SPRING
2022

CSEE4840 PROJECT

RIVER RAID - DESIGN DOCUMENT

XINHAO SU, ROJAN BANMALI

1) Abstract

This design document describes the preliminary implementation of Atari's River Raid game on an Altera DE1-SOC FPGA board. In this implementation, the graphics, audio and user inputs are handled by the FPGA hardware, whereas game logic, which includes, level design, player movement, collision detection, score keeping, etc. are done in software.

2) Design Overview

Figure 1 shows the organization of hardware modules and logic for the entire game. Figure 2 shows bit assignments for the memory layout used.

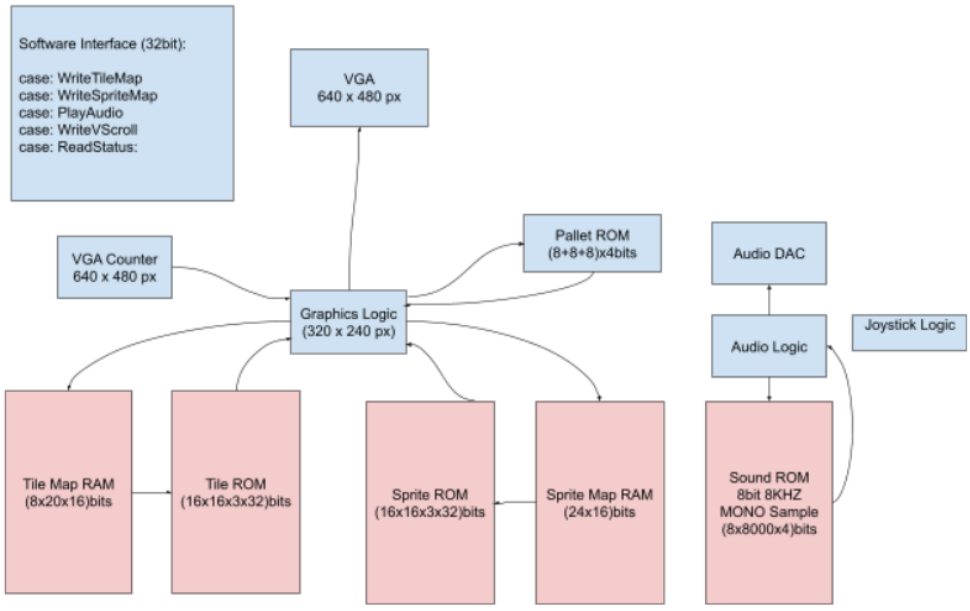

Figure 1 : Hardware components

Tile Map bit assignment (8bits)

x	Tile_Id (5 bits)	Pal_Id (2bits)

Sprite Map bit assignment (24 bits)

Pos_x (9 bits)	Pos_y (8 bits)	Spite_Id (5 bits)	Pal_Id (2bits)

Palate bit assignment (24 bits)

Red (8 bits)	Green (8 bits)	Blue(8 bits)

Tile/ Sprite ROM bit assignment (3 bit color per pixel)

px0_b2	px0_b1	px0_b0

Figure 2 : Memory bit assignment

a. Memory Organization

The memory bit assignment used in the hardware is organized as follows:

i. Tile Map RAM
1. Pal_Id (LSB): 2 bits Palette ID. This number selects palette from 4 available color palettes.
2. Tile_Id: 5 bits Tile ID. This selects tile from 32 available 16x16 tile artworks.

ii. Sprite Map RAM
1. Pal_id (LSB) 2 bits Palette ID. This number selects palette from 4 available color palettes.
2. Sprite_Id: 5 bits Tile ID. This selects tile from 32 available 16x16 sprite artworks.
3. Pos_y: 8 bits Sprite Y coordinate.
4. Pos_x: 9 bits Sprite X coordinate.

iii. Palette ROM
1. Blue (LSB): 8 bits. Blue color value.
2. Green (LSB): 8 bits. Green color value.
3. Red (LSB): 8 bits. Red color value.

iv. Tile/ Sprite ROM
1. px0_b2, px0_b1, px0_b0: 3bits pixel 0 value. Next row defines pixel 1 value, next pixel 2, and so on. 3bits will give 8 different shades.

b. Graphics Logic

Graphics Logic uses: VGA counter, Tile Map RAM, Tile ROM, Sprite Map RAM, Sprite ROM and Palette ROM as shown in Figure 1. The original resolution of the game was 160 x 192 pixels but we will design the game for 320 x 240 resolution and stretch it to fit 640 x 480 resolution. To stretch the resolution we can simply use 640 x 480 VGA counter and divide the horizontal and vertical counter by 2. The artwork for the game will be stored as 3bit per pixel sprites and tiles in Sprite and Tile ROMs. The size of each sprite and tile will be 16x16 pixels and there will be about 32 sprites and 32 tiles. The tiles will be combined to form the background of the game. The sprites, which are player ship, enemy ship, missile, etc., will be layered on top of the background. Since the graphics resolution is 320 x 240 and the tile size is 16x16, the screen will have 20 (cols) x 15 (rows) tiles. To make vertical scrolling easier, we will add additional invisible row making the background size 20 x 16 tiles as shown in Figure 2. In the figure VScroll is an 8bit variable that counts from 0 to 255, which means as VScroll counts up, it goes through all 16 row tiles (16x16=256) and then automatically wraps around. As the vertical scrolling increments new tile map row data is sent by the software and added to at the end of 15 row x 16pixel boundary.

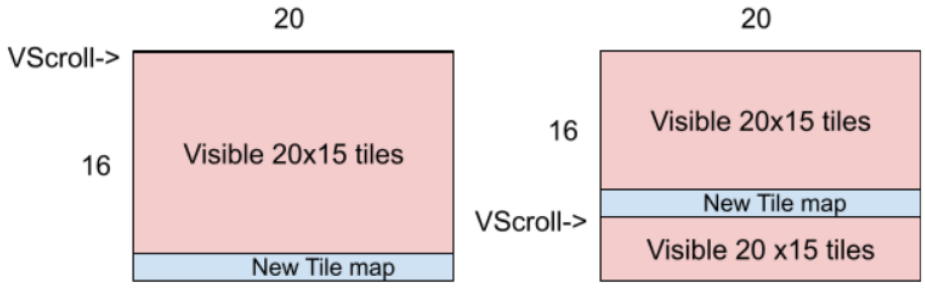

Figure 3 : Vertical Scrolling of background

Using Tile map RAM data, based on hcount, vcount and VScroll, address for Tile ROM can be calculated. The output of Tile ROM will be a pixel value, which will be mapped to a palette defined in Tile map RAM. The pseudo code to find the final tile pixel value is shown below.

```
TileMapRAM_address = hcount/16 + [( vcount + VScroll)/16]*20
TileMapData = TileMapRam [TileMapRAM_address]
TileROM_address = TileMapData*256 + (hcount%16) + [(vcount+VScroll)%16*16]
TilePixelVal = TileROM [TileROM_address];
```

Similarly, the software generates sprites by writing to Sprite Map RAM. The position of the sprites can be controlled by writing to Sprite MAP RAM. See figure 2 for Sprite RAM bit assignments.

The graphics logic layers sprites on top of the tiled background. The sprites are always kept apart so there are no overlaps.

c. Audio Logic

Four (4) one-second 8 KHz mono audio samples will be used for the game. Same sample will be sent to both left and right channel of the audio DAC.
The audio clips needed for the game is as follows:

	Sample (8hkz, mono, 1sec)	Tigger Condition
1	Fly	Play sound in loop while flying
2	Fire	Missile Fire
3	Refuel	When going over flue tank
4	Explode	When player or enemy is destroyed

Table 1: Audio Sample List

The software will send messages to hardware to trigger audio clips stored in ROM.

d. Player Input (Joystick) Logic

The game will be designed to be played with an Atari Joystick connected to DE1-SOC GPIO_0 header. Figure 4 shows the pinout of Atari Joystick. Figure 5 shows pinout for DE1-SOC GPIO_0 header. Table 2 shows GPIO_0 and Joystick pin connections. All GPIO used will be tied to VCC3p3 using 100k pull up resistors. Switches of the joystick will be denounced in hardware.

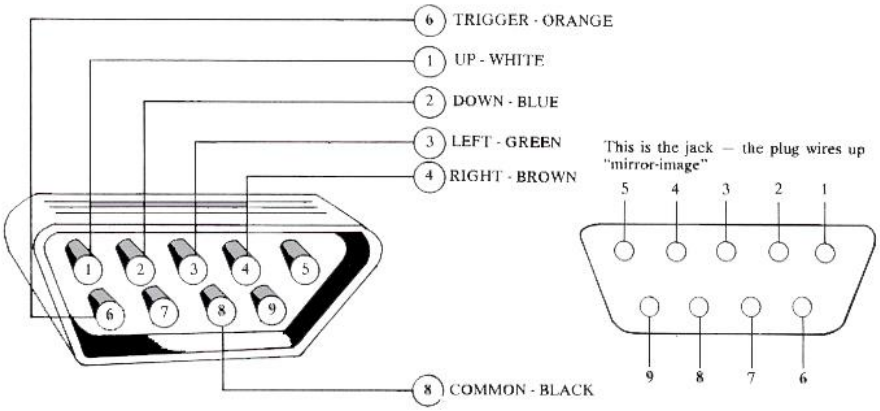

Figure 4 : Atari Joystick Pinout

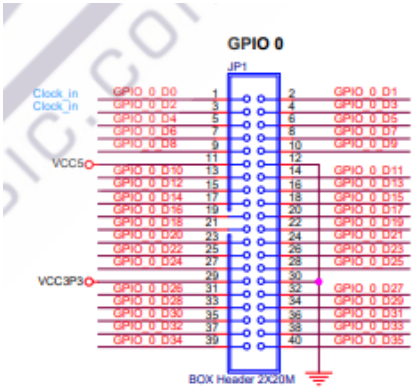

Figure 5 : DE1-SOC GPIO pins

GPIO 0 Pins	Pull up resistor	Atari Joystick Pins
29 (VCC3p3)		
30 (GND)		8
31 (GPIO_0_DB26)	100K pull up resistor to 29	1
33 (GPIO_0_DB28)	100K pull up resistor to 29	2
35 (GPIO_0_DB30)	100K pull up resistor to 29	3
37 (GPIO_0_DB32)	100K pull up resistor to 29	4
39 (GPIO_0_DB34)	100K pull up resistor to 29	6

Table 2: GPIO_0 to Joystick pin assignments

e. Software

The software for the game will be written entirely in C and will run under Linux operating system. Figure 6 shows the hardware(FPGA) and software interface. Linux kernel uses river_raid.c driver to communicate with the hardware, which runs in Kernel space. The main game runs in User space.

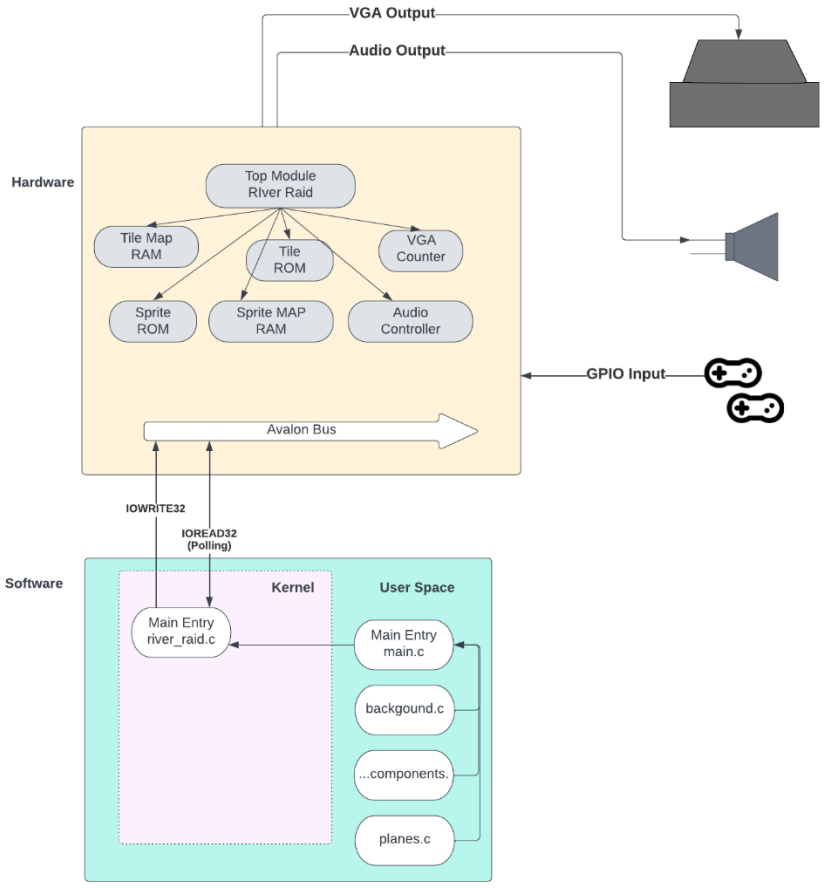

Figure 6 : Hardware Software interface

The data width of the data transfer to/from the software to/from the FPGA is 32 bits. Figure 7 shows the commands available for the software hardware interface. The following are brief descriptions of the commands.

i. WriteTileMap command writes Tile_Id and Pal_Id (palette id) in Tile Map RAM at address TileMap_Addr.
ii. WriteSpriteMap command writes Pal_Id, Sprite_Id, Pos_x (x position) and Pos_y(y position) of the sprite in Sprite Map RAM at address SpriteMap_Addr.
iii. PlayAudio command plays audio sample assigned to Audio_Id.
iv. WriteVScroll command writes vertical scroll value.

v. ReadStatus command reads the joystick switch status and VSync status. VSync status is used to send new row of tile map. ReadStatus is called from the software in polling fashion.

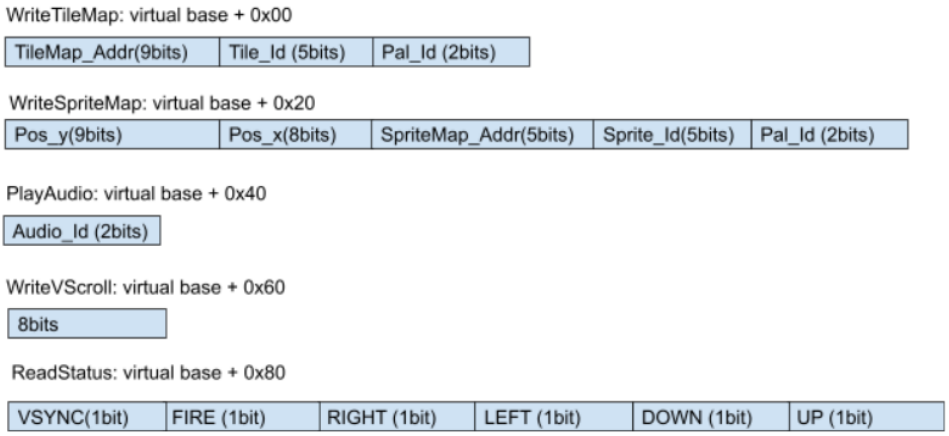

Figure 7 : Software Hardware interface commands

f. Game Logic

Game level, which includes tile map and enemy sprite arrangements, are hardcoded in the software and are transferred to the hardware as the game progresses using software interface mentioned above. The movements of enemy sprites will not use any AI. The movements of enemy sprites will simply be triggered at random times. Collision detection will be also done in software by testing overlap in sprite coordinates.

3) Memory Budget

Table 3 below shows preliminary memory estimated for the game.

Category	Per Size	Quantity	Size
Background Tile Map	8 bit	320 (20 row * 16 col) tiles	2,560 bits
Background Tile	16 * 16 * 3 (3bit color)	32 tile artwork	24,576 bits
Sprite Map	24 bit	32 sprites	768 bits
Sprite Assets	16 * 16 * 3 (3bit color)	32 sprite artwork	24,576 bits
Audio Sample	8 bit * 8KHz * 1 second	4 samples	256,000 bits
Palette	24bits (RGB) * 8 shades	4 palettes	768 bits
Total			309,248 bits

Table 3 : Preliminary memory usage estimate

4) References:

1) https://fileadmin.cs.lth.se/cs/Education/EDA385/HT11/student_doc/final_reports/spaceshooter.pdf
2) https://www.atariarchives.org/creativeatari/Joytricks.php
3) https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/DE1-SoC%20schematic.pdf