

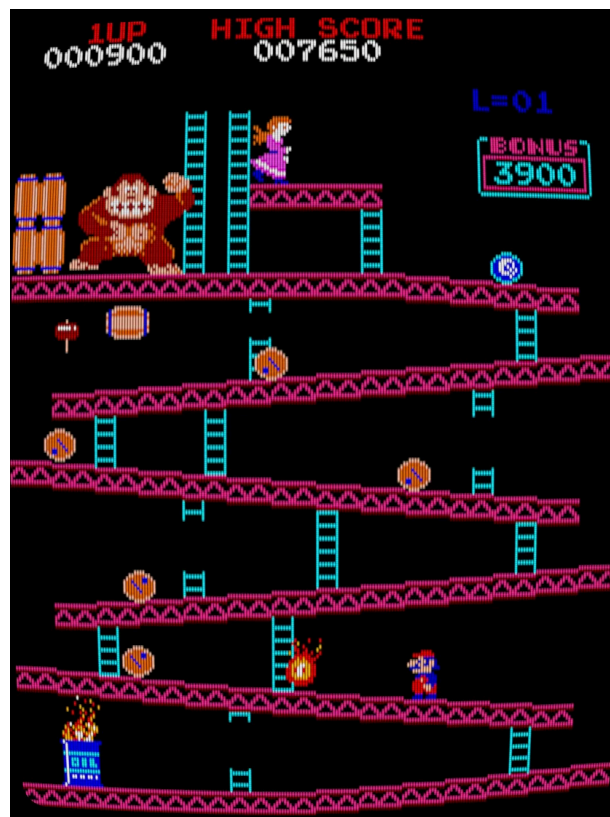
# Design Document

## Donkey Kong



Ania Róża Krzyżańska (ark2219) , Sean Stothers (sps2308),  
Ines Khouider (ik2512)

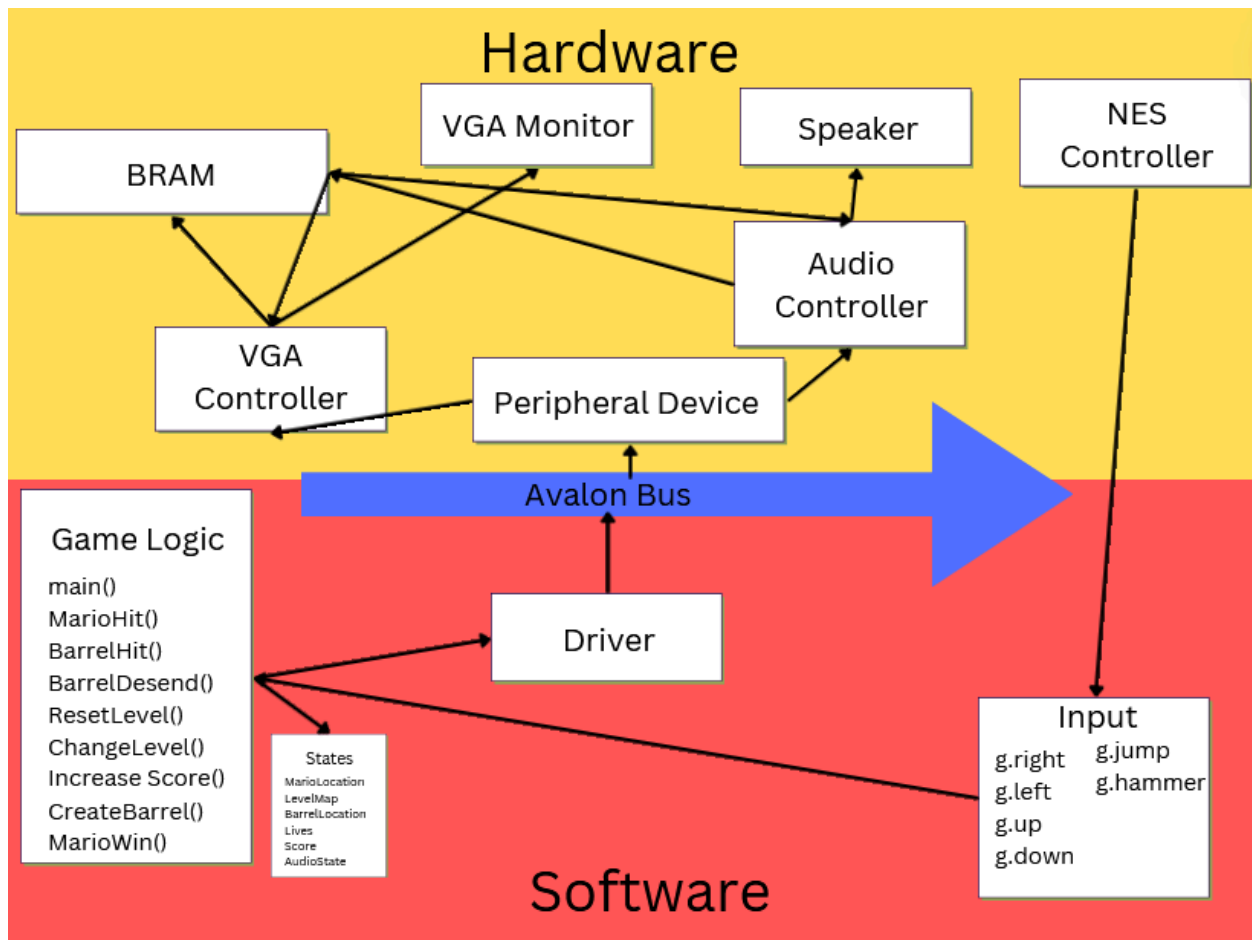
|  |           |
|--|-----------|
| <b>Introduction</b>                    | <b>2</b>  |
| <b>Systems Block Diagram</b>           | <b>2</b>  |
| <b>Algorithms</b>                      | <b>3</b>  |
| Game Logic                             | 3         |
| Algorithm for Graphics Generation      | 4         |
| Algorithm for Sound Generation         | 5         |
| <b>Resource Budgets</b>                | <b>5</b>  |
| <b>The Hardware/Software interface</b> | <b>8</b>  |
| <b>Sources</b>                         | <b>10</b> |



# Introduction

We are going to use an FPGA to recreate the game Donkey Kong using the FPGA. The player will control the Mario Sprite using an NES controller, as Fire Sprites and Barrels come barreling down to prevent Mario from reaching the top of the map. Mario will be able to climb ladders and jump to avoid the obstacles. Audio clips will be played at relevant times during the game, such as when the game is won, or when Mario loses a life.

## Systems Block Diagram



The game logic, driver and input, as well as the states of the game are tracked in the software, taken from the NES controller. The driver feeds information into the Avalon bus which goes into the peripheral device that feeds information into the controllers until it goes into the monitor and speaker.

# Algorithms

## Game Logic

We'll be using a NES controller to control the movements of Mario and to start the game. The following controls will be implemented:

**Move\_left/Move\_right** - at a high level, pressing the left button corresponds to shifting the mario sprite over to the left and pressing the right button corresponds to shifting the mario sprite to the right

More specifically, a left press following a right press first changes the displayed sprite to a sprite of mario facing left with his front leg extended leftward and also shifts the sprite left some number of pixels (exact number will be determined through testing). Further left presses will rotate cyclically between displaying mario with his front leg extended, mario with both legs together and mario with his back leg extended to create the illusion of walking.

Regardless of the "version" of mario displayed, the sprite will shift left the same number of pixels. Pressing and holding the left button results in continual motion leftward until a barrier (ie the edge of the screen) is hit.

The same holds for a right press following a left press but with rightward motion and mario facing rightward.

### Jump

when the designated button (different from the up/down and left/right buttons) is pressed, the mario sprite displayed will be mario in a jumping position (facing left or right based on his position before the press) and will be shifted up a certain number of pixels gradually and then lowered back down the same number of pixels. If a left or right key is pressed while this occurs, the Mario sprite display won't change, but the sprite will likewise shift left or right accordingly.

**Climb\_up/climb\_down** - pressing the up button near a ladder shifts mario up and changes the sprite displayed

when the up button is pressed and the mario sprite is within the some experimentally determined range of pixels near a ladder, the sprite displayed will change to a sprite of mario from behind. If up is continually pressed or pressed and held down, there will be a cyclic rotation displaying mario between two poses (with left foot raised and with right foot raised) to create the illusion of him climbing the ladder until the sprite reaches some threshold of pixels at the top of the ladder, at which point the sprite displayed will show mario from behind and he will not be able to move up any further. Each press within the appropriate range of a ladder also simultaneously corresponds to shifting the mario sprite up a certain number of pixels.

The same procedure applies to downward motion except the sprite can only move downward when the down button is pressed and the sprite is at the top of a ladder and stops when hitting a ramp at the bottom of the ladder.

**Start** - pressing the designated “start” button will start the first level, i.e. display the appropriate background with ramps, start the rolling barrels, display the score of 0, lives set to 3, etc.

**Restart** - pressing the designated restart button will reset the number of lives to three, set the displayed score to zero, set the level back to Level 1 and move the mario sprite back to the bottom left corner of the screen in its initial position

**Hammer** - when the mario sprite is within some given range of a hammer (one of two sprites that is positioned on the “course” at the start of the level) the mario and hammer sprites appear to “combine” (ie the hammer sprite is no longer displayed and the mario sprite changes to cycle between two states, one with hammer up and one with the hammer down).

The left and right buttons are functional in this state, but mario cannot climb up, climb down or jump. This state only lasts a specified amount of time (around 5 seconds). See the “enemies” section for more information on destroying barrels and the “audio” section for information about the song that plays.

Enemies - In our case, the “enemies” are the barrels that Mario must avoid. If Mario is hit by a barrel, he dies (ie lives decrement by 1). If the number of lives reaches 0, the game ends

Barrels will also be sprites, moving along the ramps and eventually leaving the field of view when they reach the bottom left corner of the screen

Updating the displayed points:

Jumping over a barrel- +100

Destroying barrel -+300

Paulina:

If a mario sprite is within a certain range of pixels of the Paulina sprite, the level ends

## Algorithm for Graphics Generation

Preloaded sprites will be stored in the sprite generator table to be consulted by the sprite controller and displayed line by line accordingly based on the state of the game. The background ramps and ladders as well as Paulina will be tiles instead of sprites.

## Algorithm for Sound Generation

The audio in Donkey Kong uses a mix of sound effects in the form of pre-sampled audio and background music generated using a triangle wave channel. In general, there are two channels, one for sound effects and one for music.

To play the music, we will break the music into notes, where each note would have the frequency to be played by the triangle wave generator. 0Hz would be used to represent a rest.

To play the sound effects, each pre-sampled audio segment will be stored in a table in memory, and each byte will be sent to the DAC at the other end of the channel at that set frequency (which will be about two times the highest frequency component in the sample, obeying Nyquist sampling theorem).

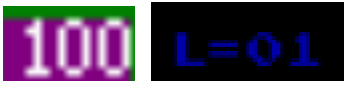
Since the sound effects and music have their own channels and only one sound effect is played at a time there doesn't need to be a mechanism to prioritize sound effects or sound effects vs music.






## Resource Budgets

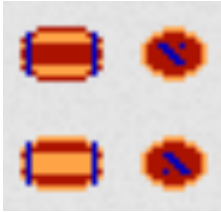


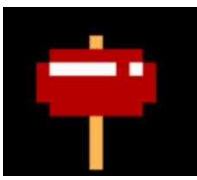
We need to make sure all our sprites will fit into the BRAM of the FPGA. We have 256 Kb on the FPGA.

We want the sprites to have colors like in the original Donkey Kong game, meaning 8 bits per color and 24 bits per pixel.

### Sprites

| Category        | Graphics  | Size (bits) | Number of images | Total Size (bits)                                     |
|-----------------|---|-------------|------------------|---|
| Score and lives |  | 13 x 15     | 12               | $24 \times 12 \times 13 \times 15 \times 3 = 140,400$ |
| Mario           | Climbing  | 40 x 30     | 14               | $24 \times 40 \times 30 \times 14 = 403,200$          |
|                 | Left  |             |                  |   |

|             |   |   |   |  |  |
|-------------|---|---|---|--|--|
|             | Right   |  |   |  |  |
|             | Hammer Left   |  |   |  |  |
|             | Hammer Right  |  |   |  |  |
| Donkey Kong |   | 50x50   | 4 | $24 \times 50 \times 50 \times 4$<br>$= 240,000$ |  |
| Heart       |  | 16x16   |   | $24 \times 16 \times 16 \times 1$<br>$= 6,144$   |  |
| Barrel      |   | 16x16   | 2 | $24 \times 16 \times 16 \times 4$<br>$= 24,576$  |  |

|          |   |       |   |   |
|----------|---|-------|---|---|
|          |  |       |   |   |
| Ladders  |  | 20x30 | 1 | $24 \times 20 \times 30 \times 1 = 14400$ |
| Platform |  | 20x30 | 1 | $24 \times 20 \times 30 \times 1 = 14400$ |
| Hammer   |  | 10x10 | 1 | $24 \times 10 \times 10 \times 1 = 2400$  |
| Total    |   |       |   | 845,520                                   |

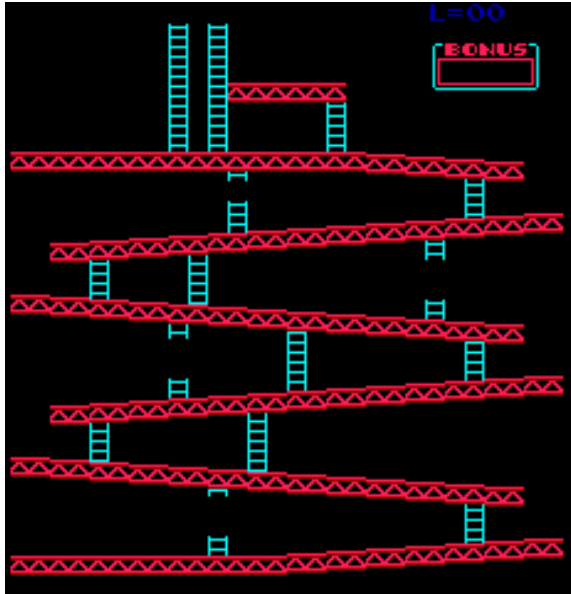
## Audio

|                  | Time (s)           | $f_s$ (KHz) → 12 is 9 bits | Memory (bit)                         |
|------------------|--------------------|----------------------------|--------------------------------------|
| Mario Die        | 4                  | 11.025                     | $9 \times (4 \times 8000) = 288,000$ |
| Mario Win        | 5 (last 5 seconds) | 11.025                     | $9 \times (5 \times 8000) = 360,000$ |
| Hammer Smash     | 2                  | 11.025                     | $9 \times (2 \times 8000) = 144,000$ |
| Jump over barrel | 1                  | 11.025                     | $9 \times 8000 = 72,000$             |
| Jump             | 1                  | 11.025                     | $9 \times 8000 = 72,000$             |
| Background Music | 1                  | 11.025                     | $9 \times 8000 = 72,000$             |
| Total            |                    |                            | 1,008,000                            |

From both audio and video we have 1,853,520 bits is 231.69 Kb which is less than 256 bytes. We need to be careful about our storage though as the sprites and audio is about 90% of our storage.

## Maps

To make the game more challenging we are introducing 3 levels. Each screen is about 550x550 pixels and is populated with platforms and ladders laid onto each other to look like longer pieces, and to give a slanting effect. Pauline, along with the ladders and platforms will only change positions when the levels change.



## The Hardware/Software interface

Mario (x,y, state)

Barrels (x, y, state)

Donkey Kong 2 bits

Heart 1 bit

Score - 6 digit decimal number -> minimum 20 bits

Each register contains 1 byte. Data is stored in the big endian format.

|              |                    |
|--------------|--------------------|
| Register 1/2 | Mario X MSB/LSB    |
| Register 3/4 | Mario Y MSB/LSB    |
| Register 5/6 | Barrel 1 X MSB/LSB |



|                   |  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------------------|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Register 7/8      | Barrel 1 Y MSB/LSB   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Registers 9 -> 44 | Repeat of 5/6/7/8 for the other 9 barrels  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Register 45/46    | Hammer X MSB/LSB   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Register 47/48    | Hammer Y MSB/LSB   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Register 49/50/51 | Score MSB/.../LSB  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Register 52/53    | <p>52</p> <table border="1" style="width: 100%; text-align: center;"> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td> </tr> </table> <p>Bit 0: Display Barrel 1<br/> Bit 1: Display Barrel 2<br/> Bit 2: Display Barrel 3<br/> Bit 3: Display Barrel 4<br/> Bit 4: Display Barrel 5<br/> Bit 5: Display Barrel 6<br/> Bit 6: Display Barrel 7<br/> Bit 7: Display Barrel 8</p> <p>53</p> <table border="1" style="width: 100%; text-align: center;"> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td> </tr> </table> <p>Bit 0: Display Barrel 9<br/> Bit 1: Display Barrel 10</p> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0                 | 1  | 2 | 3 | 4 | 5 | 6 | 7 |   |   |   |   |   |   |   |   |   |   |
| 0                 | 1  | 2 | 3 | 4 | 5 | 6 | 7 |   |   |   |   |   |   |   |   |   |   |
| Register 54       | <table border="1" style="width: 100%; text-align: center;"> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td> </tr> </table> <p>Bit 0: Display/Hide Heart<br/> Bit 1: Release Barrel<br/> Bit 2: Mario/Barrel Collision<br/> Bit 3: Mario/Hammer Collision (pick up hammer)<br/> Bit 4: Climb Ladder</p>   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |   |   |   |   |   |   |   |   |
| 0                 | 1  | 2 | 3 | 4 | 5 | 6 | 7 |   |   |   |   |   |   |   |   |   |   |

|             |   |   |   |   |   |   |   |   |   |
|-------------|---|---|---|---|---|---|---|---|---|
|             | <p>Bit 5: Hammer/Barrel Collision</p> <p>Bit 6: Barrel/Ladder Collision</p> <p>Bit 7: Jump over barrel</p>  |   |   |   |   |   |   |   |   |
| Register 55 | <table border="1"> <tr> <td>0</td> <td>1</td> <td>2</td> <td>3</td> <td>4</td> <td>5</td> <td>6</td> <td>7</td> </tr> </table> <p>Bit 0: Mario facing right</p> <p>Bit 1: Mario facing left</p> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0           | 1   | 2 | 3 | 4 | 5 | 6 | 7 |   |   |

## Sources

[http://www.ee.ic.ac.uk/pcheung/teaching/ee2\\_digital/de1-soc\\_user\\_manual.pdf](http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/de1-soc_user_manual.pdf)

<https://www.mariouniverse.com/sprites-nes-dk/>

<https://github.com/furrykef/dkdasm/blob/master/dkong-snd.asm>

<https://github.com/furrykef/dkdasm/blob/master/dkong-snd.asm>