Arcade Machine - Pong Game

User Guide

January 28, 2020

Contents

List of Tables

List of Figures

1 Introduction

Arcade Machine is a hardware architecture designed to implement the well know game - Pong - in verilog, which will be developed to be deployed in the Digilent Basys2 board. The players make use of four buttons in a PS/2 Keyboard in order to control the position of the rackets. The game is displayed in a monitor using VGA output. Also, the score of the game is displayed in a seven segment display. Additionaly, two buttons are used to start the game and reset it.

For this project, the picoVersat - minimal hardware controller - will be used as a state machine. Specialized hardware will be created to handle the PS/2, the VGA and the seven segment display interfaces.

2 Block Diagram

In this section, a block diagram of the arcade machine's hardware is showed in Fig. 1. As mentioned previously, picoVersat is used as a state machine and four other hardware modules will be developed (which are all connected to the data bus of the picoVersat). Object Display Controller is the hardware responsible for the VGA output of the game (paddles and ball), Score Display Controller is the hardware responsible for the output of the score of the players to the seven segment displays, Paddle Controller is the hardware responsible for the acquisition of the PS/2 key presses in the keyboard and Start Game Register is responsible to register the state of the switch that command the start of the game. A deeper insight in the function of each one of the modules is given in the Section 4.

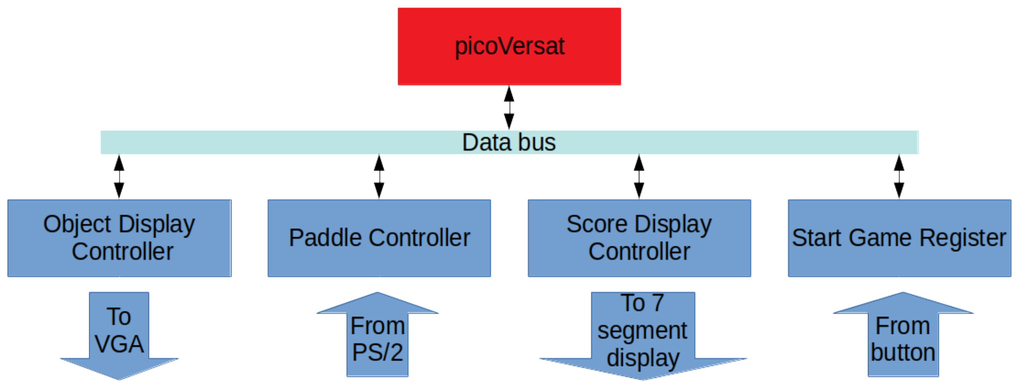


Figure 1: Block Diagram

3 Interface Signals

The interface signals of the Arcade machine project are described in Table 1. To implement this verilog project in a development board such as the basys2, you have to describe the connections between these signals and the correspondent IO ports.

Name	Direction	Description
clk	IN	Clock signal.
rst	IN	Reset signal.
btn	IN	When this signal is HIGH the game starts.
VGA Interface		
OutRed[2:0]	OUT	Intensity of red color.
OutGreen[2:0]	OUT	Intensity of green color.
OutBlue[2:0]	OUT	Intensity of blue color.
HSYNC	OUT	Horizontal sweep.
VSYNC	OUT	Vertical sweep.
7 Segment Display Interface		
an[3:0]	OUT	Selects the display to be light up.
seg[6:0]	OUT	Selects the segment of the display to be light up.
dp	OUT	Light up decimal point of the display selected by an
PS/2 Interface		
PS2D	IN	Serial data transmitted by the keyboard.
PS2C	IN	Clock driven by the keyboard.

Table 1: Interface signals.

3.1 VGA Interface

VGA works through sweeping screen methodology, which means each pixel is drawn one by one following a left to right and up to down pattern. The HS and VS signal transmit pulses each time the display has to change row or start a new screen correspondingly. In between pulses the red, green and blue signals are changed accordingly in order to indicate which color should be drawn.

In figure 2 there is a demonstration of the timing between the signals, where can be seen that the video signal is only valid between the HS and VS signal pulses that marks a start of a new row or screen. The displays resolution used in this project is 640 by 480 with a refresh rate of 60Hz for these specifications the signal timings are shown in figure 3.

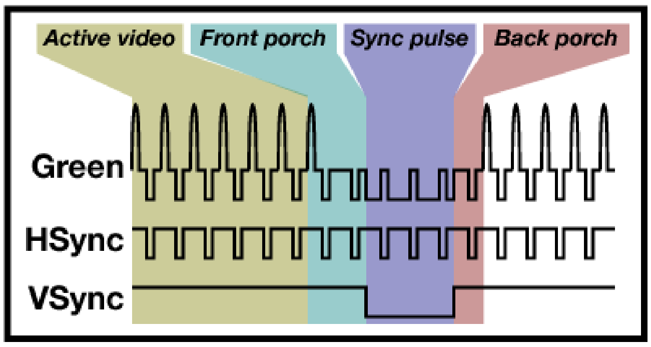
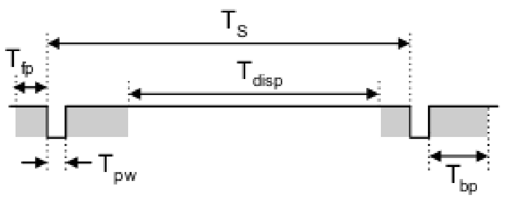


Figure 2: VGA timing diagram



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 3: VGA timing specifications for a 640 by 480 display

3.2 7 Segment Display Interface

In this project it is used four 7 segment displays, two for each player score, showing a maximum of 99 points for each player. However there are only a common bus to control each one of the segments of all the displays, the cathode bus, in this way to make each display to show a different number we have to make a time division multiplexing of this segment control bus signal accordingly to the enable signals of each 7 segment display. The anode bus has four signals one for each 7 segment display and it is used as a display enable since all the anodes of the segments of one display are connected to one of that four signals.

In figure 4 it is shown a timing diagram of the time division multiplexing done with the 7 segment displays.

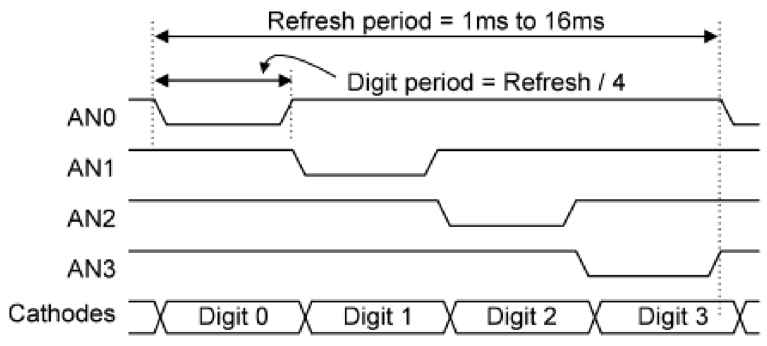
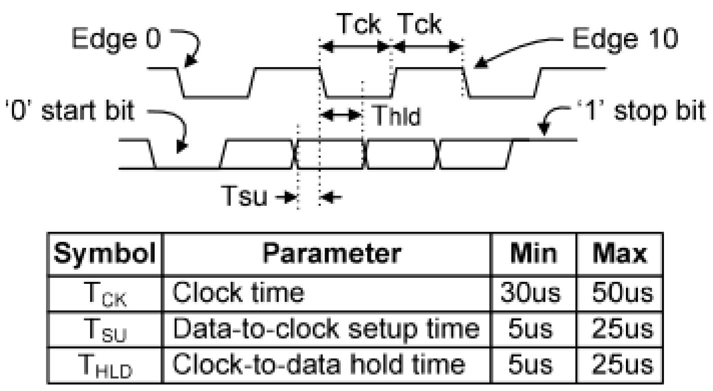


Figure 4: 7 segment display timing diagram

3.3 PS/2 Interface

The PS/2 interface is a typical synchronous serial bus interface, where a clock signal is driven by the transmitting device, in this case the keyboard, at the same time as the data signal, containing the binary code of the key pressed. This code is 8 bit long preceded by a start bit and followed by a stop bit. Some keys need more than a code to be identified.

In figure 5 it is shown a timing diagram of the specifications that the input signal form the keyboard has to respect in order to work with this system.



Symbol	Parameter	Min	Max
T_{CK}	Clock time	30us	50us
T_{SU}	Data-to-clock setup time	5us	25us
T_{HLD}	Clock-to-data hold time	5us	25us

Figure 5: Timing specifications of the PS/2 input

4 Peripherals

In this section, a deeper insight on the architecture of each one of the modules connected to the data bus will be given. Due to the fact that the picoVersat is a third party IP, its structure will not be analysed in this document and can be seen in the following document: picoVersat Documentation.
Four different subsections are presented, each one relating to each one of the modules.

4.1 Object Display Controller

The object display controller module, represented in figure 6, is responsible to deal with the visual representation of the game, it receives as input through the data bus of the picoVersat the coordinates of the upper leftmost point of the objects to be drawn on the screen. This coordinates are stored in the object position registers and are feed to the object detection sub-modules.

The object detection modules receive a bunch of parameters as input, such as the coordinates of the upper leftmost point of an object and the width (B1,B2,etc.) and height (C1,C2,etc.) of an object. Then according to the sweep coordinates received also as input, it determines if the pixel pointed by these sweep coordinates belongs to this rectangular shaped object and outputs a HIGH signal if this condition is true.

The frame detection module works similar to the object detection modules, but it only receives the thickness of the frame as a parameter (A5), this module instead of detecting the sweep coordinates inside of a rectangular shapes, it detects if the sweep coordinates are inside the frame that is around the borders of the screen.

The output of the detection sub-modules described earlier are then encoded and used to address a color memory that is connected directly to the Red, Green and Blue buses that outputs a color to the VGA screen accordingly to the existence of an object in the pixel being drawn at that time.

The VGA controller sub-module generates both the sweep signals used to drive the VGA screen (HS and VS) and the sweep coordinates used in the detection sub-modules, in order to synchronize the color being deployed to the VGA screen with the signals VS and HS that triggers the screen sweeping.

The display resolution can be configured, changing the following parameters of the sub-module VGA controller:

HEIGHT: height of the screen in number of pixels;

WIDTH: width of the screen in number of pixels;

PERIOD_H: period of the Horizontal synchronize signals, measured in number of pixel clocks;

PERIOD_V: period of the Vertical synchronize signals, measured in number of horizontal lines;

PULSE_H: pulse width of the horizontal synchronize signal, measured in number of pixel clocks;

PULSE_V: pulse width of the vertical synchronize signal, measured in number of horizontal lines;

BACK_H: back porch parameter of the horizontal synchronize signal, measured in number of pixel clocks;

BACK_V: back porch parameter of the vertical synchronize signal, measured in number of horizontal lines;

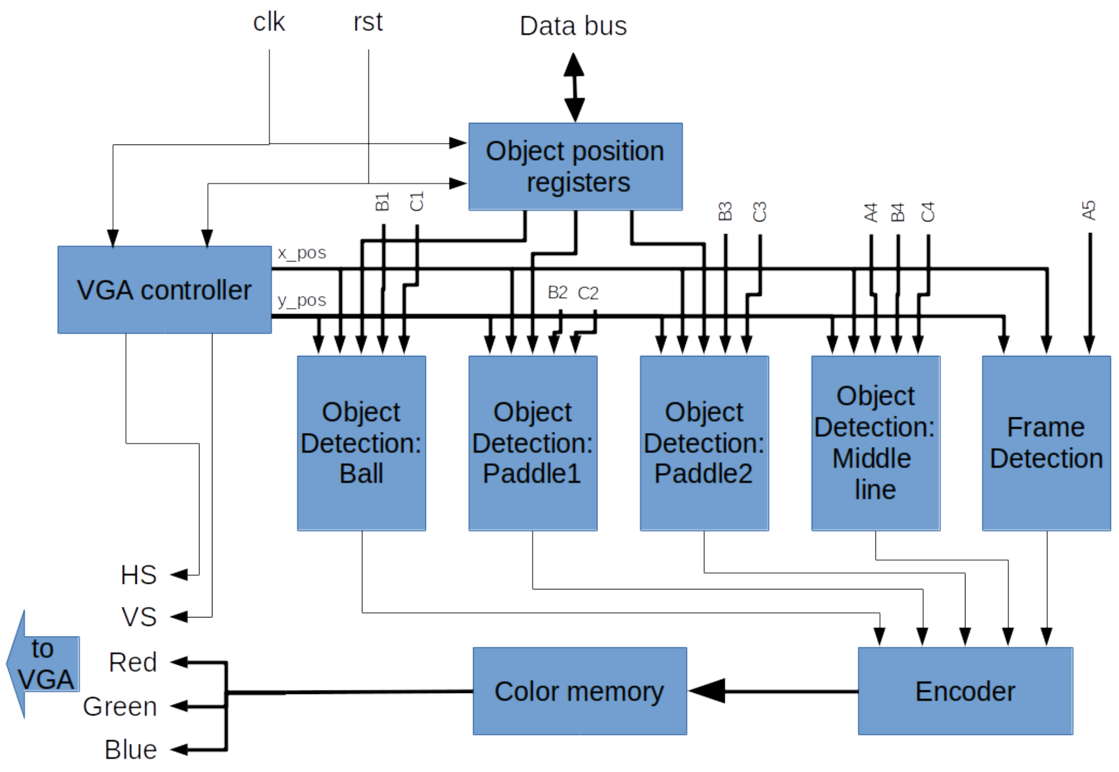


Figure 6: Object Display Controller Hardware Diagram

4.2 Paddle Controller

The Paddle Controller is a module responsible for getting the input from a PS/2 keyboard and register the paddles position.

The picoVersat can retrieve at any time the position of the paddles by reading the value of the paddle movement counter. The counter has an upper and lower limit, so that the paddles don't move more than the boundaries/limits of the game.

From the PS/2 input, it determines which was the key that was pressed. If the key is equal to the defined keys, it sends a signal to a bank of counters to increase/decrease the paddle position. To be able to do that, a clock divider is used to create a signal from each N clock of the FPGA clock (which is 50 MHz) to only move the paddles with a given period.

To know more about the PS/2 protocol and how it works, you can explore the following documentation: PS/2 Protocol.

A simplified block diagram of the Paddle Controller can be seen in Figure 7.

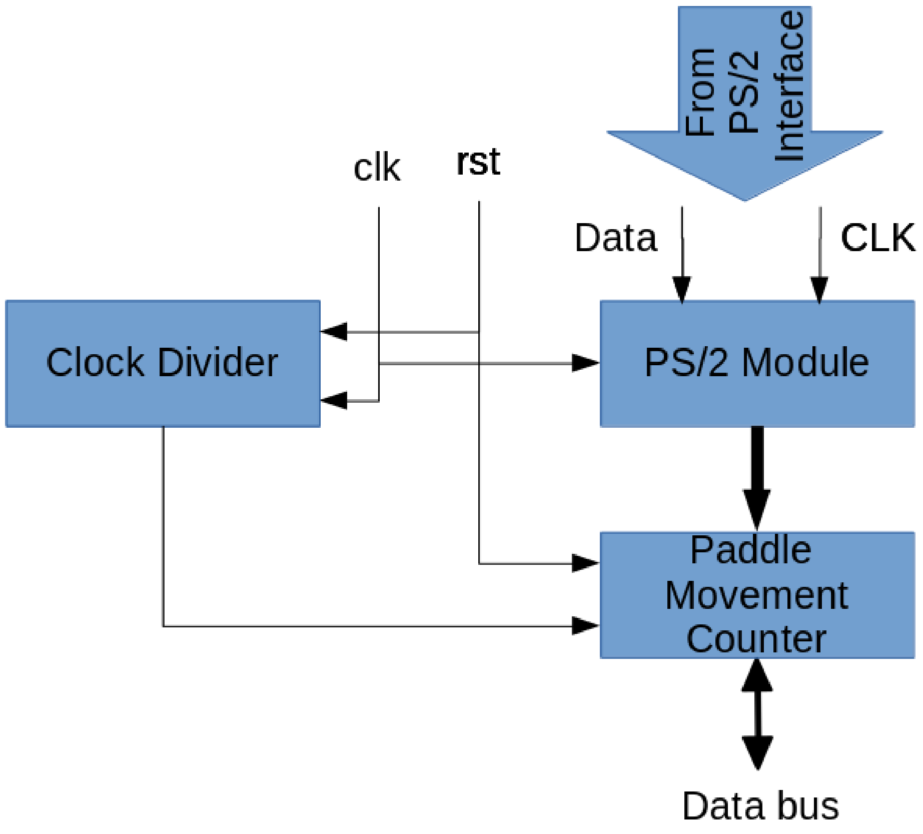


Figure 7: Paddle Controller Hardware Diagram

4.2.1 PS/2 Module

The PS/2 module is used to convert the signals from the PS/2 keyboard into a code. Its output is this code. For example, when the key ”W” is pressed, the PS/2 module outputs the code 0x1D (hexadecimal). When the key ”W” is released, two codes are output, the code 0xE0 and the code 0x1D (0xE0 is an extended code, meaning release of key).

This module can be used independently. For example, We could have 2 Paddle Movement Counters using the same PS/2 module.

4.2.2 Parameters of Paddle Controller

The Paddle Controller was designed to be as much parameterizable as possible. This parameters are:

PLAYER1_UP_KEY sets the code for the key which corresponds to upper movement by the paddle of the player 1 (default is 0x1D which corresponds to the ”W” key);

PLAYER1_DOWN_KEY sets the code for the key which corresponds to downward movement by the paddle of the player 1 (default is 0x1B which corresponds to the ”S” key);

PLAYER2_UP_KEY: sets the code for the key which corresponds to upper movement by the paddle of the player 2 (default is 0x44 which corresponds to the ”O” key);

PLAYER2_DOWN_KEY: sets the code for the key which corresponds to downward movement by the paddle of the player 2 (default is 0x4B which corresponds to the ”L” key);

SCREEN_HEIGHT: sets the number of pixels corresponding to the screen height (default is 480 pixels);

PADDLE_LENGTH: sets the number of pixels corresponding to the paddle length (default is 40 pixels);

START_POS: corresponds to the starting position of the paddles in pixels (default value is a relation between screen height and paddle length - (screen height - paddle length)/2);

FRAME_WIDTH: sets the number of pixels of the border of the game (default is 10 pixels);

MOTION_STEPS: sets the number of pixels that are incremented/decremented when paddles move (default value is 10 pixels);

BOTTOM_POS: sets the maximum position to which the paddle can move, in pixels (default is a relation between screen height, paddle length and frame width);

COUNT: sets the period in which the paddle positions should be changed, in nanosseconds(default is 1250000 nanosseconds).

4.3 Score Display Controller

The Score Display Controller is a module responsible for the display of the score in the seven segment displays.

It receives two signals from the picoVersat from the data bus, increment score from player one and increment score from player two. This means that the registers with the scores of each player are inside this module (they are divided in four sets of registers, two sets for the tenths of the number of the score of each player and two sets for the units of the score of each player). Also, there is a memory that is used to store the convertion between number to seven segment display representation.

To show the four digits in the seven segment display (score from 0 to 99 for each player), the seven segment display needs to be refreshed every 1ms to 16ms. To be able to do that, the module has a clock divider that creates a signal from every N clock of the clock of the FPGA (which is 50 MHz). This signal selects the output for the anode signal (which is stored in a bank of registers) and the input to the memory that translates the number to the digit in the display.

As input, it also has a reset signal, which resets the score of both players to zero.

A simplified block diagram of the Score Display Controller can be seen in Figure 8.

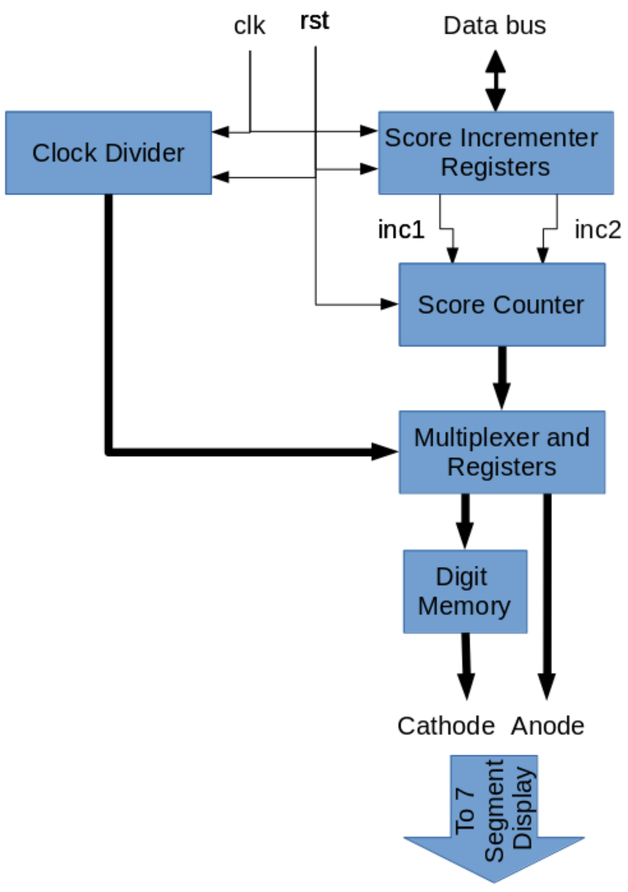


Figure 8: Score Display Controller Hardware Diagram

4.4 Start Game register

The start game register is connected directly to a push button. The register can be read by the picoVersat through the data bus, in order to determine if the game can start.

5 Memory Map

The memory map of the system, as seen by picoVersat programs, is given in Table 2.

Mnemonic	**Address**	**Read/Write**	**Read Latency**	**Description**
MEM_BASE	0	Read+Write	1	User programs and data
REGF_BASE	1024	Read+Write	0	Register file peripheral
OBJECT_BASE	1044	Write only	NA	Object Display Controller
PADDLE_BASE	1048	Read only	0	Paddle Controller
SCORE_BASE	1050	Write only	NA	Score Display Controller
START_BASE	1052	Read only	0	Start Game Register

Table 2: Memory map base addresses

6 Code Analysis

In this section, the code used to program the picoVersat, in order to run the Arcade Machine, is analyzed. A simplified flowchart of the code can be seen in the Figure 9.

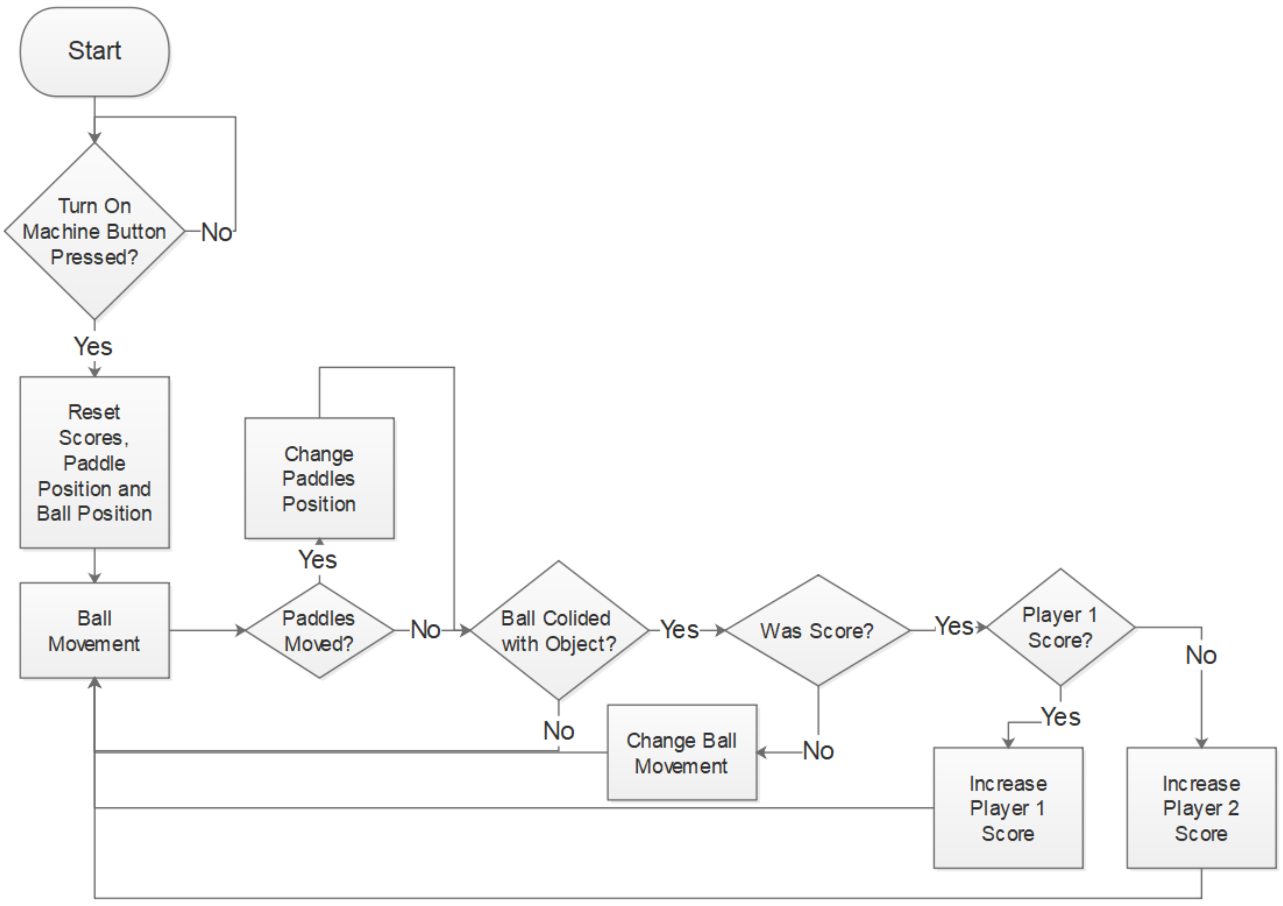


Figure 9: Paddle Controller Hardware Diagram

The arcade machine - pong game works in the following way. After the Basys 2 platform is turned on, picoVersat will start working and it will stay in a state in which it checks if the start push button is pressed. It will stay in that state until the push button is pressed, once pressed the game starts.

After starting the game, everything needs to be initialized: the scores, the paddles position and the ball position. After the initialization the ball starts moving.

If the keys which correspond to a movement of a paddle are pressed, then the paddle needs to move in the monitor (the image displayed needs to show the paddle in the new position) and we need to store this value (in order to detect collisions between the ball and the paddles). If the ball hits any object without being a goal (paddles or border of the window game), the movement of the ball needs to be changed (for example, if it was moving from right to left now it needs to move from left to right). Otherwise, if it hits an object and it was a goal, then it is needed to verify who scored the goal, increasing the correspondent player score.

7 Implementation Results

Althought the arcade machine with the pong game was developed in a Digilent Basys 2, all the modules and the program developed (assembly program running on picoVERSAT) should work in any FPGA with sufficient resources and adapted pinout configuration.

To simulate and implement the Arcade Machine the instructions given in the `README` file should be followed. The following decription shows how to play the pong game in the arcade machine deployed in the Basys 2.

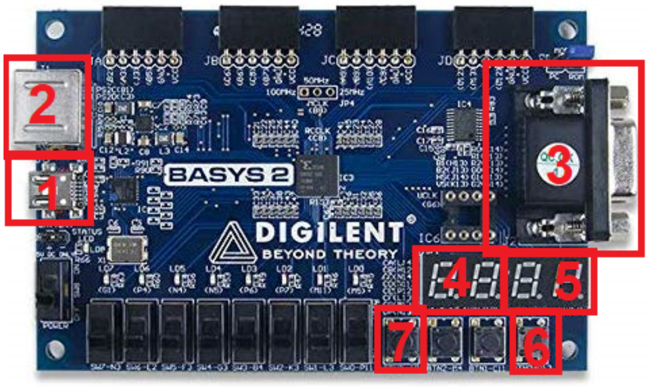


Figure 10: Arcade Machine Basys 2 inputs and outputs.

The inputs and outputs of the Basys 2 are the following (according to Fig. 10):

1 Basys 2 power on switch;

2 Basys 2 PS/2 connector, to connect the PS/2 keyboard;

3 Basys 2 VGA port, to connect a monitor;

4 Player 1 score counter;

5 Player 2 score counter;

6 Restart Arcade Machine button;

7 Start Game button.

When connecting the monitor to the VGA port, the game image should appear immediately. If the arcade machine is using the default values for the keyboard keys, key ”W” moves the paddle from player 1 up, key ”S” moves the paddle of player 1 down, key ”O” moves the paddle of player 2 up and ”key” ”L” moves the paddle of player 2 down. To start the game, press the start button (paddles and ball don’t move until you press this button). If you want to restart the game, press the restart button.

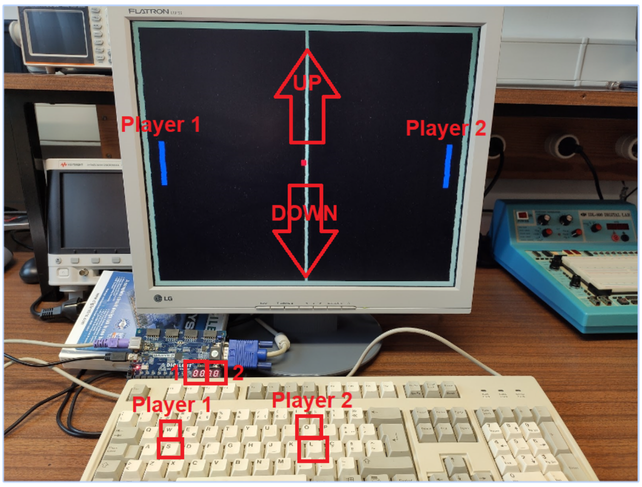


Figure 11: Arcade Machine setup with keyboard and monitor, showing the default keys, the score counters and the movement directions.

The gameplay is show in the following video: `https://drive.google.com/open?id=1HeaeJE78_7Xhp4eXOEJKMP0hSvF_SwxM`.

Once a player reaches 99 points, which is the maximum value that can be displayed by the seven segment display, the game ends and waits for the start of a new game (by pressing the start button on the Basys 2). This behaviour is shown by the following video: `https://drive.google.com/open?id=1RJiS_UIbxHn7yL1VT4FF28gns7YCDyTk`.

The following subsections 7.1 and 7.2 show the resource usage and timing results using Xilinx Spartan-3E FPGA (XC3S100E) as target device.

7.1 Resource Usage

To meet the timing constraints, the modules hierarchy should not be kept in order to reach further optimizations. Table 3 shows the resource usage with those optimizations, however to be able to estimate the resource usage by sub-module implementation, the Arcade Machine was also implemented keeping the hierarchy even thought it doesn't meet the timing constraints. Table 4 shows estimations of the resource usage by sub-module.

	Used	**Available**	**Utilization**
Slice Flip-flops	332	1920	17%
4-Input LUT	1211	1920	63%
4-Input LUT without route-through	1095	1920	57%
Occupied Slices	699	960	72%
IOBs	29	83	34%
BRAM	2	4	50%

Table 3: Target device resource usage by resource type

Table 3 shows this system can be successfully implemented in the target device, XC3S100E, using half the BRAMs and 63% of LUTs. Not counting with the LUTs used to route-throught, the resource usage of LUTs decrease to 57%, being just a little bit above half the capacity of the target device. In this way this hardware architecture shows the ability of being implemented in cost-effective low end devices.

The BRAM usage could be reduced using a smaller program memory. The designed architecture has a default program memory with a capacity for 1024 instructions, however the pong game only needs 152 instructions. The 1024 instruction capacity gives room for different games that may be developed over this architecture.

	Slice Flip-flops (%)	**4-Input LUT (%)**	**BRAM (%)**
PicoVersat	23	61	100
Object Display Controller	37	24	0
Paddle Controller	30	10	0
Score Display Controller	10	5	0

Table 4: Resource usage profiling by sub-module implementation

Table 4 shows the resource usage percentage by sub-module relative to the total resource usage of the Arcade Machine. The PicoVersat module uses most of the resources, to further reduce the resource usage, a state machine can be used to connect all the sub-modules instead of the PicoVersat, however some of the flexibility given by the software driven PicoVersat might be lost.

The Object Display Controller does not use any BRAM because this is an object driven display controller, only object proprieties must be stored not the actual values of every pixel. This approach saves memory usage, making this hardware architecture compatible with target devices with small memory capacities.

7.2 Timing Results

The Arcade Machine can be implemented using a **50 MHz** clock frequency, this is a clock period of **20 ns**. With this timing constraint and making use of further optimizations by not keeping hierarchy the worst case slack is shown in table 5.

	Worst Case Slack
Setup	0.070ns
Hold	0.844ns

Table 5: Worst Case Slack for a clock period of 20 ns

The Setup Worst Case Slack shown in table 5 is given by an internal path of the PicoVersat module. This path source is the RAM and ends in the controller of the PicoVersat.