ABSTRACT

This report describes how a SNES-like system running the classic game Bomberman was developed and implemented on a Nexys 3 development board. Custom IPs were written in VHDL and the software, running on a Xilinx MicroBlaze soft processor, was written in C. Four original Nintendo gamepads was used for player input, and a speaker was added for playback of music and sound effects. The project turned out great, instructions on how to connect gamepads and a speaker is included in this report along with timing diagrams. The project took seven weeks and was done as a part of the course “Design of Embedded Systems - Advanced Course” taken at LTH in Lund, Sweden.

INTRODUCTION

In this chapter the concept behind and goals of the project is explained as well as a few design ideas. An overview of the system architecture is also presented. In the following chapters detailed descriptions of the custom IPs and program code will be given along with installation instructions for anyone wanting to test the game. Lastly there will be a discussion about problems that occurred, possible improvements to the design and thoughts about how the project turned out.

Concept

In the year 1995 the game Super Bomberman 2 was released in Europe by Hudson Soft for the Super Nintendo Entertainment System (SNES). In it, the player controlled a small “bomberman” that navigated a maze and could place bombs that blasted both obstructions and enemies out of the way. It contained both a single player storyline and a multiplayer component. The multiplayer component was the only focus of this project. In it up to four players battled it out in a maze-like arena collecting powerups and bombs until only one player remained. Figure X.X and X.Y shows a screendump of the original game alongside with a picture of the FPGA running the finished project.

Architecture

The original goals for the project was to implement a working Bomberman multiplayer game for up to four players without powerups or any sound. Original Nintendo gamepads were to be used, as well as original graphic elements and the game was to be displayed in 640x480 resolution @ 60 Hz using a VGA interface. The system was to be designed as software independent as possible, meaning that in theory any game could be played by running a different program.

The final system achieved all this, as well as support for music and sound effects, and the software was extended to include three different power ups improving the overall feel of the game. The graphics components, music and sound effects ended up being stored on the onboard flash memory since it would be too big to store in program memory.

In figure X.X a block schematic of the finished hardware is shown. All communication with the peripherals is done over the AXI bus. The custom IPs are implemented with the AXI Memory bus interface, so that they can easily be connected to any system using the AXI BRAM Controller IP, and memory mapped by the MicroBlaze. The GPU outputs the VGA RGB signals as well as an interrupt signal (IRQ). The IRQ is connected to the MicroBlaze using an AXI Interrupt Controller, meaning that more interrupts could easily be added and handled by the system if wanted (a keyboard interrupt for example). The serial flash memory is accessed using the AXI Quad SPI Interface core. This core is configured to run in a read-only mode, reducing the complexity of the data transfer and eliminating the need to read or write from any registers to set up a transfer. The flash memory is pre-loaded with the game components using Digilent Adept (REF). (Som förslag på förbättringar skulle man kunna lägga till att man flashar in download.bit filen så att den laddas varje gång man slår på strömmen). The VGA timing generator was made available in a previous course and simply generates a horizontal and vertical counter used by the GPU as well as VGA synchronisation signals for the monitor.

HARDWARE

In this chapter the custom IPs will be explained in further detail. All components were written in VHDL and tested using ISim in the Xilinx ISE Project Navigator.

Gpu

The GPUs task was to render the graphical elements in the game, meaning it had to decide the color of each pixel at the exact time it was needed by the screen. It was designed to be simple to program, requiring as little information from the user as possible, but still versatile and capable of displaying a lot of graphics elements, keeping in mind that it should be able to render any SNES-like game, not only Bomberman. It also had to meet the timing constraints of the system. Figure X.X shows a slightly simplified version of the entire GPU.

There are two components that make up the graphics. Tiles that are used for background, and sprites that are used for foreground objects. Both tiles and sprites are 16x16 pixels large, however, tiles are aligned to a grid and cannot be moved independently. Sprites on the other hand can be offset by any value from this grid. Sprites and tiles are made up by bitmaps, and a palette index. The bitmaps are made up by 4-bit color indexes, meaning that each separate sprite or tile can contain up to 16 unique colors. To color the sprite or tile the color index needs to be looked up in a palette. Each palette contains up to 16 colors, and by changing the palette index, the same sprite or tile can be rendered in different colors, and different sprites or tiles can use the same palette. This saves a lot of memory. Figure X.X shows an example of how the bitmap - palette combination works.

The GPU is programmed by writing commands to the Sprite Attribute Memory (SAM), in the Sprite Handler. For every line, the sprite handler checks the SAM to see if any sprites should be shown. If so it calculates the correct pixel offset and looks up the correct color index

Sound generator

A simple way to generate audio signals is to use pulse width modulation (PWM). It works by changing the duty-cycle of a square wave. When passed through a low-pass filter (or a speaker), the square wave will be integrated into an analog signal, if the duty-cycle is increased the analog signal will rise and if decreased the analog signal will fall. Figure X.X shows an example of this. Audio encoded into raw (pulse code modulated) data only contains different duty-cycle values, that when modulated correctly will generate a good sounding audio wave.

Figure X.X shows a simplified schematic of the sound generator. Internally, an address is being incremented at the sample rate, reading new sample data from the dual-port BRAM to the PWM component. This address will correctly overflow when reaching the last address of the BRAM so that it will never run out of data. Data samples are continuously being fed into the sound buffer by the MicroBlaze. When one half of the buffer is being sampled by the PWM component, the other is being updated by the CPU. To know which half to update, the msb of the cyclic sample address is wired to the data\_out signal, available for polling at any time by the CPU.

On the PWM side of the generator, the data sample is stored in a register and updated at the sample frequency. To generate the square wave an 8-bit counter is incremented at the system clock frequency. At the start of the counter the output signal is set to high. The counter value is then compared to the sample data and when the values are equal the output signal is set to low. When the counter reaches 256 it overflows and the output signal is once again set to high.

The sample frequency was chosen by dividing the system clock frequency with 10 times the resolution of the duty-cycle. This means that the PWM\_out signal will switch exactly 10 times for every audio sample, this being thought to give a clearer more even sound. 8-bit data samples were used resulting in a resolution of 256 different duty-cycle widths and a sample frequency of about 39.1 KHz. This also means that two samples can be stored at each memory location, since the BRAMs are 18 bits wide.

SOFTWARE

INSTALLATION

PROBLEMS AND CONCLUSIONS

CONTRIBUTIONS

REFERENCES

Super Bomberman 2 <http://en.wikipedia.org/wiki/Super_Bomberman_2>

Digilent Adept <http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT2>