# Battle City Game

Team Name: Tank

Group Member: Han Cui(hc2737) James Thompson(jlt2160)

Li Qi(lq2156)

Instructor: Stephen Edwards

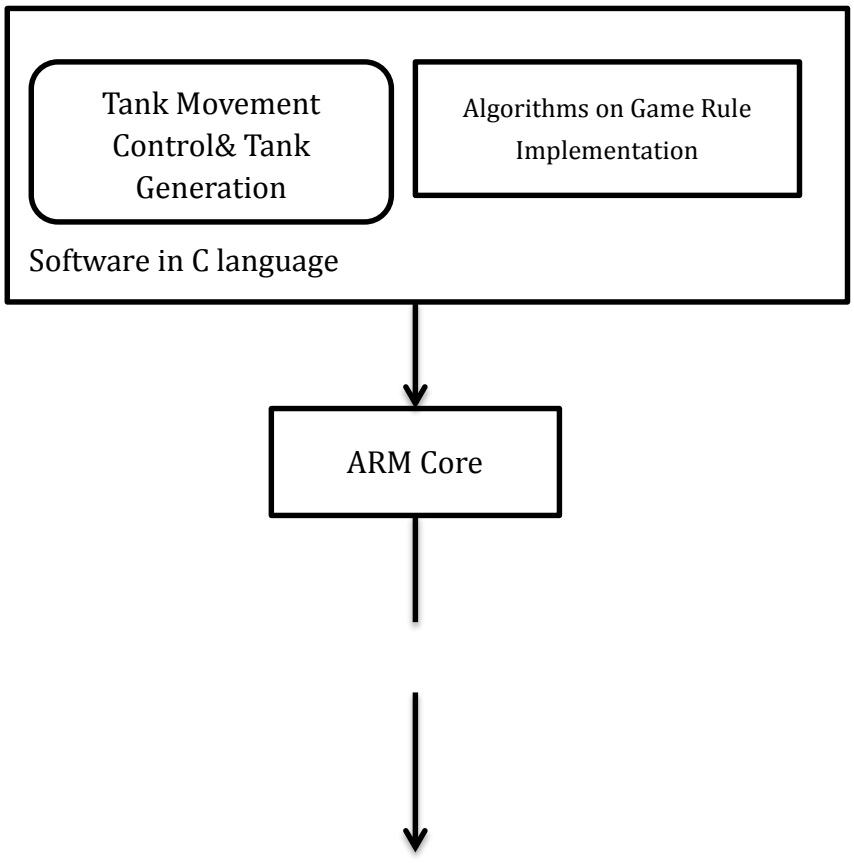
# 1.Design Overview

The project's purpose is to build the classic video game "Battle City" using synthesizable Verilog HDL and C language on a FPGA board. Users should be able to play this 3rd-person multidirectional shooting game using keyboard as controller and monitor for display. We will implement three different difficult levels, which are easy, medium and hard, to this game. Differences among these three levels are mainly presented as different moving speed and firing rate. To finish this, we need to design hardware units using with Verilog HDL for functions of configuring VGA and total game display, scoring on FPGA board, sound effects and keyboard control; as well as to design software units using C language for tank generation, movement and application of game rules.

## 1.1 Game Rules

The basic rule of the game is that players should control a tank to reach the higher level of game by destroying all the enemy tanks while keeping the home safe from enemy tanks. There are 2 enemy tanks in total to be generated at random positions on the edge of the game window in the beginning of each game. With different difficult levels, enemy tanks' moving speed and fire rate would increase drastically at higher-level games, thus harder for players to win. Player's tank would generate at a fixed position right next to the home for every game and should expect 2 chances in total to restart before the end of each round. It's considered players losing the game when they used all the chances of restarting before finishing all the levels of the game when there are totally 3 levels to pass.

## 1.2 Overall Structure Description



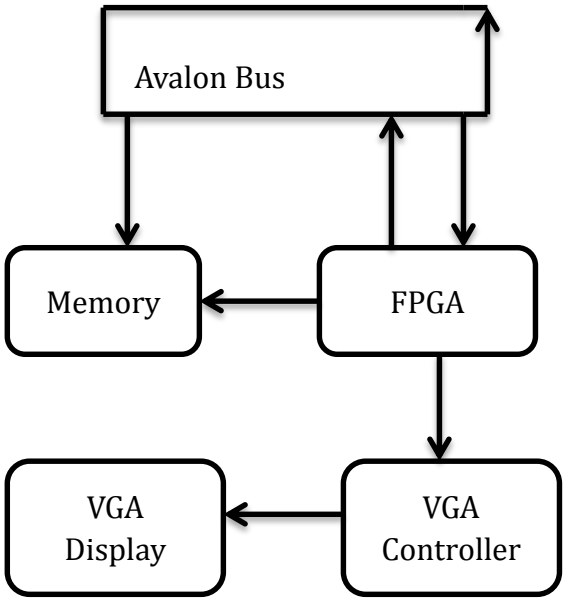



Figure 1. Overall Structure

Our design includes software part that is in C language. This part contains code to randomly generate enemy tanks on the edge of the window and depict user's tank movement, which allows users to control the tanks with keyboard; algorithms that apply the game rule, which would determine if players have died, lost or won, as well as to allow players to pause, resume, start and exit the game.

Using ARM core and Avalon bus we could bridge the software in C language and hardware designed in Verilog HDL to function together. Also the hardware contains VGA display that would generate everything belongs to the game window. And the current game window should be generated with keyboard command the previous game window stored in the memory. The function of the hardware design would be achieved on the DE2 FPGA board provided.

# 2.Hardware

### 2.1 Keyboard

Keyboard is used to control the total game process; all the functions of the game should be realized with the use of keyboard. These functions include systematic functions such as pause, resume, start and exit; as well as gaming functions such as going forward, going back, making turns and fire. The functions of the keyboard are designed both in Verilog HDL and C language. In this design, we use following keys on keyboard and their functions are also listed below:

| Keys | Functions |
|---|---|

| | |
|---|---|
| **W,A,S,D** | Move tank up, left, down and right |
| ↑ , ↓ | Move selection between start and exit |
| **Space** | Tank fires |
| **Esc** | Pause&Resume the game |

**2.2 VGA Block**

VGA is hardware designed in Verilog HDL to correctly connect the FPGA board with functions loaded to the monitor, as well as to display everything in the gaming window. VGA display would interact with memory which consists of RAM and decoder to generate display at the current moment with respect to the display stored as the previous game window. This is the core part in terms of game display.

In VGA display, there are kinds of elements to be shown:

2.2.1. User tank

2.2.2. Enemy tanks

According to the design overview, there are only two enemy tanks in each round during the game. Notice that the only difference between user tank and enemy tank is the color of appearance. The user tank is brown and the enemy tank is silver.

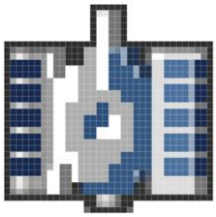


Figure2. The image of enemy tank.

2.2.3. Obstacles in the map (need modified)

We intend to design three obstacles during each ground in the map. These obstacles' position are certain in each round. These three obstacles are:

Trees: Tank can move across these trees and both tank itself and there bullets become invisible while crossing trees.

Steel: Tank cannot move across the steel or fire a bullet to destroy it.

Brick: Tank cannot move across the brick however a bullet can destroy one brick.

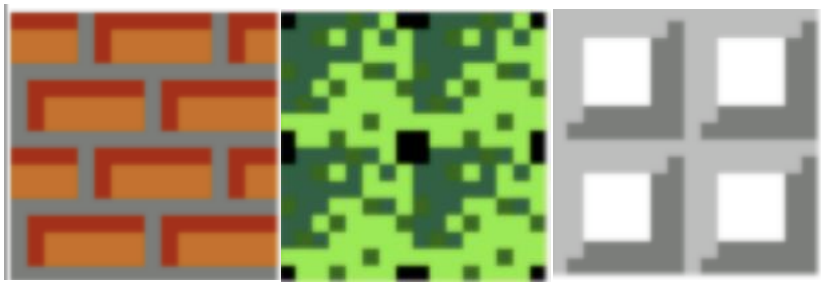


Figure 3. The image of brick, tree and steel.

2.2.4. Bullet and explosions effect.

2.2.5 Game information. Remain lives and times for example.

2.2.6. Home base

### 2.3 Memory

In a video game buffer, each pixel has 30 bits. We design that all user and enemy tanks, obstacles are in the same size. And game information will also occupy some memory. We haven't decided the final size of each element so the total memory for displaying cannot be calculated right now. We will fix this problem soon.

### 2.4 Audio Block

We will use 6 different sounds in this game. The name and description of these sounds are shown below:

| Name | Description |
|---|---|
| **Game start** | A welcome music |
| **Fire** | Tank firing |
| **Movement** | Tank moving |
| **Hit** | When tank is hit |
| **Blast** | When tank or home base is blasted |

## 3. Software(need modified)

Software is also very important in our design. We will write in C and Verilog HDL to achieve our goals in the project. We will try to achieve these functions to make the game work:

3.1. Adapting and Importing image through DPI to systemVerilog (on software side), for purpose of importing tank and wall images. During a level in each frame, the software driver will import an array of coordinates for each moving sprite in the game (the player, enemy tanks and projectiles). Background tiles will be updated in another array to account for destroyed bricks.

Levels containing the layout of bricks and starting positions of enemy tanks will be stored in text files. Potentially these can be written as ASCII files where each type of tile is represented by a letter. When a level is first loaded the driver will read from the correct numbered level file and import the map to the hardware. This allows for a simple form of level customization by means of editing the text files. If we have time we could implement a level editor in the hardware, which was included in the original Battle City arcade game.

3.2. Random generation position of AI tanks (on the edge of game window).

Pseudo code:

```
Int r = rand() % 640//assuming the window size is 640X320
Void initial_generate_config(unsigned int tank_num)
Swith(tank_num)
{  case 0;
```

```
        generate_one_tank(320,320,tank_num)// user tank
    case 1;
        generate_one_tank(r,40,tank_num)//enemy tank 1
    case 2;
       generate_one_tank(r,40,tank_num)//enemy tank 2
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[ta
nk_num],tank_num);
elseif(explosion_flag[tank_num]==Generate2)
generate_one_tank(tank_x[tank_num],tank_y[tank_num],tank_direction[ta
nk_num],tank_num);
else if(explosion_flag[tank_num]==Generate1)
```

3.3. AI tank actions. To design for different difficulty levels of game, the speed and firing rate of enemy tanks will increase as the level increases. We could use the same moving path algorithm and still have different difficulty. The AI movement algorithm we plan to use is the A-star algorithm using Manhattan Distance (i.e. 4 directions of movement) as the heuristic. The frequency of movements along the shortest path from enemy coordinate to player coordinate will be determined by the level difficulty.

3.4. User control.

Pseudo code:

```
void keyboardcontrol() {
if (transferred == sizeof(packet)) {
if “w” key is pressed//up
tank_direction_buffer[0]=up;
else if “s” key is pressed//down
tank_direction_buffer[0]=down;
else if “a” key is pressed//left
tank_direction_buffer[0]=left;
else if “d” key is pressed//right
tank_direction_buffer[0]=right;
else if “ESC” key is pressed//stop
tank_direction_buffer[0]=stop;
if “space” key is pressed//shoot
bullet_buffer[0]=shoot;
}
}
#endif
```

3.5. Determination on gaming status( game over or winning )

Player starts with a fixed number of lives (3 by default). Each time the tank or the home base is hit with a bullet a life is lost and the level restarts. This can be represented as a

value `current_lives` which decrements each time a bullet coordinate collides with a tank or home coordinate. Game over screen renders when `current_lives == 0`. The `level` counter keeps track of the current level. When both enemy tanks have been destroyed level will increment and the next level will be loaded from a text file. Game win screen renders when all levels have been completed,i.e. when `level==MAX_LEVELS`, which is currently planned to be set to 3.

5.Appendix

1. Worm Craft report from Spring 2014 CSEE 4840 class"
http://www.cs.columbia.edu/~sedwards/classes/2014/4840/reports/WormCraft.pdf
2. Avalon memory-mapped interface specifications
http://www.cs.columbia.edu/~sedwards/classes/2008/4840/mnl_avalon_spec.pdf
3. Altera's Avalon communication fabric
http://www.cs.columbia.edu/~sedwards/classes/2015/4840/avalon.pdf