Space Force Duo

Charlie Devivero and Javier Garcia

November 10, 2009

1 Overview

The Space Force Duo (SFD) project consists of two FPGA's connected through a serial port enabling real-time communication of both voice and data between the two terminals. SFD implements a two-player collaborative game where each player takes control of a spacecraft fighting enemy units. There will be a host terminal and a client terminal. The host will be determined by the first player who logs in to a terminal. Additionally, the game will feature different "camera views," giving the player the illusion that the game is being played in 3D. What this consists of is taking the rendered image, and applying matrix transformations to project the image in a perspective view.

The SFD will include the following modules:

- Debouncer: Debounces and synchronizes the push buttons.
- AC97 Interface: Processes sound from microphone and to the headphones.
- Filter: Passes audio through a low-pass filter
- Recorder: Receives sound data for filtering and transmission.
- Playback: Receives sound data for filtering and playback.
- Deserializer: Receives data serially from the RS-232 port.

- Network Packet Receiver: Receives data from the Deserializer as a packet, and parses it out to the appropriate modules.
- Network Packet Parser: Structures game data into packets for network transmission.
- Serializer: Sends data serially through the RS-232 port.
- Master FSM: Processes input from the other terminal and handles game logic.
- VGA Generator: Produces the appropriate sync signals to produce a VGA (640x480 pixels) monitor output.
- Pixel Generator: Creates game sprites and Graphical User Interface (GUI).
- Matrix Transformer: uses matrix operations to transform an image frame to produce a sense of "3D" perspective on screen.
- Display Generator: Reads final screen frame and outputs it to the VGA port.

2 Modules and Responsibilities

2.1 Audio Controller (Devivero)

2.1.1 AC97 Interface

This module is the pre-built interface module from Lab 4 that outputs sound data from the microphone, and can take sound data as input to be played back on the headphones. It can be tested by supplying a simple square wave tone to the input, or routing the output from the microphone to the sound input.

2.1.2 Filter

This module is based on the Lab #4 implementation, which takes in 8-bit samples and stores them into a 32-byte buffer. The entire buffer is used to perform the filter calculations. The result cuts out audio with frequencies higher than 3kHz. An audio sample can be passed through the filter, and then observed on the logic analyzer to prove that it is filtering sound data correctly. Alternatively, ModelSim can be used to test this module in the same way.

2.1.3 Recorder

This module takes in 8-bit samples from the AC97 Interface, and passes them to the Filter module. Filtered audio samples are then placed in a 16-byte buffer and passed on to the Network Packet Parser for network transmission. To test this module, audio samples can be recorded to a BRAM instead, and played back on the same FPGA.

2.1.4 Playback

This module receives a 16-byte audio sample from the Network Packet Receiver, and parses individual 8-bit audio samples to a Filter module. Filtered audio samples are then passed on to the AC97 Interface to be played back. To test this module, audio samples can be retrieved from a BRAM and played back on the same FPGA.

2.2 Network Controller (Devivero)

2.2.1 Deserializer

This module receives serially transmitted data from the other terminal, following a standard RS-232 signaling scheme of 1 start bit, followed by 8 bits of data (LSB first), and end with 1 stop bit. Throughput will be initially set to 19,200 bps, though this can change according

to bandwidth needs. This data will be placed in a 1-byte buffer, and when full, will be transfered to the Network Packet Receiver.

The logic analyzer will be useful for checking if the signals are being sent correctly at the precise timings.

2.2.2 Network Packet Receiver

This module receives data from the Deserializer and places it in a 16-byte buffer. When the buffer is filled, this means a full network packet is received, and can then be parsed out to the correct modules.

2.2.3 Network Packet Parser

This module takes in game data (sprite positions, game state, voice data) and organizes it into a 16-byte buffer. This buffer serves as the structure for the 16-byte packet of data, to be passed on byte by byte to the Serializer, for network transmission.

2.2.4 Serializer

This module receives a byte of data from the Network Packet Parser and is transmitted serially to the other terminal, bit by bit, out of the RS-232 port, following a standard RS-232 signaling scheme of 1 start bit, followed by 8 bits of data (LSB first), and end transmission with 1 stop bit. The transmission will occur at a bit rate of 19,200 bps, though this can be changed depending on bandwidth needs.

2.3 Game Logic (Garcia)

2.3.1 Master FSM

This module handles game initialization, determines which terminal is the host and the client (according to who logs in the system first), computes the state of the game according to the game logic and user input, determines what data to send over the network, and handles data coming in from the network.

2.4 Display Controller (Garcia)

2.4.1 VGA Generator

This module produces the correct horizontal and vertical sync signals, as well as pixel counters, to produce a 640x480 VGA image on a monitor. These signals are passed on to the Pixel Generator module.

2.4.2 Pixel Generator

This module takes in game data input (such as player positions) to produce sprites at the correct locations on screen. It generates an image frame to be buffered on a ZBT RAM.

2.4.3 Matrix Transformer

This module reads an image buffer from a ZBT RAM, and performs matrix operations on the array of pixels to transform the image, giving it a "three-dimensional" perspective view of the game screen. This transformed image is stored in a ZBT RAM.

2.4.4 Display Generator

This module reads the final game image frame from the ZBT RAM, and outputs the appropriate signals to the VGA port.

3 Block Diagram