The Design Document for CSEE 4840 Embedded System Design

Jiayi Wang (jw4462)
Xuanbo Xu (xx2440)
Yizhi Wang (yw4174)
Yiyang Peng (yp2655)

Contents

Introduction

Tank Go! is a 2D multiplayer-competitive game focusing on the combined development of hardware and software using DE1-SoC by converting the user input to the game instruction of the tanks and implementing the custom Video Display Processor (VDP) to render the graphics. Our system architecture is designed to achieve a balance between the display logic and the game logic to generate a fluent visual display of tanks, bullets, chickens, etc. For the hardware side, the display will be on an 800*600 pixels VGA screen with a 40Hz clock, and all the modules like bullets and chickens will be written to the ROM to save the on-chip memory. For the software side, each component is designed to communicate with each other, as the game engine is responsible for sending the render request to the VDP and sending the user data to the server, while the server synchs the game status of all users using the Networking module and database. We have a detailed design of optimizing the resource usage of DE1-SoC which is explained below in the resource budget part. Overall, Tank Go! is a fun and responsive game for multiusers to play with designed by the combined use of software and hardware.

System Block Diagram

Based on the design we came up with, here are nine blocks that contain the major functionalities to achieve the goal. The user can operate the joystick physically, and these inputs will be converted to the electronic signal passed to the game engine. The game engine will control the action of the tank and generate render requests for the VGA driver handled by the graphics accelerator in between, and furtherly displayed by the VGA display. In the meantime, the game engine will send the action and the status of the user in the game to the multi-functional server using the networking module. The server will synchronize the users' game status, manage the online conversations, and store the data in the database.

Figure 1 below shows the design components and the interactions between them:

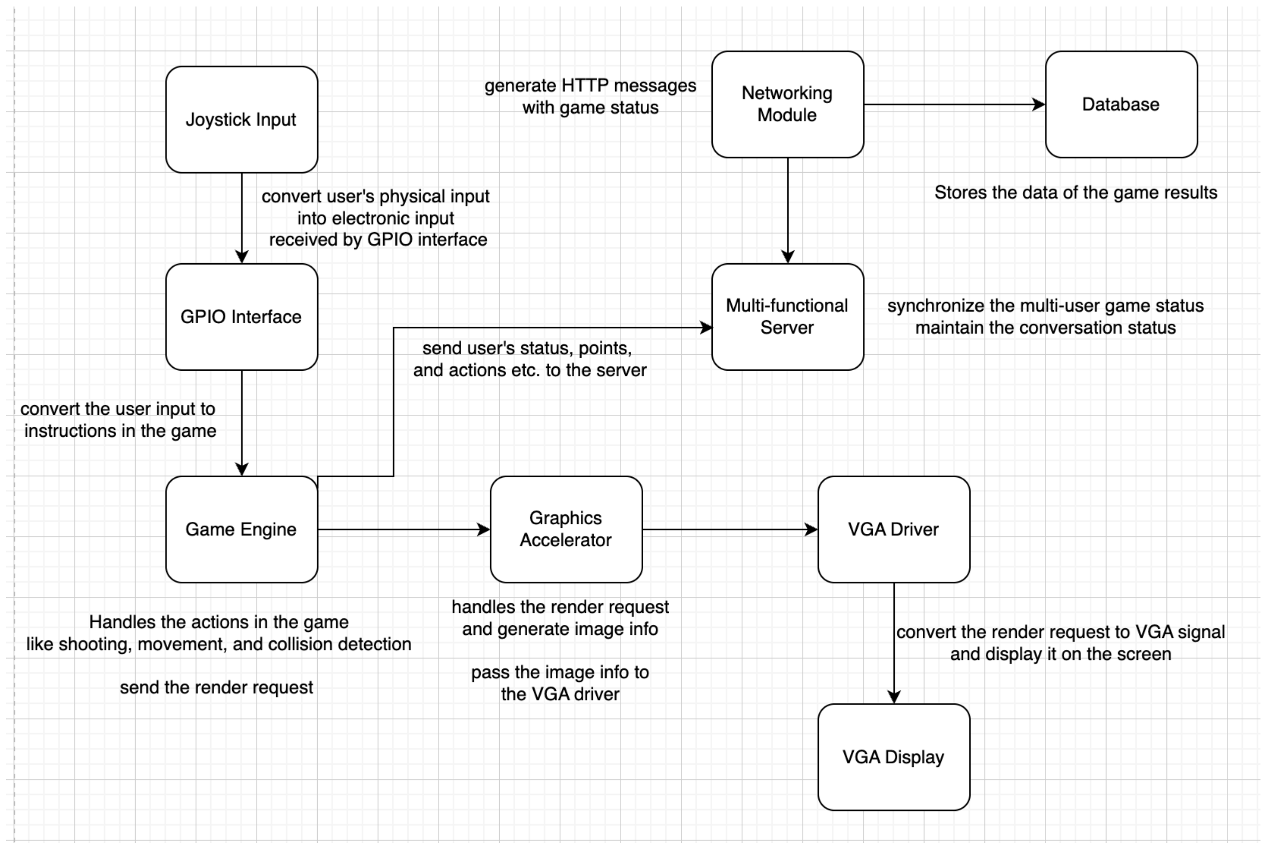

Joystick Input
convert user's physical input into electronic input received by GPIO interface
GPIO Interface
convert the user input to instructions in the game
Game Engine
Handles the actions in the game like shooting, movement, and collision detection
send the render request
send user's status, points, and actions etc. to the server
generate HTTP messages with game status
Networking Module
Database
Stores the data of the game results
Multi-functional Server
synchronize the multi-user game status maintain the conversation status
Graphics Accelerator
handles the render request and generate image info
pass the image info to the VGA driver
VGA Driver
convert the render request to VGA signal and display it on the screen
VGA Display

Figure 1. System Block Diagram of Tank Go.

Algorithms

Display logic

We will design a game engine render logic to render images by converting PNG photos to MIF files and reading them from rom. However, since the size of the screen was 800 x 600 pixels, large menu screens of text would require large amounts of memory, ~150KB each (large in the context of the 256 KB on-chip memory of the DE1). To prevent loading up the memory with large text, we will develop a separate module to represent the start and end screens by taking a much smaller image with the necessary text and expanding them to fit the screen.

We will design a VGA core to read pixel data from different sources based on the state of the game. We will implement the logic to calculate all necessary signals to control and render the VGA monitor, such as HSYNC, VSYNC, pixel color, and so on. It will be running with a 40 MHz clock. The VGA screen is 800*600 pixels, and 60 Hz. The resulting frame rate is good enough for our game display. Our game screens will be controlled by a state machine, which will take signals for starting, winning, and losing to switch between the screens.

Game Logic

Our game will consist of four components: tank, bullet, chicken, and red man.

We will develop the logic to process the user inputs to move the tanks on the screen, so that when users press once, the duration that the signal rises is shortened to only one clock cycle to adapt to the rest of the game logic. Tank position is updated based on the filtered user input, and the final location information, which is the top left coordinates, will be sent to interact with other game components and the render logic to draw the tank.

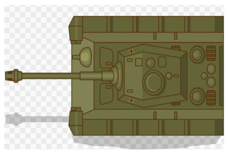
Bullets are controlled by hardware, so when users press the ‘shoot’ button, it will activate the bullets, and then their positions will be calculated and updated along the y-axis automatically. When the bullet reaches the end of the screen or hits any other game components, its position will always start from the point, and it will not be rendered until the user enters another ‘shoot’ signal. Moreover, bullets will speed up if tanks touch the red man. The change of speed is manipulated in the power-up logic which is described later. Bullet could also kill the chicken to make the game more interesting, but the main functionality of the bullet is to hit the other tank three times, and try to win this game.

Chicken will be bouncing inside the region between two tanks, every time it touches the boundary, it will change its direction symmetrically.

Another important game component, the red man, so that if the tank touches the red man, its bullet speed goes up. This is done by periodically checking if there is an overlap between the red man position and the tank position. Moreover, if the tank tries to approach the red man, the red man will start running away until it successfully maintains a safe distance about 350 pixels from the tank. The red man will appear randomly on each side of the tank for 500s, and then change to the next random place.

Resource Budgets

We will resize and crop each image and store the graphics data in ROM.

Category	Graphics	Size(bits)	# of images	Total Size(bits)
Tanks		55*100	2	132000
Bullets		18*30	2	12960
Power-up		30*30	1	21600
Chicken		34*28	1	22848

Total memory budget is 189408 bits which is less than the on-chip memory of DE1-SOC.

The Hardware/Software Interface

The hardware/software interface centered around a custom Video Display Processor (VDP) and control input system. The VDP handles rendering and display, while the input system manages player controls. This description outlines the control and status registers necessary for interfacing software with this hardware, facilitating the development of "Tank Go."

Video Display Processor (VDP) Interface

Control Registers:

1. Mode Register (8 bits)
    - Bits 0-2: Screen Mode (000 = Text Mode, 001 = Graphics Mode, 010 = Advanced Graphics Mode, etc.)
    - Bit 3: Interrupt Enable (0 = Disabled, 1 = Enabled)
    - Bit 4: Screen Enable (0 = Off, 1 = On)
    - Bits 5-7: Reserved for future use

2. Screen Resolution Register (8 bits)
    - Bits 0-3: Horizontal Resolution (0000 = 256 pixels, 0001 = 320 pixels, etc.)
    - Bits 4-7: Vertical Resolution (0000 = 192 pixels, 0001 = 224 pixels, etc.)

3. Color Depth Register (4 bits)
    - Bits 0-3: Color Depth (0000 = 2bpp, 0001 = 4bpp, 0010 = 8bpp, etc.)

4. Sprite Control Register (8 bits)
    - Bits 0-3: Maximum Sprites per Line (0000 = 8 sprites, 0001 = 16 sprites, etc.)
    - Bit 4: Sprite Size (0 = 8x8, 1 = 16x16)
    - Bits 5-7: Reserved

Status Registers:

1. V-Blank Status Register (1 bit)
    - Bit 0: V-Blank Status (0 = Not in V-Blank, 1 = In V-Blank)

2. Sprite Collision Register (1 bit)

- Bit 0: Sprite Collision (0 = No collision, 1 = Collision detected)

3. Display Status Register (8 bits)

- Bit 0: Frame Complete (0 = Rendering, 1 = Frame ready)
- Bits 1-7: Reserved

<u>**Control Input System Interface**</u>

<u>Control Registers:</u>

1. Input Configuration Register (8 bits)

- Bits 0-1: Number of Players (00 = 1 player, 01 = 2 players, etc.)
- Bits 2-7: Reserved

<u>Status Registers:</u>

1. Player 1 Input Status (8 bits)

- Bit 0: Up (0 = Not pressed, 1 = Pressed)
- Bit 1: Down
- Bit 2: Left
- Bit 3: Right
- Bit 4: Fire
- Bits 5-7: Reserved

2. Player 2 Input Status (8 bits)

- Identical to Player 1 Input Status

<u>Usage:</u>

- Configure the VDP to the desired screen mode, resolution, and color depth using the Mode Register, Screen Resolution Register, and Color Depth Register.

- Monitor the V-Blank Status Register to synchronize frame updates.
- Check the Sprite Collision Register to handle in-game sprite interactions.
- Read the Player Input Status Registers to respond to player actions in real-time.