Project proposal eda 385
space shooter

Jonas Johannesson et07jj5, Philip Ljungkvist et07pl3

1 Introduction

The idea of this project is to design a space shoot 'em up game. A space ship is controlled by the player in a two dimensional space. There will be obstacles and enemy spacecrafts that the player must avoid or destroy using the ships on board cannon. It will be displayed on a VGA monitor and controlled with a rotary encoder and a pushbutton. The design will consist of a general purpose CPU and a hardware graphics accelerator. The CPU decides the content of the game screen and the hardware is responsible for drawing this content to the screen. A conceptual image of the game can be seen in figure(1).

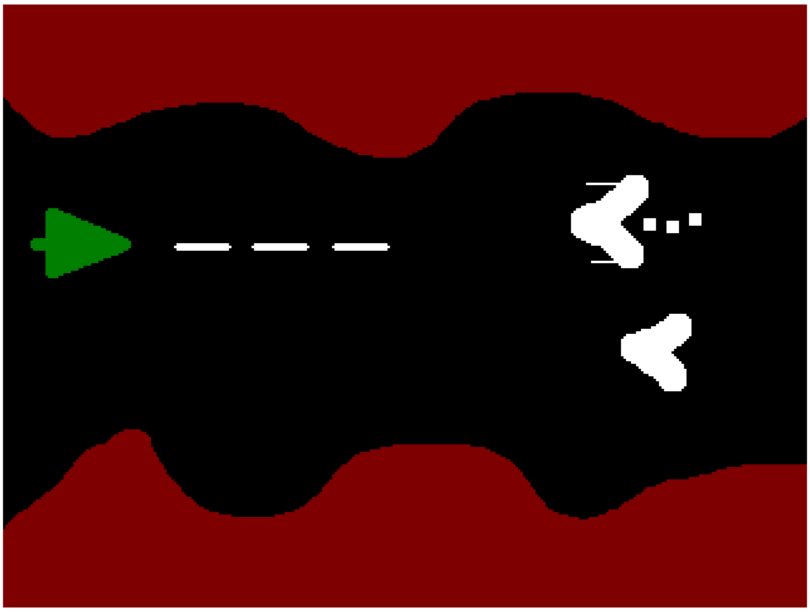

Figure 1: Concept image of how the game will look

2 Detailed description

A block diagram of the system can be seen in figure(2)

2.1 On chip communication

For communication between the different hardware cores the PLB bus is used.

2.2 I/O processes

The input from the user to the game is done with a controller that consists of a rotary decoder and a pushbutton. The rotary decoder is used to control the players spaceship sideways and the pushbutton is used to control the spaceships cannon that can be used to destroy enemy spacecrafts. When you turn the rotary decoder a bit pattern is generated from which you can decide if the wheel is turned clockwise or counter clockwise. The latest bit pattern from the

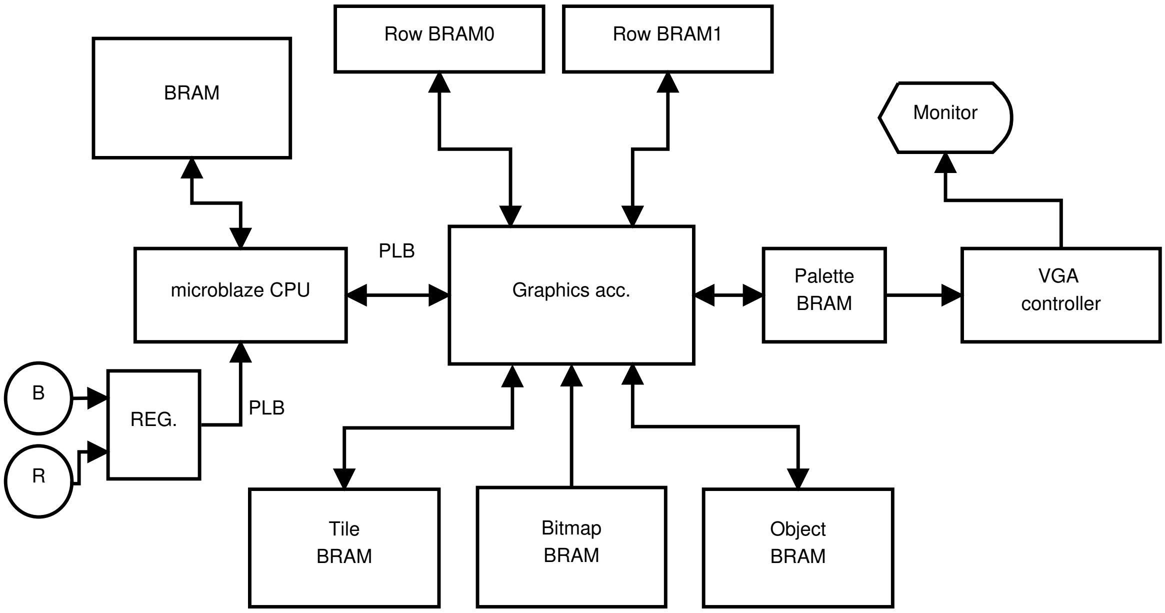

Figure 2: Overview of the systems components

rotary encoder and the state of the pushbutton is kept in registers which the CPU can read. After each screen refresh the CPU receives an interrupt and during this the CPU checks the controller registers for changes.

The output from the game is graphics on a VGA-monitor. The resolution is 320x240 (640x480 @ 60Hz). The graphics are composed of 16x16-pixel tiles stored in BRAM. These tiles are drawn to the monitor in real-time by the graphics accelerator based on values in two separate BRAM, one for the background and one for foreground objects i.e. spacecrafts and cannon fire.

2.3 Memory

The system uses seven different memories, hopefully all can be implemented using BRAM. The memories are:

- Instruction / Data memory for the microblaze CPU.
- Two 320 * log2(#colors) bit row rams used to buffer the next row of object-pixels on the monitor
- 16*16*log2(#tiles) * log2(#colors) bit Bitmap RAM to store the different graphical elements.
- 21*15 * log2(#tiles) bit Tile RAM used to store the background map.
- #objects * (9+8+log2(#tiles)+1) bit Object RAM holds coordinates and other information about the different objects on screen.
- #colors*#palettes * 8 bit Palette RAM holds the different colors available.

2.4 Graphics accelerator

The background is made up of a 20*15 grid of tiles making each tile 16*16 pixels in size. Every pixel is three bits. The available tiles are stored in the Bitmap RAM. The screen scrolls with 1 pixel resolution. The offset of the background is stored in a 9bit register and is referred to as scroll value. The information stored in the tile RAM combined with the scroll value and screen position forms an address to the Bitmap RAM and the value at that address is drawn to the screen. The position of the objects are stored in the object RAM. A line buffer is used to store what pixels of the next line that are part of an object. This is done while the current line buffer is drawn to the screen.

2.5 Software

The CPU is interrupted after each screen refresh. At interrupt the following will happen:

- Send information about object position and background scroll etc to the graphics accelerator.
- Read button registers.
- Handle collisions.
- Calculate positions for objects.
- Calculate scroll value.
- Background update.
- Wait until interrupted.

3 Time plan

To make it easier to finish the project in time and to make sure that the group members know what to do, a time plan was constructed, see figure (3)

Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Philip & Jonas	**Jonas**	**Philip**	**Philip & Jonas**	**Philip**	**Philip**	**Phlip & Jonas**
make project proposal	construct skeleton	construct interface and	start with interrupt routine	calculate positions	finishing collision	write report.
and presentation	for hardware architecture	registers for buttons	for cpu	for objects	handling	
	Philip	**Jonas**	**Philip**	**Jonas**	**Philip & Jonas**	
	generate bitmaps for	implement row buffering	Uppdate graph. memories	algorithms for choosing	testing of whole system	
	graphics memory			background and objects		
			Jonas	**Philip**	**Jonas**	
	Philip & Jonas	testing	Calculate scrollvalue	start on collision handling	start on report	
	Start construct graph. Acc.					
	posible to draw background		testing	testing		
	testing of graph. Acc.					

Figure 3: Time plan for the project, describing what to do and when to do it.