_vga,
                        output logic [1:0]  tank_direction_vga //0-up, 1-down, 2-left, 3-right
                            );

                    `define BULLET_NUMBER 5'd7
                    `define ENEMY_NUMBER   5'd5
              /* Define variables */
```

```
/* Command Signal */
    logic [31:0]  game_command = 32'd0;

    /* Figure coordinates */
    //player1
    logic [10:0]  tank1_x=11'd320;
    logic [9:0]           tank1_y=10'd416;

    logic [10:0]  tank1_x_temp=11'd320;
    logic [9:0]           tank1_y_temp=10'd416;

    //play2
    logic [10:0]  tank2_x=11'd796;
    logic [9:0]           tank2_y=10'd416;

    //enemy tank
    logic [10:0]  tank_enemy_x[0:4];    ///enemy position
    logic [9:0]           tank_enemy_y[0:4];

    //bullet
     logic [10:0] bullet_x[0:6];
     logic [9:0]  bullet_y[0:6];

     /* Figure Display Flag */
     // tank1
     logic                    tank1_exist_flag = 1'd0;                //1-exist 2-dismiss
     logic [3:0]  tank1_type_flag = 4'd7;              //0~2 tank, 3-generation1, 4-generation2, 5-explosion1, 6-explosion2,
7-no explosion
     logic [2:0]  tank1_direction_flag = 3'd0;      //0- nothing, 1-up, 2-down,3-left, 4-right,5-static
     //tank2
     logic                tank2_exist_flag = 1'd0;                //1-exist 2-dismiss
     logic [3:0]  tank2_type_flag = 4'd7;              //0~2 tank, 3-generation1, 4-generation2, 5-explosion1, 6-explosion2,
7-no explosion
     logic [2:0]  tank2_direction_flag = 3'd0;      //0- nothing, 1-up, 2-down,3-left, 4-right,5-static
     //tank enemy
     logic [4:0]  tank_enemy_exist_flag = 5'd0;          //1-exist 2-dismiss
     logic [3:0]  tank_enemy_type_flag[0:4];           //0~2 tank, 3-generation1, 4-generation2, 5-explosion1, 6-explosion2,
7-no explosion
     logic [2:0]  tank_enemy_direction_flag[0:4];   //0- nothing, 1-up, 2-down,3-left, 4-right,5-static

     // brick flag
     logic [69:0] brick_sub_flag = 70'd0;
     //bullet
     logic [6:0]  bullet_exist_flag = 7'd0;

   logic [9:0]       enemy_life_counter= 10'd5;
     logic [3:0]  our_life_counter     = 3'd6;
```

```
assign brick_sub_flag_vga = brick_sub_flag;

/* FSM variable */
logic [1:0]  stage = 2'd0;
integer                 i = 0;

/* FSM */
always_ff @(posedge clk) begin
 if(reset) begin
        //disable all ports
        game_stage_vga <=3'd0;
        tank1_bullet_audio      <= 1'd0;
        tank2_bullet_audio      <= 1'd0;
        move_audio        <=    1'd0;
        //clear all regs
        game_command <= 32'd0;
        tank1_x<=11'd320;
        tank1_y<=10'd416;
        tank2_x<=11'd796;
        tank2_y<=10'd416;
        for(i = 0; i <= 4; i = i+1) begin
            tank_enemy_x[i] <= 11'd64;
            tank_enemy_y[i] <= 10'd32;
            tank_enemy_type_flag[i] <= 4'd7;
            tank_enemy_direction_flag[i] <= 3'd0;
        end

        tank1_exist_flag <= 1'd1;                   //1-exist 2-dismiss
        tank2_exist_flag <= 1'd1;                   //1-exist 2-dismiss
        tank_enemy_exist_flag <= 5'd0;      //1-exist 2-dismiss
        tank1_type_flag <= 4'd7;            //0~2 tank, 3-generation1, 4-generation2, 5-explosion1, 6-explosion2,
7-no explosion
        tank2_type_flag <= 4'd7;            //0~2 tank, 3-generation1, 4-generation2, 5-explosion1, 6-explosion2,
7-no explosion
        tank1_direction_flag <= 3'd0;    //0- nothing, 1-up, 2-down,3-left, 4-right,5-static
        tank2_direction_flag <= 3'd0;    //0- nothing, 1-up, 2-down,3-left, 4-right,5-static
        bullet_exist_flag <= 7'd0;
        brick_sub_flag <= 70'd0;
        stage <= 2'd0;
        game_stage_vga <= 3'd0;
 end
 else
            if (chipselect && write )           //use enable signal
                case(address)
                //msg for tanks
                7'd0:begin
                        tank1_exist_flag <= writedata[28];
                        tank1_type_flag <= writedata[24:21];
                        tank1_direction_flag <= writedata[27:25];
```

```
                    tank1_x_temp <= writedata[20:10];
                    tank1_y_temp <= writedata[9:0];
                end
            7'd1:begin
                    tank2_exist_flag <= writedata[28];
                    tank2_type_flag <= writedata[24:21];
                    tank2_direction_flag <= writedata[27:25];
                    tank2_x <= writedata[20:10];
                    tank2_y <= writedata[9:0];
                end
            //msg for brick
            7'd44: begin
                brick_sub_flag[31:0] <= writedata;
                end
            7'd45: begin
                brick_sub_flag[63:32] <= writedata;
                end
            7'd46: begin
                brick_sub_flag[69:64] <= writedata[5:0];
                end
            //msg for command
            7'd127:begin
                game_command <= writedata;
                enemy_life_counter <= writedata[29:20];
                our_life_counter    <= writedata[19:17];
                end
            default begin
            if((address >= 7'd2) && (address <= 7'd6)) begin
                                                    tank_enemy_exist_flag[address - 7'd2] <= writedata[28];
                                                    tank_enemy_type_flag[address - 7'd2] <= writedata[24:21];
                                                    tank_enemy_direction_flag[address - 7'd2] <= writedata[27:25];
                                                    tank_enemy_x[address - 7'd2] <= writedata[20:10];
                                                    tank_enemy_y[address - 7'd2] <= writedata[9:0];
                                                    end
            if((address >= 7'd22) && (address <= 7'd28)) begin
                                                    bullet_exist_flag[address - 7'd22] <= writedata[21];
                                                    bullet_x[address - 7'd22] <= writedata[20:10];
                                                    bullet_y[address - 7'd22] <= writedata[9:0];
                                                    end
                                end
            endcase
        case(stage)
        /* Game Stage 1: Welcome page */
        2'd0:begin
```

```
                                        game_stage_vga <= 3'd0;
                                        if(game_command[31:30] == 2'd1)
                                            stage <= 3'd1;
                                    end // stage 1 end
                                /* Game Stage 2: game play */
                                2'd1:begin
                                        if(vcount_vga >= 450) begin
                                            tank1_x <=tank1_x_temp;
                                            tank1_y <= tank1_y_temp;
                                        end
                                        move_audio      <=      1'd1;
                                        game_stage_vga <= 3'd1;

                                        // tank1 display
                                        tank1_flag_vga                                  <=      ((tank1_type_flag   ==
4'd0)&          (tank1_exist_flag)          &           (hcount_vga>=tank1_x)&(hcount_vga<=(tank1_x+11'd
63))&(vcount_vga>=tank1_y)&(vcount_vga<=(tank1_y+10'd 31)));
                                        tank1_generation_flag_vga[0]    <=      ((tank1_type_flag   ==      4'd3)&
(hcount_vga>=tank1_x)&(hcount_vga<=(tank1_x+11'd 63))&(vcount_vga>=tank1_y)&(vcount_vga<=(tank1_y+10'd 31)));
                                        tank1_generation_flag_vga[1]    <=      ((tank1_type_flag   ==      4'd4)&
(hcount_vga>=tank1_x)&(hcount_vga<=(tank1_x+11'd 63))&(vcount_vga>=tank1_y)&(vcount_vga<=(tank1_y+10'd 31)));
                                        tank1_explosion_flag_vga[0]     <=      ((tank1_type_flag   ==      4'd5)&
(hcount_vga>=tank1_x)&(hcount_vga<=(tank1_x+11'd 63))&(vcount_vga>=tank1_y)&(vcount_vga<=(tank1_y+10'd 31)));
                                        tank1_explosion_flag_vga[1]     <=      ((tank1_type_flag   ==      4'd6)&
(hcount_vga>=tank1_x)&(hcount_vga<=(tank1_x+11'd 63))&(vcount_vga>=tank1_y)&(vcount_vga<=(tank1_y+10'd 31)));

                                        if((tank1_exist_flag)   &   (hcount_vga>=tank1_x)&(hcount_vga<=(tank1_x+11'd
63))&(vcount_vga>=tank1_y)&(vcount_vga<=(tank1_y+10'd 31)))
                                        begin
                                            //disable other figure flag
                                            tank2_flag_vga <= 1'd0;
                                            bullet_flag_vga <= 1'd0;
                                            tank_enemy_flag_vga <= 1'd0;
                                        tank_x_vga<=tank1_x;
                                            tank_y_vga<=tank1_y;

                                            case(tank1_direction_flag)
                                                3'd1:
                                                    tank_direction_vga <= 2'd0;
                                                3'd2:
                                                    tank_direction_vga <= 2'd1;
                                                3'd3:
                                                    tank_direction_vga <= 2'd2;
                                                3'd4:
                                                    tank_direction_vga <= 2'd3;

                                            endcase
                                        end
                                        //tank2 display
```

```
tank2_flag_vga <= ((tank2_type_flag == 4'd0)& (tank2_exist_flag) & (hcount_vga>=tank2_x)&(hcount_vga<=(tank2_x+11'd 63))&(vcount_vga>=tank2_y)&(vcount_vga<=(tank2_y+10'd 31)));
tank2_generation_flag_vga[0] <= ((tank2_type_flag == 4'd3)& (hcount_vga>=tank2_x)&(hcount_vga<=(tank2_x+11'd 63))&(vcount_vga>=tank2_y)&(vcount_vga<=(tank2_y+10'd 31)));
tank2_generation_flag_vga[1] <= ((tank2_type_flag == 4'd4)& (hcount_vga>=tank2_x)&(hcount_vga<=(tank2_x+11'd 63))&(vcount_vga>=tank2_y)&(vcount_vga<=(tank2_y+10'd 31)));
tank2_explosion_flag_vga[0] <= ((tank2_type_flag == 4'd5)& (hcount_vga>=tank2_x)&(hcount_vga<=(tank2_x+11'd 63))&(vcount_vga>=tank2_y)&(vcount_vga<=(tank2_y+10'd 31)));
tank2_explosion_flag_vga[1] <= ((tank2_type_flag == 4'd6)& (hcount_vga>=tank2_x)&(hcount_vga<=(tank2_x+11'd 63))&(vcount_vga>=tank2_y)&(vcount_vga<=(tank2_y+10'd 31)));


if((tank2_exist_flag) & (hcount_vga>=tank2_x)&(hcount_vga<=(tank2_x+11'd 63))&(vcount_vga>=tank2_y)&(vcount_vga<=(tank2_y+10'd 31)))
begin
    //disable other figure flag
    tank1_flag_vga <= 1'd0;
    bullet_flag_vga <= 1'd0;
    tank_enemy_flag_vga <= 1'd0;

  tank_x_vga<=tank2_x;
    tank_y_vga<=tank2_y;

    case(tank2_direction_flag)
        3'd1:
            tank_direction_vga <= 2'd0;
        3'd2:
            tank_direction_vga <= 2'd1;
        3'd3:
            tank_direction_vga <= 2'd2;
        3'd4:
            tank_direction_vga <= 2'd3;

    endcase
end

//tank ememy display

tank_enemy_flag_vga <= (((tank_enemy_type_flag[0] == 4'd2)&&((tank_enemy_exist_flag[0]) & (hcount_vga>=tank_enemy_x[0])&(hcount_vga<=(tank_enemy_x[0]+11'd 63))&(vcount_vga>=tank_enemy_y[0])&(vcount_vga<=(tank_enemy_y[0]+10'd 31))))

||((tank_enemy_type_flag[1] == 4'd2)&&((tank_enemy_exist_flag[1]) & (hcount_vga>=tank_enemy_x[1])&(hcount_vga<=(tank_enemy_x[1]+11'd 63))&(vcount_vga>=tank_enemy_y[1])&(vcount_vga<=(tank_enemy_y[1]+10'd 31))))

||((tank_enemy_type_flag[2] == 4'd2)&&((tank_enemy_exist_flag[2]) & (hcount_vga>=tank_enemy_x[2])&(hcount_vga<=(tank_enemy_x[2]+11'd
```

```
63))&(vcount_vga>=tank_enemy_y[2])&(vcount_vga<=(tank_enemy_y[2]+10'd 31))))

    ||((tank_enemy_type_flag[3] == 4'd2)&&((tank_enemy_exist_flag[3]) &
(hcount_vga>=tank_enemy_x[3])&(hcount_vga<=(tank_enemy_x[3]+11'd
63))&(vcount_vga>=tank_enemy_y[3])&(vcount_vga<=(tank_enemy_y[3]+10'd 31))))

    ||((tank_enemy_type_flag[4] == 4'd2)&&((tank_enemy_exist_flag[4]) &
(hcount_vga>=tank_enemy_x[4])&(hcount_vga<=(tank_enemy_x[4]+11'd
63))&(vcount_vga>=tank_enemy_y[4])&(vcount_vga<=(tank_enemy_y[4]+10'd 31))));

                tank_generation_flag_vga[0]<= (((tank_enemy_type_flag[0] ==
4'd3)&&((tank_enemy_exist_flag[0]) & (hcount_vga>=tank_enemy_x[0])&(hcount_vga<=(tank_enemy_x[0]+11'd
63))&(vcount_vga>=tank_enemy_y[0])&(vcount_vga<=(tank_enemy_y[0]+10'd 31))))

    ||((tank_enemy_type_flag[1] == 4'd3)&&((tank_enemy_exist_flag[1]) &
(hcount_vga>=tank_enemy_x[1])&(hcount_vga<=(tank_enemy_x[1]+11'd
63))&(vcount_vga>=tank_enemy_y[1])&(vcount_vga<=(tank_enemy_y[1]+10'd 31))))

    ||((tank_enemy_type_flag[2] == 4'd3)&&((tank_enemy_exist_flag[2]) &
(hcount_vga>=tank_enemy_x[2])&(hcount_vga<=(tank_enemy_x[2]+11'd
63))&(vcount_vga>=tank_enemy_y[2])&(vcount_vga<=(tank_enemy_y[2]+10'd 31))))

    ||((tank_enemy_type_flag[3] == 4'd3)&&((tank_enemy_exist_flag[3]) &
(hcount_vga>=tank_enemy_x[3])&(hcount_vga<=(tank_enemy_x[3]+11'd
63))&(vcount_vga>=tank_enemy_y[3])&(vcount_vga<=(tank_enemy_y[3]+10'd 31))))

    ||((tank_enemy_type_flag[4] == 4'd3)&&((tank_enemy_exist_flag[4]) &
(hcount_vga>=tank_enemy_x[4])&(hcount_vga<=(tank_enemy_x[4]+11'd
63))&(vcount_vga>=tank_enemy_y[4])&(vcount_vga<=(tank_enemy_y[4]+10'd 31))));

                tank_generation_flag_vga[1]<= (((tank_enemy_type_flag[0] ==
4'd4)&&((tank_enemy_exist_flag[0]) & (hcount_vga>=tank_enemy_x[0])&(hcount_vga<=(tank_enemy_x[0]+11'd
63))&(vcount_vga>=tank_enemy_y[0])&(vcount_vga<=(tank_enemy_y[0]+10'd 31))))

    ||((tank_enemy_type_flag[1] == 4'd4)&&((tank_enemy_exist_flag[1]) &
(hcount_vga>=tank_enemy_x[1])&(hcount_vga<=(tank_enemy_x[1]+11'd
63))&(vcount_vga>=tank_enemy_y[1])&(vcount_vga<=(tank_enemy_y[1]+10'd 31))))

    ||((tank_enemy_type_flag[2] == 4'd4)&&((tank_enemy_exist_flag[2]) &
(hcount_vga>=tank_enemy_x[2])&(hcount_vga<=(tank_enemy_x[2]+11'd
63))&(vcount_vga>=tank_enemy_y[2])&(vcount_vga<=(tank_enemy_y[2]+10'd 31))))

    ||((tank_enemy_type_flag[3] == 4'd4)&&((tank_enemy_exist_flag[3]) &
(hcount_vga>=tank_enemy_x[3])&(hcount_vga<=(tank_enemy_x[3]+11'd
63))&(vcount_vga>=tank_enemy_y[3])&(vcount_vga<=(tank_enemy_y[3]+10'd 31))))

    ||((tank_enemy_type_flag[4] == 4'd4)&&((tank_enemy_exist_flag[4]) &
(hcount_vga>=tank_enemy_x[4])&(hcount_vga<=(tank_enemy_x[4]+11'd
63))&(vcount_vga>=tank_enemy_y[4])&(vcount_vga<=(tank_enemy_y[4]+10'd 31))));
```

```
tank_explosion_flag_vga[0] <= (((tank_enemy_type_flag[0] == 4'd5)&&((tank_enemy_exist_flag[0]) & (hcount_vga>=tank_enemy_x[0])&(hcount_vga<=(tank_enemy_x[0]+11'd 63))&(vcount_vga>=tank_enemy_y[0])&(vcount_vga<=(tank_enemy_y[0]+10'd 31))))

||((tank_enemy_type_flag[1] == 4'd5)&&((tank_enemy_exist_flag[1]) & (hcount_vga>=tank_enemy_x[1])&(hcount_vga<=(tank_enemy_x[1]+11'd 63))&(vcount_vga>=tank_enemy_y[1])&(vcount_vga<=(tank_enemy_y[1]+10'd 31))))

||((tank_enemy_type_flag[2] == 4'd5)&&((tank_enemy_exist_flag[2]) & (hcount_vga>=tank_enemy_x[2])&(hcount_vga<=(tank_enemy_x[2]+11'd 63))&(vcount_vga>=tank_enemy_y[2])&(vcount_vga<=(tank_enemy_y[2]+10'd 31))))

||((tank_enemy_type_flag[3] == 4'd5)&&((tank_enemy_exist_flag[3]) & (hcount_vga>=tank_enemy_x[3])&(hcount_vga<=(tank_enemy_x[3]+11'd 63))&(vcount_vga>=tank_enemy_y[3])&(vcount_vga<=(tank_enemy_y[3]+10'd 31))))

||((tank_enemy_type_flag[4] == 4'd5)&&((tank_enemy_exist_flag[4]) & (hcount_vga>=tank_enemy_x[4])&(hcount_vga<=(tank_enemy_x[4]+11'd 63))&(vcount_vga>=tank_enemy_y[4])&(vcount_vga<=(tank_enemy_y[4]+10'd 31))));

tank_explosion_flag_vga[1] <= (((tank_enemy_type_flag[0] == 4'd6)&&((tank_enemy_exist_flag[0]) & (hcount_vga>=tank_enemy_x[0])&(hcount_vga<=(tank_enemy_x[0]+11'd 63))&(vcount_vga>=tank_enemy_y[0])&(vcount_vga<=(tank_enemy_y[0]+10'd 31))))

||((tank_enemy_type_flag[1] == 4'd6)&&((tank_enemy_exist_flag[1]) & (hcount_vga>=tank_enemy_x[1])&(hcount_vga<=(tank_enemy_x[1]+11'd 63))&(vcount_vga>=tank_enemy_y[1])&(vcount_vga<=(tank_enemy_y[1]+10'd 31))))

||((tank_enemy_type_flag[2] == 4'd6)&&((tank_enemy_exist_flag[2]) & (hcount_vga>=tank_enemy_x[2])&(hcount_vga<=(tank_enemy_x[2]+11'd 63))&(vcount_vga>=tank_enemy_y[2])&(vcount_vga<=(tank_enemy_y[2]+10'd 31))))

||((tank_enemy_type_flag[3] == 4'd6)&&((tank_enemy_exist_flag[3]) & (hcount_vga>=tank_enemy_x[3])&(hcount_vga<=(tank_enemy_x[3]+11'd 63))&(vcount_vga>=tank_enemy_y[3])&(vcount_vga<=(tank_enemy_y[3]+10'd 31))))

||((tank_enemy_type_flag[4] == 4'd6)&&((tank_enemy_exist_flag[4]) & (hcount_vga>=tank_enemy_x[4])&(hcount_vga<=(tank_enemy_x[4]+11'd 63))&(vcount_vga>=tank_enemy_y[4])&(vcount_vga<=(tank_enemy_y[4]+10'd 31))));

for(i = 0; i <= 4; i = i+1)
begin
    if((tank_enemy_exist_flag[i]) & (hcount_vga>=tank_enemy_x[i])&(hcount_vga<=(tank_enemy_x[i]+11'd 63))&(vcount_vga>=tank_enemy_y[i])&(vcount_vga<=(tank_enemy_y[i]+10'd 31)))
    begin
        //disable other figure flag
        tank1_flag_vga <= 1'd0;
```

```
                                        bullet_flag_vga <= 1'd0;
                                        tank2_flag_vga <= 1'd0;


                                        // not sure if there is a delay in assigning the rom address when hcount
and vcount are within the tank area
                                        tank_x_vga<=tank_enemy_x[i];
                                        tank_y_vga<=tank_enemy_y[i];

                                        case(tank_enemy_direction_flag[i])
                                            3'd1:
                                                tank_direction_vga <= 2'd0;
                                            3'd2:
                                                tank_direction_vga <= 2'd1;
                                            3'd3:
                                                tank_direction_vga <= 2'd2;
                                            3'd4:
                                                tank_direction_vga <= 2'd3;

                                        endcase
                                    end
                                end // for loop end

                                    bullet_flag_vga
<=(((((hcount_vga-bullet_x[0])*(hcount_vga-bullet_x[0])/4+(vcount_vga-bullet_y[0])*(vcount_vga-bullet_y[0]))<=22'd4)          &&
(bullet_exist_flag[0]))

    ||((((hcount_vga-bullet_x[1])*(hcount_vga-bullet_x[1])/4+(vcount_vga-bullet_y[1])*(vcount_vga-bullet_y[1]))<=22'd4)          &&
(bullet_exist_flag[1]))

    ||((((hcount_vga-bullet_x[2])*(hcount_vga-bullet_x[2])/4+(vcount_vga-bullet_y[2])*(vcount_vga-bullet_y[2]))<=22'd4)          &&
(bullet_exist_flag[2]))

    ||((((hcount_vga-bullet_x[3])*(hcount_vga-bullet_x[3])/4+(vcount_vga-bullet_y[3])*(vcount_vga-bullet_y[3]))<=22'd4)          &&
(bullet_exist_flag[3]))

    ||((((hcount_vga-bullet_x[4])*(hcount_vga-bullet_x[4])/4+(vcount_vga-bullet_y[4])*(vcount_vga-bullet_y[4]))<=22'd4)          &&
(bullet_exist_flag[4]))

    ||((((hcount_vga-bullet_x[5])*(hcount_vga-bullet_x[5])/4+(vcount_vga-bullet_y[5])*(vcount_vga-bullet_y[5]))<=22'd4)          &&
(bullet_exist_flag[5]))

    ||((((hcount_vga-bullet_x[6])*(hcount_vga-bullet_x[6])/4+(vcount_vga-bullet_y[6])*(vcount_vga-bullet_y[6]))<=22'd4)          &&
(bullet_exist_flag[6])));

                                    // bullet display
                                    for(i = 0; i<=6; i = i+1)
                                    begin
```

```
        if(((((hcount_vga-bullet_x[i])*(hcount_vga-bullet_x[i])/4+(vcount_vga-bullet_y[i])*(vcount_vga-bullet_y[i]))<=22'd4)  &&
bullet_exist_flag[i]))
                                                        begin
                                                                //disable other figure flag
                                                                tank1_flag_vga <= 1'd0;
                                                                tank_enemy_flag_vga <= 1'd0;
                                                                tank2_flag_vga <= 1'd0;
                                                                tank_explosion_flag_vga <= 2'd0;
                                                                tank_generation_flag_vga <= 2'd0;
                                                                //bullet coordinate
                                                                bullet_x_vga <= bullet_x[i];
                                                                bullet_y_vga <= bullet_y[i];

                                                        end
                                                end // for loop end

                                        // bullet audio
                                                tank1_bullet_audio <= bullet_exist_flag[0];
                                                tank2_bullet_audio <= bullet_exist_flag[1];
                                        // explosion audio
                                        tank_explosion_audio <= ((tank1_type_flag == 4'd5) || (tank1_type_flag == 4'd6)
                                                                                                ||(tank2_type_flag == 4'd5) ||
(tank2_type_flag == 4'd6));
                                                // determine life figure displayed
                                                case(enemy_life_counter)
                                                        10'd5: enemy_life_sub_flag_vga<= 5'd31;
                                                        10'd4: enemy_life_sub_flag_vga<= 5'd15;
                                                        10'd3: enemy_life_sub_flag_vga<= 5'd7;
                                                        10'd2: enemy_life_sub_flag_vga<= 5'd3;
                                                        10'd1: enemy_life_sub_flag_vga<= 5'd1;
                                                        10'd0: enemy_life_sub_flag_vga<= 5'd0;
                                                        default:enemy_life_sub_flag_vga<= 5'd31;
                                                endcase

                                                case(our_life_counter)
                                                        3'd6: our_life_sub_flag_vga<= 6'd63;
                                                        3'd5: our_life_sub_flag_vga<= 6'd31;
                                                        3'd4: our_life_sub_flag_vga<= 6'd15;
                                                        3'd3: our_life_sub_flag_vga<= 6'd7;
                                                        3'd2: our_life_sub_flag_vga<= 6'd3;
                                                        3'd1: our_life_sub_flag_vga<= 6'd1;
                                                        3'd0: our_life_sub_flag_vga<= 6'd0;
                                                        default:our_life_sub_flag_vga<= 6'd63;
                                                endcase

                                                // determine the stage status
                                                if((game_command[31:30] == 2'd3) || (game_command[31:30] == 2'd2))
                                                        stage <= 2'd2;
```

```
                end     // stage 2 end

            /* Game Stage 3: Game Win page */
        2'd2:begin
                move_audio          <=      1'd0;
                game_stage_vga <= 3'd2;
                if(game_command[31:30] == 2'd0)
                        stage <= 2'd0;
            end //stage 3 end

        endcase

    end

endmodule
```

```
/*
 * Battle Tank Game audio top module
 *
 * Team: Battle Tank
 * Columbia University
 */
module audio (
    input   OSC_50_B4A,
    inout   AUD_ADCLRCK,
    input   AUD_ADCDAT,
    inout   AUD_DACLRCK,
    output AUD_DACDAT,
    output AUD_XCK,
    inout   AUD_BCLK,
    output AUD_I2C_SCLK,
    inout   AUD_I2C_SDAT,
    output AUD_MUTE,

    input   [3:0] KEY,
    input   [3:0] SW,
        input tank_explosion_audio,
        input tank1_bullet_audio,
        input tank2_bullet_audio,
        input move_audio
);

wire reset = !KEY[0];
wire main_clk;
wire audio_clk;

wire [1:0] audio_data_record_req;
wire [1:0] audio_data_play_req;
wire [15:0] audio_output;
wire [15:0] audio_input;

clock_pll pll (
    .refclk (OSC_50_B4A),
    .rst (reset),
    .outclk_0 (audio_clk),
    .outclk_1 (main_clk)
);

i2c_controller i2c_controller(
    .clk (main_clk),
    .reset (reset),
    .i2c_sclk (AUD_I2C_SCLK),
    .i2c_sdat (AUD_I2C_SDAT)
);
```

```
assign AUD_XCK = audio_clk;
assign AUD_MUTE = (SW != 4'b0);



audio_codec_controller audio_codec_controller (
    .clk (audio_clk),
    .reset (reset),
    .audio_data_record_req (audio_data_record_req),
    .audio_data_play_req (audio_data_play_req),
    .audio_output (audio_output),
    .audio_input (audio_input),
    .channel_sel (2'b11), // 11-set both left and right channel;10-left channel; 01-right channel

    .AUD_ADCLRCK (AUD_ADCLRCK),
    .AUD_ADCDAT (AUD_ADCDAT),
    .AUD_DACLRCK (AUD_DACLRCK),
    .AUD_DACDAT (AUD_DACDAT),
    .AUD_BCLK (AUD_BCLK)
);

audio_data_controller audio_data_controller (
    .clk (audio_clk),
    .audio_data_record_req          (audio_data_record_req),
    .audio_data_play_req            (audio_data_play_req),
    .audio_output                   (audio_output),
    .audio_input                        (audio_input),
    .control                            (SW),
        .tank_explosion_audio       (tank_explosion_audio),
        .tank1_bullet_audio             (tank1_bullet_audio),
        .tank2_bullet_audio             (tank2_bullet_audio),
        .move_audio                         (move_audio)
);

Endmodule
```

```
/*
 * Battle Tank Game audio codec controller
 *
 * Team: Battle Tank
 * Columbia University
 */
module audio_codec_controller (
    input   clk,
    input   reset,
    output [1:0]   audio_data_record_req,
    output [1:0]   audio_data_play_req,
    input   [15:0] audio_output,
    output [15:0] audio_input,
    // 1 - left, 0 - right
    input   [1:0] channel_sel,

    output AUD_ADCLRCK,
    input AUD_ADCDAT,
    output AUD_DACLRCK,
    output AUD_DACDAT,
    output AUD_BCLK
);

reg [7:0] lrck_divider;
reg [1:0] bclk_divider;

reg [15:0] shift_out;
reg [15:0] shift_temp;
reg [15:0] shift_in;

wire lrck = !lrck_divider[7];

assign AUD_ADCLRCK = lrck;
assign AUD_DACLRCK = lrck;
assign AUD_BCLK = bclk_divider[1];
assign AUD_DACDAT = shift_out[15];

always @(posedge clk) begin
    if (reset) begin
        lrck_divider <= 8'hff;
        bclk_divider <= 2'b11;
    end else begin
        lrck_divider <= lrck_divider + 1'b1;
        bclk_divider <= bclk_divider + 1'b1;
    end
end

assign audio_data_record_req[1] = (lrck_divider == 8'h40);
assign audio_data_record_req[0] = (lrck_divider == 8'hc0);
```

```
assign audio_input = shift_in;
assign audio_data_play_req[1] = (lrck_divider == 8'hfe); //read left channel data before rising edge
assign audio_data_play_req[0] = (lrck_divider == 8'h7e); //read right channel data before falling edge

wire lrck_low = (lrck_divider == 8'h7f); // right channel
wire lrck_high = (lrck_divider == 8'hff); // left channel
// high right after bclk is set
wire bclk_high = (bclk_divider == 2'b10 && !lrck_divider[6]);
// high right before bclk is cleared
wire bclk_low = (bclk_divider == 2'b11 && !lrck_divider[6]);

always @(posedge clk) begin
    if (reset) begin
        shift_out <= 16'h0;
        shift_in <= 16'h0;
        shift_in <= 16'h0;
    end else if (lrck_high || lrck_low) begin
        // check if current channel is selected
        if (channel_sel[lrck_high]) begin
            shift_out <= audio_output;
            shift_temp <= audio_output;
            shift_in <= 16'h0;
        // repeat the sample from the other channel if not
        end else shift_out <= shift_temp;
    end else if (bclk_high == 1) begin
        // only read in if channel is selected
        if (channel_sel[lrck])
            shift_in <= {shift_in[14:0], AUD_ADCDAT};
    end else if (bclk_low == 1) begin
        shift_out <= {shift_out[14:0], 1'b0};
    end
end

endmodule
```

```
/*
 * Battle Tank Game audio data controller
 *
 * Team: Battle Tank
 * Columbia University
 */
module audio_data_controller (
    input   clk,
    // 1 - left, 0 - right
    input   [1:0]   audio_data_record_req,
    input   [1:0]    audio_data_play_req,
    output [15:0] audio_output,
    input   [15:0] audio_input,
    input   [3:0]   control,

        input tank_explosion_audio,
        input tank1_bullet_audio,
        input tank2_bullet_audio,
        input move_audio,

        input   [31:0] writedata,
        input                   write,
        input                   chipselect,
        input   [6:0]       address
);

//reg [15:0] romdata [0:11131];
reg [6:0]   index = 7'd0;
reg [15:0] last_sample;
reg [15:0] dat;

reg [13:0] fire_l_addr;
wire [15:0] fire_l_data;
reg [13:0] fire_r_addr;
wire [15:0] fire_r_data;

reg [13:0] explosion_addr;
wire [15:0] explosion_data;

reg [13:0] move_addr;
wire [15:0] move_data;

assign audio_output = dat;

parameter PLAY      = 1;
parameter EXPLOSION = 2;
parameter MOVE          = 3;

attack_L fire_l (    .address(fire_l_addr),      .clock(clk),   .q(fire_l_data));
```

```
fire_R fire_r (         .address(fire_r_addr),       .clock(clk),   .q(fire_r_data));
explosion explosion (       .address(explosion_addr), .clock(clk),   .q(explosion_data));
self_move self_move (     .address(move_addr),       .clock(clk),   .q(move_data));

assign bullet_status = tank1_bullet_audio;
assign move_status = move_audio;
assign explosion_status = tank_explosion_audio;

always @(posedge clk) begin

        if(bullet_status ==0) begin
                fire_r_addr<= 14'd00;
                fire_l_addr<= 14'd00;
         end
        if(explosion_status == 0)
                explosion_addr <= 15'd0;

        //load left channel data
         if (audio_data_play_req[1]) begin
          if (control[PLAY] && bullet_status) begin
                                dat <= fire_l_data;
                                if (fire_l_addr == 14'd 7535)
                                        fire_l_addr<= 14'd7535;
                                else
                                        fire_l_addr <= fire_l_addr + 1'b1;end
           else if (control[EXPLOSION] && explosion_status) begin
                                dat <= explosion_data;
                                if (explosion_addr == 15'd 30575)
                                        explosion_addr<= 15'd30575;
                                else
                                        explosion_addr <= explosion_addr + 1'b1;end
              else if (control[MOVE] &&   move_status) begin
                                dat <= move_data;
                                if (move_addr == 15'd 21359)
                                        move_addr<= 15'd00;
                                else
                                        move_addr <= move_addr + 1'b1;end
                    else
                dat <= 16'd0;
        end


        //load right channel data
         if (audio_data_play_req[0]) begin
          if (control[PLAY] && bullet_status) begin
                                dat <= fire_r_data;
                                if (fire_r_addr == 14'd 7535)
                                        fire_r_addr<= 14'd7535;
                                else
```

```
                                        fire_r_addr <= fire_r_addr + 1'b1;end
        else if (control[EXPLOSION] && explosion_status) begin
                                    dat <= explosion_data;
                                    if (explosion_addr == 15'd 30575)
                                        explosion_addr<= 15'd30575;
                                    else
                                        explosion_addr <= explosion_addr + 1'b1;end
            else if (control[MOVE] && move_status) begin
                                    dat <= move_data;
                                    if (move_addr == 15'd 21359)
                                        move_addr<= 15'd00;
                                    else
                                        move_addr <= move_addr + 1'b1;end
                else
             dat <= 16'd0;
    end

end

endmodule
```

```
/*
 * Battle Tank Game audio I2C configuration
 *
 * Team: Battle Tank
 * Columbia University
 */
module i2c_controller(
      input clk,
      input reset,

      output i2c_sclk,
      inout   i2c_sdat
);

reg [23:0] i2c_data;
reg [15:0] lut_data;
reg [3:0]    lut_index = 4'd0;

parameter LAST_INDEX = 4'ha;

reg   i2c_start = 1'b0;
wire i2c_done;
wire i2c_ack;

i2c i2c (
      .clk (clk),
      .i2c_sclk (i2c_sclk),
      .i2c_sdat (i2c_sdat),
      .i2c_data (i2c_data),
      .start (i2c_start),
      .done (i2c_done),
      .ack (i2c_ack)
);

always @(*) begin
      case (lut_index)
            4'h0: lut_data <= 16'h0c10; // power on everything except out
            4'h1: lut_data <= 16'h0017; // left input
            4'h2: lut_data <= 16'h0217; // right input
            4'h3: lut_data <= 16'h0479; // left output
            4'h4: lut_data <= 16'h0679; // right output
            4'h5: lut_data <= 16'h08d4; // analog path
            4'h6: lut_data <= 16'h0a04; // digital path
            4'h7: lut_data <= 16'h0e01; // digital IF:01-left; 00-right; 10-I2S; 11-DSP
            4'h8: lut_data <= 16'h1020; // sampling rate
            4'h9: lut_data <= 16'h0c00; // power on everything
            4'ha: lut_data <= 16'h1201; // activate
            default: lut_data <= 16'h0000;
      endcase
```

```
end

reg [1:0] control_state = 2'b00;

always @(posedge clk) begin
    if (reset) begin
        lut_index <= 4'd0;
        i2c_start <= 1'b0;
        control_state <= 2'b00;
    end else begin
        case (control_state)
            2'b00: begin
                i2c_start <= 1'b1;
                i2c_data <= {8'h34, lut_data};
                control_state <= 2'b01;
            end
            2'b01: begin
                i2c_start <= 1'b0;
                control_state <= 2'b10;
            end
            2'b10: if (i2c_done) begin
                if (i2c_ack) begin
                    if (lut_index == LAST_INDEX)
                        control_state <= 2'b11;
                    else begin
                        lut_index <= lut_index + 1'b1;
                        control_state <= 2'b00;
                    end
                end else
                    control_state <= 2'b00;
            end
        endcase
    end
end

endmodule
```

```
module i2c(
    input   clk,

    output i2c_sclk,
    inout   i2c_sdat,

    input   start,
    output done,
    output ack,

    input [23:0] i2c_data
);

reg [23:0] data;

reg [4:0] stage;
reg [6:0] sclk_divider;
reg clock_en = 1'b0;

// don't toggle the clock unless we're sending data
// clock will also be kept high when sending START and STOP symbols
assign i2c_sclk = (!clock_en) || sclk_divider[6];
wire midlow = (sclk_divider == 7'h1f);

reg sdat = 1'b1;
// rely on pull-up resistor to set SDAT high
assign i2c_sdat = (sdat) ? 1'bz : 1'b0;

reg [2:0] acks;

parameter LAST_STAGE = 5'd29;

assign ack = (acks == 3'b000);
assign done = (stage == LAST_STAGE);

always @(posedge clk) begin
    if (start) begin
        sclk_divider <= 7'd0;
        stage <= 5'd0;
        clock_en = 1'b0;
        sdat <= 1'b1;
        acks <= 3'b111;
        data <= i2c_data;
    end else begin
        if (sclk_divider == 7'd127) begin
            sclk_divider <= 7'd0;

            if (stage != LAST_STAGE)
                stage <= stage + 1'b1;
```

```
        case (stage)
            // after start
            5'd0:   clock_en <= 1'b1;
            // receive acks
            5'd9:   acks[0] <= i2c_sdat;
            5'd18: acks[1] <= i2c_sdat;
            5'd27: acks[2] <= i2c_sdat;
            // before stop
            5'd28: clock_en <= 1'b0;
        endcase
    end else
        sclk_divider <= sclk_divider + 1'b1;

    if (midlow) begin
        case (stage)
            // start
            5'd0:   sdat <= 1'b0;
            // byte 1
            5'd1:   sdat <= data[23];
            5'd2:   sdat <= data[22];
            5'd3:   sdat <= data[21];
            5'd4:   sdat <= data[20];
            5'd5:   sdat <= data[19];
            5'd6:   sdat <= data[18];
            5'd7:   sdat <= data[17];
            5'd8:   sdat <= data[16];
            // ack 1
            5'd9:   sdat <= 1'b1;
            // byte 2
            5'd10: sdat <= data[15];
            5'd11: sdat <= data[14];
            5'd12: sdat <= data[13];
            5'd13: sdat <= data[12];
            5'd14: sdat <= data[11];
            5'd15: sdat <= data[10];
            5'd16: sdat <= data[9];
            5'd17: sdat <= data[8];
            // ack 2
            5'd18: sdat <= 1'b1;
            // byte 3
            5'd19: sdat <= data[7];
            5'd20: sdat <= data[6];
            5'd21: sdat <= data[5];
            5'd22: sdat <= data[4];
            5'd23: sdat <= data[3];
            5'd24: sdat <= data[2];
            5'd25: sdat <= data[1];
            5'd26: sdat <= data[0];
```

```
                    // ack 3
                    5'd27: sdat <= 1'b1;
                    // stop
                    5'd28: sdat <= 1'b0;
                    5'd29: sdat <= 1'b1;
                endcase
            end
        end
    end

endmodule
```

```
/*
 * Battle Tank Game VGA Controller
 *
 * Team: Battle Tank
 * Columbia University
 */

module vga_led_controller(input logic            clk,
                  input logic                    reset,
                        input logic [5:0]    our_life_sub_flag_vga,
                        input logic [4:0]    enemy_life_sub_flag_vga,
                        input logic [2:0]    game_stage_vga,
                        input logic [10:0]  tank_x_vga,
                        input logic [9:0]    tank_y_vga,
                        input logic [69:0]  brick_sub_flag_vga,
                        input logic [10:0]  bullet_x_vga,
                        input logic [9:0]    bullet_y_vga,
                        input logic                  tank1_flag_vga,
                        input logic                  tank2_flag_vga,
                        input logic [1:0]    tank_explosion_flag_vga,
                        input logic [1:0]    tank_generation_flag_vga,
                        input logic [1:0]    tank1_explosion_flag_vga,
                        input logic [1:0]    tank1_generation_flag_vga,
                        input logic [1:0]    tank2_explosion_flag_vga,
                        input logic [1:0]    tank2_generation_flag_vga,
                        input logic           bullet_flag_vga,
                        input logic           tank_enemy_flag_vga,
                        input logic [1:0]    tank_direction_vga, //0-up, 1-down, 2-left, 3-right
                       // output hcount, vcount for game controller
                        output logic[9:0]   vcount_vga,
                        output logic[10:0] hcount_vga,

                        output logic [7:0]  VGA_R, VGA_G, VGA_B,
                  output logic                       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
                  output logic                       VGA_SYNC_n);


                /* Define variables */

                        //Game stage status
                        logic [2:0]   game_stage_status;
                       // player 1, 2 enemy tank address
                        logic   [10:0] sprite_tank_addr;
                        // Welcome page address
                        logic   [15:0] sprite_oneplayer_addr;
                        // Gamewin page address
                        logic   [15:0] sprite_game_win_addr;
                        //sides address
                        logic   [10:0] sprite_life_flag_addr;
```

```
    // wecome page mif data
    logic   [23:0] sprite_oneplayer_data;
    // game win page vga
    logic   [23:0] sprite_game_win_data;
    // player1 tank mif data
    logic   [23:0] sprite_tank1up_data;
    logic   [23:0] sprite_tank1down_data;
    logic   [23:0] sprite_tank1left_data;
    logic   [23:0] sprite_tank1right_data;
    // player2 tank mif data
    logic   [23:0] sprite_tank2up_data;
    logic   [23:0] sprite_tank2down_data;
    logic   [23:0] sprite_tank2left_data;
    logic   [23:0] sprite_tank2right_data;
    // enemy tank mif data
    logic   [23:0] sprite_tank_enemy_up_data;
    logic   [23:0] sprite_tank_enemy_down_data;
    logic   [23:0] sprite_tank_enemy_right_data;
    logic   [23:0] sprite_tank_enemy_left_data;
    // generation mif data
    logic   [23:0] sprite_generation1_data;
    logic   [23:0] sprite_generation2_data;
    logic   [23:0] sprite_explosion1_data;
    logic   [23:0] sprite_explosion2_data;
    //sides mif data
    logic   [23:0] sprite_life_flag_data;
   // play1 tank vga
logic   [7:0] VGA_R_tank1, VGA_G_tank1, VGA_B_tank1;
   // play2 tank vga
logic   [7:0] VGA_R_tank2, VGA_G_tank2, VGA_B_tank2;
    // generation1 vga
logic   [7:0] VGA_R_generation1, VGA_G_generation1, VGA_B_generation1;
    // generation2 vga
logic   [7:0] VGA_R_generation2, VGA_G_generation2, VGA_B_generation2;
    // explosion1 vga
logic   [7:0] VGA_R_explosion1, VGA_G_explosion1, VGA_B_explosion1;
    // explosion2 vga
logic   [7:0] VGA_R_explosion2, VGA_G_explosion2, VGA_B_explosion2;
   // enemy tank vga
logic   [7:0] VGA_R_enemy_tank, VGA_G_enemy_tank, VGA_B_enemy_tank;
   // map vga
    logic   [7:0] VGA_R_brick, VGA_G_brick, VGA_B_brick;
    logic   [7:0] VGA_R_grass, VGA_G_grass, VGA_B_grass;
    logic   [7:0] VGA_R_water, VGA_G_water, VGA_B_water;
    logic   [7:0] VGA_R_steel, VGA_G_steel, VGA_B_steel;
    logic   [7:0] VGA_R_homebase, VGA_G_homebase, VGA_B_homebase;
    // welcome page vga
    logic   [7:0] VGA_R_oneplayer, VGA_G_oneplayer, VGA_B_oneplayer;
   // game win page vga
```

```
    logic   [7:0] VGA_R_game_win, VGA_G_game_win, VGA_B_game_win;
    //sides vga
    logic   [7:0] VGA_R_our_life, VGA_G_our_life, VGA_B_our_life;
    logic   [7:0] VGA_R_enemy_life, VGA_G_enemy_life, VGA_B_enemy_life;
    logic   [7:0] VGA_R_life_flag, VGA_G_life_flag, VGA_B_life_flag;
     /*flag for display*/
     logic                  grass;
    // player1 tank flag
     logic                  tank1;
     logic                          tank2;
     //generation and explosion fag
     logic [1:0]                    tank_generation;
     logic [1:0]                    tank_explosion;
     logic [1:0]                    tank1_generation;
     logic [1:0]                    tank1_explosion;
     logic [1:0]                    tank2_generation;
     logic [1:0]                    tank2_explosion;
     // enemy tank
     logic                  generate_enemy;
     //map flag
     logic                  brick;
     logic [69:0]           brick_sub_flag; //flag=0 means the subbrick can be existed
   logic                    water;
     logic                  steel;
     logic                  homebase;
     logic                          grey_background;
     logic                  bullet;
     //welcome page flag
     logic                  oneplayer;
     // gamewin page flag
     logic                          game_win_flag;
     //sides flag
     logic                  our_life_flag;
     logic [5:0]            our_life_sub;
     logic [5:0]            our_life_sub_flag;
     logic                  enemy_life_flag;
     logic [4:0]            enemy_life_sub;
     logic [4:0]            enemy_life_sub_flag;
     logic                  life_flag_flag;
     // vga pointer
     logic   [9:0]                  vcount;
logic   [10:0]                  hcount;
          // assign vcount and hcount output
          assign vcount_vga         = vcount;
          assign hcount_vga                 = hcount;
          assign brick_sub_flag     = brick_sub_flag_vga;
          assign tank1                      = tank1_flag_vga;
          assign tank2                      = tank2_flag_vga;
          assign generate_enemy     = tank_enemy_flag_vga;
```

```
                    assign game_stage_status          = game_stage_vga;
                    assign bullet                             = bullet_flag_vga;
                    assign tank_generation       = tank_generation_flag_vga;
                    assign tank_explosion              = tank_explosion_flag_vga;
                    assign tank1_generation     = tank1_generation_flag_vga;
                    assign tank1_explosion             = tank1_explosion_flag_vga;
                    assign tank2_generation     = tank2_generation_flag_vga;
                    assign tank2_explosion             = tank2_explosion_flag_vga;
                    assign enemy_life_sub_flag = enemy_life_sub_flag_vga;
                    assign our_life_sub_flag  = our_life_sub_flag_vga;



                    //vga display controller
                    vga_led_display led_led_display(.clk50(clk), .*);
                    map1 map1(.clk(clk),.*);
                    //enemy tank mif rom
                    tank_enemy_up tank_enemy_up(.address(sprite_tank_addr),       .clock(clk),.q(sprite_tank_enemy_up_data));
                    tank_enemy_down                                                  tank_enemy_down(.address(sprite_tank_addr),
.clock(clk),.q(sprite_tank_enemy_down_data));
                    tank_enemy_left tank_enemy_left(.address(sprite_tank_addr),     .clock(clk),.q(sprite_tank_enemy_left_data));
                    tank_enemy_right tank_enemy_right(.address(sprite_tank_addr), .clock(clk),.q(sprite_tank_enemy_right_data));
                    // generation and explosion mif rom
                    generation1 generation1(.address(sprite_tank_addr), .clock(clk),.q(sprite_generation1_data));
                    generation2 generation2(.address(sprite_tank_addr), .clock(clk),.q(sprite_generation2_data));
                    explosion1 explosion1(.address(sprite_tank_addr),   .clock(clk),.q(sprite_explosion1_data));
                    explosion2 explosion2(.address(sprite_tank_addr),   .clock(clk),.q(sprite_explosion2_data));
                    // player1 tank mif rom
                    tank1_up tankup_draw1(.address(sprite_tank_addr), .clock(clk),.q(sprite_tank1up_data));
                    tank1_down tankdown_draw1(.address(sprite_tank_addr), .clock(clk),.q(sprite_tank1down_data));
                    tank1_left tankleft_draw1(.address(sprite_tank_addr),       .clock(clk),.q(sprite_tank1left_data));
                    tank1_right tankright_draw1(.address(sprite_tank_addr),    .clock(clk),.q(sprite_tank1right_data));
                    // player2 tank mif rom
                    tank2_up tankup_draw2(.address(sprite_tank_addr), .clock(clk),.q(sprite_tank2up_data));
                    tank2_down tankdown_draw2(.address(sprite_tank_addr), .clock(clk),.q(sprite_tank2down_data));
                    tank2_left tankleft_draw2(.address(sprite_tank_addr),       .clock(clk),.q(sprite_tank2left_data));
                    tank2_right tankright_draw2(.address(sprite_tank_addr),    .clock(clk),.q(sprite_tank2right_data));
                    //welcome screen
                    begin1player begin1player(.address(sprite_oneplayer_addr),.clock(clk),.q(sprite_oneplayer_data));
                    game_win game_win(.address(sprite_game_win_addr),.clock(clk),.q(sprite_game_win_data));
                    //sides mif rom
                    life_flag    life_flag_draw(.address(sprite_life_flag_addr),.clock(clk),.q(sprite_life_flag_data));
/*Draw block*/
always @ ( hcount||vcount )
  begin
    /* judge the flag */
  //welcome screen
  oneplayer=((hcount>=11'd 320)&(hcount<=11'd 831)&(vcount>=10'd 128)&(vcount<=10'd 383) && (game_stage_status == 3'd0));
// gamewin screen
  game_win_flag = ((hcount>=11'd 320)&(hcount<=11'd 831)&(vcount>=10'd 128)&(vcount<=10'd 383) &&(game_stage_status ==
```

```
3'd2));
            //sides
        life_flag_flag = (((hcount>=11'd 1152)&(hcount<=11'd 1215)&(vcount>=10'd 256)&(vcount<=10'd 287))
                                        || ((hcount>=11'd 1152)&(hcount<=11'd 1215)&(vcount>=10'd 64)&(vcount<=10'd 95)));
        our_life_sub[0]=    ((hcount>=11'd    1152)&(hcount<=11'd    1215)&(vcount>=10'd    288)&(vcount<=10'd    319))    &&
(our_life_sub_flag[0]) ;
        our_life_sub[1]=    ((hcount>=11'd    1216)&(hcount<=11'd    1279)&(vcount>=10'd    288)&(vcount<=10'd    319))    &&
(our_life_sub_flag[1]);
        our_life_sub[2]=    ((hcount>=11'd    1152)&(hcount<=11'd    1215)&(vcount>=10'd    320)&(vcount<=10'd    351))    &&
(our_life_sub_flag[2]);
        our_life_flag= |our_life_sub;

    enemy_life_sub[0]=    ((hcount>=11'd    1152)&(hcount<=11'd    1215)&(vcount>=10'd    96)&(vcount<=10'd    127))    &&
(enemy_life_sub_flag[0]);
        enemy_life_sub[1]=    ((hcount>=11'd    1216)&(hcount<=11'd    1279)&(vcount>=10'd    96)&(vcount<=10'd    127))    &&
(enemy_life_sub_flag[1]);
        enemy_life_sub[2]=    ((hcount>=11'd    1152)&(hcount<=11'd    1215)&(vcount>=10'd    128)&(vcount<=10'd    159))    &&
(enemy_life_sub_flag[2]);
        enemy_life_sub[3]=    ((hcount>=11'd    1216)&(hcount<=11'd    1279)&(vcount>=10'd    128)&(vcount<=10'd    159))    &&
(enemy_life_sub_flag[3]);
        enemy_life_sub[4]=    ((hcount>=11'd    1152)&(hcount<=11'd    1215)&(vcount>=10'd    160)&(vcount<=10'd    191))    &&
(enemy_life_sub_flag[4]);
        enemy_life_flag= |enemy_life_sub;


   /* vga data from rom */

        /*welcome screen*/
        if(game_stage_status == 3'd0 ||game_stage_status == 3'd3 ) begin
            if(oneplayer) begin
                    VGA_B_oneplayer = sprite_oneplayer_data[7:0];
                    VGA_G_oneplayer = sprite_oneplayer_data[15:8];
                    VGA_R_oneplayer = sprite_oneplayer_data[23:16];
                    sprite_oneplayer_addr=(hcount-11'd 320)/2+(vcount-10'd 128)*256;
            end
    end

        // gamewin screen
        if(game_win_flag) begin
                    VGA_B_game_win = sprite_game_win_data[7:0];
                    VGA_G_game_win = sprite_game_win_data[15:8];
                    VGA_R_game_win = sprite_game_win_data[23:16];
                    sprite_game_win_addr=(hcount-11'd 320)/2+(vcount-10'd 128)*256;
        end

            //sides vga data
        if(our_life_flag) begin
                    VGA_B_our_life = sprite_tank1up_data[7:0];
                    VGA_G_our_life = sprite_tank1up_data[15:8];
```

```
                        VGA_R_our_life = sprite_tank1up_data[23:16];

                        sprite_tank_addr=hcount[5:1]+vcount[4:0]*32;
end
if(enemy_life_flag) begin
                        VGA_B_enemy_life = sprite_tank_enemy_up_data[7:0];
                        VGA_G_enemy_life = sprite_tank_enemy_up_data[15:8];
                        VGA_R_enemy_life = sprite_tank_enemy_up_data[23:16];
                        sprite_tank_addr =hcount[5:1]+vcount[4:0]*32;
end
if(life_flag_flag) begin
                        VGA_B_life_flag = sprite_life_flag_data[7:0];
                        VGA_G_life_flag = sprite_life_flag_data[15:8];
                        VGA_R_life_flag = sprite_life_flag_data[23:16];
                        sprite_life_flag_addr =hcount[5:1]+vcount[4:0]*32;
end

//player1 tank
 if(tank1) begin
          if(tank_direction_vga==2'd0) begin
          VGA_B_tank1= sprite_tank1up_data[7:0];
          VGA_G_tank1= sprite_tank1up_data[15:8];
          VGA_R_tank1= sprite_tank1up_data[23:16];
            end
   if(tank_direction_vga==2'd1) begin
          VGA_B_tank1= sprite_tank1down_data[7:0];
          VGA_G_tank1= sprite_tank1down_data[15:8];
          VGA_R_tank1= sprite_tank1down_data[23:16];
            end
            if(tank_direction_vga==2'd2) begin
          VGA_B_tank1= sprite_tank1left_data[7:0];
          VGA_G_tank1= sprite_tank1left_data[15:8];
          VGA_R_tank1= sprite_tank1left_data[23:16];
            end
   if(tank_direction_vga==2'd3) begin
                  VGA_B_tank1= sprite_tank1right_data[7:0];
          VGA_G_tank1= sprite_tank1right_data[15:8];
          VGA_R_tank1= sprite_tank1right_data[23:16];
            end

     sprite_tank_addr=(hcount-tank_x_vga)/2+(vcount-tank_y_vga)*32;
end

   //player2 tank
   if(tank2) begin
          if(tank_direction_vga==2'd0) begin
          VGA_B_tank2= sprite_tank2up_data[7:0];
          VGA_G_tank2= sprite_tank2up_data[15:8];
          VGA_R_tank2= sprite_tank2up_data[23:16];
```

```
            end
    if(tank_direction_vga==2'd1) begin
            VGA_B_tank2= sprite_tank2down_data[7:0];
            VGA_G_tank2= sprite_tank2down_data[15:8];
            VGA_R_tank2= sprite_tank2down_data[23:16];
            end
            if(tank_direction_vga==2'd2) begin
            VGA_B_tank2= sprite_tank2left_data[7:0];
            VGA_G_tank2= sprite_tank2left_data[15:8];
            VGA_R_tank2= sprite_tank2left_data[23:16];
            end
    if(tank_direction_vga==2'd3) begin
                    VGA_B_tank2= sprite_tank2right_data[7:0];
            VGA_G_tank2= sprite_tank2right_data[15:8];
            VGA_R_tank2= sprite_tank2right_data[23:16];
            end

     sprite_tank_addr=(hcount-tank_x_vga)/2+(vcount-tank_y_vga)*32;
  end
     // explosion
         if(tank_generation[0] || tank1_generation[0] || tank2_generation[0]) begin
            VGA_B_generation1= sprite_generation1_data[7:0];
            VGA_G_generation1= sprite_generation1_data[15:8];
            VGA_R_generation1= sprite_generation1_data[23:16];
                    sprite_tank_addr=(hcount[5:1]-tank_x_vga[5:1])+(vcount[4:0]-tank_y_vga[4:0])*32;
            end
     else if(tank_generation[1] || tank1_generation[1] || tank2_generation[1]) begin
            VGA_B_generation2 = sprite_generation2_data[7:0];
            VGA_G_generation2= sprite_generation2_data[15:8];
            VGA_R_generation2= sprite_generation2_data[23:16];
                    sprite_tank_addr=(hcount[5:1]-tank_x_vga[5:1])+(vcount[4:0]-tank_y_vga[4:0])*32;
            end
            else if(tank_explosion[0] || tank1_explosion[0] || tank2_explosion[0]) begin
            VGA_B_explosion1= sprite_explosion1_data[7:0];
            VGA_G_explosion1= sprite_explosion1_data[15:8];
            VGA_R_explosion1= sprite_explosion1_data[23:16];
                    sprite_tank_addr=(hcount[5:1]-tank_x_vga[5:1])+(vcount[4:0]-tank_y_vga[4:0])*32;
            end
            else if(tank_explosion[1] || tank1_explosion[1] || tank2_explosion[1]) begin
            VGA_B_explosion2= sprite_explosion2_data[7:0];
            VGA_G_explosion2= sprite_explosion2_data[15:8];
            VGA_R_explosion2= sprite_explosion2_data[23:16];
                    sprite_tank_addr=(hcount[5:1]-tank_x_vga[5:1])+(vcount[4:0]-tank_y_vga[4:0])*32;
            end


//enemy tank generate
if (generate_enemy) begin
    if(tank_direction_vga==2'd0) begin //up
```

```
                VGA_B_enemy_tank= sprite_tank_enemy_up_data[7:0];
                VGA_G_enemy_tank= sprite_tank_enemy_up_data[15:8];
                VGA_R_enemy_tank= sprite_tank_enemy_up_data[23:16];
                  end
            if(tank_direction_vga==2'd1) begin //down
                VGA_B_enemy_tank= sprite_tank_enemy_down_data[7:0];
                VGA_G_enemy_tank= sprite_tank_enemy_down_data[15:8];
                VGA_R_enemy_tank= sprite_tank_enemy_down_data[23:16];
                  end
            if(tank_direction_vga==2'd2) begin//left
                VGA_B_enemy_tank= sprite_tank_enemy_left_data[7:0];
                VGA_G_enemy_tank= sprite_tank_enemy_left_data[15:8];
                VGA_R_enemy_tank= sprite_tank_enemy_left_data[23:16];
                  end
            if(tank_direction_vga==2'd3) begin//right
                      VGA_B_enemy_tank= sprite_tank_enemy_right_data[7:0];
                VGA_G_enemy_tank= sprite_tank_enemy_right_data[15:8];
                VGA_R_enemy_tank= sprite_tank_enemy_right_data[23:16];
                  end
        sprite_tank_addr=(hcount-tank_x_vga)/2+(vcount-tank_y_vga)*32;
              end

          end


endmodule
```

```
/*
 * Battle Tank Game VGA Display
 *
 * Team: Battle Tank
 * Columbia University
 */

module vga_led_display(
        input logic        clk50, reset,
        input logic   [7:0] VGA_R_tank1, VGA_G_tank1, VGA_B_tank1,
        input logic   [7:0] VGA_R_tank2, VGA_G_tank2, VGA_B_tank2,
        // input for enemy tank
        input logic   [7:0] VGA_R_enemy_tank, VGA_G_enemy_tank, VGA_B_enemy_tank,
        input logic   [7:0] VGA_R_brick, VGA_G_brick, VGA_B_brick,
        input logic   [7:0] VGA_R_grass, VGA_G_grass, VGA_B_grass,
        input logic   [7:0] VGA_R_water, VGA_G_water, VGA_B_water,
        input logic   [7:0] VGA_R_steel, VGA_G_steel, VGA_B_steel,
        input logic   [7:0] VGA_R_homebase, VGA_G_homebase, VGA_B_homebase,
        input logic   [7:0] VGA_R_oneplayer, VGA_G_oneplayer, VGA_B_oneplayer,
        input logic   [7:0] VGA_R_game_win, VGA_G_game_win, VGA_B_game_win,
        input logic   [7:0] VGA_R_generation1, VGA_G_generation1, VGA_B_generation1,
        input logic   [7:0] VGA_R_generation2, VGA_G_generation2, VGA_B_generation2,
        input logic   [7:0] VGA_R_explosion1, VGA_G_explosion1, VGA_B_explosion1,
        input logic   [7:0] VGA_R_explosion2, VGA_G_explosion2, VGA_B_explosion2,
        input logic   [7:0] VGA_R_our_life, VGA_G_our_life, VGA_B_our_life,
        input logic   [7:0] VGA_R_enemy_life, VGA_G_enemy_life, VGA_B_enemy_life,
        input logic   [7:0] VGA_R_life_flag, VGA_G_life_flag, VGA_B_life_flag,
       /*flag*/
        input  logic                tank1,
        input  logic                tank2,
        input  logic                brick,//
        input  logic                grass,
        input logic                 water,
        input logic                 steel,
        input logic                 homebase,
        input logic                 grey_background,
        input logic                 bullet,
        input logic                 generate_enemy,
        input logic   [2:0]                 game_stage_status,
        input logic                 oneplayer,
        input logic                                 game_win_flag,
        input logic   [1:0]                 tank_generation,
        input logic [1:0]                   tank_explosion,
        input logic   [1:0]                 tank1_generation,
        input logic [1:0]                   tank1_explosion,
        input logic   [1:0]                 tank2_generation,
        input logic [1:0]                   tank2_explosion,
        input logic                 our_life_flag,
        input logic                 enemy_life_flag,
```

```
    input logic                 life_flag_flag,
     output logic [7:0]                 VGA_R, VGA_G, VGA_B,
     output logic                       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n,
     output logic [9:0]                     vcount,
     output logic [10:0]                        hcount);

parameter HACTIVE       = 11'd 1280,
          HFRONT_PORCH = 11'd 32,
          HSYNC         = 11'd 192,
          HBACK_PORCH   = 11'd 96,
          HTOTAL        = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; //1600

parameter VACTIVE       = 10'd 480,
          VFRONT_PORCH = 10'd 10,
          VSYNC         = 10'd 2,
          VBACK_PORCH   = 10'd 33,
          VTOTAL        = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; //525

logic                           endOfLine;

always_ff @(posedge clk50 or posedge reset)
  if (reset)            begin
                        hcount <= 0;
                                            end
  else if (endOfLine) hcount <= 0;
  else                  begin
                          hcount <= hcount + 11'd 1;
                                        end

assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic                           endOfField;

always_ff @(posedge clk50 or posedge reset)
  if (reset)            begin
                              vcount <= 0;
                                            end
  else if (endOfLine)
       begin
    if (endOfField)     vcount <= 0;
    else            begin
                              vcount <= vcount + 10'd 1;
                                        end
          end

assign endOfField = vcount == VTOTAL - 1;
assign VGA_HS = !( (hcount[10:7] == 4'b1010) & (hcount[6] | hcount[5]));
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
```

```
    assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for VGA
    assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                         !( vcount[9] | (vcount[8:5] == 4'b1111) );
    assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge

    always_comb begin
        {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Black
            if(game_stage_status == 3'd0) begin
                                if(oneplayer)
                                  {VGA_R, VGA_G, VGA_B} = {VGA_R_oneplayer, VGA_G_oneplayer, VGA_B_oneplayer};

            end
            else if(game_stage_status == 3'd1)       begin
                                if(our_life_flag)
                                  {VGA_R, VGA_G, VGA_B} = {VGA_R_our_life, VGA_G_our_life, VGA_B_our_life};
                                else if(enemy_life_flag)
                                  {VGA_R, VGA_G, VGA_B} = {VGA_R_enemy_life, VGA_G_enemy_life, VGA_B_enemy_life};
                                else if(life_flag_flag)
                                  {VGA_R, VGA_G, VGA_B} = {VGA_R_life_flag, VGA_G_life_flag, VGA_B_life_flag};
                                else begin
                                    if(grey_background)                                         //sprite priority up to down
                                      {VGA_R, VGA_G, VGA_B} = {8'd192, 8'd192, 8'd192};
                                    if (brick)
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_brick, VGA_G_brick, VGA_B_brick};
                                    if (water)
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_water, VGA_G_water, VGA_B_water};
                                    if (steel)
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_steel, VGA_G_steel, VGA_B_steel};
                                    if (homebase)
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_homebase, VGA_G_homebase, VGA_B_homebase};
                                    if(generate_enemy)
                                      {VGA_R,      VGA_G,      VGA_B}      =      {VGA_R_enemy_tank,      VGA_G_enemy_tank,
VGA_B_enemy_tank};
                                    if (bullet)
                                      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                                    if (tank1)
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_tank1, VGA_G_tank1, VGA_B_tank1};
                                    if (tank2)
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_tank2, VGA_G_tank2, VGA_B_tank2};
                                    if (tank_generation[0] || tank1_generation[0] || tank2_generation[0])
                                      {VGA_R,      VGA_G,      VGA_B}      =      {VGA_R_generation1,      VGA_G_generation1,
VGA_B_generation1};
                                    if (tank_generation[1] || tank1_generation[1] || tank2_generation[1])
                                      {VGA_R,      VGA_G,      VGA_B}      =      {VGA_R_generation2,      VGA_G_generation2,
VGA_B_generation2};
                                    if (tank_explosion[0] || tank1_explosion[0] || tank2_explosion[0])
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_explosion1, VGA_G_explosion1, VGA_B_explosion1};
                                    if (tank_explosion[1] || tank1_explosion[1] || tank2_explosion[1])
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_explosion2, VGA_G_explosion2, VGA_B_explosion2};
```

```
                                    if (grass & (VGA_B_grass!=8'h0) & (VGA_R_grass!=8'h0) & (VGA_G_grass!=8'h0))
                                      begin
                                      {VGA_R, VGA_G, VGA_B} = {VGA_R_grass, VGA_G_grass, VGA_B_grass};
                                      end
                                end

            end
            else if(game_stage_status == 3'd2) begin
                if(game_win_flag)
                        {VGA_R, VGA_G, VGA_B} = {VGA_R_game_win, VGA_G_game_win, VGA_B_game_win};
            end
    end

endmodule // VGA_LED_Emulator
```

```
/*
 * Battle Tank Game MAP1
 *
 * Team: Battle Tank
 * Columbia University
 */

module map1(        input logic                             clk,
                            input logic[10:0]                       hcount,
                            input logic[9:0]                vcount,
                        input logic[69:0]               brick_sub_flag, //flag=0 means the subbrick can be existed


                            output logic   [7:0]            VGA_R_brick, VGA_G_brick, VGA_B_brick,
                            output logic   [7:0]            VGA_R_grass, VGA_G_grass, VGA_B_grass,
                            output logic   [7:0]            VGA_R_water, VGA_G_water, VGA_B_water,
                            output logic   [7:0]            VGA_R_steel, VGA_G_steel, VGA_B_steel,
                            output logic   [7:0]            VGA_R_homebase, VGA_G_homebase, VGA_B_homebase,
                            output logic                brick,                  //
                            output logic                grass,
                            output logic                water,
                            output logic                steel,
                            output logic                homebase,
                            output logic                grey_background
                        );

                        /* Define variables */
                        //map obstacle address
                        logic   [10:0] sprite_brick_addr;
                        logic   [10:0] sprite_grass_addr;
                        logic   [10:0] sprite_water_addr;
                        logic   [10:0] sprite_steel_addr;
                        logic   [10:0] sprite_homebase_addr;
                        // map mif data
                        logic   [23:0] sprite_brick_data;
                        logic   [23:0] sprite_grass_data;
                        logic   [23:0] sprite_water_data;
                        logic   [23:0] sprite_steel_data;
                        logic   [23:0] sprite_homebase_data;
                         /*flag for display*/
                        logic [69:0]            brick_sub= 70'd0;           //70 subs in total 16*16
                        // map mif rom
                        brick16     brick_draw(.address(sprite_brick_addr),.clock(clk),.q(sprite_brick_data));
                        grass     grass_draw(.address(sprite_grass_addr),.clock(clk),.q(sprite_grass_data));
                        water     water_draw(.address(sprite_water_addr),.clock(clk),.q(sprite_water_data));
                        steel     steel_draw(.address(sprite_steel_addr),.clock(clk),.q(sprite_steel_data));
                        homebase     homebase_draw(.address(sprite_homebase_addr),.clock(clk),.q(sprite_homebase_data));

        /*Draw block*/
```

```
always @ ( hcount||vcount )
  begin
   /* judge the flag */
   // homebase
  homebase= (hcount>=11'd 512)&(hcount<=11'd 575)&(vcount>=10'd 416)&(vcount<=10'd 447);

  //brick
  brick_sub[0]= ((hcount>=11'd 704)&(hcount<=11'd 735)&(vcount>=10'd 32)&(vcount<=10'd 47)) && (brick_sub_flag[0]) ;
  brick_sub[1]= ((hcount>=11'd 736)&(hcount<=11'd 767)&(vcount>=10'd 32)&(vcount<=10'd 47)) && (brick_sub_flag[1]);
  brick_sub[2]= ((hcount>=11'd 704)&(hcount<=11'd 735)&(vcount>=10'd 48)&(vcount<=10'd 63)) && (brick_sub_flag[2]);
  brick_sub[3]= ((hcount>=11'd 736)&(hcount<=11'd 767)&(vcount>=10'd 48)&(vcount<=10'd 63)) && (brick_sub_flag[3]);

  //||((hcount>=11'd 896)&(hcount<=11'd 959)&(vcount>=10'd 32)&(vcount<=10'd 63))
  brick_sub[4]= ((hcount>=11'd 896)&(hcount<=11'd 927)&(vcount>=10'd 32)&(vcount<=10'd 47)) && (brick_sub_flag[4]);
  brick_sub[5]= ((hcount>=11'd 928)&(hcount<=11'd 959)&(vcount>=10'd 32)&(vcount<=10'd 47)) && (brick_sub_flag[5]);
  brick_sub[6]= ((hcount>=11'd 896)&(hcount<=11'd 927)&(vcount>=10'd 48)&(vcount<=10'd 63)) && (brick_sub_flag[6]);
  brick_sub[7]= ((hcount>=11'd 928)&(hcount<=11'd 959)&(vcount>=10'd 48)&(vcount<=10'd 63)) && (brick_sub_flag[7]);

 //||((hcount>=11'd 128)&(hcount<=11'd 191)&(vcount>=10'd 64)&(vcount<=10'd 95))
  brick_sub[8]= ((hcount>=11'd 128)&(hcount<=11'd 159)&(vcount>=10'd 64)&(vcount<=10'd 79)) && (brick_sub_flag[8]);
  brick_sub[9]= ((hcount>=11'd 160)&(hcount<=11'd 191)&(vcount>=10'd 64)&(vcount<=10'd 79)) && (brick_sub_flag[9]);
  brick_sub[10]= ((hcount>=11'd 128)&(hcount<=11'd 159)&(vcount>=10'd 80)&(vcount<=10'd 95)) && (brick_sub_flag[10]);
  brick_sub[11]= ((hcount>=11'd 160)&(hcount<=11'd 191)&(vcount>=10'd 80)&(vcount<=10'd 95)) && (brick_sub_flag[11]);

 //||((hcount>=11'd 512)&(hcount<=11'd 575)&(vcount>=10'd 64)&(vcount<=10'd 95))
  brick_sub[12]= ((hcount>=11'd 512)&(hcount<=11'd 543)&(vcount>=10'd 64)&(vcount<=10'd 79)) && (brick_sub_flag[12]);
  brick_sub[13]= ((hcount>=11'd 544)&(hcount<=11'd 575)&(vcount>=10'd 64)&(vcount<=10'd 79)) && (brick_sub_flag[13]);

  brick_sub[14]= ((hcount>=11'd 512)&(hcount<=11'd 543)&(vcount>=10'd 80)&(vcount<=10'd 95)) && (brick_sub_flag[14]);

  brick_sub[15]= ((hcount>=11'd 544)&(hcount<=11'd 575)&(vcount>=10'd 80)&(vcount<=10'd 95)) && (brick_sub_flag[15]);

  // ||((hcount>=11'd 320)&(hcount<=11'd 383)&(vcount>=10'd 96)&(vcount<=10'd 127))
  brick_sub[16]= ((hcount>=11'd 320)&(hcount<=11'd 351)&(vcount>=10'd 96)&(vcount<=10'd 111)) && (brick_sub_flag[16]);
  brick_sub[17]= ((hcount>=11'd 352)&(hcount<=11'd 383)&(vcount>=10'd 96)&(vcount<=10'd 111)) && (brick_sub_flag[17]);
  brick_sub[18]= ((hcount>=11'd 320)&(hcount<=11'd 351)&(vcount>=10'd 112)&(vcount<=10'd 127)) && (brick_sub_flag[18]);
  brick_sub[19]= ((hcount>=11'd 352)&(hcount<=11'd 383)&(vcount>=10'd 112)&(vcount<=10'd 127)) && (brick_sub_flag[19]);

//||((hcount>=11'd 704)&(hcount<=11'd 767)&(vcount>=10'd 128)&(vcount<=10'd 159))
  brick_sub[20]= ((hcount>=11'd 704)&(hcount<=11'd 735)&(vcount>=10'd 128)&(vcount<=10'd 143)) && (brick_sub_flag[20]);
  brick_sub[21]= ((hcount>=11'd 736)&(hcount<=11'd 767)&(vcount>=10'd 128)&(vcount<=10'd 143)) && (brick_sub_flag[21]);

  brick_sub[22]= ((hcount>=11'd 704)&(hcount<=11'd 735)&(vcount>=10'd 144)&(vcount<=10'd 159)) && (brick_sub_flag[22]);
  brick_sub[23]= ((hcount>=11'd 736)&(hcount<=11'd 767)&(vcount>=10'd 144)&(vcount<=10'd 159)) && (brick_sub_flag[23]);

//   ||((hcount>=11'd 1024)&(hcount<=11'd 1087)&(vcount>=10'd 128)&(vcount<=10'd 159))
  brick_sub[24]= ((hcount>=11'd 1024)&(hcount<=11'd 1055)&(vcount>=10'd 128)&(vcount<=10'd 143)) && (brick_sub_flag[24]);
```

```
        brick_sub[25]= ((hcount>=11'd 1056)&(hcount<=11'd 1087)&(vcount>=10'd 128)&(vcount<=10'd 143)) && (brick_sub_flag[25]);

        brick_sub[26]= ((hcount>=11'd 1024)&(hcount<=11'd 1055)&(vcount>=10'd 144)&(vcount<=10'd 159)) && (brick_sub_flag[26]);

        brick_sub[27]= ((hcount>=11'd 1056)&(hcount<=11'd 1087)&(vcount>=10'd 144)&(vcount<=10'd 159)) && (brick_sub_flag[27]);


//    ||((hcount>=11'd 192)&(hcount<=11'd 255)&(vcount>=10'd 192)&(vcount<=10'd 223))
        brick_sub[28]= ((hcount>=11'd 192)&(hcount<=11'd 223)&(vcount>=10'd 192)&(vcount<=10'd 207)) && (brick_sub_flag[28]);
        brick_sub[29]= ((hcount>=11'd 224)&(hcount<=11'd 255)&(vcount>=10'd 192)&(vcount<=10'd 207)) && (brick_sub_flag[29]);
        brick_sub[30]= ((hcount>=11'd 192)&(hcount<=11'd 223)&(vcount>=10'd 208)&(vcount<=10'd 223)) && (brick_sub_flag[30]);
        brick_sub[31]= ((hcount>=11'd 224)&(hcount<=11'd 255)&(vcount>=10'd 208)&(vcount<=10'd 223)) && (brick_sub_flag[31]);

       //||((hcount>=11'd 768)&(hcount<=11'd 799)&(vcount>=10'd 192)&(vcount<=10'd 223))
        brick_sub[32]= ((hcount>=11'd 768)&(hcount<=11'd 799)&(vcount>=10'd 192)&(vcount<=10'd 207)) && (brick_sub_flag[32]);
        brick_sub[33]= ((hcount>=11'd 768)&(hcount<=11'd 799)&(vcount>=10'd 208)&(vcount<=10'd 223)) && (brick_sub_flag[33]);

 //||((hcount>=11'd 128)&(hcount<=11'd 191)&(vcount>=10'd 256)&(vcount<=10'd 287))
        brick_sub[34]= ((hcount>=11'd 128)&(hcount<=11'd 159)&(vcount>=10'd 256)&(vcount<=10'd 271)) && (brick_sub_flag[34]);
        brick_sub[35]= ((hcount>=11'd 160)&(hcount<=11'd 191)&(vcount>=10'd 256)&(vcount<=10'd 271)) && (brick_sub_flag[35]);
        brick_sub[36]= ((hcount>=11'd 128)&(hcount<=11'd 159)&(vcount>=10'd 272)&(vcount<=10'd 287)) && (brick_sub_flag[36]);
        brick_sub[37]= ((hcount>=11'd 160)&(hcount<=11'd 191)&(vcount>=10'd 272)&(vcount<=10'd 287)) && (brick_sub_flag[37]);


        //||((hcount>=11'd 384)&(hcount<=11'd 447)&(vcount>=10'd 256)&(vcount<=10'd 287))
        brick_sub[38]= ((hcount>=11'd 384)&(hcount<=11'd 415)&(vcount>=10'd 256)&(vcount<=10'd 271)) && (brick_sub_flag[38]);
        brick_sub[39]= ((hcount>=11'd 416)&(hcount<=11'd 447)&(vcount>=10'd 256)&(vcount<=10'd 271)) && (brick_sub_flag[39]);
        brick_sub[40]= ((hcount>=11'd 384)&(hcount<=11'd 415)&(vcount>=10'd 272)&(vcount<=10'd 287)) && (brick_sub_flag[40]);
        brick_sub[41]= ((hcount>=11'd 416)&(hcount<=11'd 447)&(vcount>=10'd 272)&(vcount<=10'd 287)) && (brick_sub_flag[41]);

       // ||((hcount>=11'd 768)&(hcount<=11'd 831)&(vcount>=10'd 288)&(vcount<=10'd 319))
        brick_sub[42]= ((hcount>=11'd 768)&(hcount<=11'd 799)&(vcount>=10'd 288)&(vcount<=10'd 303)) && (brick_sub_flag[42]);

        brick_sub[43]= ((hcount>=11'd 800)&(hcount<=11'd 831)&(vcount>=10'd 288)&(vcount<=10'd 303)) && (brick_sub_flag[43]);

        brick_sub[44]= ((hcount>=11'd 768)&(hcount<=11'd 799)&(vcount>=10'd 304)&(vcount<=10'd 319)) && (brick_sub_flag[44]);

        brick_sub[45]= ((hcount>=11'd 800)&(hcount<=11'd 831)&(vcount>=10'd 304)&(vcount<=10'd 319)) && (brick_sub_flag[45]);

   // ||((hcount>=11'd 192)&(hcount<=11'd 255)&(vcount>=10'd 384)&(vcount<=10'd 415))
        brick_sub[46]= ((hcount>=11'd 192)&(hcount<=11'd 223)&(vcount>=10'd 384)&(vcount<=10'd 395)) && (brick_sub_flag[46]);

        brick_sub[47]= ((hcount>=11'd 224)&(hcount<=11'd 255)&(vcount>=10'd 384)&(vcount<=10'd 395)) && (brick_sub_flag[47]);

        brick_sub[48]= ((hcount>=11'd 192)&(hcount<=11'd 223)&(vcount>=10'd 396)&(vcount<=10'd 415)) && (brick_sub_flag[48]);

        brick_sub[49]= ((hcount>=11'd 224)&(hcount<=11'd 255)&(vcount>=10'd 396)&(vcount<=10'd 415)) && (brick_sub_flag[49]);
```

```
//||((hcount>=11'd 320)&(hcount<=11'd 383)&(vcount>=10'd 384)&(vcount<=10'd 415))
      brick_sub[50]=  ((hcount>=11'd  320)&(hcount<=11'd  351)&(vcount>=10'd  384)&(vcount<=10'd  399))  &&  (brick_sub_flag[50]);

      brick_sub[51]=  ((hcount>=11'd  352)&(hcount<=11'd  383)&(vcount>=10'd  384)&(vcount<=10'd  399))  &&  (brick_sub_flag[51]);

      brick_sub[52]=  ((hcount>=11'd  320)&(hcount<=11'd  351)&(vcount>=10'd  400)&(vcount<=10'd  415))  &&  (brick_sub_flag[52]);

      brick_sub[53]=  ((hcount>=11'd  352)&(hcount<=11'd  383)&(vcount>=10'd  400)&(vcount<=10'd  415))  &&  (brick_sub_flag[53]);


//    ||((hcount>=11'd 64)&(hcount<=11'd 127)&(vcount>=10'd 416)&(vcount<=10'd 447))
      brick_sub[54]= ((hcount>=11'd 64)&(hcount<=11'd 95)&(vcount>=10'd 416)&(vcount<=10'd 431)) && (brick_sub_flag[54]);

      brick_sub[55]= ((hcount>=11'd 96)&(hcount<=11'd 127)&(vcount>=10'd 416)&(vcount<=10'd 431)) && (brick_sub_flag[55]);
      brick_sub[56]= ((hcount>=11'd 64)&(hcount<=11'd 95)&(vcount>=10'd 432)&(vcount<=10'd 447)) && (brick_sub_flag[56]);
      brick_sub[57]= ((hcount>=11'd 96)&(hcount<=11'd 127)&(vcount>=10'd 432)&(vcount<=10'd 447)) && (brick_sub_flag[57]);

      //||((hcount>=11'd 706)&(hcount<=11'd 769)&(vcount>=10'd 416)&(vcount<=10'd 447))
      brick_sub[58]=  ((hcount>=11'd  706)&(hcount<=11'd  737)&(vcount>=10'd  416)&(vcount<=10'd  431))  &&  (brick_sub_flag[58]);

      brick_sub[59]= ((hcount>=11'd 738)&(hcount<=11'd 769)&(vcount>=10'd 416)&(vcount<=10'd 431)) && (brick_sub_flag[59]);
      brick_sub[60]= ((hcount>=11'd 706)&(hcount<=11'd 737)&(vcount>=10'd 432)&(vcount<=10'd 447)) && (brick_sub_flag[60]);
      brick_sub[61]=  ((hcount>=11'd  738)&(hcount<=11'd  769)&(vcount>=10'd  432)&(vcount<=10'd  447))  &&  (brick_sub_flag[61]);


     // ||((hcount>=11'd 480)&(hcount<=11'd 607)&(vcount>=10'd 400)&(vcount<=10'd 415))
      brick_sub[62]=  ((hcount>=11'd  480)&(hcount<=11'd  511)&(vcount>=10'd  400)&(vcount<=10'd  415))  &&  (brick_sub_flag[62]);

      brick_sub[63]=  ((hcount>=11'd  512)&(hcount<=11'd  543)&(vcount>=10'd  400)&(vcount<=10'd  415))  &&  (brick_sub_flag[63]);

      brick_sub[64]=  ((hcount>=11'd  544)&(hcount<=11'd  575)&(vcount>=10'd  400)&(vcount<=10'd  415))  &&  (brick_sub_flag[64]);

      brick_sub[65]=  ((hcount>=11'd  576)&(hcount<=11'd  607)&(vcount>=10'd  400)&(vcount<=10'd  415))  &&  (brick_sub_flag[65]);


//    ||((hcount>=11'd 480)&(hcount<=11'd 511)&(vcount>=10'd 416)&(vcount<=10'd 447))
      brick_sub[66]=  ((hcount>=11'd  480)&(hcount<=11'd  511)&(vcount>=10'd  416)&(vcount<=10'd  431))  &&  (brick_sub_flag[66]);

      brick_sub[67]=  ((hcount>=11'd  480)&(hcount<=11'd  511)&(vcount>=10'd  432)&(vcount<=10'd  447))  &&  (brick_sub_flag[67]);


     //||((hcount>=11'd 576)&(hcount<=11'd 607)&(vcount>=10'd 416)&(vcount<=10'd 447))
      brick_sub[68]=  ((hcount>=11'd  576)&(hcount<=11'd  607)&(vcount>=10'd  416)&(vcount<=10'd  431))  &&  (brick_sub_flag[68]);

      brick_sub[69]= ((hcount>=11'd 576)&(hcount<=11'd 607)&(vcount>=10'd 432)&(vcount<=10'd 447)) && (brick_sub_flag[69]);
      brick= |brick_sub;
      //grass
     grass= (((hcount>=11'd 384319)&(hcount<=11'd 575)&(vcount>=10'd 96)&(vcount<=10'd 159))
                              ||((hcount>=11'd 512)&(hcount<=11'd 575)&(vcount>=10'd 160)&(vcount<=10'd 191))
```

```
                        ||((hcount>=11'd 64)&(hcount<=11'd 191)&(vcount>=10'd 192)&(vcount<=10'd 255))
                        ||((hcount>=11'd 192)&(hcount<=11'd 255)&(vcount>=10'd 224)&(vcount<=10'd 255))
                        ||((hcount>=11'd 320)&(hcount<=11'd 447)&(vcount>=10'd 288)&(vcount<=10'd 255))
                        ||((hcount>=11'd 320)&(hcount<=11'd 383)&(vcount>=10'd 320)&(vcount<=10'd 383))
                        ||((hcount>=11'd 576)&(hcount<=11'd 703)&(vcount>=10'd 288)&(vcount<=10'd 383))
                        ||((hcount>=11'd 320)&(hcount<=11'd 447)&(vcount>=10'd 288)&(vcount<=10'd 319))
                        ||((hcount>=11'd 896)&(hcount<=11'd 1087)&(vcount>=10'd 384)&(vcount<=10'd 447))
                            );
    //water
    water=(((hcount>=11'd 192)&(hcount<=11'd 319)&(vcount>=10'd 64)&(vcount<=10'd 127))
                        ||((hcount>=11'd 576)&(hcount<=11'd 703)&(vcount>=10'd 96)&(vcount<=10'd 159))
                        ||((hcount>=11'd 320)&(hcount<=11'd 447)&(vcount>=10'd 160)&(vcount<=10'd 223))
                        ||((hcount>=11'd 192)&(hcount<=11'd 319)&(vcount>=10'd 256)&(vcount<=10'd 319))
                        ||((hcount>=11'd 448)&(hcount<=11'd 575)&(vcount>=10'd 288)&(vcount<=10'd 351))
                        ||((hcount>=11'd 704)&(hcount<=11'd 831)&(vcount>=10'd 320)&(vcount<=10'd 383))
                        ||((hcount>=11'd 960)&(hcount<=11'd 1087)&(vcount>=10'd 224)&(vcount<=10'd 287))
                            );
    //steel
    steel=(((hcount>=11'd 384)&(hcount<=11'd 447)&(vcount>=10'd 64)&(vcount<=10'd 95))
                        ||((hcount>=11'd 768)&(hcount<=11'd 831)&(vcount>=10'd 96)&(vcount<=10'd 127))
                        ||((hcount>=11'd 128)&(hcount<=11'd 191)&(vcount>=10'd 160)&(vcount<=10'd 191))
                        ||((hcount>=11'd 448)&(hcount<=11'd 511)&(vcount>=10'd 192)&(vcount<=10'd 223))
                        ||((hcount>=11'd 640)&(hcount<=11'd 703)&(vcount>=10'd 224)&(vcount<=10'd 255))
                        ||((hcount>=11'd 832)&(hcount<=11'd 895)&(vcount>=10'd 224)&(vcount<=10'd 255))
                        ||((hcount>=11'd 64)&(hcount<=11'd 127)&(vcount>=10'd 288)&(vcount<=10'd 319))
                        ||((hcount>=11'd 256)&(hcount<=11'd 319)&(vcount>=10'd 352)&(vcount<=10'd 383))
                        ||((hcount>=11'd 768)&(hcount<=11'd 831)&(vcount>=10'd 416)&(vcount<=10'd 447))
                            );

      grey_background=((vcount<=31)||(vcount>=448)||(hcount<=63)||(hcount>=1088));

//vga data from rom
    /*homebase*/
    if(homebase)
        begin
            VGA_B_homebase= sprite_homebase_data[7:0];
            VGA_G_homebase= sprite_homebase_data[15:8];
            VGA_R_homebase= sprite_homebase_data[23:16];
            sprite_homebase_addr=(hcount-11'd 448)/2+(vcount-10'd 416)*32;
             end

    /*      brick mif:16*16 */
             if (brick) begin
       VGA_B_brick= sprite_brick_data[7:0];
           VGA_G_brick= sprite_brick_data[15:8];
           VGA_R_brick= sprite_brick_data[23:16];
       sprite_brick_addr=hcount[4:1]+vcount[3:0]*16;
             end
```

```
        /*grass*/
         if (grass) begin
                VGA_B_grass= sprite_grass_data[7:0];
                VGA_G_grass= sprite_grass_data[15:8];
                VGA_R_grass= sprite_grass_data[23:16];
                sprite_grass_addr=hcount[5:1]+vcount[4:0]*32;
               end
        /*water*/
    if(water)begin

                VGA_B_water= sprite_water_data[7:0];
                VGA_G_water= sprite_water_data[15:8];
                VGA_R_water= sprite_water_data[23:16];
                sprite_water_addr=hcount[5:1]+vcount[4:0]*32;
               end
        /*steel*/
    if(steel)begin
                        VGA_B_steel= sprite_steel_data[7:0];
                VGA_G_steel= sprite_steel_data[15:8];
                VGA_R_steel= sprite_steel_data[23:16];
                sprite_steel_addr=hcount[5:1]+vcount[4:0]*32;
               end
        end
endmodule
```

Software code

```
/*
  * Userspace program that communicates with the led_vga device driver
  * primarily through ioctls
  *
  * Stephen A. Edwards
  * Columbia University
  */
#include <stdlib.h>
#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "global.h"
#include "joystick.h"
#include "writemessage.h"
#include "gamecontroller.h"
#include "bullet_map.h"
#include "tank_bullet.h"
#include "explosion.h"
#include "initial.h"
#include "iowrite.h"
#include "tank.h"
#include "initial_tank_generation.h"
#include "Tank_map.h"

int main()
{
// initial keyboard end

    int clear_key = 1;
    vga_led_arg_t vla;
    static const char filename[] = "/dev/vga_led";

if ( (joystick = openjoystick(&endpoint_address)) == NULL ) {
            fprintf(stderr, "Did not find a joystick\n");
                exit(1);
       }

//initial position of every object and its flag,
```

```
    printf("VGA axis Userspace program started\n");

    if ( (vga_led_fd = open(filename, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n", filename);
        return -1;
    }

    printf("initial state: ");
    print_segment_info();




    printf("current state: ");
    print_segment_info();



// initial keyboard end

        while (start==welcome){                    // wait for begin game
      gamecontroller();

                }

    // begin game ////////////////////////////////////////////////////////////////////////////////////////
    initial_game();
    unsigned int k;
    unsigned int initial_tank_i;

          while(initial_tanks==1)
{
    for (initial_tank_i=0;initial_tank_i<5;initial_tank_i++){
          generate_initial_tanks(initial_tank_i);
            iowrite();}

  if(tank_type[4]==2)
    initial_tanks=0;

}



//control the movement of every object
    while(start==begin_game) {


          gamecontroller();    // Furture need to put it in another thread
        unsigned int tank_num_i;
```

```
for (tank_num_i=0;tank_num_i<OWNTANK;tank_num_i++){      // generate for own tank
        initial_loop();


      tank(tank_num_i);


    // bullet module
    bullet_module(bullet_buffer[tank_num_i], tank_num_i);
  }


for (tank_num_i=OWNTANK;tank_num_i<MAXTANK;tank_num_i++){      // generate for enemy tank
           initial_loop();


      tank(tank_num_i);

    // bullet module
       unsigned int random_bullet = rand()%100;                                  //random for enemy tank
        if (random_bullet==0)
              bullet_buffer[tank_num_i]=shoot;
        else if (random_bullet!=0)
              bullet_buffer[tank_num_i]=0;



    bullet_module(bullet_buffer[tank_num_i], tank_num_i);

          }

    // module for explosion
          unsigned int tank_explosion_i;
        for (tank_explosion_i=0;tank_explosion_i<7;tank_explosion_i++){
            tank_explosion(tank_explosion_i);}


      //command_message update
      if (tank_enemy_remaining==0 )
         { start=gameover_win; }
      else if (tank_own_remaining==0)
         { start=gameover_lose;}


        command_message = (start<<30) + ((tank_enemy_remaining)<<20) +
((tank_own_remaining)<<17);
```

```
            iowrite();// transmit message


     }
////////////////////////////////////////////////////////////

   // game over ////////////////////////////////////////////////////////////////////
     while (1) {
           command_message = (start<<30) + ((tank_enemy_remaining)<<20) +
((tank_own_remaining)<<17);
              iowrite();// transmit message

       }


   // end of program




   printf("VGA LED Userspace program terminating\n");
   return 0;
}
```

```
#ifndef INITIAL_H_
#define INITIAL_H_


#include <stdlib.h>
#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "global.h"


void initial_game(void){
   unsigned int i;

   //initial bullet
   for (i=0;i<MAXBULLET;i++){    //initial the relative messages for bullets
      message_bullet[i]=0;
      bullet_flag[i]=0;
      bullet_x[i]=0;
      bullet_y[i]=0;
      bullet_buffer[i]=0;
      bullet_direction[i]=0;

    }

    //initial tank
    for (i=0;i<OWNTANK+ENEMYTANK;i++){ //initial the relative messages for tanks
     message_tank[i]=0;

      }



    //initial subbrick(temporate reserved)
    for (i=0;i<3;i++) {      //temporate used for subbrick flag
     message_subbrick[i]=0xFFFFFFFF;
   }

       for(i=0;i<Numsubbrick;i++){
       sub_brick_flag[i]=1;}

    // initial iomessage
    for (i=0;i<maxmessage;i++)
       iomessage[i]=0;
```

```
}



// This function for initialing at every loop beginning and every use for bullet_map and tank_map
void initial_loop(void){
       //initial bullet map
      unsigned int i;
       bullet_map_homebase=0;
    bullet_map_steel=0;
    bullet_map_grey_background=0;
    bullet_map_tank=0;
      for (i=0; i<OWNTANK+ENEMYTANK;i++ ){
          bullet_map_tank_sub[i]=0;
      }

    bullet_map_brick=0;
      for (i=0;i<Numsubbrick;i++){
          bullet_map_brick_sub[i]=0;
      }

    bullet_map_bullet=0;
      for(i=0;i<MAXBULLET;i++){
          bullet_map_bullet_sub[i]=0;
    }

}



#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include "global.h"

    void initial_generate_config(unsigned int tank_num)
     {
          switch(tank_num)
                {      case 0:
                            generate_one_tank(320,416,up,tank_num);
                           break;//player1
                       case 1:
                            generate_one_tank(640,416,up,tank_num);
                           break;//player2
                       case 2:
                            generate_one_tank(64,32,down,tank_num);
                           break;//enemy 1
                       case 3:
                            generate_one_tank(512,32,down,tank_num);
                           break;//enemy 2
                       case 4:
                           generate_one_tank(1024,32,down,tank_num);
                           break;//enemy 3
                       default:
                                 break;

                }
      }// This function defines the initial 5 tanks birthplace: 2 players + 3 enemy

void generate_initial_tanks(unsigned int tank_num)
{
      {
                       if(explosion_flag[tank_num]==no_explosion)

generate_one_tank(tank_x[tank_num],tank_y[tank_num],tank_direction[tank_num],tank_num);

                       else if(explosion_flag[tank_num]==Generate2)

generate_one_tank(tank_x[tank_num],tank_y[tank_num],tank_direction[tank_num],tank_num);

                       else if(explosion_flag[tank_num]==Generate1)
                       {    if( generate_delay==Generate_Delay)
                          initial_generate_config(tank_num);
                     else

generate_one_tank(tank_x[tank_num],tank_y[tank_num],tank_direction[tank_num],tank_num);

                       }
                          message_tank[tank_num]=                (tank_flag[tank_num]<<28)                +
(tank_direction[tank_num]<<25)                                                                              +
```

```
(tank_type[tank_num]<<21)+(tank_x[tank_num]<<10)+tank_y[tank_num];
        }
}// This function is used to generate the 5 tanks when the game begin.
```

```
#ifndef gamecontroller_h
#define gamecontroller_h

#include <stdlib.h>
#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "global.h"
#include "joystick.h"




struct usb_joystick_packet packet;
int transferred;
struct libusb_device_handle *joystick;
uint8_t endpoint_address;


void gamecontroller (){



    libusb_interrupt_transfer(joystick, endpoint_address,(unsigned char *) &packet,
sizeof(packet),&transferred, 10000);
                if (transferred == sizeof(packet)) {

        if(packet.keycode[3]==0x00||packet.keycode[3]==0x10)//up
        tank_direction_buffer[0]=up;
        else if(packet.keycode[3]==0x04||packet.keycode[3]==0x14)//down
        tank_direction_buffer[0]=down;
        else if (packet.keycode[3]==0x06||packet.keycode[3]==0x16)//left
        tank_direction_buffer[0]=left;
        else if (packet.keycode[3]==0x02||packet.keycode[3]==0x12)//right
        tank_direction_buffer[0]=right;
        else if (packet.keycode[3]==0x0f)//stop
        tank_direction_buffer[0]=stop;

        if(packet.keycode[3]==0x1f||packet.keycode[3]==0x10||packet.keycode[3]==0x14
                        ||packet.keycode[3]==0x16||packet.keycode[3]==0x12)//shoot
        bullet_buffer[0]=shoot;
        else
```

```
        bullet_buffer[0]=0;

        if(packet.keycode[4]==0x20&&start_flag==1)//game start button
            {    start=begin_game;
                 start_flag=0;
            }

                        }



}

#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"
#include "Tank_map.h"

/*      The standard of the transmission message of Tank                        */
// use 32 bit message to transmit the tank info.: 0~9: y position; 10~20: x position; 21~24: Tank Type; 25~27: Tank Direction; 28: TankFlag    31~29: Reserved
// This file defines the tank's generation, direction and moving modules.

//////////////////////////////////////////////////////tank_module_begin////////////////////////////////////
//////////

void generate_one_tank(unsigned int x, unsigned int y,unsigned int direction, unsigned int tank_num)
{
      if(explosion_flag[tank_num]==no_explosion)
          tank_type[tank_num]=tank_final_state[tank_num];

           else if(explosion_flag[tank_num]==Generate2)
                {          tank_type[tank_num]=Generate2;
                        if (generate_delay==0){
                           explosion_flag[tank_num]=no_explosion;
                           generate_delay=Generate_Delay;}
                        else if (generate_delay!=0)
                           generate_delay--;
                }
           else if( explosion_flag[tank_num]==Generate1)
           {
           tank_flag[tank_num]=Tank_exist;
           tank_x[tank_num]=x;
           tank_y[tank_num]=y;
           tank_x_temp[tank_num]=x;
           tank_y_temp[tank_num]=y;
           tank_direction[tank_num]=direction;

                      if(tank_num!=0 && tank_num!=1)
           tank_direction_buffer[tank_num]=direction;
           tank_type[tank_num]=Generate1;

           if (generate_delay==0){
                        explosion_flag[tank_num]=Generate2;
                        generate_delay=Generate_Delay;}
                     else if (generate_delay!=0)
                        generate_delay--;
           }

}//This is the tank generation function unit. Later it'll be called to generate each single tank.
```

```
void generate_config(unsigned int tank_num)
{
	unsigned int random = rand()%100;
		if(tank_own_remaining > 0 )
			{
				if(tank_num==0 )
			generate_one_tank(320,416,up,tank_num);

		 else if (tank_num==1){
				tank_direction_buffer[tank_num]=0;
		 generate_one_tank(640,416,up,tank_num);
		}
		 }

	 if((tank_enemy_remaining>0) && (tank_num !=Player1_Tank) &&(tank_num !=Player2_Tank))
		 {
			if(random<33){
		generate_one_tank(64,32,down,tank_num);
			}

		 else if(random<66){
			generate_one_tank(512,32,down,tank_num);
	    }

		 else if (random<100){
			generate_one_tank(1024,32,down,tank_num); }
		 }
```

}//This function defines the 5 birth places,1 for each player, 3 for enemy. Tanks will be generated in a place accoding to their number.

```
void tank_generate(unsigned int tank_num)
{		if(tank_type[tank_num]==notank){
			tank_flag[tank_num]=Tank_miss;
			return;}
	if((tank_num<OWNTANK && tank_own_remaining > 0 )||(OWNTANK<=tank_num &&
```

```
tank_num<MAXTANK                    &&tank_enemy_remaining>4 ) )
                {
                        if(explosion_flag[tank_num]==no_explosion)

generate_one_tank(tank_x[tank_num],tank_y[tank_num],tank_direction[tank_num],tank_num);

                  else if(explosion_flag[tank_num]==Generate2){

generate_one_tank(tank_x[tank_num],tank_y[tank_num],tank_direction[tank_num],tank_num);

                }
                        else if(explosion_flag[tank_num]==Generate1) {
                       if( generate_delay==Generate_Delay)
                     generate_config(tank_num);
                  else

generate_one_tank(tank_x[tank_num],tank_y[tank_num],tank_direction[tank_num],tank_num);
                }
             }

}// Generation process was separeted into 3 phases. Each phase has its own graph shown.
  // Genarate1, Genarate2 and no_explosion are flags indicating each phase.


void player_tank_direction( unsigned int tank_num )
{
        switch(tank_direction_buffer[tank_num])
        {
              case up:       tank_direction[tank_num]=Tank_up; break;


                      case down:    tank_direction[tank_num]=Tank_down;break;


              case left:    tank_direction[tank_num]=Tank_left;break;


                      case right: tank_direction[tank_num]=Tank_right;break;


                      case stop:    tank_direction[tank_num]= tank_direction[tank_num];    break;

                  }
}


void move_attempt(unsigned int tank_num)
{
     switch(tank_direction_buffer[tank_num])
```

```
        {   case up:      {
                tank_x_temp[tank_num]=tank_x_temp[tank_num];
                    tank_y_temp[tank_num]=tank_y_temp[tank_num]-step;
                    break;
                }
             case down:   {
                tank_x_temp[tank_num]=tank_x_temp[tank_num];
              tank_y_temp[tank_num]=tank_y_temp[tank_num]+step;
                    break;
             }
             case left:   {
             tank_x_temp[tank_num]=tank_x_temp[tank_num]-2*step;
                tank_y_temp[tank_num]=tank_y_temp[tank_num];
                    break;
             }
             case right: {
              tank_x_temp[tank_num]=tank_x_temp[tank_num]+2*step;
              tank_y_temp[tank_num]=tank_y_temp[tank_num];
              break;
             }
             case stop:   {
              tank_direction[tank_num]= tank_direction[tank_num];
                tank_x_temp[tank_num]=tank_x_temp[tank_num];
                tank_y_temp[tank_num]=tank_y_temp[tank_num];
              break;
             }
    }
}


void enemy_tank_direction(unsigned int tank_num)
    {
                        int random = rand()%100;
                    if(random<25)
                    {
                    tank_direction[tank_num]=Tank_up;
                    tank_direction_buffer[tank_num]=up;
                    }

                    else if(random<50)
                    {
                    tank_direction[tank_num]=Tank_down;
                    tank_direction_buffer[tank_num]=down;
                    }
                    else if(random<75)
                    {
                    tank_direction[tank_num]=Tank_left;
                    tank_direction_buffer[tank_num]=left;
                    }
```

```
                    else
                    {
                    tank_direction[tank_num]=Tank_right;
                    tank_direction_buffer[tank_num]=right;
                    }

    }
```
// This function is used for setting the tank's direction mainly including 2 parts: player part and enemy part.

// Player tank's direction information comes from the game controller corresponding to the 'case' part in the program.

// The enemy keeps its direction if there is no obstacle in front. Otherwise the enemy will chooses a direction based on a random value.

```
void tank_move(unsigned int tank_num)
{

        if(tank_num!=0 && tank_num!=1 && tank_stop_flag[tank_num]==1)
        enemy_tank_direction(tank_num);

        if(tank_stop_flag[tank_num] && explosion_flag[tank_num]==no_explosion)
        {
        tank_x_temp[tank_num]=tank_x[tank_num];
        tank_y_temp[tank_num]=tank_y[tank_num];

        }
        else if (explosion_flag[tank_num]==no_explosion)
        {
        tank_x[tank_num]=tank_x_temp[tank_num];
        tank_y[tank_num]=tank_y_temp[tank_num];
        }

}
```
// This function defines the moving rules of the tank. If there's no obstacle in front, tanks move as the designated direction. Otherwise, the tanks will stop.

```
void tank(unsigned int tank_num )
{
 if(explosion_flag[tank_num]==Generate1
||explosion_flag[tank_num]==Generate2||explosion_flag[tank_num]==no_explosion)
 tank_generate(tank_num);

 if(tank_num<OWNTANK)
 player_tank_direction(tank_num);

 move_attempt(tank_num);

 generate_flag(tank_num);

 tank_move(tank_num);
```

```
  ///form tank message sent to hardware///
  message_tank[tank_num]= (tank_flag[tank_num]<<28) + (tank_direction[tank_num]<<25) + (tank_type[tank_num]<<21)+(tank_x[tank_num]<<10)+tank_y[tank_num];


}// tank module: including tank's generation, set direction and move functions putting them all together.
```

```
#ifndef level1_H_
#define level1_H_


#include "global.h"
#include <math.h>


int tank_meet_flag[MAXTANK];
int sides_meet_flag;

unsigned int NotMoveover(unsigned int sprite);
//                              Y     X
unsigned int level1[15][18]=
{   //32*32 for each block        map does not have greybackground--need to add offset when giving coordinate:x-64 y-32          //8 for greybackground
   //   0       1       2       3       4       5       6       7       8       9       10    11      12      13      14      15      16      17      18    19      20      21      22      23      24      25      26      27      28      29      30      31      32
      {0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8},//0

      {0xf8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf1,0x00,0x00,0xf1,0x00,0x00,0xf8},//1

      {0xf8,0x00,0xf1,0xf4,0xf4,0x00,0xf2,0x00,0xf1,0x00,0x00,0xf1,0x00,0x00,0x00,0x00,0x00,0x00,0xf8},//2

      {0xf8,0x00,0x00,0xf4,0xf4,0xf1,0xf3,0xf3,0xf3,0xf4,0xf4,0x00,0xf2,0x00,0x00,0x00,0xf2,0xf8},//3

      {0xf8,0x00,0x00,0x00,0x00,0x00,0xf3,0xf3,0xf3,0xf4,0xf4,0xf1,0x00,0x00,0x00,0x00,0x00,0xf1,0xf8},//4

      {0xf8,0x00,0xf2,0x00,0x00,0xf4,0xf4,0x00,0xf3,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8},//5

      {0xf8,0xf3,0xf3,0xf1,0x00,0xf4,0xf4,0xf2,0x00,0x00,0x00,0x00,0xa1,0x00,0x00,0x00,0x00,0x00,0xf8},//6

      {0xf8,0xf3,0xf3,0xf3,0x00,0x00,0x00,0x00,0x00,0x00,0xf2,0x00,0x00,0xf2,0x00,0xf4,0xf4,0xf8},//7

      {0xf8,0x00,0xf1,0xf4,0xf4,0x00,0xf1,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf4,0xf4,0xf8},//8

      {0xf8,0xf2,0x00,0xf4,0xf4,0xf3,0xf3,0xf4,0xf4,0xf3,0xf3,0x00,0xf1,0x00,0x00,0x00,0x00,0xf8},//9

      {0xf8,0x00,0x00,0x00,0x00,0xf3,0x00,0xf4,0xf4,0xf3,0xf3,0xf4,0xf4,0x00,0x00,0x00,0x00,0xf8},//10

      {0xf8,0x00,0x00,0x00,0xf2,0xf3,0x00,0x00,0x00,0xf3,0xf3,0xf4,0xf4,0x00,0x00,0x00,0x00,0xf8},//11

      {0xf8,0x00,0x00,0xf1,0x00,0xf1,0x00,0x11,0x31,0x21,0x00,0x00,0x00,0x00,0x00,0xf3,0xf3,0xf3,0xf8},//12

      {0xf8,0xf1,0x00,0x00,0x00,0x00,0x00,0x51,0xf5,0xa1,0x00,0xf1,0xf2,0x00,0xf3,0xf3,0xf3,0xf8},//13

      {0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8,0xf8} //14

};
```

```
unsigned int TYPES (unsigned int block)
        {
            return block & 0x0f;
        }

unsigned int BLOCKS (unsigned int block)
        {
            return ((block) & (0xf0))>>4;
        }




//check each tank to determine whether tanks can move or not
void TankCheck(unsigned int x_coord, unsigned int y_coord,unsigned int dir, unsigned int tank_num)

{
   int TankCheck_i;
   int tank_meet_count=0;

   for(TankCheck_i=0;TankCheck_i<MAXTANK;TankCheck_i++)
   {

        if(TankCheck_i != tank_num && tank_flag[TankCheck_i]==Tank_exist)
        {
           if(dir == Tank_up)      //check tank collision when moving up
           {
               tank_meet_count             +=             ((x_coord>=tank_x[TankCheck_i])             &&
(x_coord<=tank_x[TankCheck_i]+63)                &&(y_coord>=tank_y[TankCheck_i])             &&
(y_coord<=(tank_y[TankCheck_i]+31)))          ||          (((x_coord+61)>=tank_x[TankCheck_i])          &&
((x_coord+61)<=tank_x[TankCheck_i]+63)             &&(y_coord>=tank_y[TankCheck_i])             &&
(y_coord<=(tank_y[TankCheck_i]+31))) ;

           }
           if(dir == Tank_down)    //check tank collision when moving down
           {
               tank_meet_count             +=             ((x_coord>=tank_x[TankCheck_i])             &&
(x_coord<=tank_x[TankCheck_i]+63)             &&((y_coord+31)>=tank_y[TankCheck_i])             &&
((y_coord+31)<=(tank_y[TankCheck_i]+31)))        ||        (((x_coord+61)>=tank_x[TankCheck_i])       &&
((x_coord+61)<=tank_x[TankCheck_i]+63)          &&((y_coord+31)>=tank_y[TankCheck_i])          &&
((y_coord+31)<=(tank_y[TankCheck_i]+31)) );
           }
           if(dir == Tank_left) //check tank collision when moving left
           {
               tank_meet_count             +=             ((x_coord>=tank_x[TankCheck_i])             &&
(x_coord<=tank_x[TankCheck_i]+63)                &&(y_coord>=tank_y[TankCheck_i])             &&
(y_coord<=(tank_y[TankCheck_i]+31)))          ||          ((x_coord>=tank_x[TankCheck_i])          &&
```

```
(x_coord<=tank_x[TankCheck_i]+63) &&((y_coord+31)>=tank_y[TankCheck_i]) &&
((y_coord+31)<=(tank_y[TankCheck_i]+31)));
            }
            if(dir == Tank_right) //check tank collision when moving right
            {
                tank_meet_count += (((x_coord+61)>=tank_x[TankCheck_i]) &&
((x_coord+61)<=tank_x[TankCheck_i]+63) &&(y_coord>=tank_y[TankCheck_i]) &&
(y_coord<=(tank_y[TankCheck_i]+31))) || (((x_coord+61)>=tank_x[TankCheck_i]) &&
((x_coord+61)<=tank_x[TankCheck_i]+63) &&((y_coord+31)>=tank_y[TankCheck_i]) &&
((y_coord+31)<=(tank_y[TankCheck_i]+31)));
            }
        }
   }

      if(tank_meet_count != 0)
          tank_meet_flag[tank_num]=1;
}




//updates map after bullet make brick disappear
void setBrickMap(unsigned int bullet_x_coord, unsigned int bullet_y_coord)
{
   int bullet_x_block=bullet_x_coord/64;
   int bullet_y_block=bullet_y_coord/32;
   int bullet_x_remainder=bullet_x_coord%64;
   int bullet_y_remainder=bullet_y_coord%32;

   if(TYPES(level1[bullet_y_block][bullet_x_block]) == 0x01)     //determine whether current block is
brick or not
      {   //enter=1;

            if((bullet_x_remainder>=0 && bullet_x_remainder<=31) && (bullet_y_remainder>=0 &&
bullet_y_remainder<=15))     //    brick at top left corner disappear
              level1[bullet_y_block][bullet_x_block]=level1[bullet_y_block][bullet_x_block] & 0x71;
            if((bullet_x_remainder>=0 && bullet_x_remainder<=31) && (bullet_y_remainder>=16 &&
bullet_y_remainder<=31))     //    brick at bottom left corner disappear
              level1[bullet_y_block][bullet_x_block]=level1[bullet_y_block][bullet_x_block] & 0xd1;
            if(((bullet_x_remainder>=32 && bullet_x_remainder<=63)) && (bullet_y_remainder>=0 &&
bullet_y_remainder<=15))     //    brick at top right corner disappear
              level1[bullet_y_block][bullet_x_block]=level1[bullet_y_block][bullet_x_block] & 0xb1;
            if(((bullet_x_remainder>=32 && bullet_x_remainder<=63)) && (bullet_y_remainder>=16 &&
bullet_y_remainder<=31))     //    brick at bottom right corner disappear
              level1[bullet_y_block][bullet_x_block]=level1[bullet_y_block][bullet_x_block] & 0xe1;

    }
}
```

```
//tank can not move over these things
unsigned int NotMoveover(unsigned int sprite)
{
      if(sprite==Brick || sprite==Steel || sprite==Water || sprite==Homebase || sprite==Sprite_Player1 ||
sprite==Sprite_Player2 || sprite==Greybackground)

            return 1;
      else
            return 0;
}



//check whether there is an obstacle in certain position
unsigned int TankObstacleCheck(unsigned int x_coord, unsigned int y_coord)
{
    int x_block = x_coord/64;
    int y_block = y_coord/32;

  if (TYPES(level1[y_block][x_block]) == 0x0)       // no obstacle appears
            return 0;

 else if(TYPES(level1[y_block][x_block]) == 0x1)    //the obstacle is brick and check its appearance
{
            if                    ((BLOCKS(level1[y_block][x_block])                    ==                    0x1)&&
NotMoveover(TYPES(level1[y_block][x_block])))
            {
                  if        ((x_coord)>=(x_block*64+32)        &&        (x_coord)<=(x_block*64+63)        &&
(y_coord>=(y_block*32+15)) && (y_coord<=(y_block*32+31)))
                        return 1;
                  else
                        return 0;
            }
            if                ((BLOCKS(level1[y_block][x_block])                ==                0x2)                &&
NotMoveover(TYPES(level1[y_block][x_block])))
             {
                  if        ((x_coord)>=(x_block*64)        &&        ((x_coord)<=(x_block*64+31))        &&
(y_coord>=(y_block*32+15)) && (y_coord<=(y_block*32+31)))
                     return 1;
                  else
                     return 0;
             }
            if                ((BLOCKS(level1[y_block][x_block])                ==                0x3)                &&
NotMoveover(TYPES(level1[y_block][x_block])))
            {       // enter=1;
                  if ((y_coord>=(y_block*32+15)) && (y_coord<=(y_block*32+31)))
                    return 1;
```

```
                else
                  return 0;
              }
            if              ((BLOCKS(level1[y_block][x_block])          ==          0x4)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
            {
               if((x_coord)>=(x_block*64+32) && ((x_coord)<=(x_block*64+63)) && (y_coord>=y_block*32)
&& (y_coord<=(y_block*32+14)))
                  return 1;
               else
                  return 0;
            }
            if              ((BLOCKS(level1[y_block][x_block])          ==          0x5)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
            {
               if((x_coord)>=(x_block*64+32) && ((x_coord)<=(x_block*64+63)) && (y_coord>=y_block*32)
&& (y_coord<=(y_block*32+31)))
                  return 1;
            else
                  return 0;
            }
            if              ((BLOCKS(level1[y_block][x_block])          ==          0x6)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
            {
                if(((x_coord)>=(x_block*64+32)          &&          ((x_coord)<=(x_block*64+63))          &&
(y_coord>=y_block*32) && (y_coord<=(y_block*32+14)))          //0x04
                  ||((x_coord)>=(x_block*64)          &&          ((x_coord)<=(x_block*64+31))          &&
(y_coord>=(y_block*32+15)) && (y_coord<=(y_block*32+31))))    //0x02
                  return 1;
                else
                  return 0;
            }
            if              ((BLOCKS(level1[y_block][x_block])          ==          0x7)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
            {
               if((x_coord)>=(x_block*64)  &&  ((x_coord)<=(x_block*64+31))  &&  (y_coord>=y_block*32)
&& (y_coord<=(y_block*32+14)))    //this part does not have brick
                  return 0;
               else
                  return 1;
            }
            if              ((BLOCKS(level1[y_block][x_block])          ==          0x8)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
               {
                 if((x_coord)>=(x_block*64)  &&  ((x_coord)<=(x_block*64+31))  &&  (y_coord>=y_block*32)
&& (y_coord<=(y_block*32+14)))
                  return 1;
                else
                  return 0;
```

```
            }
        if          ((BLOCKS(level1[y_block][x_block])          ==          0x9)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
        {
            if(((x_coord)>=(x_block*64) && ((x_coord)<=(x_block*64+31)) && (y_coord>=y_block*32)
&& (y_coord<=(y_block*32+14))) //0x8
              ||((x_coord)>=(x_block*64+32)      &&      (x_coord)<=(x_block*64+63)      &&
(y_coord>=(y_block*32+15)) && (y_coord<=(y_block*32+31)))) //0x1
              return 1;
             else
              return 0;
        }
        if          ((BLOCKS(level1[y_block][x_block])          ==          0xa)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
        {
          if((x_coord)>=(x_block*64+32) && ((x_coord)<=(x_block*64+63)) && (y_coord>=y_block*32)
&& (y_coord<=(y_block*32+31)))      //without brick
              return 0;
          else
              return 1;
        }
        if          ((BLOCKS(level1[y_block][x_block])          ==          0xb)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
        {
          if((x_coord)>=(x_block*64+32) && ((x_coord)<=(x_block*64+63)) && (y_coord>=y_block*32)
&& (y_coord<=(y_block*32+14)))   //without brick
              return 0;
          else
              return 1;
        }
        if          ((BLOCKS(level1[y_block][x_block])          ==          0xc)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
        {
          if ((y_coord>=(y_block*32+15)) && (y_coord<=(y_block*32+31)))          //without brick
              return 0;
          else
              return 1;
        }
        if          ((BLOCKS(level1[y_block][x_block])          ==          0xd)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
        {
          if((x_coord)>=(x_block*64)          &&          ((x_coord)<=(x_block*64+31))          &&
(y_coord>=(y_block*32+15)) && (y_coord<=(y_block*32+31)))      //without brick
              return 0;
          else
              return 1;
        }
        if          ((BLOCKS(level1[y_block][x_block])          ==          0xe)          &&
NotMoveover(TYPES(level1[y_block][x_block])))
```

```
            {
               if((x_coord)>=(x_block*64+32)          &&          (x_coord)<=(x_block*64+63)          &&
(y_coord>=(y_block*32+15)) && (y_coord<=(y_block*32+31)))    //without brick
                  return 0;
                else
                  return 1;
            }
            if              ((BLOCKS(level1[y_block][x_block])              ==              0xf)              &&
NotMoveover(TYPES(level1[y_block][x_block])))
                  return 1;
            else
                  return 0;
}


    else    if    (TYPES(level1[y_block][x_block])    ==0x2    ||    TYPES(level1[y_block][x_block])    ==4    ||
TYPES(level1[y_block][x_block]) ==5 || TYPES(level1[y_block][x_block]) ==6)
          {
                return 1;
          }
          else
         {return 0;}
}

//check whether the tank can move up or not
unsigned int cannotMoveup(unsigned int x_coord, unsigned int y_coord)
{
      return    TankObstacleCheck(x_coord,y_coord)    ||    TankObstacleCheck(x_coord+16,y_coord)    ||
TankObstacleCheck(x_coord+32,y_coord)        ||        TankObstacleCheck(x_coord+48,y_coord)        ||
TankObstacleCheck(x_coord+62,y_coord);
}

//check whether the tank can move down or not
unsigned int cannotMovedown(unsigned int x_coord, unsigned int y_coord)
{
      return TankObstacleCheck(x_coord,y_coord+31) || TankObstacleCheck(x_coord+16,y_coord+31) ||
TankObstacleCheck(x_coord+32,y_coord+31)      ||      TankObstacleCheck(x_coord+48,y_coord)      ||
TankObstacleCheck(x_coord+62,y_coord+31);

}

//check whether the tank can move left or not
unsigned int cannotMoveleft(unsigned int x_coord, unsigned int y_coord)
{
      return    TankObstacleCheck(x_coord,y_coord)    ||    TankObstacleCheck(x_coord,y_coord+8)    ||
TankObstacleCheck(x_coord,y_coord+16)        ||        TankObstacleCheck(x_coord,y_coord+24)        ||
TankObstacleCheck(x_coord,y_coord+31);
}
```

```
//check whether the tank can move right or not
unsigned int cannotMoveright(unsigned int x_coord, unsigned int y_coord)
{
      return  TankObstacleCheck(x_coord+62,y_coord)  ||  TankObstacleCheck(x_coord+62,y_coord+8)  ||
TankObstacleCheck(x_coord+62,y_coord+16)     ||     TankObstacleCheck(x_coord+62,y_coord+24)     ||
TankObstacleCheck(x_coord+62,y_coord+31)
;
}

//check whether tank can move over backgrounds or not:up down left right        call function above
void cannotTankMove (unsigned int x_coord, unsigned int y_coord, unsigned int dir)
{
                                                    // sides_meet_flag = 1 means tank cannot move over
  if(dir==Tank_up)
        sides_meet_flag= cannotMoveup(x_coord, y_coord);
  if(dir==Tank_down)
        sides_meet_flag= cannotMovedown(x_coord, y_coord);
  if(dir==Tank_left)
        sides_meet_flag= cannotMoveleft(x_coord, y_coord);
  if(dir==Tank_right)
        sides_meet_flag= cannotMoveright(x_coord, y_coord);
  //if((x_coord==62  &&  dir==3)  ||  (x_coord==1026  &&  dir==4)  ||  (y_coord==31  &&  dir==1)||
(y_coord==417 && dir==2))//check next position
     if(x_coord<=62 || x_coord>=1026 || y_coord<=31 || y_coord>=417)
        sides_meet_flag= 1;

}


//////////////////////////////////////////////generate stop flag for each tank//////////////////////////
void generate_flag(unsigned int tank_num)
{
      tank_meet_flag[tank_num] = 0;
            tank_stop_flag[tank_num] = 0;
            sides_meet_flag = 0 ;

            setBrickMap(bullet_x[tank_num], bullet_y[tank_num]);

            cannotTankMove
(tank_x_temp[tank_num],tank_y_temp[tank_num],tank_direction[tank_num]);

      TankCheck(tank_x_temp[tank_num],tank_y_temp[tank_num],tank_direction[tank_num],
tank_num);

            tank_stop_flag[tank_num] = sides_meet_flag || tank_meet_flag[tank_num];

}
```

//

#endif

```
/*
 * tank_bullet.h
 *
 *          Author: Tank_Group

*/
#ifndef BULLET_H
#define BULLET_H


#include <stdlib.h>
#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <math.h>

#include "global.h"

void bullet_module(unsigned int shoot_button, unsigned int numbullet){

   int i=0;
        unsigned int obstacle=0;
        unsigned int sub_bullet_temp[3]={0,0,0};
        // means no bullet at present
        if (shoot_button!=shoot    && bullet_flag[numbullet]==Bullet_miss ){

           return;
      }


      // initial the bullet position
         if        (shoot_button==shoot                &&        bullet_flag[numbullet]==Bullet_miss        &&
tank_flag[numbullet]==Tank_exist&&explosion_flag[numbullet]==no_explosion){
             // have shoot flag and bullets are not exist and tank exist, generate the bullet
            bullet_flag[numbullet]=Bullet_exist;

                if (tank_direction[numbullet]==Tank_up){
                    bullet_x[numbullet]=tank_x[numbullet]+32;
                    bullet_y[numbullet]=tank_y[numbullet];
                    bullet_direction[numbullet]=bullet_up;
                }
                else if(tank_direction[numbullet]==Tank_down){
                   bullet_x[numbullet]=tank_x[numbullet]+32;
                    bullet_y[numbullet]=tank_y[numbullet]+32;
```

```
                bullet_direction[numbullet]=bullet_down;
            }
            else if (tank_direction[numbullet]==Tank_left){
                bullet_x[numbullet]=tank_x[numbullet];
                bullet_y[numbullet]=tank_y[numbullet]+16;
                bullet_direction[numbullet]=bullet_left;

            }
            else if (tank_direction[numbullet]==Tank_right){
                bullet_x[numbullet]=tank_x[numbullet]+64;
                bullet_y[numbullet]=tank_y[numbullet]+16;
                bullet_direction[numbullet]=bullet_right;
            }


        //change the value of transmit bullet message

message_bullet[numbullet]=(bullet_flag[numbullet]<<21)+(bullet_x[numbullet]<<10)+bullet_y[numbullet
];

        return;
        }
        // achieve the moving of bullet
        else if (bullet_flag[numbullet]==Bullet_exist){
           if (bullet_direction[numbullet]==bullet_up){
                bullet_y[numbullet]=bullet_y[numbullet]-3;
           }
           else if (bullet_direction[numbullet]==bullet_down){
              bullet_y[numbullet]=bullet_y[numbullet]+3;
           }
           else if (bullet_direction[numbullet]==bullet_right){
              bullet_x[numbullet]=bullet_x[numbullet]+6;
           }
           else if (bullet_direction[numbullet]==bullet_left){
              bullet_x[numbullet]=bullet_x[numbullet]-6;
           }


        }

        // call the function of map    // justify the exist or miss of bullet
        bullet_map(bullet_x[numbullet] , bullet_y[numbullet], numbullet);


        obstacle=(bullet_map_brick    ||    bullet_map_grey_background    ||    bullet_map_steel
||bullet_map_tank || bullet_map_bullet || bullet_map_homebase );//tank and bullet need to change
the version!!!

        if (obstacle){                                    // meet the obstacle, bullet miss
```

```
   bullet_flag[numbullet]=Bullet_miss;
   bullet_direction[numbullet]=bullet_static;


if (bullet_map_homebase)    {                                          // homebase miss
                        start=gameover_lose;
   }


     for(i=0;i<32;i++){                              // meet subbrick, subbrick miss    temporal reserved
      if (bullet_map_brick_sub[i]!=0){
      sub_brick_flag[i]=brick_miss;

      }
     sub_bullet_temp[0]= sub_bullet_temp[0]+ (sub_brick_flag[i]<<i);


     if (bullet_map_brick_sub[i+32]!=0){
          sub_brick_flag[i+32]=brick_miss;
     }
     sub_bullet_temp[1]= sub_bullet_temp[1]+ (sub_brick_flag[i+32]<<i);
   }


     for(i=0;i<(Numsubrick%32);i++){
          if (bullet_map_brick_sub[i+64]!=0){
       sub_brick_flag[i+64]=brick_miss ;
       }
     sub_bullet_temp[2]= sub_bullet_temp[2]+ (sub_brick_flag[i+64]<<i);
   }

     // change the value of transmit homebase and subbrick
                 for (i=0;i<3;i++){
                message_subbrick[i]=sub_bullet_temp[i];
               }

  // meet tank, tank miss
  if (numbullet<OWNTANK){    // own bullet meet enemy tank
   for (i=OWNTANK;i<OWNTANK+ENEMYTANK;i++){
          if (bullet_map_tank_sub[i]==sub_tank_hit){

                explosion_flag[i]=Explosion1;

           tank_enemy_remaining=tank_enemy_remaining-1;
          }
   }
  }

else if (numbullet>OWNTANK && numbullet< (OWNTANK+ENEMYTANK)){    // enemy bullet meet
```

```
own tank
            for (i=0;i<OWNTANK;i++){
                  if (bullet_map_tank_sub[i]==sub_tank_hit){

                        explosion_flag[i]=Explosion1;
                 tank_own_remaining=tank_own_remaining-1;

                  }
             }
            }


             for (i=0; i<MAXBULLET;i++){                        // meet other bullet, other bullet miss

                  if (bullet_map_bullet_sub[i]==sub_bullet_hit){
                   bullet_flag[i]=Bullet_miss;
                   message_bullet[i]=message_bullet[i] & 0xFFDFFFFF;

                  }


                }


          }
                 //change the value of transmit bullet message

message_bullet[numbullet]=(bullet_flag[numbullet]<<21)+(bullet_x[numbullet]<<10)+bullet_y[numbullet
];


}
#endif
```

```
#ifndef EXPLOSION_H_
#define EXPLOSION_H_

#include <stdlib.h>
#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#include "global.h"


void tank_explosion(unsigned int numexplosion){

   if (explosion_flag[numexplosion]== Explosion2){
         if (generate_delay==0){
             explosion_flag[numexplosion]=Generate1;
             generate_delay=Generate_Delay;
             tank_flag[numexplosion]=Tank_miss;
             tank_type[numexplosion]=Explosion2;
             if((numexplosion>1 && tank_enemy_remaining<4) ||(numexplosion<2 &&
tank_own_remaining<0) ) // 1)the enemy tank less than 5,dont't generate the new one
                 tank_type[numexplosion]=notank;
// 2)the own tank less than 2,dont't generate the new one
              }
         else if (generate_delay!=0)
            {generate_delay--;}


       ///form tank message sent to hardware///
         message_tank[numexplosion]= (tank_flag[numexplosion]<<28) +
(tank_direction[numexplosion]<<25) +
(tank_type[numexplosion]<<21)+(tank_x[numexplosion]<<10)+tank_y[numexplosion]; // generate the
explosion of explosion2
         // return;
     return;
   }

   else if (explosion_flag[numexplosion]==Explosion1){
      if (generate_delay==0){
        explosion_flag[numexplosion]=Explosion2;
        generate_delay=Generate_Delay;
        tank_type[numexplosion]=Explosion1;}
      else if (generate_delay!=0)
             {generate_delay--;}
```

```
    ///form tank message sent to hardware///
        message_tank[numexplosion]= (tank_flag[numexplosion]<<28) +
(tank_direction[numexplosion]<<25) +
(tank_type[numexplosion]<<21)+(tank_x[numexplosion]<<10)+tank_y[numexplosion];     // generate the
explosion of explosion1

    return;
   }

    else if(explosion_flag[numexplosion]==no_explosion){
          return;
     }

}


#endif
```

```
#ifndef _WRITEMESSAGE_H
#define _WRITEMESSAGE_H

#include <stdlib.h>
#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "global.h"



/* Read and print the segment values */
void print_segment_info() {
    vga_led_arg_t vla;
    int i;

    for (i = 0 ; i < VGA_LED_DIGITS; i++) {
        vla.axis = i;
        if (ioctl(vga_led_fd, VGA_LED_READ_DIGIT, &vla)) {
            perror("ioctl(VGA_LED_READ_DIGIT) failed");
            return;
        }
        printf("%02x ", vla.direction);
    }
    printf("\n");
}

/* Write the contents of the array to the display */
void write_segments(const unsigned int dirs[VGA_LED_DIGITS])
{
    vga_led_arg_t vla;
    int i;
    for (i = 0 ; i < VGA_LED_DIGITS ; i++) {
        vla.axis = i;
        vla.direction = dirs[i];
        if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
            perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
            return;
        }
    }
}


#endif
```

```
#ifndef _IOWRITE_H
#define _IOWRITE_H

#include <stdlib.h>
#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "global.h"

void iowrite(void){
	unsigned int io_i;

	for (io_i=0;io_i<MAXTANK; io_i++){
	    iomessage [io_i]=message_tank[io_i];
   }

	for (io_i=0;io_i<MAXBULLET; io_i++){
	     iomessage [io_i+maxtankmessage]=message_bullet[io_i];
   }

	for (io_i=0;io_i<maxsubbrick; io_i++){
	     iomessage [io_i+maxtankmessage+maxbulletmessage]=message_subbrick[io_i];
   }

	  iomessage [127]=command_message;

   write_segments(iomessage);
	usleep(iodelay);

}



#endif
```