StrEEt Fight Project Design

CSEE4840 Embedded Systems

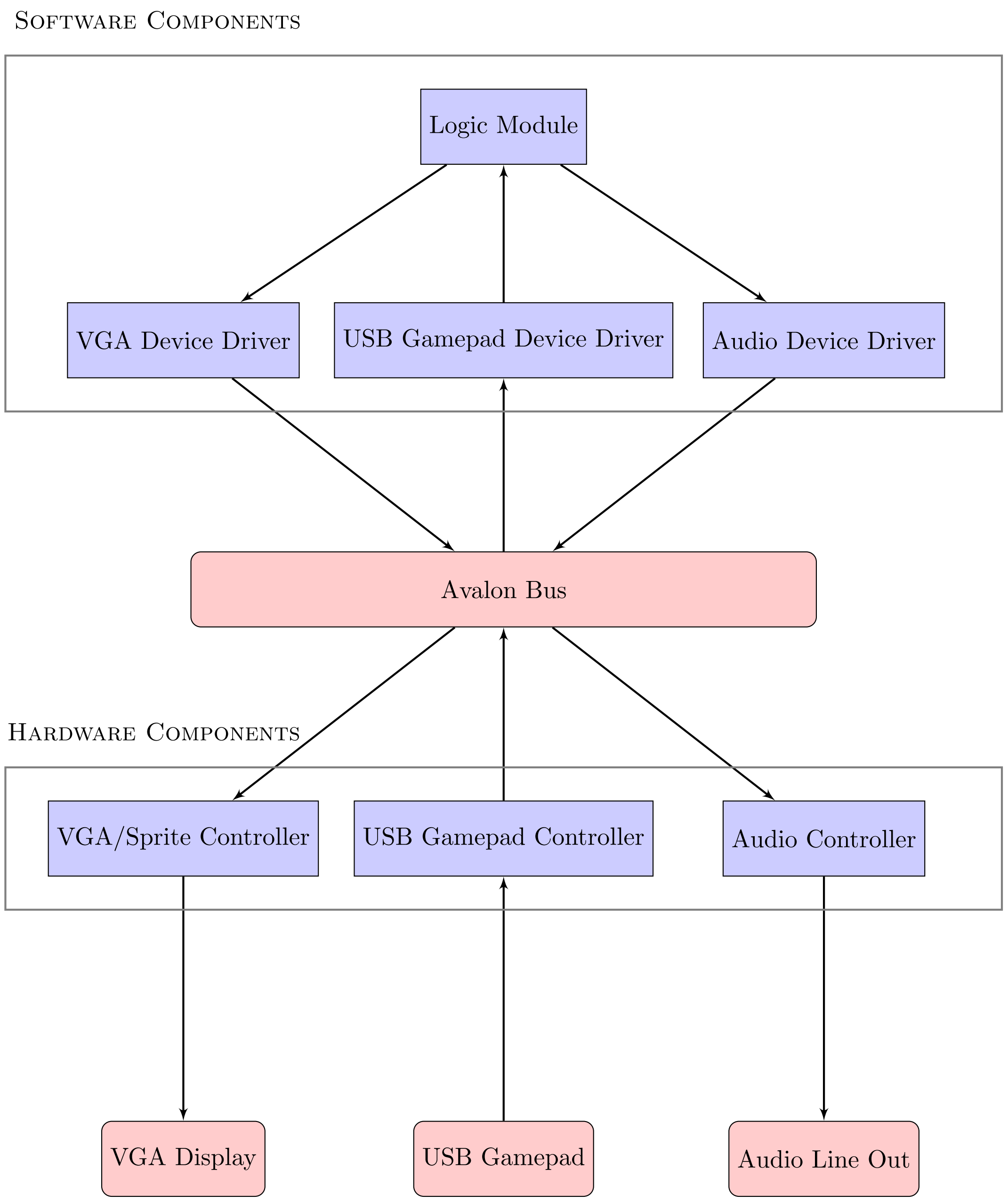
Alan Armero aa3938
Cansu Cabuk cc4455
Daniel Mesko dpm2153

March 29, 2019

1 Introduction

We propose a 2D, 2-player street fighting game, where players interact with the game via USB gamepads and a VGA monitor. The gamepad has a joystick and two buttons. The joystick controls the movement of the player: moving the joystick left or right moves the player left or right, moving the joystick up makes the player jump, and moving the joystick down makes the player duck. The two buttons control punches and kicks, respectively. On the VGA monitor we will display a background terrain, three hearts above each player, and two sprites representing the players. When a player punches, kicks, or takes a hit, an appropriate sprite is displayed. All other times, an idle sprite is displayed. The players will be on opposite sides of the screen, facing one another, with their respective health hearts above their corners. When one player hits the other, the other player incurs damage and loses a heart. When one player loses all of their hearts, the other player wins. When a player is hit or a player dies, an appropriate sound is emitted.

2 System Architecture



2.1 Software Components

The game logic and device drivers will be written in C.

2.1.1 Logic Module

The game logic is controlled by a logic module at the highest level. The body of this module will begin by forking two new threads. Each of these two threads will then enter an infinite loop that waits for input from one of the two USB gamepads, controlled by one of the two players. The main thread will also enter an infinite loop where it continuously checks the two players' states and calls appropriate device drivers to represent those states. State information is stored globally all three threads. Player state includes player position, health, and current behavior (idle, ducking, punching, etc.) On input from the gamepad, the two worker threads update player state accordingly. The main thread will then read state changes of the two players, update game state information if necessary, and forward relevant information to the device drivers. This information includes x and y coordinates of the players and the sprites that represent their current states, to be sent to the the VGA driver, and the "pow"/"death" sounds, to be sent to the audio driver. In some cases, additional sprites are sent to the VGA driver. For example, when a player lands a punch/kick (the player punches/kicks while adjacent to the opponent), a "pow" sprite is sent to the VGA device driver.

When a player's health reaches zero, the main thread indicates to the VGA device driver to display the "dead player" sprite for that player, and indicates to the audio device driver to emit the "death" sound. It then exits the infinite loop and terminates the game.

2.1.2 VGA Device Driver

The VGA device driver receives sprite information from the logic module and outputs appropriate information to the VGA hardware controller. Specifically it receives sprite number and x and y coordinates from the logic module and forwards them to the VGA hardware controller for all of the sprites being represented in a particular frame. After it receives all of the sprites for a frame, it makes a "render" call to the VGA hardware controller, which will render the corresponding frame. The device driver, as with all of the drivers, exposes `ioctl` system calls to the logic module and uses memory mapped device registers internally to communicate with the hardware controller. For the VGA driver, the device registers will be used to send x and y coordinates and sprite number to the hardware controller.

2.1.3 USB Gamepad Device Driver

The USB gamepad device driver will receive information (via memory-mapped device registers) from the USB hardware controller indicating what kind of action was performed: joystick up, joystick down, joystick left, joystick right, button one pressed, button two pressed. It will then decode this information into a corresponding magic number which it will forward to the logic module. The logic module worker threads will receive this information when they call a function to wait for gamepad input, at the beginning of the infinite loop.

2.1.4 Audio Device Driver

The audio device driver will receive a magic number from the logic module indicating which of the two sounds to emit. It will then forward that number (by placing it in a memory-mapped device register) to the audio hardware controller, which will fetch the appropriate synthesized sound from `SDRAM` and put it out on the audio line out.

2.2 Hardware Components

The device controllers will be written in SystemVerilog.

2.2.1 VGA/Sprite Controller

The VGA/Sprite controller receives from the VGA device driver an x and y position for a given sprite and adds it to a global queue for that frame. It continues to receive sprite information from the VGA device driver until a "render" bit is set. At that point, it renders the frame pixel by pixel, traversing the sprite queue to determine if any of the sprites need to be rendered at that pixel. If this is the case, the controller pulls the appropriate sprite image (RBG values for all pixels) from `DDR3` memory and displays the appropriate pixel within that sprite. If there is no sprite to display, the appropriate background tile for that pixel is fetched from `DDR3` memory and displayed. After a frame is displayed, the sprite queue is cleared.

2.2.2 Audio Controller

The audio controller will receive a magic number from the device driver indicating which of the two sounds ("pow" or "die") to emit. When it receives a number, it will fetch the appropriate synthesized sound from `SDRAM` and put it out on the audio line out on the DE1-SoC. It will do so by interacting with the I^{2}C Multiplexer on the DE1-SoC.

2.2.3 USB Gamepad Controller

The USB gamepad controller receives USB keycodes from the gamepad and forwards them to the USB gamepad device driver. We will use the controller included with the Gamelec 2-Player Arcade Buttons and Joystick DIY Controller Kit (available at `https://www.amazon.com/Gamelec-2-Player-Controller-Raspberry-Joysticks/`

`dp/B077FRWMKF`).

3 Sprites/Background Tiles

To save memory, we will use the same sprites for both players, but mirror the images, so that the two players are facing one another. The `ioctl` calls made to the VGA device driver by the logic module will contain an extra bit that indicates whether the given sprite is for player 1 or for player 2. The VGA driver will pass the extra bit along to the hardware controller, which will mirror the sprite it pulls from memory, if necessary. The full list of sprites and background tiles is shown below.

1. Idle Sprite
2. Duck Sprite
3. Punch Sprite
4. Kick Sprite
5. Walk Sprite
6. Float Sprite
7. Dead Sprite
8. ”pow/spark” Sprite
9. Arrow Sprite
10. Health Heart Sprite
11. Floor Tile

4 Potential Bottlenecks

4.1 Memory Usage

Since we have many sprites to render per frame and the hardware controller is responsible for pulling these sprites from memory, we may encounter an issue where the monitor refreshes before we can render the entire frame.

4.2 Computation Speed

The logic module performs many non-trivial computations to update game and player state before calling the VGA device driver, so it is possible that this will slow down the frame rate and make the game glitchy.

5 Project Plan

5.1 Milestone 1

By April 5th we aim to have the VGA/Sprite controller and the VGA device driver complete. To test these components without the logic module and USB gamepad support, we intend to hardcode a few `ioctl` calls that test the capability of the VGA driver/controller to display the various sprites and alter their locations.

5.2 Milestone 2

By April 19th we aim to have the Audio and USB gamepad device drivers and controllers complete. At this point we should be able to interpret input from the USB gamepads and alter the locations of the sprites accordingly, as well as output our two sounds properly.

5.3 Milestone 3

By May 3rd we aim to have the logic module complete and have nearly completed debugging and testing the game. At this point players and their sprites should respond properly to the various forms of gamepad input. Health heart sprites should disappear when a player takes a hit, and the game should terminate when a player loses all hearts. Sounds should be emitted at the appropriate times.