ICOBS Game

Raffael Rocha Daltoé

April 11, 2024

Abstract

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 Today, it is obsolete, but some people continue to play it, striving to set new records.

1 Introduction

The development of this game was based on the ICOBS processor, utilizing the Instruction Set Architecture (ISA) of RISC-V. The goal was to implement a game using the C language and establish the connection with the processor. For communication, we utilized the AHB-Lite protocol, developed with specifications from the Advanced Microcontroller Bus Architecture (AMBA) set by ARM.

2 Software Layer

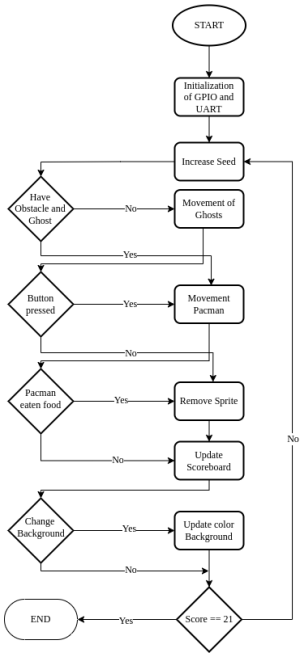

Figure 1: Flowchart of C code.

2.1 Creation of Sprites

2.2 Game

The game PACMAN features four ghosts, one main character, and a map with obstacles. The obstacles were created in the software using a *struct* with four fields, with the X and Y positions, both with

minimum and maximum positions, as it was necessary to define the limits of a rectangle-shaped object. The food and the position of the main character basically use the same structure but were divided into 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 needs to verify if the PACMAN is not surpassing the limits of the map or passing through an obstacle. For each obstacle, a rectangle was defined to represent it on the map, and a function was used to print the position of the PACMAN and manually set it. Each time the player presses a button (BTNR, BTND, BTNL, or BTNU), there is a comparison to see if there is a collision. If there is, the PACMAN cannot move to the desired position. Ghosts do not rely on Pacman's movement; they move freely, depending only 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() checks if the PACMAN has arrived at the Food. If it has, the program changes the value of the register of Foods and sends it to the Hardware level to unset the sprite of the Food and increase a counter of eaten foods. A vector structure with the position of each food was used, and inside the while loop of each iteration of the puppet with the map, the verifyEats() function checks if it has been eaten.

2.5 Switches

The user can control the color 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 more interesting.

2.6 Structure Movement

To move the PACMAN, it was necessary to create the struct Position, which is used each time to check if the PACMAN has arrived at an obstacle. For this structure, only the position on the X-axis and the position on the Y-axis are used. In the end, the ghosts were hard to control. The same idea as the obstacles was used to check if there is some obstacle and to verify it for 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 established by defining GPIO pins in the C code to connect with GPIOC and GPIOA, which are responsible for the buttons and the switches, respectively. This was accomplished using the structure already implemented in the C code, found in the file *arch_gpio.h*.

3 Hardware Layer

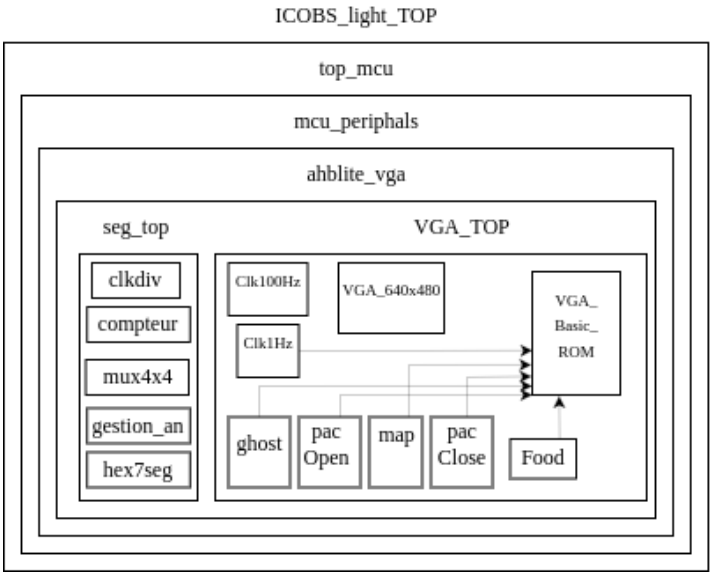

Figure 2: Schematic of ICOBS.

Register	Description
X0,Y0	PACMAN - MOVEMENT
X1,Y1	GHOST0
X2,Y2	GHOST1
X3,Y3	GHOST2
X4,Y4	GHOST3
Score1	Score on 7SEG
Score2	Score on 7SEG
Score3	Score on 7SEG
Score4	Score on 7SEG
Status	Win or Lose
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 where were declared are in the Table 1 above, was necessary to use at the total twelve registers.

3.2 Memory Efficiency

Embedded systems have limited memory, so it was necessary to economize space on BASYS3. It has almost 1,800 Kbits of RAM and non-volatile storage, including a 32Mbit non-volatile serial Flash device. The division of the map into sprites began after Vivado reported memory exceeded. The idea was to divide the map into four parts and use one-quarter of the map, replicating the VHDL code for the sprite of the same map because the map is symmetric.

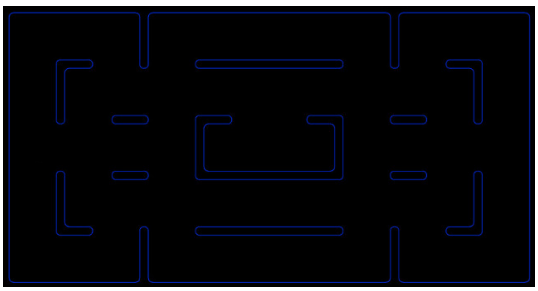

Figure 3: Original Map.

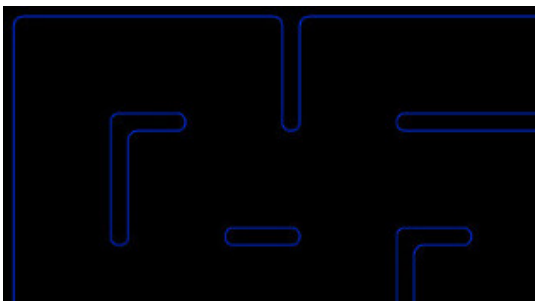

Figure 4: Cut map.

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. However, 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 better understand this concept below.

The same sprite was used for all the ghosts in the game. To change their color, the VHDL code utilized OR and AND logic operations. Only five Block Memory units were created: Food, Map, Pacman with mouth closed, Pacman with mouth opened, and the Ghost.

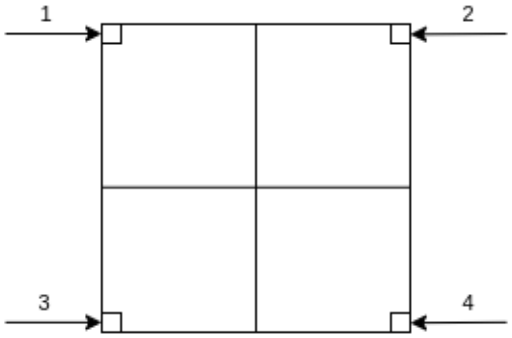

Figure 5: Read of ROM.

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 the last one to the Ghost.

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.

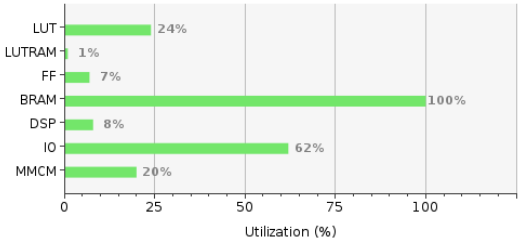

Figure 7: Cut map.

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.

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