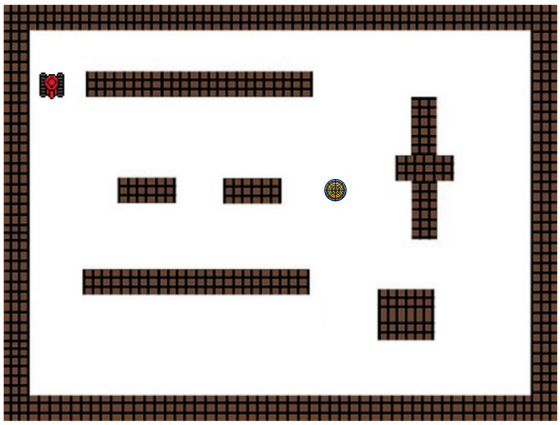
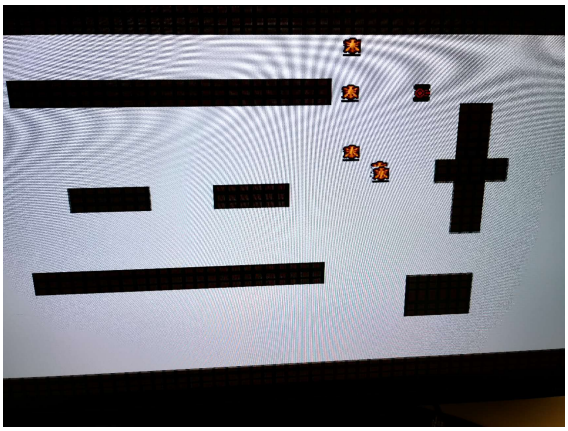
1 Introduction

Player can use the four buttons on the FPGA to independently control the tank's movement in the four directions: up, down, left, and right. Player sends command the tank to fire the laser cannon to eliminate the target.
The keys for Player are:
M18 = up direction
P18 = down direction
P17 = left direction
M17 = right direction
N17 = fire
CPU RESET = reset (start a new game)
Game Rules: The player can control the tank's movement in four directions—up, down, left, and right—using the four buttons on the FPGA. Additionally, the tank can fire a laser cannon by pressing a button.
Hardware: NEXYS A7 100T
EDA Tools: Xilinx Vivado Suite (for Synthesis, Place and Route and downloading the implemented design to the FPGA Board).



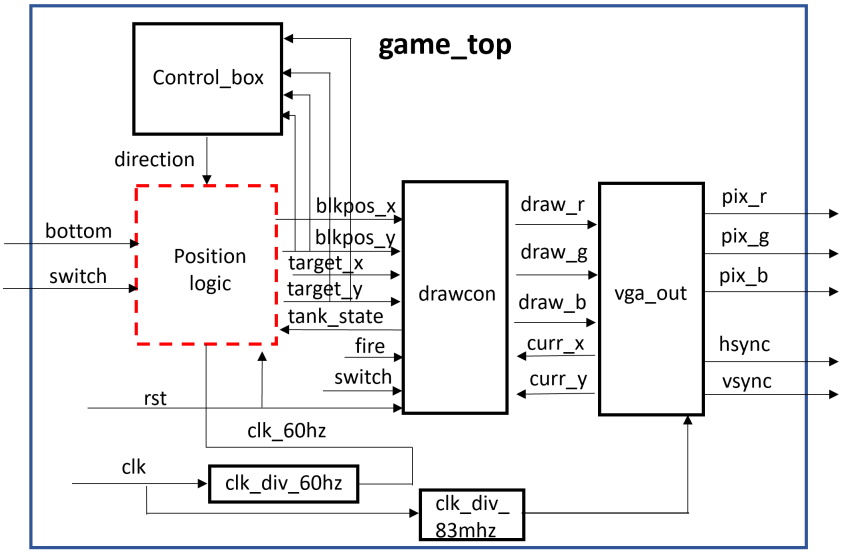
The real demonstration figure show as follows:



2 block diagram

The block diagram is shown as below. It contains several submodules, including clk_count, vga_unit, dawn_con, control_box and position_logic. I will explain individual modules below.

clk_counter: clk_counter helps us downsample the 1 MHZ input signal into the low frequency we what, which control the box movement. We adjust the frequency instead of using 60hz, as we test the box with frequency of 60hz, it still move too fast.
vga_unit: This part help display on the VGA screen. It takes RGB (red, green, blue) pixel values as inputs and generates the necessary signals for displaying these colors on a VGA screen. We just follow the orders in the tutorial and implement the block without any modifications.
drawcon: This block is the most complicated part in this project. It mainly takes blocks position and button control states as the input. In our case, we have one moving block tank, and maximum to five block targets, which we give the variable name blkpos_x to blkpos_x_6. Our tank have four different states according to our commands (up, down, left and right) so the button control states are also necessary as the input to set different pixel. When the curr_x and curr_y move to the exactly position of the block, we give the address to the pixel memory we have saved and display on the background.
position logic: The position logic controls the block position, where we take input as button command and the target state. To be noticed, the target state represents the enemy target is boomed or not as we don't expect to see a boom target is still moving. For tank position, it is control by the giving command (up, down, left and right) along with the clock cycle. After the reset button is pressed, the tank will back to the designated position.
control_box: This module is to give direction to enemy box. It is difficult to change moving strategy during the game. So we just apply the algorithm which push the enemy box approach the tank position. This block we take input as tank position and target position, and it will give output feedback as direction.



3 Some tips and our method

3.1 image generation

The images are full of three RGB channels. When we need to utilize the image as memory unit in FPGA, we need to turn the format into coe file. The image is linearized into a one-dimensional array of pixel values, and through the pixel value, it is encoded into hexadecimal. We implement it with python PIL package. The part of the code is shown below:

```python
def image_to_coe(image_path, output_path, width, height):
    # Open the image
    image = Image.open(image_path)
    # Resize the image
    image = image.resize((width, height))

    # Convert the image to a numpy array
    image_data = np.array(image)
    # image_data=image_data[3:4,:,:]
    print(image_data.shape)

    # Check if the image is not already in 'RGB' mode
    if image_data.shape[2] != 3:
        raise ValueError("Image must be in RGB format")

    # Prepare the .coe file header
    header = "; Sample .COE file\n"
    header += "memory_initialization_radix=16;\n"
    header += "memory_initialization_vector=\n"

    # Convert the pixel values to 12-bit hexadecimal and prepare the data
    data_str = ""
    for row in image_data:
        for pixel in row:
            # Scale each RGB value to 4 bits and combine them to get a 12
            r, g, b = [format(val >> 4, '1X') for val in pixel]  # Scale
            rgb_12bit = r + g + b
            data_str += rgb_12bit + ",\n"

    # Remove the last comma and newline
    data_str = data_str.rstrip(",\n")

    # Write to the .coe file
    with open(output_path, 'w') as file:
        file.write(header + data_str + ";")
```

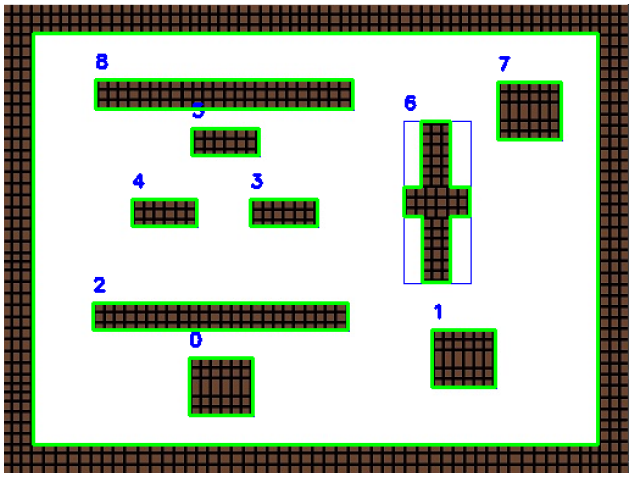
3.2 background setup

For the background setup, the display screen has the of 1280 $\times$ 800. However, when we try to directly put the whole map into block memory generator, it shows the error that the size is out of the range. We come up with two ideas, one is that we work on the algorithm level, which involves calculating the value of nearby pixels to approximate and compress the image. The other idea is to use a clock and read periodly for the same pixel with specific ratio of origin image and background size. We utilize the second method here and first we use python PIL module to resize the image into 512 $\times$ 384, and then we use python to turn into smaller coe file. When we finally deploy it on the design, we give a clk and scale ratio that helps build a frame buffer. The code behave as follows:

```
always @(posedge clk) begin
    // Scaling current screen coordinates to memory coordinates
    // Using the reciprocal of the scaling factor
    scaled_x = (curr_x * X_SCALE) >> 10;
    scaled_y = (curr_y * Y_SCALE) >> 10;

    // Making sure the scaled coordinates do not exceed the memory bounds
    if (scaled_x >= 512) scaled_x = 511;
    if (scaled_y >= 384) scaled_y = 383;

    // Combining the scaled coordinates to form the memory address
    address_reg = {scaled_y, scaled_x};
end
```

3.3 barrier setup

For barrier setup, we initially come up with two ideas. One is we detect through the background, and to check if the background is white (white means the area allows to move) and send the signal to position logic for control. The other way is to use high level code to detect the barrier boundary area first and directly set the hard constraint in pixel level. Since our barrier is all the rectangular and regular shape, we can easily extract information from high level code and apply in the background.

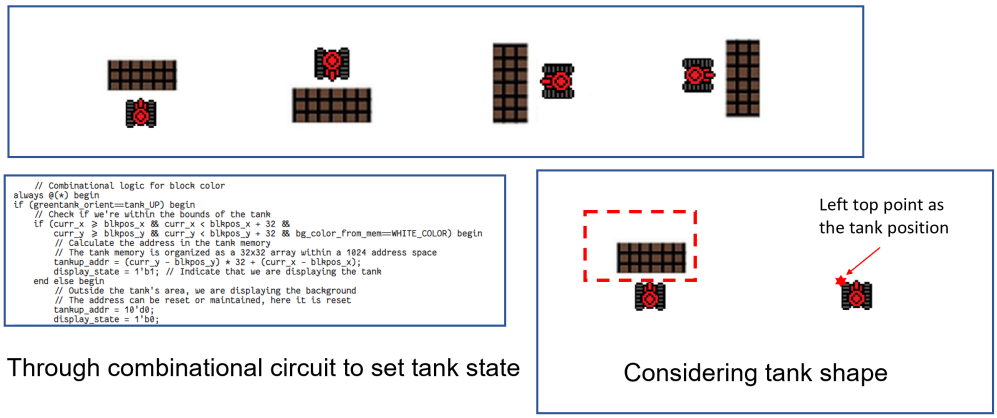


We use the opencv package in python and the details show in below.

```
# Read the image
image = cv2.imread('map.jpg')

image = cv2.resize(image, (1280, 800))

# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding to get the white areas
_, thresh = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY)

# Find contours using cv2.RETR_CCOMP to get all the contours
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
```

3.4 draw of tank shape

For the tank state, it has four different shapes including up, down, left and right. And for the constraint, we also need to take the tank shape into consideration as shown below. The blkpos_x and blkpos_y represent the left top point so the barrier constraint should add the tank weight and height (32×32). Every time we have a movement command, we will check the function if the position is forbidden area to set the barrier constraint. For the target and tank display, it is purely combinational circuit which depends on blkpos and current state. The logic of enemy box is the same as tank, and the only difference is controlled by direction signal in control box.



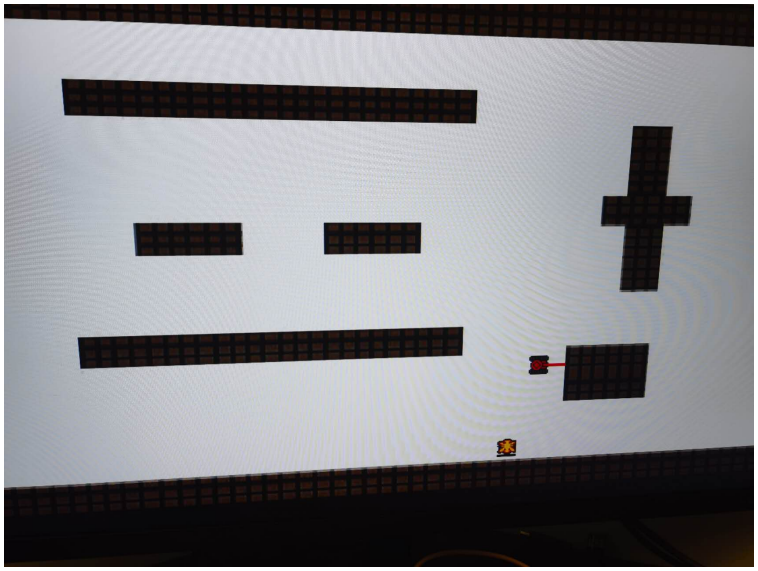
```
    // Combinational logic for block color
always @(*) begin
if (greentank_orient==tank_UP) begin
    // Check if we're within the bounds of the tank
    if (curr_x >= blkpos_x && curr_x < blkpos_x + 32 &&
        curr_y >= blkpos_y && curr_y < blkpos_y + 32 && bg_color_from_mem==WHITE_COLOR) begin
        // Calculate the address in the tank memory
        // The tank memory is organized as a 32x32 array within a 1024 address space
        tankup_addr = (curr_y - blkpos_y) * 32 + (curr_x - blkpos_x);
        display_state = 1'b1; // Indicate that we are displaying the tank
    end else begin
        // Outside the tank's area, we are displaying the background
        // The address can be reset or maintained, here it is reset
        tankup_addr = 10'd0;
        display_state = 1'b0;
```

Through combinational circuit to set tank state

Considering tank shape

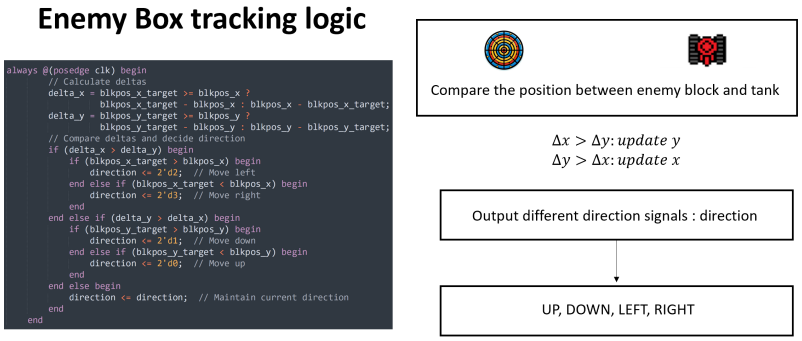
3.5 laser fire

The laser fire module is to fire a red laser with length of 80 and width of 4. When the laser touch the barrier, it will not show and activate. We set the laser function in drawcon module, which is combinational circuit. And also when the laser position is overlap with target position, the target state will be set as exploded. For the barrier constraint, we use the function check if the position is forbidden area which can add the barrier constraints. There is a small bug in it that we don't have time to fix it. When the laser touch the barrier, it will not stop but pass it. We also set the function to check the forbidden area and the active area is only the barrier. The real test shown as below:



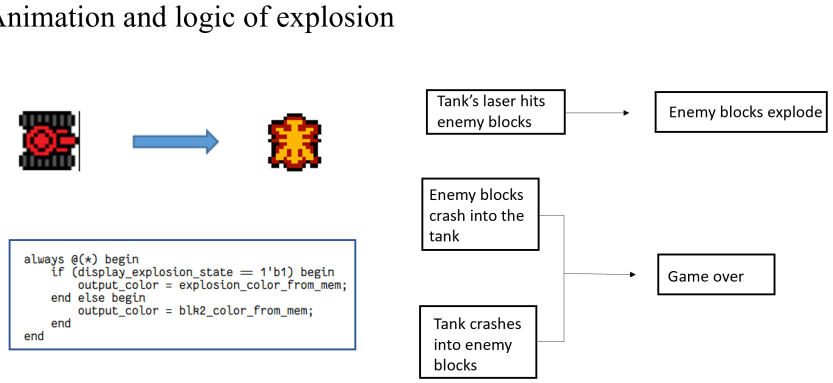
3.6 enemy Box tracking logic

The enemy box tracking logic is implemented in the control box module. It is to compare the position difference with x and y dimension. It will update the enemy position accordingly. However, it will not help pass the barrier in the background.



3.7 animation and logic of explosion

When the tank laser fire to the target, the target will be exploded. The logic is simple and easy. We got a explosion state for individual enemy box, and once the laser position is overlapped with the enemy box, the state of enemy box will be exploded. We also give the explosion state into position logic due to two main reasons. One is that once the target explode, it will no longer move and we will set the state as static. The other reason is that we set the other boxes will only appear and activate after the first enemy box is exploded. That also means if the first one is not exploded, the rest of enemy target will not show up. This is also for better demonstration. We have five enemy explosion state initially, and we can control each tank individually through the dependence of the explosion state.



3.8 testbench

In the beginning of the project, we utilize the test bench a lot. For the VGA ouput, we spend lots of time for debug the output. We simply give the test by giving a specific clock and see the wave. When we have the interactions with the output input buttons, the most used test bench in this project is shown below, where we give random input signals (up,

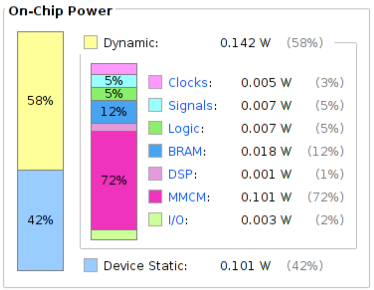
down, left and right). In this part, we rely on the waves but we can still improve the test part with some high level languages to display the value and result. In submodule test, we have tested for the enemy tracking logic, where how we apply the updating strategy to the enemy box and target tanks. For this module test, it is quite simple and we just instantiate the module and give random for position with output of direction. As project goes larger, we rely on the directly test on the FPGA board. It is also the simplest but effective way to directly test on the board as our time is very limited.

```
// Random input generator task
task randomize_inputs;
begin
    // Initialize all inputs to 0
    up = 0;
    down = 0;
    left = 0;
    right = 0;
    fire = 0;

    // Randomly select which input to activate
    case ($random % 5)
        0: up = 1;
        1: down = 1;
        2: left = 1;
        3: right = 1;
        4: fire = 1;
    endcase
end
endtask

// Test Bench Logic
initial begin
    // Initialize Inputs
    clk = 0;
    rst = 1;
    up = 0;
    down = 0;
    left = 0;
    right = 0;
    fire = 0;

    // Reset the system
    #100;
    rst = 0;

    // Continuously generate random inputs
    forever begin
        #20; // Change the inputs every 20ns
        randomize_inputs();
    end
end
```

3.9 hardware resource utilization

The hardware resource is shown below. We utilizes 2602 LUTs in total, which is a quite large. And also when we try to add a game over screen at the end, it shows the errors in implementation. It is due to limited hardware resources in the board and it meets some error in the opt_design. The target board only compatible with 270 sites but our design requires 278 sites.

Hardware resources utilization



Module name	LUTS
game top	2602
clk_counter	6
drawcon	460
position_logic	1930
vga_unit	315