ICOBS Game

Raffael Rocha Daltoé

April 7, 2024

Abstract

This game was developed for the class of Architecture of Circuits Numeric 2, the focus was to develop the game PACMAN, which is a traditional game from before 2000. Today is obsolate but some people continue to play, striving to set new records.

1 Introduction

The development of this game was based on the architecture of the process ICOBS that use the Instruction Set Architecture (ISA) of RISC-V, the goal was to implement a game using the language C and make the link with the processor. To do the communication we used the protocol AHB-Lite, where was developed with specification set AMBA (Advanced Microcontroller Bus Architecture) by ARM.

2 Software Layer

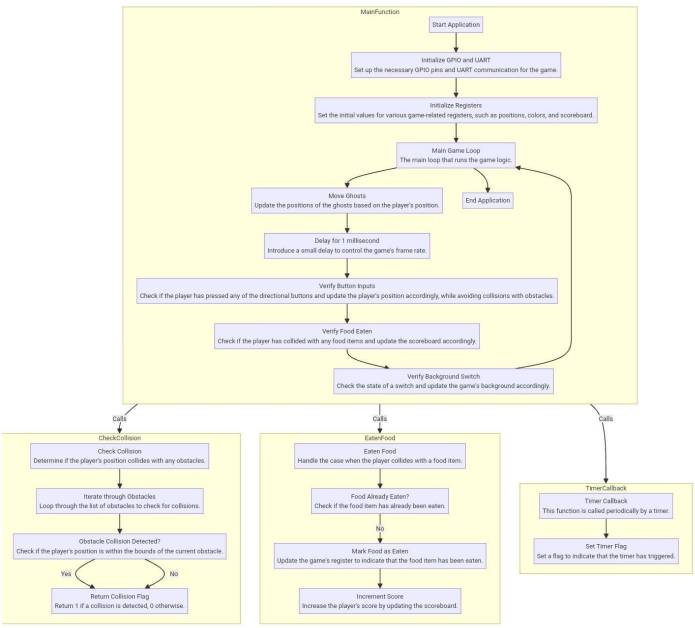

Figure 1: Flowchart of C code.

2.1 Creation of Sprites

2.2 Game

The game PACMAN, features four ghosts, one main character and the map with obstacles, the obstacles were created in Software using a struct with four fields, with the position X and position Y, but both

with minimum and maximum position, because was necessary to define the limits of the rectangle. The Food and Position of the main puppet basically use the same structure, but they were divided in two structures, called Food_Pos and Position, respectively.

2.3 Structure of Collision

The goal here was to create a good way to manage the comparison at each iteration when the puppet is moved, because at each time when the PACMAN is moved the software need to verify if the PACMAN not is surpassing the limits of the map or passing by a obstacle. To each obstacle was defined a rectangle to represents on the map, was used a function to print the position of the PACMAN and manually set. Each time when the player a button, BTNR,BTND,BTNL or BTNU, have a comparison to see if have a collision, if have he can not pass to the desired position. Ghosts do not rely on Pacman's movement, meaning they move freely; they only depend on Pacman's current location. If Pacman is against a wall, with a block separating him from the ghosts, the implemented algorithm isn't intelligent enough to navigate around the block, but it still manages to reach Pacman.

2.4 Structure of Foods

The function verifyEats() see if the PACMAN arrived in the Food, if he arrived, the program change the value of the register of Foods and send to the Hardware level to unset the sprite of the Food and increase a counter of eaten foods. Was used a vector structure with the position of each food, and inside of the while loop each iteration of the puppet with the map, the verifyEats() see if he eaten.

2.5 Switches

The user can control the colour of the Background, with the function verifySwitchBackground(), the user just needs to change the position of the Switches on the BASYS3. The user can also use the 12th switch to control the game's difficulty.
At the start of the game, the difficulty is set to low, but if the user activates switch number 12, the game becomes interesting.

2.6 Structure Movement

To move the PACMAN was necessary to create the struct Position, where is used each time to see if the PACMAN arrived in a obstacle, to this structure is used only the position on the axis X and the position on the axis Y.
At the final the ghosts were hard to control, was used the same idea of the obstacles, to see if have some obstacle and to verify it to each ghost.

2.7 Optimization

Using -O3, -fdce, and -flto are compiler flags that enhance the performance and efficiency of compiled programs. The -O3 flag activates aggressive optimizations in the compiler, potentially improving execution speed by rearranging code, inlining functions, and utilizing specific hardware instructions. The -fdce flag stands for "Dead Code Elimination," which removes code that doesn't affect the outcome, reducing the size and possibly increasing the speed. Lastly, -flto enables "Link Time Optimization," allowing the compiler to optimize across all modules as a whole, leading to better optimization decisions and performance improvements. These flags can significantly optimize the final executable, though they may increase compile time.

2.8 Connection with the Hardware

The connection with the hardware was created the defines in C code to connect the GPIOC and GPIOA, which are responsible by the buttons and the switch respectively. Using the structure already implemented in C code, at the file *arch_gpio.h*.

3 Hardware Layer

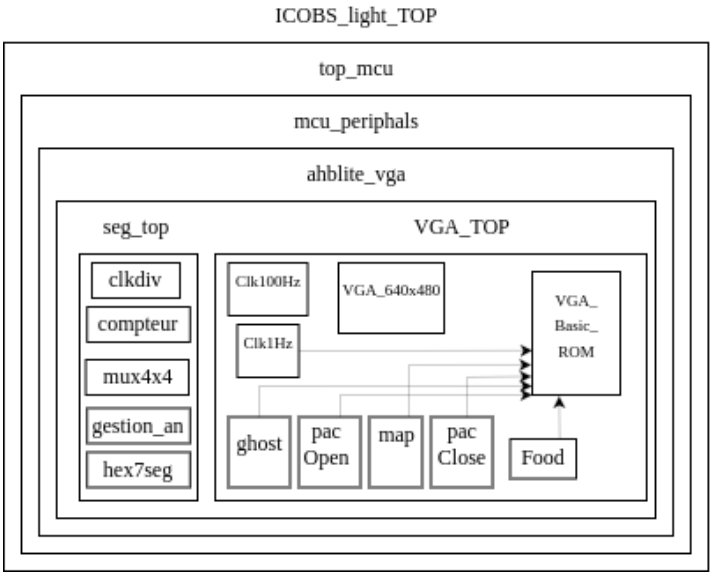

Figure 2: Schematic of ICOBS.

Register	Description
X0,Y0	PACMAN - MOVEMENT
X1,Y1	GHOST0
X2,Y2	GHOST1
X3,Y3	GHOST2
X4,Y4	GHOST3
Scoreboard	Score on 7SEG
Register_Foods	Register of foods
BACKGROUND	SW

Table 1: Table of registers.

3.1 New peripheral

The new peripheral was called *ahblite_vga.vhd* and the registers are declared in the text following this section.

3.2 Economy of Memory

Embedded Systems have few memory and with this limitation, was necessary to economize space on BASYS3, because it has almost 1.800 Kbits of RAM and non-volatile storage, it includes a 32Mbit non-volatile serial Flash device. The division of map to sprite started after the Vivado reported the exceeded of memory, the idea was to divide the map in four parts, and used 1 over 4 of the map and replicate in VHDL code the sprite of the same map, because the map is symmetric.

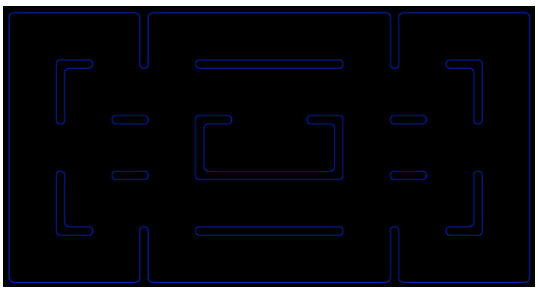

Figure 3: Original Map.

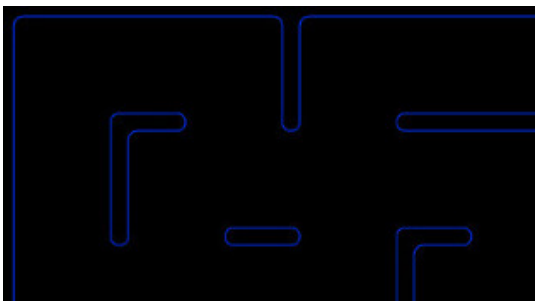

Figure 4: Cut map.

The idea was to mirror the map image. For example, if we're in the first quadrant(arrow 1), we print the image normally on the screen, but if we're in the second quadrant(arrow 2), we need to mirror the image horizontally, meaning we need to read the ROM addresses from right to left. The most "critical" case is the bottom right quadrant(arrow 4) because we need to mirror both horizontally and vertically. So we literally read the memory completely in reverse. We can see better below the idea of the map.

And was used the same sprite to all ghosts of the game, to change their color was used the OR logic and AND logic in the VHDL code. Was created only five Block Memory, they are: Food, Map, Pacman with mouth closed, Pacman with mouth opened and to the Ghost.

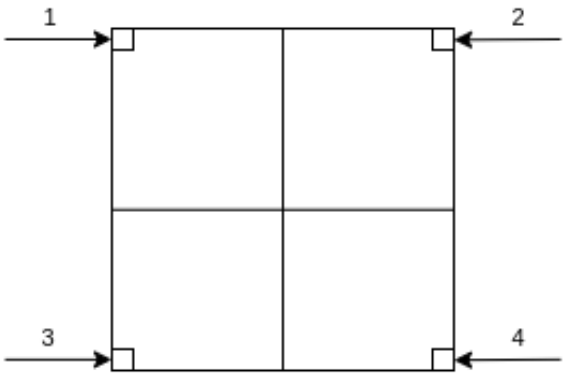

Figure 5: Read of ROM.

At the image below, it's possible to see how much resources was used on this game.

Resource	Utilization	Available	Utilization %
LUT	4994	20800	24.01
LUTRAM	44	9600	0.46
FF	2778	41600	6.68
BRAM	50	50	100.00
DSP	7	90	7.78
IO	66	106	62.26
MMCM	1	5	20.00

Figure 6: Original Map.

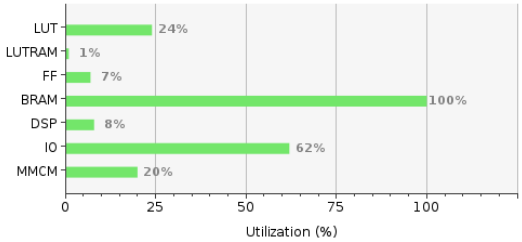

Figure 7: Cut map.

I've optimized our FPGA to the fullest, reaching 100% utilization of the Block RAM (BRAM). This full capacity is harnessed to create seven distinct memory blocks. These blocks are crucial as they work in tandem to generate and manage a total of 33 sprites on the screen simultaneously. Such an array of sprites enriches the visual dynamics and overall gaming experience.

3.3 Creation of Sprites

The critical part to create the sprites was to replicate the map, after this, the idea was to create a animation of the PACMAN, when he open and close the mouth, then i used a clock of 1Hz, each 1 second the mouth of the PACMAN open and close. To create the sprite of the PACMAN and ghosts was necessary to make the link the position of the registers on the level Software, and when the register change the position of the sprite change. At the total the code have 31 sprites, including the 4 sprites of the maps 2 sprite of the pacman, 4 sprites of ghosts and the last is 20 to the foods.

4 Conclusion

This work provided a lot of experience on the area of integration Software and Hardware, using the VGA also. How important is to control the memory of the embedded system and if we change a little thing a lot of things doesn't work. Was possible to see how hard is to debug a embedded system using the VGA Control, because to create a test bench not represent the real life at the final.

References

5 References

1. Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics
2. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2