strEEtfight
CSEE 4840 Embedded Systems

Alan Armero
aa3938

Cansu Cabuk
cc4455

Daniel Mesko
dpm2153

May 14, 2019

Contents

Chapter 1

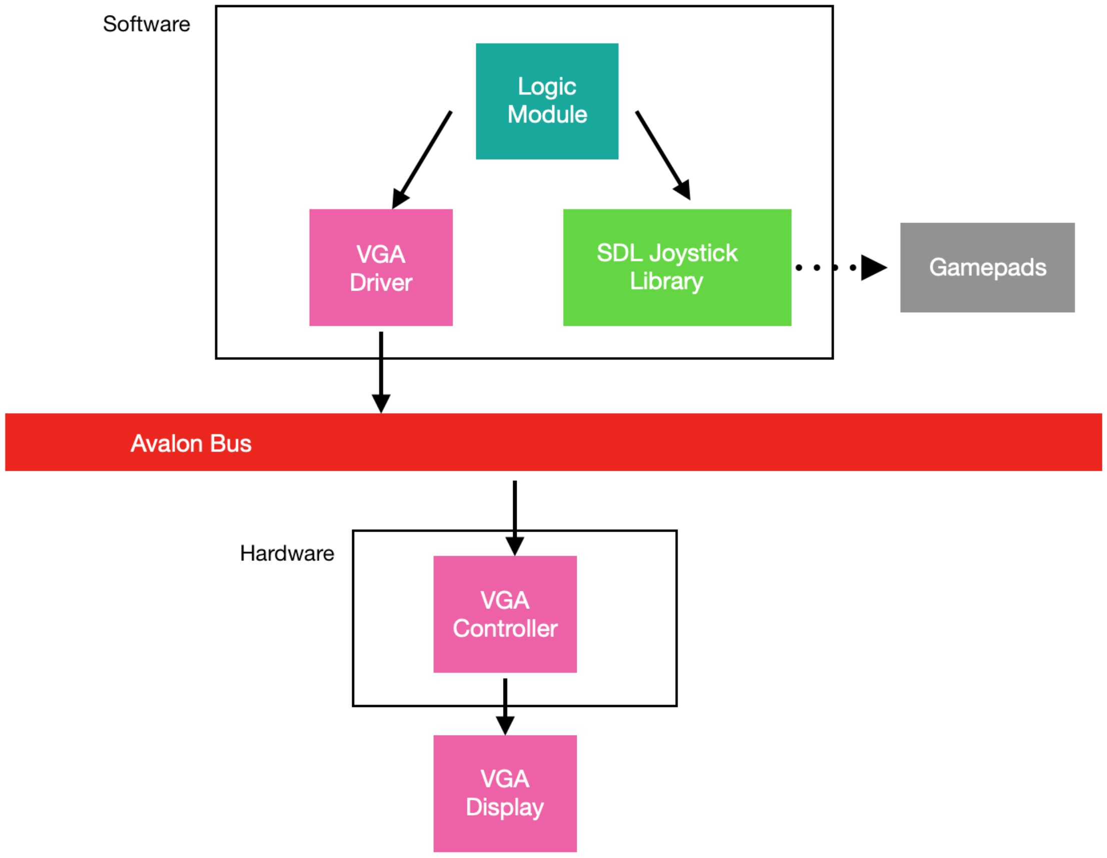
Introduction

We present a 2D, two-player street fighting game implemented using the Altera DE1-SoC Development Kit. Players are assigned a character in the shape of a block with a specific color and they use a VGA monitor and gamepads to interact with the game. The gamepads each have a joystick and two buttons. The joystick is used to control the character's movements which are moving left and right, and jumping. All of these actions are performed by moving the joystick in the desired direction. The two buttons are used to throw punches and kicks. If a punch or a kick lands (the puncher/kicker is adjacent to the other player), then the other player incurs damage and loses a life. However, if the other player was jumping at the time that the punch was thrown, damage is not incurred. When a player loses all of their lives, the game starts over.

Chapter 2

Design Overview

StrEEtfight can logically be separated into software and hardware components. The software controls game logic and device communication, and the hardware controls the physical devices. The overall structure of these components is illustrated by the diagram below.



Chapter 3

Software Components

All of the software for StrEEtfight is written in C. We made use of the C standard library for math functions and threading, and the SDL (Simple DirectMedia Layer) libraries for communicating with the joysticks (documentation and source code available at `https://libsdl.org/`). Our game is compiled with the GNU C Compiler. Compilation and linking was automated with *make*.

3.1 Logic Module

The game logic is controlled by a logic module at the highest level. It performs the following tasks:

1. Register a signal handler for `SIGINT` to allow safe termination of the game - free allocated memory, close the joysticks, and close the VGA device descriptor.
2. Initialize the various sub-modules:
   - Fork the joystick handler thread, initialize joysticks via SDL.
   - Allocate scene object list and set initial states for player objects.
   - Open the VGA device and retain corresponding file descriptor.
3. Enter the game loop: call other sub-modules that do the following:
   - Update joystick input states.
   - Update player states based on input states.
   - Render each player by reading player states and passing the relevant information to the VGA device driver.

3.1.1 Input

When initialized, the input sub-module forks a thread that will handle the joystick input. This thread enters an infinite loop where it waits for interrupts from the joysticks via the `PollEvent` function of the SDL library. When it receives an interrupt, it parses the SDL `event` object that is returned and extracts the relevant information regarding the joystick button states or joystick axis motion. This information is stored in a set of global input bitmasks corresponding to each joystick, and is read by the main thread in the logic module's game loop. These bitmasks are protected by `pthread mutexes`.

Various other functions are exposed by the input sub-module so that other modules can attain specific attributes of the bitmasks, such as the buttons that went up this frame, or X-axis values.

3.1.2 Scene

When initialized, the scene sub-module allocates a linked list of `scene_objects` that can represent any entity that can effect game state. There are only two `scene_objects` after initialization - the objects representing the two players. A `scene_object` contains behavior information and a position vector for the object, as well as a pointer to more specific object data - in the case of the player objects, these are health and a boolean value indicating whether the player is player 1 or player 2.

The scene sub-module exposes a `list_iter` function that takes a function pointer as an argument and traverses the scene object list, calling the supplied function for each `scene_object`, with the object as the sole parameter. This allows other sub-modules to traverse the `scene_object` list and perform their own specific actions on each object.

3.1.3 Player State

When initialized, the player state sub-module instantiates each player's `scene_object` object and sets initial values for health, player behavior, and position.

The `player_update` function is called by `list_iter` from the scene sub-module for every object in the scene list. It updates a player's state based on the global input bitmasks - for example, update player behavior and position if one of the joystick axes moved, or decrement player health if a hit was thrown by the other player within range.

3.1.4 Render

When initialized, the render sub-module opens the VGA device and retains the file descriptor. It also contains a function that is passed to to `list_iter` to read a `scene_object` and pass relevant information to the VGA device driver for rendering.

3.2 VGA Device Driver

The VGA device driver manages the communication between the logic module (specifically the render sub-module), and the VGA device. It exposes an `ioctl` system call that takes a pointer to a `vga_device_arg_t` object, which contains the following information:

- Player 1's position on the X axis.
- Player 1's position on the Y axis.
- Player 2's position on the X axis.
- Player 2's position on the Y axis.
- Player 1's health (0-3).
- Player 2's health (0-3).

This ioctl is called only by the render function, to pass information about player locations and health to the VGA device.

Chapter 4

Hardware Components

The hardware descriptions for StrEEtfight were written in SystemVerilog and compiled onto the DE1-SOC FPGA using Intel's Qsys and Quartus software.

4.1 VGA Display Controller

The VGA display controller decodes the input address to determine what data is being sent over the `writedata` vector. Asynchronously, it stores the `writedata` in the appropriate register, corresponding to player 1's x-position, player 1's y-position, player 2's x-position, player 2's y-position, player 1's health, and player 2's health. Combinationally, it outputs the appropriate red, green, and blue vectors for the pixel currently being drawn. To determine what color to output, it compares the current pixel's location with the locations of either player (plus or minus a pre-defined width, representing the players as boxes), and the fixed locations of the heart squares. If the pixel is within the bounds of either player, it outputs the corresponding player color - red for player 1 and red for player 2. If it is within the location of one of the heart squares, it checks if that player's health is high enough for this particular heart to be drawn. If this is the case, the heart color drawn. For all other pixel locations, it draws a black background color.

Chapter 5

Individual Contributions

As with every project, teamwork was really important in this game implementation and each team member's contribution was really important to this project. Daniel worked on getting the joysticks to clearly work using the SDL library, as well as incorporating them into the whole design in addition to working on VGA driver. Alan worked on software modules like scene, state, input and render and also put great effort into writing the SDRAM controller and hardware buffer modules that we weren't able to incorporate into our project. Cansu found the sprites we were going to use and converted them to 24-bit BMP files for pulling into SDRAM.

Chapter 6

Lessons Learned and Advice

Through this project, we learned many valuable lessons on both software and hardware sides. One of the most important lessons we learned was to instead of focusing in just one aspect of the project, make sure that you focus equally on hardware and software simultaneously and keep in mind that they have to work with each other. While it is easy to believe that they will work when integrated because they work separately, this is often not the case. Another valuable lesson learned was to make sure to unit test everything in order to see they are able to perform as expected, especially the hardware components. Before starting to design the project, make sure you know the limits of the hardware you have. Dealing with memory on FPGAs can be really challenging due to space limitations, so it is very important to calculate everything in advance and double check.