CSEE4840 Final Project Design Document

Watch Out!

Shangru Li
Zachary Salzbank

Introduction
Watch Out! is a game in which the player must jump from one platform

to another in order to avoid hitting the top of the screen. The game gets progressively more difficult as time goes on. The platforms will begin to move faster and the distance between platforms will increase. The game has no ending, as long as the player can survive the increased difficulty level. The player's score increases when they jump from one platform to the next. The goal of the game is to get the highest score.

Goals

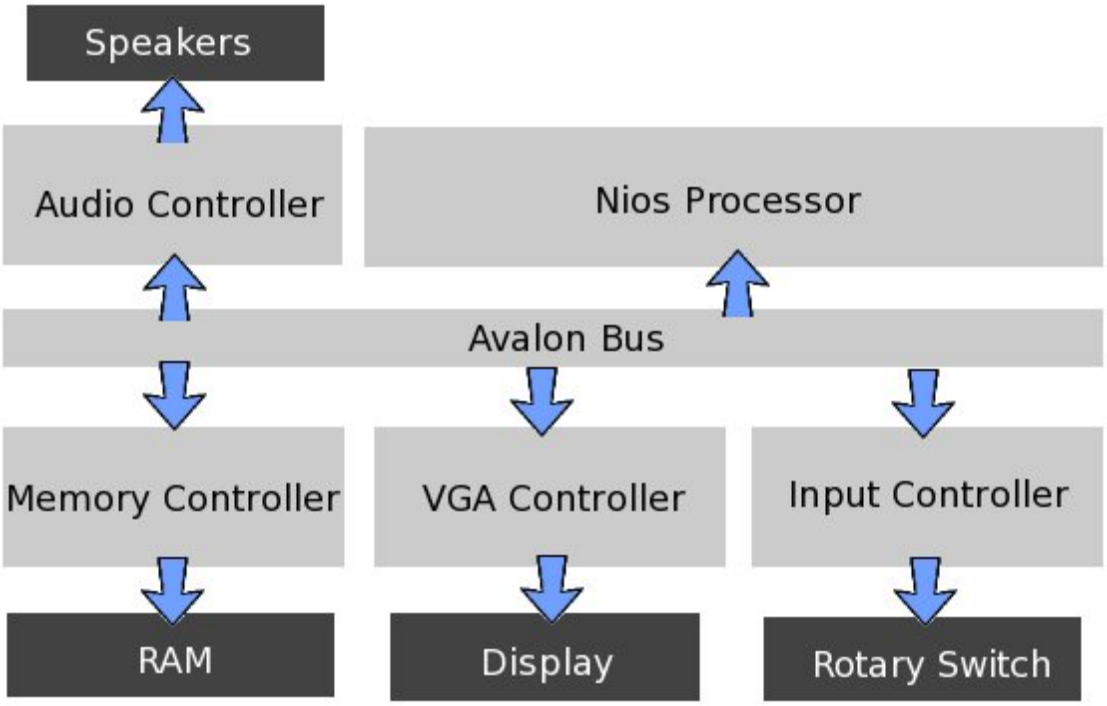
- Make four different types of platforms:
    - Standard - a platform with no special properties.
    - Time Limited - a platform that disappears 3 seconds after the player lands on it.
    - Spiked - a platform that reduces the health of the player.
    - Bounce - a platform that bounces the player higher than they could normally jump.
- Background is scrolling behind the platforms.
- Score and player health are displayed on screen.
- Player is controlled by rotary switch.
- Simple sounds when the player jumps or hits a spiked platform.
- Ability to keep track of high score for a session of play.

Hardware

In order to achieve our goals, the following hardware is required to be attached to the FPGA. Controllers must be designed for each of these components:

- VGA Display
- Speakers
- Rotary Switch

All components will be able to be controlled by a NIOS processor that will be programmed onto the FPGA. The main game software will be run on the processor.



Technical Requirements - VGA Display

The VGA display must be able to show the following elements, in positions that are determined by the software:

- Eight different platforms (of four different types, listed above)
- Player (can face multiple directions)
- Background (continuously scrolls behind elements above)

In addition, text and/or graphics indicating the score and health of the player are to be displayed in a fixed position. These are also controllable by software.

Sprite Implementation

Graphic data will be stored in ROM on the FPGA. There will be seperate ROMs for the platfroms, player and background images. The data being output by the ROMs will be determined by the current pixel being drawn on the display.
Each platform can be one of four types, so there will be four entries in the platform ROM. The man can face one of three directions, so there will be three entries in the player ROM. The background is always the same data, so there is only one entry in this ROM.

In order to display a platform, four settings must be written to memory: x position, y position,

platform type and enabled/disabled. When enabled, the other three settings are used to determine whether or not to display a platform pixel on the display. Eight platforms can be displayed at the same time. Each has it's own memory location. In order to display the man, three settings must be written to memory: x position, y position and direction. The man is always displayed on the screen. The three settings determine how to display a pixel for the man.

All the settings are accessible via the Avalon Bus on the FPGA. When the VGA controller is written to, the address is used to determine which setting is being set. The two MSBs of the address is used to determine the setting group: platform (00) or man (01). When modifying a platform, the following scheme is used to determine the setting register to modify:

Bits 4-3	Bits 2-0
Property: • X - 00 • Y - 01 • TYPE - 10 • ENABLED - 11	Platform Number • Any number 0 - 7

When modifying the man, the following scheme is used:

Bits 2-0
Property: • X - 000 • Y - 001 • DIRECTION - 010

Technical Requirements - Speakers

The speakers will be controlled by the audio controller on the FPGA. The speakers will be able to play sounds when certain events occur during the game, such as the player landing or when the game ends. Output begins when the software requests the sound.

Technical Requirements - Rotary Switch

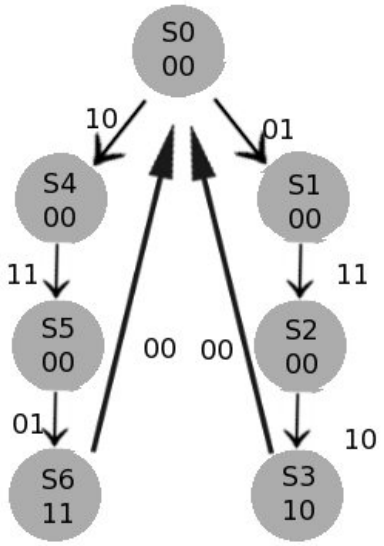
The rotary switch will be able to control the movement of the player. The switch provides analog signals which will be converted to digital and

accessible by the software for user input.

The rotary controller has two inputs, s1 and s2, going into the GPIO of the DE2 board. We can track the pattern of the sequence of s1 and s2 to determine whether it is rotated clockwise or anticlockwise. If the sequence is 00, 10, 11, 01, it is anti-clockwise. If the sequence is 00, 01, 11, 10, it is clockwise.

We can use two bits of output to represent different types of movements: output = "0X" means no movement, output = "10" means clockwise movement, and output = "11" means anti-clockwise movement.

So we can make a simple FSM to act as the digital interface of the rotary controller and the FSM's state diagram is as follows:



Software

The software will determine the position, type and speed of the platforms. They will be generated in a pseudo-random matter, such that the game is never the same twice. As time progresses, the parameters determining the randomness will change, making the game harder.

The player will be controlled by the input from the rotary switch. When the bottom of the player hits the top of a platform, the player stops moving. But, when the player misses a platform and hits the bottom of the screen, the game is over.

Software will also control the text indicating the player score and health, as well as sounds.

Milestones
The following milestones have been set for getting the game working on schedule:

- Milestone 1 -
    - Rotary Controller - reading the analogue signals from the rotary controller and output the signal digitally to the FPGA.
    - VGA Controller - Positioning and display of the fundamental objects of the game (player, platforms) via software.
- Milestone 2 -
    - VGA Controller - Movement of the background and display of player score and health.
    - Audio Controller - Software should be able to play sounds through the FPGA.
- Milestone 3 -
    - Software - Enable the different properties of the platform types.
    - Software - Program the main game logic, such as generating the new platform randomly and player/platform hit detection.