Graehme Blair

ghb130

EECS 355

Final Project Report

* Design process & methodology
* The system architecture by components / processes
* Board implementation & peripherals
* Simulation figures and testing methodology, including sample input/output data
* Synthesis results, including memory, clocks, and resource utilization

**Component Descriptions**

**Counter:**

**Desired Behavior:**

Accept clock, reset and hold signals and output a pulse in a consistent manner.

**Implementation:**

A simple state machine with two states forms the skeleton of this component. The first state, increment, adds one to a clocked signal on every rising edge. Increment transitions to the hold state when the incoming hold signal is high. While in the hold state the component holds the pulse output low and waits for the hold signal to go low again, at which point it resumes counting in the increment state. The signal keeping track of count was set to be 18 bits wide so that it would eventually roll over back to zero (18 bits was chosen because it produced the most fluid motion onscreen without being too fast). Whenever the counter detects that count is zero while in the increment state it inverts the pulse output signal, thus producing a steady on-off signal.

**Simulation:**

No simulation was made for this component as it would be impractical to simulate the millions of cycles required to see proper output. It was tested through trial and error on the board.

**PS2:**

**Desired Behavior:**

Take in keyboard clock and data signals, a 50mhz clock and a reset signal. Output a 2 bit speed for each tank and a 1 bit fire signal for each bullet.

**Implementation:**

**Brief Subcomponent Description:**

PS2 uses a keyboard component and a oneshot component. Keyboard is passed the keyboard clock/data, 50mhz clock, reset and read signals and outputs scan code and scan ready signals. Oneshot takes in a clock and trigger and outputs a pulse. By passing the scan ready signal from the keyboard component to the oneshot component and the oneshot component’s pulse signal to the keyboard component, the pulse from the oneshot component will only trigger a rising edge when a new key has been pressed.

PS2 consists of the above-mentioned components, keyboard and oneshot, and a process triggered by the rising edge of the pulse from oneshot. Within this process, the component looks at the scan code received from the keyboard. If it is a break code it sets a flag; if it is a key that is used for controlling something and the flag is low, sets that key’s bit high; otherwise it sets the key’s bit low.

This procedure allows the component to keep track of which keys are pressed continuously, so you can play the game by holding down keys. At the end of the component there is a combinational process that takes the bits corresponding to used keys and applies some logic to translate it into the desired outputs.

**Simulation:**

No simulation was made for this component as it functions purely in conjunction with the development board.

**GraphicsOut:**

**Desired Behavior:**

Take in (x,y) coordinates for both tanks and both bullets and generate the required signals to draw them onscreen using the VGA interface.

**Implementation:**

**Brief Subcomponent Description:**

Graphics out makes use of two components: vga\_sync and pixelGenerator. Vga\_sync simply takes in a 50mhz clock and generates the signals required to synchronize the output of the pixelGenerator to the monitor’s scan. pixelGenerator takes in (x,y) positions for the tanks and bullets and the current pixel being scanned(generated by vga\_sync) and outputs the RGB value for the current pixel.

As GraphicsOut just provides an interface for accessing vga\_sync and pixelGenerator, pixelGenerator will be discussed here, as it is the largest component. The pixel generator instantiates 5 roms that contain pixel information for tanks A and B, bullets A and B, and the background. It then takes the current pixel being scanned and checks to see if it is within the bounds of bullet A, bullet B, tank A, tank B in that order. If it is in within an object, it then reads the color from that objects rom into a signal. If the color read is the chroma key, it reads and uses the color from the background instead, otherwise it writes the color read to the RGB output for that pixel. If the current pixel is not within any components it writes the color from the background rom to the RGB output.

**Simulation:**

No simulation was made for this component as it functions purely in conjunction with the development board.