# Arcade Machine - Pong Game

## User Guide

January 28, 2020

# Contents

# List of Tables

# List of Figures

# 1 Introduction

The proposed project is the implementation of an Arcade Machine, more specifically, the implementation of the well know game - Pong - in verilog, which will be designed 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 by using VGA output. Also, the score of the game is displayed in the seven segment display. Additionaly, two switches are used to turn on 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 score.

# 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.

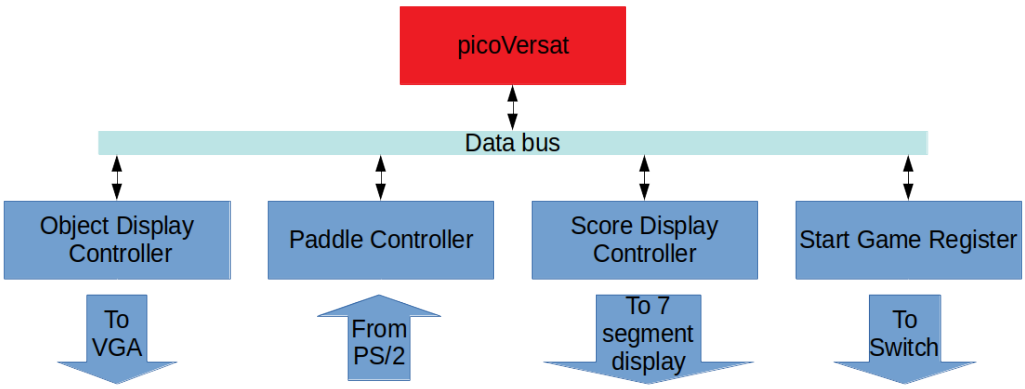



Figure 1: Block Diagram

# 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. |
| start_game | IN | When this signal is HIGH the game starts. |
| **VGA Interface** | | |
| red[2:0] | OUT | Intensity of red color. |
| green[2:0] | OUT | Intensity of green color. |
| blue[1:0] | OUT | Intensity of blue color. |
| HS | OUT | Horizontal sweep. |
| VS | OUT | Vertical sweep. |
| **7 Segment Display Interface** | | |
| anode[3:0] | OUT | Selects the display to be light up. |
| cathode[7:0] | OUT | Selects the segment of the display to be light up. |
| **PS/2 Interface** | | |
| data | IN | Serial data transmitted by the keyboard. |
| clk_ps2 | 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.

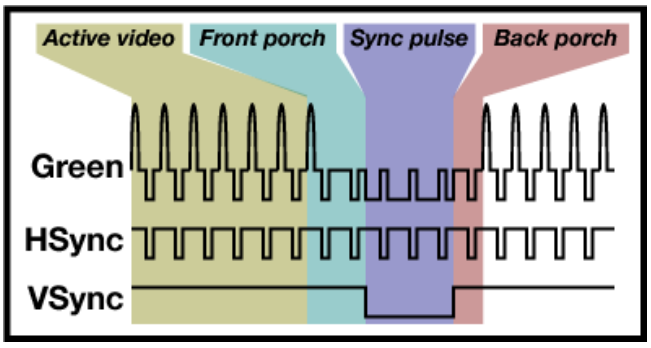
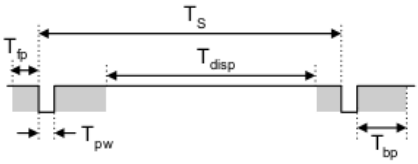



Figure 2: VGA timing diagram

## 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




| 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

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.

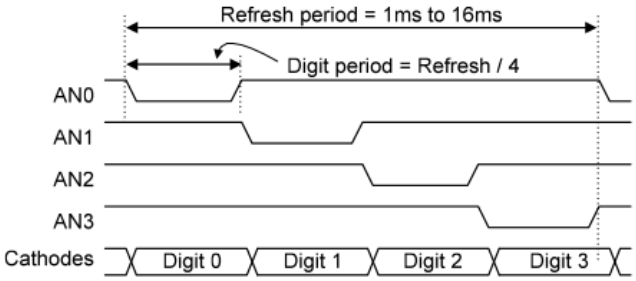



Figure 4: 7 segment display timing diagram

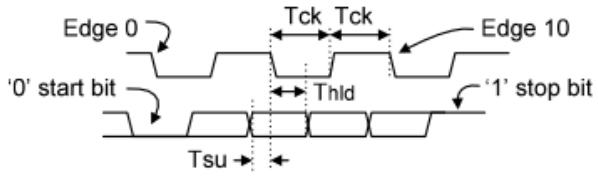
### 3.3 PS/2 Interface

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.

## 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




| 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

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 thought 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.

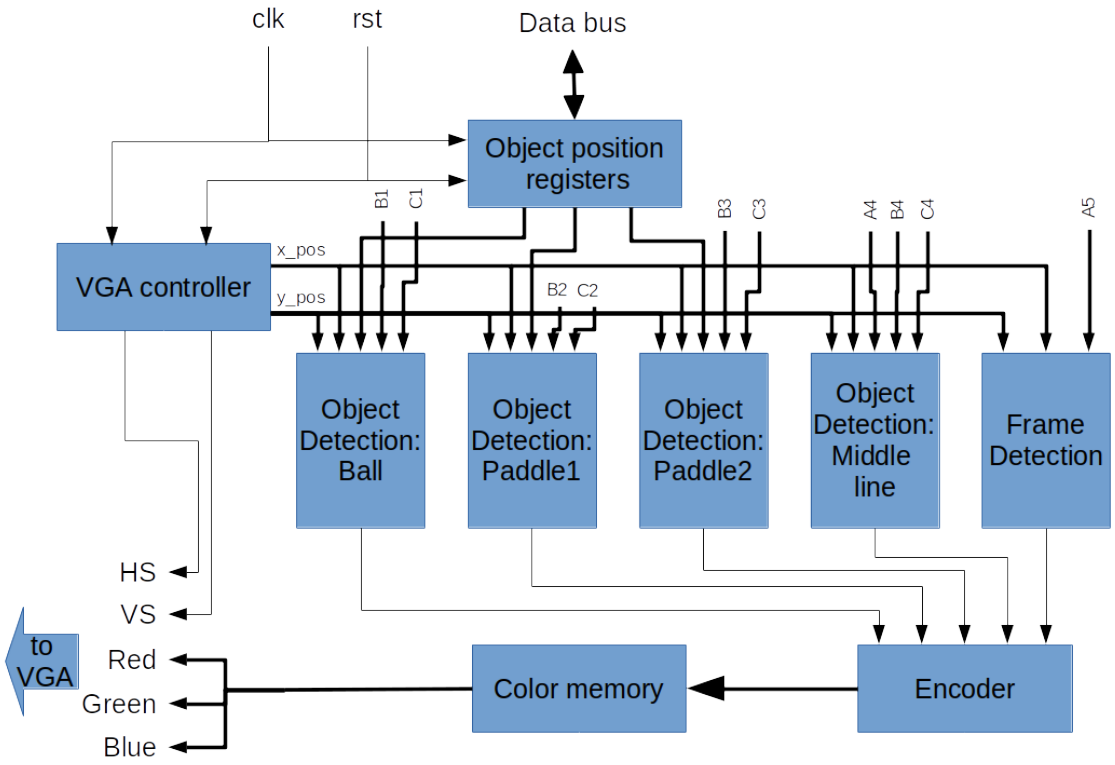



Figure 6: Object Display Controller Hardware Diagram

## 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.

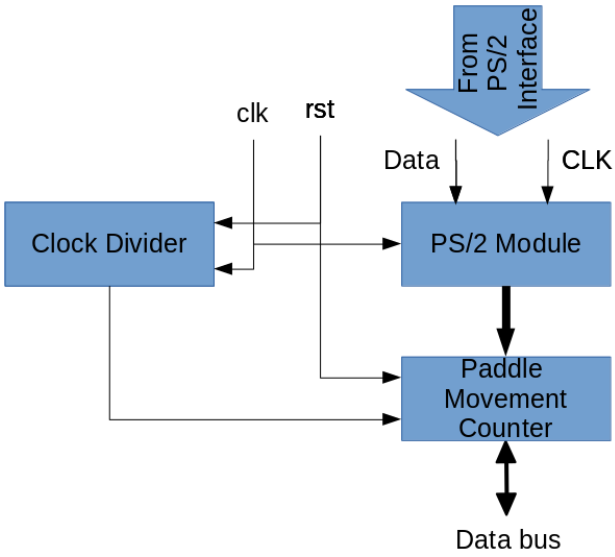



Figure 7: Paddle Controller Hardware Diagram

## 4.3 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.

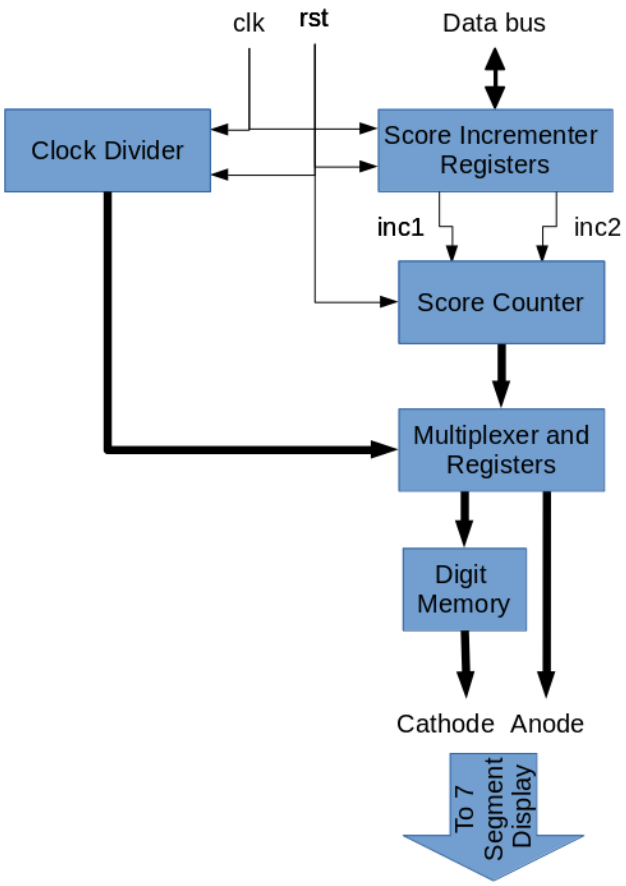



Figure 8: Score Display Controller Hardware Diagram

### 4.4 Start Game register

The start game register is to be connected directly to a switch. 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 | 1040 | Write only | NA | Object Display Controller |
| PADDLE_BASE | 1043 | Read only | 0 | Paddle Controller |
| SCORE_BASE | 1045 | Write only | NA | Score Display Controller |
| START_BASE | 1046 | Read only | 0 | Start Game Register |

Table 2: Memory map base addresses

# 6 Code Analysis

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

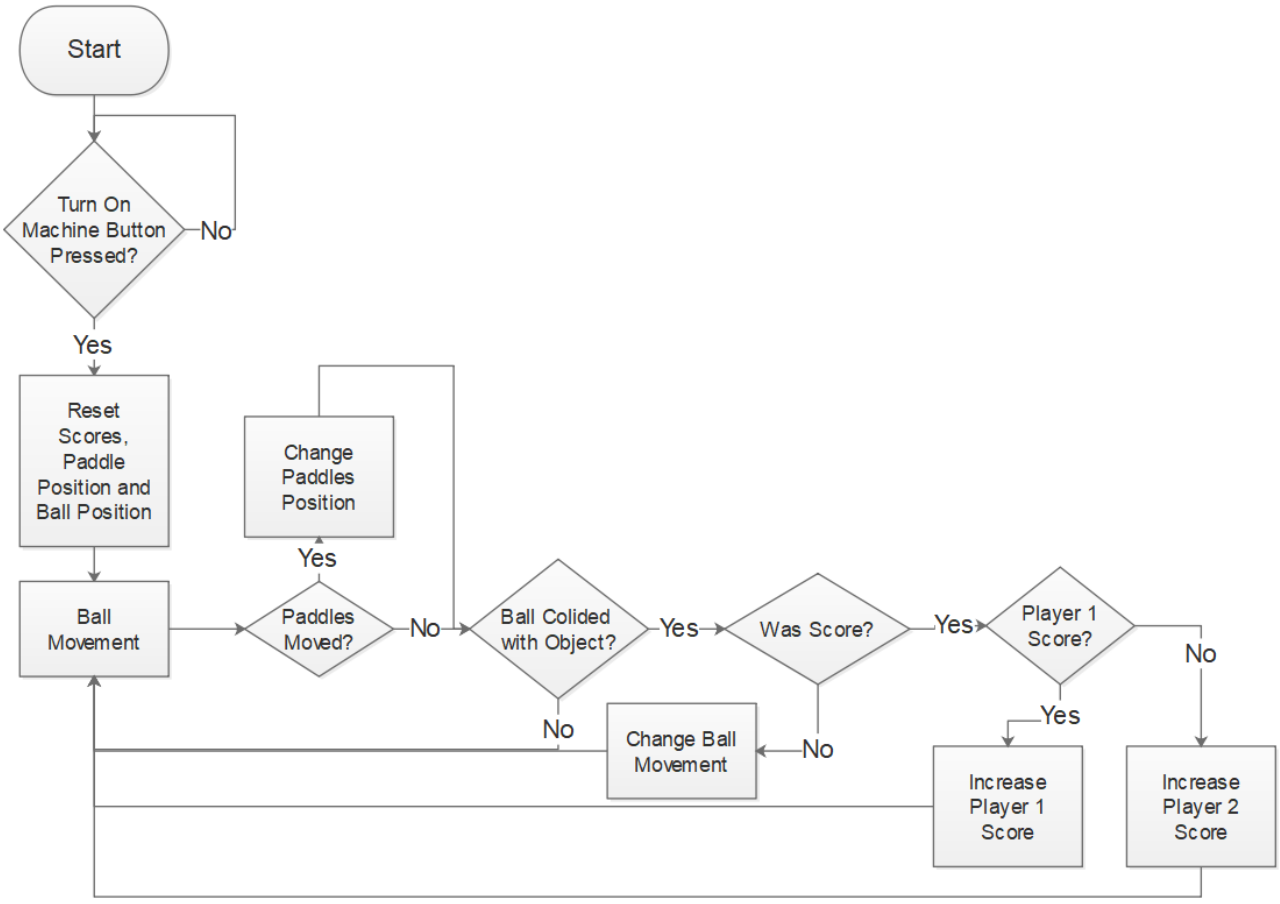



Figure 9: Paddle Controller Hardware Diagram

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 a switch is on or off (which correlates to the arcade machine being on or off, respectively). If it is off, nothing is done. If the switch is on, then the arcade machine is on and the game can start.

After turning on the arcade machine, everything needs to be initialized: the scores, the paddles position and the ball position (initialize all the modules). After some time, the game can start. The ball will start 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

# 8 Conclusions