2011-10-31

Bomberman

A "Design of Embedded Systems - Advanced Course" Project

Linus Sandén, Mikael Göransson & Michael Lennartsson
et07ls4@student.lth.se, et07mg7@student.lth.se, mt06ml8@student.lth.se

Abstract

This project implements a classic game in the style of the NES game Bomberman. It uses the Digilent Nexys 2-1200 FPGA Board, a PS/2 Keyboard and a VGA screen capable of displaying 640x480 pixels. The timer that controls the gameplay and the keyboard are connected through an interrupt controller to the microblaze soft processor. The microblaze processor handles the game logics and a graphics accelerator receives changes in the game grid and outputs the full game graphics to the VGA screen.

Innehåll

Abstract..2
Innehåll..3
1. Introduction..4
2. Hardware..4
  2.1 Graphics Accelerator..4
    2.1.1 Components..5
    2.1.2 Processes..5
    2.1.3 SW/HW transfer protocol..6
  2.2 Timer..6
  2.3 PS/2..6
  2.4 Device utilization summary..6
3 Software..7
4 Installation Instructions..9
5 Lessons and conclusions..9
6 Contributions..9

1. Introduction
This project implement the classical game Bomberman on an Digilent Nexys 2-1200 FPGA Board. It is implemented as a player versus player game. The players are controlled using a PS/2 Keyboard and the game is displayed on an VGA screen in 640x480 pixel resolution.

The final project architecture is not far from the initial plan. The main difference is the addition of a timer module that provides interrupts to control the game flow. It was also decided to skip the audio part due to lack of time. The final architecture is shown in figure 1 below.

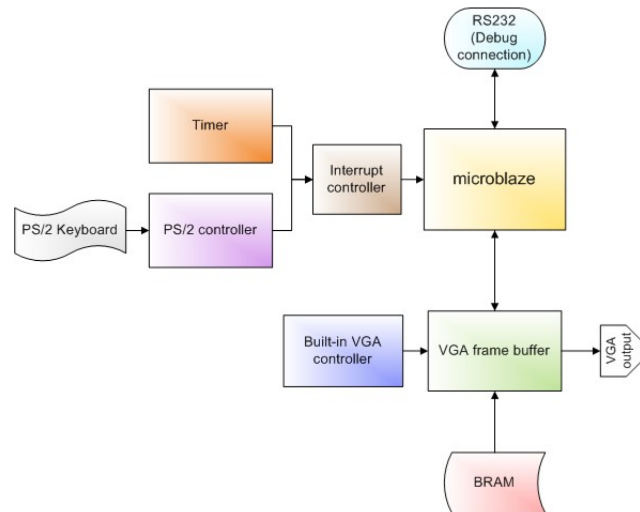

Figure 1. Final architecture

The custom hardware consists of a VGA frame buffer that uses the VGA controller and bram to create the VGA output. The frame buffer keeps track of how the playfield should look and the position of the players. It fetches the graphic pictures from the bram and fetches what pixel should be displayed from the VGA controller. The playfield and player positions are updated by reading information from the microblaze through the PLB.

The software control the game logic by using the timer and keyboard interrupts. The timer interrupts keep track of when bombs should explode and for how long they should explode. The keyboard interrupts control the two player's movement and when they place bombs. Changes in the game are sent to the hardware that then displays it.

2. Hardware

2.1 Graphics Accelerator
Most of the graphics accelerator shown in figure 2 was custom made. It includes a Xilinx VGA controller IP from the fpga documentation that provides the pixel counters and sync signals. It also includes an image bram that provides the graphics of the game.

The logic consists of a 15x20 matrix dividing the whole screen in 32x32 pixel squares. The matrix contains 4 bit values representing the different possible images the squares should contain. The logic also holds the two players vertical and horizontal positions. Both the matrix values and the positions are calculated by the microblaze and updated through the PLB.

The memory access process updates the bram address every time the pixel counters change. Every change in the bram output causes the rendering process to update the VGA output register. That register then updates the output to the screen every clock.

2.1.1 Components

The six images are written into the same bram as 32x32 pixels of 8 bit values. They are accessed by an image offset selecting which image to take from together with the position off the wanted image pixel.

The graphics accelerator use the digilent 640x480 VGA controller to get blank signal, horizontal and vertical counters and horizontal and vertical sync. The blank signal describes if the counters are inside the monitor display range. By using the 50 MHz system clock as input to the controller the output horizontal counter is twice as fast as intended. This is compensated for in the graphic accelerator by not using the LSB in horizontal counter.

2.1.2 Processes

Every time the pixel counters or the player position is updated the memory access process calculates the address to the image bram so it can fetch the desired pixel value. The players have precedence over the game board images and player 1 has precedence over player 2. Since the players can move almost anywhere on the play field the pixel position off the image in the bram is calculated by subtracting the player position from the pixel counter values corresponding to the current game board square (5 least significant bits). The game board image is dependent on the current value of the game board matrix square for the current pixel counter values. By reading the fire image column first instead of row first the image is rotated and can then be used as both vertical fire and horizontal fire.

The rendering process updates the VGA output register depending on the pixel counters and the output from the bram. The same player image is used for both players by changing all red pixels to blue when the second player image is being displayed.
The CPU input process decodes the information sent from the microblaze and updates the player position or game board matrix value. The protocol used for the communication is shown in figure 3. The bits are read different depending on if its player position info or game board info.

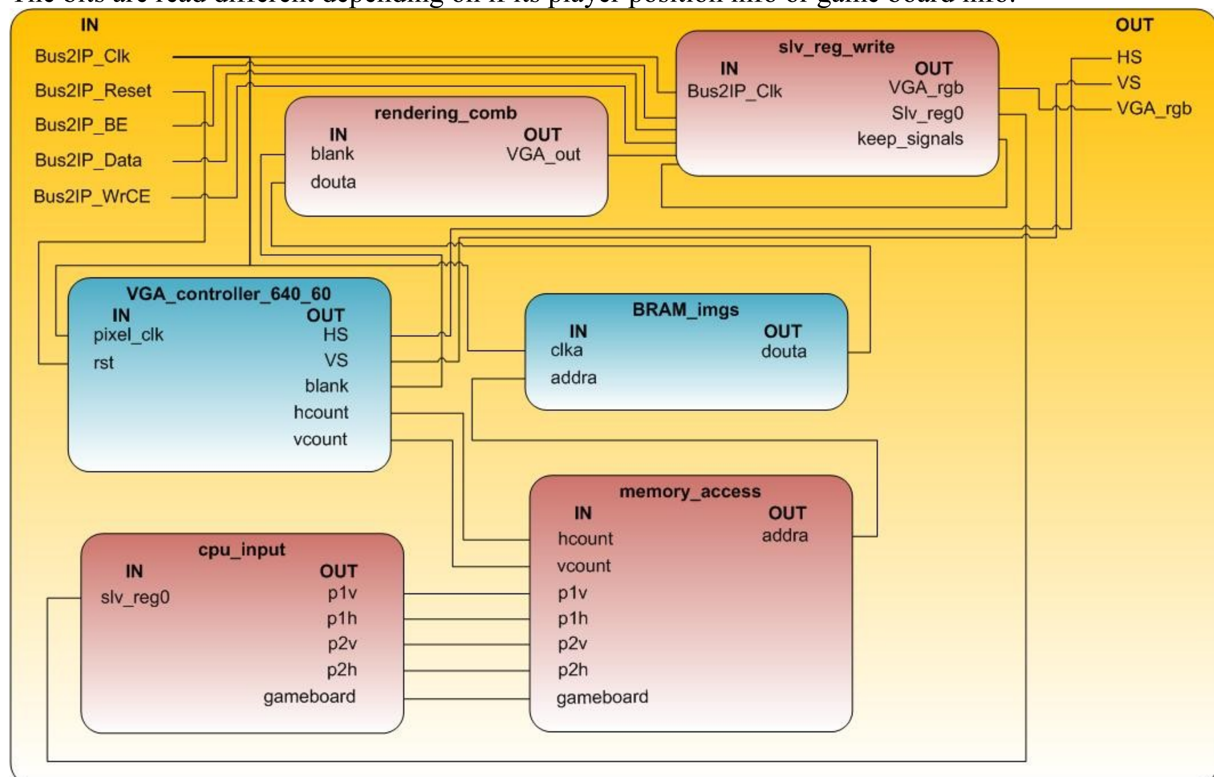

Figure 2.Hardware architecture of the graphic accelerator

2.1.3 SW/HW transfer protocol

The synchronous process writes the data sent from the microblaze through the PLB to the slave register and updates all registers. One player position or two game board updates can be sent in every update of the register.

Figure 3. Transfer protocol between microblaze and graphic accelerator. First row show bit position with 0 as MSB, second row show the info contained in the bits.

2.2 Timer

A timer IP-block is generated to provide the timing interrupts controlling the game. The code generating the interrupts was heavily inspired by fpgadeveloper.com[1]. The timer reads from the PLB bus to two registers to get the timer counter and a start flag. When the start flag is set a synchronous process decreases the counter. When the counter reaches zero an interrupt is generated by setting an interrupt flag in the data that is sent back to the microblaze.

2.3 PS/2

To get input from the keyboard a ps/2 IP-block is generated in XPS. On keystroke this core generates an interrupt that invokes the interrupt handler in the microblaze who handles the codes sent from the keyboard.

2.4 Device utilization summary

Table 1 show the hardware utilization. Notable is that the picture bram uses a big part of the available BRAMs when storing six 32x32 squares.

	Used	Available	% utilization
Slices	6182	8672	71%
Slice Flip Flops	4797	17344	27%
4 input LUTs	10404	17344	59%
BRAMs	19	28	67%
DCMs	1	8	12%

Table 1. Hardware resources used

[1] http://www.fpgadeveloper.com/2008/10/timer-with-interrupts.html

3 Software

The screen is divided into 15 horizontal rows and 20 vertical columns, these rows and columns makes a matrix of squares. The hight and the width of each of the squares are 32 pixels. Since the game board consists of eleven horizontal rows and thirteen vertical columns it does not cover the whole screen. The upper left corner of the game board is located three squares horizontal and two squares vertical from the upper left corner of the screen. The player position, p1pos and p2pos, is determined on which horizontal row and which vertical column the player is located on. The squares of the board are constructed of different tiles. The different tiles are:

- GROUND_TILE, The ground where the players are allowed to move.
- STONE_TILE, The stone blocks which can't be destroyed by bombs.
- BRICK_TILE, The brick block that can be destroyed by bombs.
- BOMB_TILE, The bomb.
- FIRE_C_TILE, The centre of en exploding bomb.
- FIRE_H_TILE, The fire in the horizontal direction.
- FIRE_V_TILE, The fire in the vertical direction.

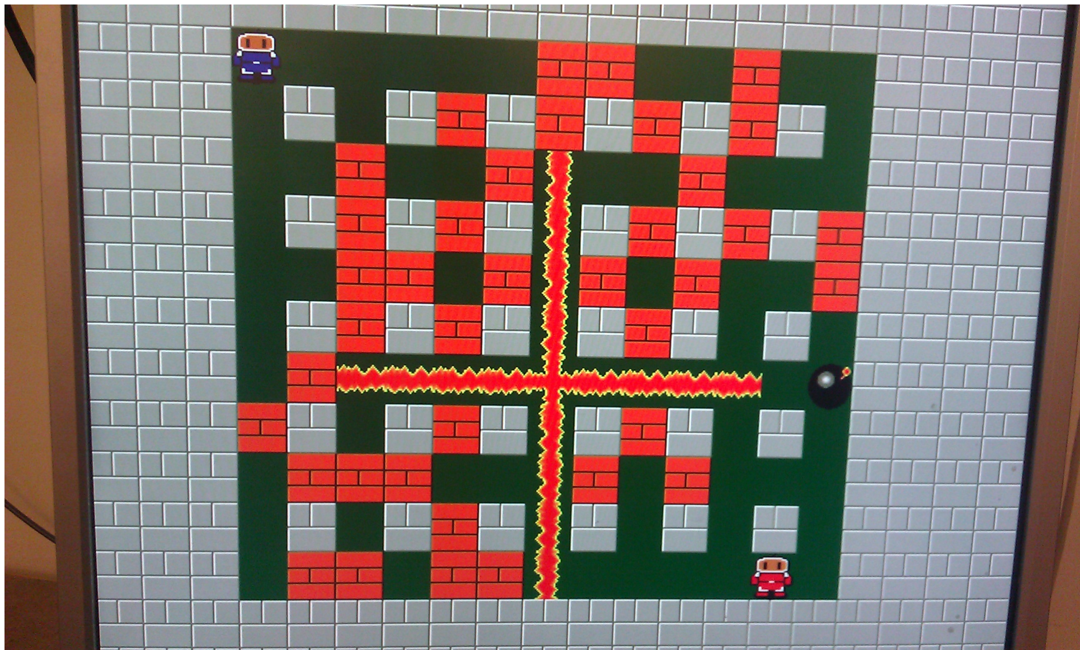

Figure 3.

The main-function of the Bomberman game contains of two while loops. The first while loop starts up the game by initializing the game board with the init-function, the interrupts from the keyboard and the hardware implemented timer are also initialized. When leaving the first while loop and entering the second everything is initialized and the game is ready to use. The second while statement contains two if statements that has the two boolean p1dead and p2dead. If either player 1 or player 2 dies p1dead or p2dead turns true and the if-statement is entered. Then the win-function is started and the game board is initialized once again and the game is restarted.

When the game is started the game board is initialized by the function called init. The init-function first draws the STONE_TILES of the game board. The game board is drawn with two for-loops starting from the upper left corner of the game board and drawing a STONE_TILE where the horizontal rows are even and the vertical columns are odd. The starting position for player 1 is at the upper left corner and starting position for player two is the lower right corner, these squares are

declared as GROUND_TILE. The rest of the squares of the game board is declared either as GROUND_TILE or BRICK_TILE with a random function.

The game does not have any functions that update regularly instead the game updates with interrupts. There are two kinds of interrupts, one that reacts to when certain keys are pressed on the keyboard. The other interrupt reacts to a hardware implemented timer. There is also an interrupt handler that keeps track on which of the interrupts that has been sent.

The function that reacts to the keyboard strokes are mapped to certain keys for the different players. The keys a, s, d, e, and r are reserved for player 1 and the keys j, k, l, I and u for player 2. The keys:

- a and j, are the left direction for player 1 respectively player 2
- s and k, are the downwards direction for player 1 respectively player 2
- d and l, are the right direction for player 1 respectively player 2
- w and I, are the upwards direction for player 1 respectively player 2
- e and l, are for placing bombs for player 1 respectively player 2

The function that reacts to the keyboard interrupts consists of a switch case statement. If any of the direction keys are pressed the case-statement checks the square in that direction. If the square is declared as a GROUND_TILE the player is allowed to move and the player position is updated and the new position is sent to the graphics accelerator. If the square is declared as less than BOMB_TILE, in other words as any of the FIRE_TILES, the boolean p1dead or p2dead is change to true for the actual player. When the e or l key is pressed a bomb is added to the game board by the function add_bomb

The add_bomb function adds a bomb to the square when player 1 or player 2 presses the add bomb key. Both players are allowed to place two bombs each at the same time on the game board, once a bomb is detonated the player is allowed to place another bomb. The add_bomb functions keeps track of how many bombs each player has places on the game board. When a bomb is dropped a timer is started, MY_TIMER_Intr_Handler keeps track of how long time it is before the bomb explodes. The fuse of a bomb is two seconds.

The hardware implemented timer ticks for every half second. MY_TIMER_Intr_Handler is the function that checks if anything happens every time a timer interrupt occurs. The only thing that reacts to the timer interrupt is the bombs. MY_TIMER_Intr_Handler checks every bomb-timer once a interrupt happens. If a timer for a bomb has expired the bomb will explode and the explode-function will be triggered.

The explode-function has the three parameters horizontal- and vertical position and the boolean clear. If clear is set to false the explode-function will draw an explosion with center of it in the horizontal- and vertical position and declare the square as a FIRE_C_TILE. After that it will check if there are any players in closest four squares in the four directions from the explosion center. If any players are detected the boolean p1dead or p2dead will be set to true and the game will restart. If no players are detected the explode-function will enter a switch case statement that checks four squares in every direction starting from square closest to the center. If the square that is checked is declared as GROUND_TILE or BOMB_TILE it will be declared as FIRE_V_TILE or FIRE_H_TILE depending in which direction that is checked after that it will continue by checking the next square. If the next square is a BRICK_TILE the function will declare it as FIRE_V_TILE or FIRE_H_TILE and finally break the case-statement. If the square is STONE_TILE it will break directly. Any FIRE_TILEs will just be ignored.
If clear is set to true the explode-function will check the squares by the same principle stated above but instead of declaring FIRE_TILEs it will replace all the FIRE_TILEs with GROUND_TILEs.

4 Installation Instructions

To run the game the file "top_downlad.bit" (located in ../BombermanISE/) can be downloaded to the FPGA-board using Digilent Adept. To edit any of the game hardware or software and recompile Xilinx ISE and XPS is recommended.

5 Lessons and conclusions

One lesson learned during hardware creation was the importance of checking all settings during memory generation. A lot of hours were lost compiling hardware with faulty memories. This ultimately caused us to end up with a single memory that stores all the sprites for the game. The biggest consequence of this is that the bombs are not visible behind the players. The gameplay where then changed to enable "shooting" where you place a bomb and not move and still surviving. This ended up being even more exciting gameplay and where therefore never changed.

A partly related issue is that the animations for seamlessly moving the players between positions had to be scrapped due to lack of time and space for code on the FPGA. This would also have required the separation of the player sprite to another memory or a very complicated algorithm for choosing which sprite to fetch from memory.

The software and the game logic implementation was pretty straight forward. An action is taken when an interrupt occurs, when the problem how to manage the two different interrupts was solved the implementation of the game logic was fairly easy. The game logic is based mainly on if-statements when an interrupt is detected. Since there are a lot of things that can happen when an interrupt occurs there are many if-statements. Due to limited memory resources the code were rewritten a little bit to be able to fit in the limited memory. For example were the player positions stored in temporary variables instead of using them every time in the if-statements. The main features of Bomberman that that we wanted achieve are implemented but there are of course improvements to be made. Due to limited time and memory we decided that we were happy how the game was working.

The most important lesson to be taken away from this project is that we obviously have the capacity to create a playable and fun game for FPGA in a month.

6 Contributions

The final report was divided into Software (Michael), Hardware (Mikael) and rest (Linus). Presentations were made by Linus. The graphics accelerator was made by Linus and Mikael, the Keyboard code was interfaced by Linus & Mikael and the timer was a joint project between all group members. The Software interrupt controlling was done by Linus & Mikael. Testing and Game Logic construction were done by all members.