Kaboom, an Atari 2600 Classic, revisited!!!

Team Name: Team Kaboom

Team members:
Laura Kamfonik
Ramakrishnan Lakshmanan
Mettu Sai Krishna Sankar
Juan Carlos Morales Torrico

Kaboom: a quick recap/abstract

Kaboom is an arcade game developed by Atari. It was one of the most popular games in the 1980's. Insanely addictive and highly challenging, it is a game one could play for hours.

In our project, we have tried to emulate this classic on a Xilinx Spartan 6 FPGA by making it accessible(user-friendly) for the present generation, with a bluetooth version of the game and also added our own innovative twist to it. In Kaboom, the central character is a bomber who drops bombs continuously. The player controls a set of three pads placed at the bottom of the display, which catch the bombs. Every bomb caught gives an extra point. For every bomb missed, a pad vanishes. The game ends if all three pads vanish. Lost pads can be regained when the user reaches a certain threshold of points.

The original Kaboom is a single player game where the bomber moves in a random manner, and the player only controls the pads. In our version, we have incorporated a single player as well as two player mode, the two player mode being one where the player(s) control both the bomber and the pads. Additionally, we have also added pause/resume functions, and a reload option if the game ends because of the player losing/winning the game. We have also incorporated audio for explosions, bomb catching, and the "Super Mario" game over tune for game over.
Most importantly, we created an android app which shows the pads and background on a smartphone touch-screen. The app allows the player to play the game via bluetooth, as mentioned above, where, when the pads are moved on the phone screen, they move in the same way on the display too.

Following are the verilog modules which we have used in our code:

- **<u>vga_display.v:</u>** This is our top module. This module is the heart of our project, and contains its set of signals which help in drawing the display, i.e. sprites, backgrounds etc. via the VGA port. This module also contains signals which control the activation/deactivation of various sounds in the game like bomb catch,explosion, "you won/game over" screens etc, lines of code for the bluetooth functionality etc.

- **score_2_dec.v:** This module takes a 10-bit binary value as an input, and provides the equivalent decimal value as the output. We have a four-digit decimal score displayed on the screen, which increments each time a bomb is caught. This module converts this score to decimal form, which is displayed on the screen through the VGA.

- **binary_to_seven.v:** This module converts a 4-bit input binary value to its equivalent 7-segment value, such that the 7-segment display shows the hex-equivalent of the input binary number. Larger binary numbers must be handled four bits at a time because the Nexsys 3 board only has one four-bit data input to the 7-segment display.

- **seven_segment.v:** This module displays the score on the on-board 7-segment display. It has a clock signal and a 16-bit data input. This module smartly cycles through each 4-bit nibble of the input and the 4-bit, active-low selector output, feeding the correct nibble to each of the four 7-segment digits. This happens in an endless loop, fast enough for the human eye to see four steady numbers continuously displayed, despite the fact that they are actually turning on and off in quick succession.

- **font_rom.v:** This module stores a 2-D array register of 16x32 bit tiles, each of which represents an alphanumeric character. The module receives a clock and a selector as input, and outputs the selected character tile. The design allows scaling of the tiles on the VGA.

- **rand_gen.v:** This module controls the bomber's movements in the 1-player mode. It consists of 2 LFSR modules of different sizes, each of which produces a pseudo-random bit every clock cycle. The two bits are combined to form a pseudo-random number.

- **LFSR_5.v:** This module contains a 5-bit Linear Feedback Shift Register. Each clock cycle, the most significant bit is passed to rand_gen.v.

- **LFSR_8.v:** contains an 8-bit Linear Feedback Shift Register. Each clock cycle, the most significant bit is passed to rand_gen.v.

- **vga_controller.v:** This is by far, the most critical module of our project. All the drawing that takes place on the display is controlled by this module. The VGA works at a clock frequency of 25 MHz. The monitors that we have used, have a resolution of 640X480 pixels. This module specifies the terminal values of the horizontal and vertical front porch, and creates horizontal and vertical beam traces. This module is used as a fundamental basis for drawing all our sprites on the display.

Important instructions/Highlights of how we worked on getting things done:

The most fundamental or critical aspect of this project is to first get the sprites to show up on the display. For that, the sprites have to be created on a graphics editing tool. We have used "Gimp" for this purpose. That done, the next objective is to extract the .coe files from each .bmp or .jpg sprite. This is done through C#, which generates the .coe files as a number of rows of 8-bit binary numbers. The .coe files are fed to the verilog code using the BRAM, which is generated via IP-coregen. Each line in the .coe file specifies an aspect of the sprite. From our code, the VGA actually reads each line, draws the corresponding pixel, increments to the next line, and this continues till the last line of the .coe file. Once the last line is reached, we have instructed the VGA to go back to the first line and repeat this process. This process is repeated over and over again, so we can actually see a stable figure/character of our game, on the display. Once the drawing of the sprites is taken care of, we can easily specify where a particular sprite has to be drawn. We can also move the sprites from one position of the display to another, via buttons on the Nexys-3 board, or hard-code them to move in a defined manner automatically.

The font ROM is an interesting feature of our game because it can be used to add new text and other features to the game, or could be easily ported to other games. Each character is stored as a 16x32 bit "tile" where 0 indicates whitespace and 1 indicates the character. Whitespace padding around the top, right, and bottom edges of the each tile preserve an appropriate spacing when the tile is scaled. The size scaling is implemented using a number of counters, and registers storing the scale factor and number of characters in the line. Vertical and horizontal counters keep track of the VGA's progress through a tile, and a character counter tracks progress through the line. A line counter can be added to handle multiple lines of characters. The scale factor is used along with a second set of vertical and horizontal counters to repeat each pixel a the correct number of times in both directions, allowing the tile to be drawn at the size desired.

The implementation also uses the scale factor to define the borders of the character blocks on the screen. Because all the tiles are the same size at any scale, an entire line of character position blocks can be defined easily in a short generate loop. The scale factor can be updated dynamically, allowing text to grow or shrink on the screen as the game runs.

Recommended improvements that could be made in the future:

We were able to achieve most of our goals with the game, which in its finished state is fun, playable, and looks good. The development process was plagued by sync issues with the sprites. We were able to stabilize the sprites somewhat by implementing resets. The root cause of these sync issues was identified near the end of the semester, but the fix would involve an entire redesign of the code structure, for which there was not enough time remaining. If we were to do the game over again, we would implement the movement and VGA drawing of the sprites within the same always block.

Conclusion:

Our final step was to build on the button-controlled movements, and got the sprites to respond to controls from a joystick on an arcade cabinet, to give that amazing experience of playing an actual arcade game!