

CSEE4840 Embedded Systems Design Document

Quinn Booth (qab2004), Ganesan Narayanan (grn2112), Ana Maria Rodriguez (amr2343)

March 30, 2023

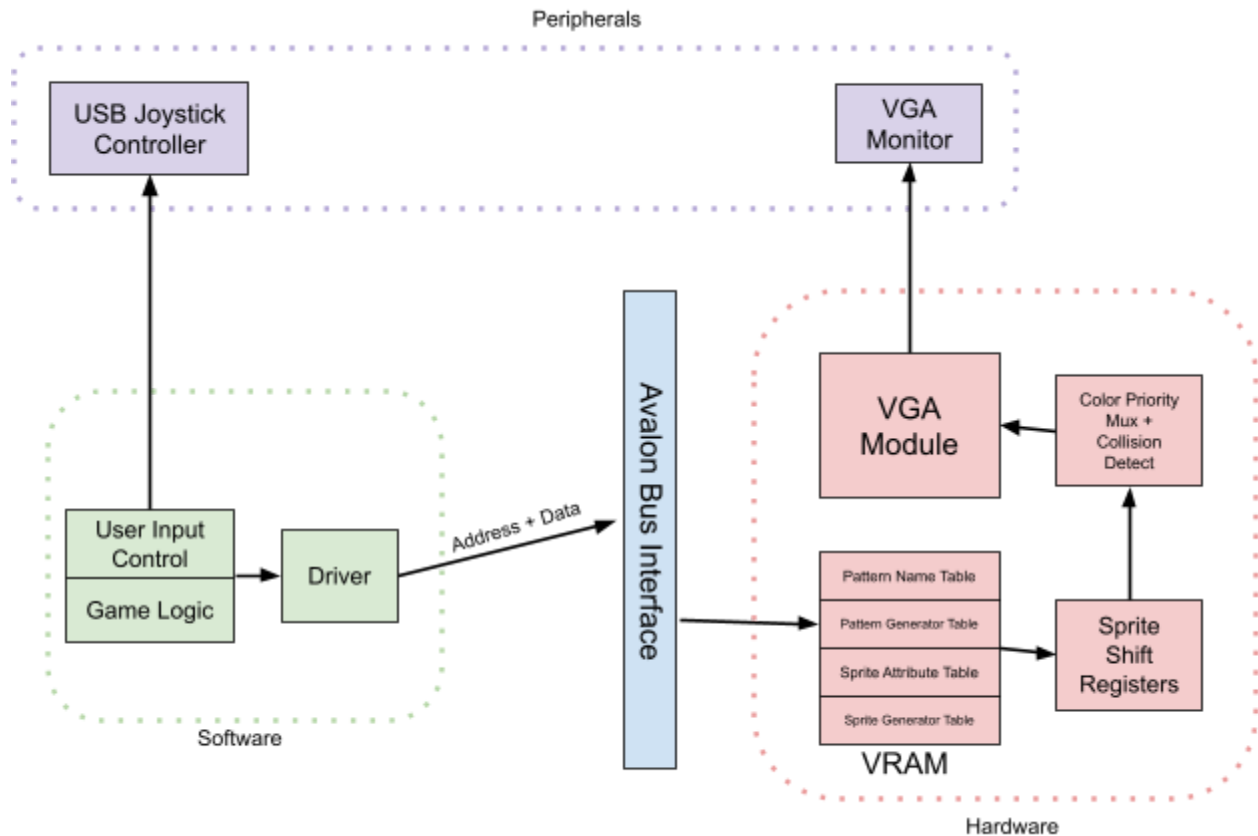
1 Introduction

In this project, our team will build a 2-player tank maze game based on the original Tank arcade game developed in 1974 by a subsidiary of Atari. Tanks is a game where two players drive tanks around in a maze viewed from above, while attempting to shoot the opposing player's tank. In a brief overview, the user will interface with the software controlling the game logic through a joystick communicating through the USB protocol. Software will communicate to the FPGA using a driver, while the FPGA will handle displaying the graphics for the game.

Going into more detail, the software components involve the main game logic .c file, which will handle the logic of tank movement, bullet shooting, and scoring. The usb .c file will recognize inputs from the USB joystick controllers so that the game logic can be carried out. Finally, the vga ball .c file device driver will communicate with the FPGA to update the graphics that will be displayed on the VGA monitor based on the game logic. The hardware peripherals include the USB joystick controllers, through which player input will be passed, and the VGA monitor, which will display the output of the game itself. The hardware consists of the VRAM on the FPGA in which all the necessary sprite tile data will be stored, registers to handle the displaying of the sprites on a per-line basis, and the vga ball .sv file that will display the requisite graphical information on the VGA display.

The USB joystick communicates to the game logic user space program through the USB protocol and libusb library. The game logic communicates with the FPGA hardware through the vga ball device driver and Avalon bus interface. The vga ball module implemented on the FPGA controls the VGA monitor hardware peripheral.

2 System Block Diagram



3 Algorithms

Hardware:

The main hardware algorithm is the logic to display the graphics. This will be done using the TMS9918 architecture with sprite-tile graphics.

We will display the static, fixed maze using tiles, with the pattern name table containing the memory address of the relevant tile for each position, and the pattern generator table containing the pixels themselves in the FPGA VRAM.

Similarly, the preloaded sprites, such as the tank and the bullet, will be stored in the sprite generator table in the RAM. The sprite attribute table will contain the memory addresses of the graphics along with the display positions of the sprites.

Displaying the sprites is done line-by-line. If the sprite will be displayed on that line, then the horizontal position will be loaded in and the counter ticked down by the hcount until zero when the sprite is actually drawn on the display pixel by pixel. The sprite drawing concludes based on the size of the sprite.

Based on our resource calculations performed in the next part, we do not anticipate exceeding the RAM size on the FPGA and so do not envision needing access to additional memory or to use additional logic in order to save on memory used, such as a color lookup table.

Software:

The objective of tanks is to shoot the enemy player's tank before they shoot your tank. There are two players controlling two separate tanks independently. The first player to kill the enemy tank wins the round, with the score displayed at the top of the screen. To simplify the game, we will have just one level with a fixed maze design. The game will conclude once one player reaches 10 round wins.

We will use a coordinate system to track the location of the tanks and the bullets. Tanks move up, down, left or right from their current location, however they cannot move through the walls of the maze but must remain in their position if movement towards a wall is attempted. Players start in opposite corners of the maze at the beginning of the game, which will restrict their initial movements.




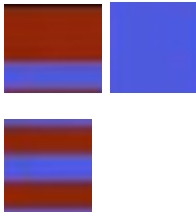
Tanks can only fire a bullet in a straight line trajectory, and will continue until it hits a wall or the enemy tank. The bullet will move faster than the tanks, the software must keep track of where a bullet leaves the tank and continue the movement until termination and recognize when this overlaps with the enemy tank position to determine the round winner.

There is no AI in our game design, however both player inputs need to be processed simultaneously, using two threads in the user program to listen to the joystick controller inputs.

4 Resource Budgets

The sprites and tiles required for our project are shown in the table below. The images for the tanks will require the most memory. For the tank, 4 images are needed for each of the 4 possible directions the tank can face (up, down, left, right), multiplied by 2 for the enemy player, giving a total of 8. The bullets fired by the tank will require just 2 images, one color for each of the players. The maze boundaries are constructed out of a single wall image. This single sized tile can be combined to form different wall shapes such as an L, straight line, etc.. To display the score, numbers are constructed out of 3 different

single tile images that are combined to create the digits 0-9. The score is displayed independently for both players, so one set for each player gives a total of 6 images.

| Name | Graphic | Size (bits) | # Required | Total Size (bits) |
|---------|---|----------------------|------------|-------------------|
| Tank |  | 16 x 16 | 8 | 49152 |
| Bullet |  | 8 x 8 | 2 | 3072 |
| Wall |  | 8 x 8 | 1 | 1536 |
| Numbers |  | 8 x 8 | 6 | 9216 |
| | | Memory Budget (bits) | | 62976 |

In addition to this memory, the pattern name table and sprite attribute table will also take up some space. Regarding the pattern name table, as the VGA display we are working with is 640x480 pixels, with 8x8 tiles this is 4800 tiles, which is 38400 bits. We have a total of 2 sprites with the tank and the bullet, so for the sprite attribute table, assuming each entry will take up 4 bytes (which should be more than enough for the address and display position), this is a total of 8 bytes, or 64 bits. This gives a total VRAM of 101440 bits, which will fit in the 256KB RAM on the FPGA.

5 The Hardware/Software Interface

Our HW/SW interface will be as follows. The requisite sprite data will be stored in the video RAM on the FPGA. The stationary graphics (maze walls) will be displayed in a fixed grid using tiles. The memory addresses of these tiles will be held in a pattern name table. Different addresses in the byte-addressable memory will correspond to the pattern name table and sprite attribute table, as per the design laid out in the TMS9918 architecture. Changing the position of sprites will be done from the software by writing data to the appropriate byte address in the RAM. As we will have to update the positions of the sprites many times throughout the gameplay, we seek to make this as simple as possible from the software side so that just the address to be written to and the new coordinates could be passed in when we want to update what is being displayed and where. We will make the size of the writedata passed through to the vga module 32 bits so that we can include all the necessary information such as the new positions for both tanks and the position for any bullets. This 32 bit value will be broken down into these respective components and written to the corresponding addresses in the VRAM.