Simple Arcade Game from Hardware Side using MicroBlaze

Milan Tucic, Ivan Kastelan
University of Novi Sad, Faculty of Technical Sciences
Novi Sad, Serbia
milan.tucic@rt-rk.com

Dragan Topalovic, Milos Nikolic
RT-RK Institute for Computer Based Systems
Novi Sad, Serbia

Abstract— **This paper proposes reconstruction of the simple arcade game Pong using MicroBlaze soft core and explains hardware side of this process. Reconstruction is split into two time phases. In the first phase hardware is developed with functionalities which, as they are, are sufficient for game implementation. This means hardware part of system is made of simple graphic controller and paddle movement controller. The second phase covers system analysis and hardware made as a result of this analysis. All upgrades are done considering MicroBlaze's flexibility. This kind of approach and organization leads to better balance between hardware and software implementation, and demonstrates advantages of designing a system in which it is possible to monitor, analyze and influence the further evolution of the system.**

Keywords-game; hardware; microblaze; pong; vhdl

I. INTRODUCTION

FPGA configured by some Hardware Description Language (HDL) provides simple and fast development of complex digital circuits and systems. Developers have accepted this approach, which has led us to a lot of examples and solutions that we have today. Using them, development is easier, more flexible and it is giving additional time for research, rather than creating system from a scratch.

Good example of available solutions today is the Xilinx Embedded Development Kit (EDK), including Intellectual Property Catalog (IP Cores) and the MicroBlaze embedded processor soft core (MicroBlaze) with a reduced instruction set computer (RISC) architecture optimized for implementation in Xilinx Field Programmable Gate Arrays (FPGAs) [1].

Ke, Eric and Winston [2] were using this solution, while working on a student project of reconstructing Pac-Man game. They split the project into software and hardware programming, and with little effort on software side they got reduced complexity on hardware side. Game logic includes collision detection, scoring and enemy artificial intelligence (AI) and it is implemented using the MicroBlaze. Hardware configuration is described in Very Large Scale Integration HDL (VHDL), and it implements graphics engine with sprites and paddle control over the MicroBlaze UART IP Core.

As one of the earliest arcade video games, Pong game is used as referent simple arcade game in this paper. It has simple game logic and two-dimensional graphics. While other arcade video games, such as Computer Space, came before it, Pong was one of the first video games to reach mainstream popularity. It is a multiplayer tennis like game, where the goal is to defeat an opponent by earning a higher score. The game was originally manufactured by Atari Incorporated (Atari), who released it in 1972.

Armandas [3] has done full reconstruction of the Pong game in VHDL. He used Nintendo Controllers as an input device for paddle control. Ball and paddle objects are represented by graphics stored in memory, separated of text symbols which are used for text generation. Logic for game, which includes ball movement and paddle movement has also been implemented in VHDL.

This paper proposes an approach to accelerate drawing of graphics with hardware-based drawing using VHDL, MicroBlaze and IP Cores, suitable for reconstruction of the simple arcade game Pong [4] as one of the earliest arcade video games. VGA Controller handles communication with VGA screen and it is combined with modules for text generation and graphics memory. Game logic (scoring, ball and paddle movement) is handled by MicroBlaze. Paddle control has been done initially over the UART IP Core, but has changed (after system analysis) to less robust Joy Peripheral. This kind of approach brought additional time for modifications to graphics memory and this system module was upgraded with hardware sprites.

The rest of the paper is organized as follows: section 2 gives the overview of the proposed system. Section 3 explains graphics modules and first approach for paddle movement. Section 4 explains modifications done after system analysis. Section 5 gives results of the proposed approach implemented on Pong game. Finally, section 6 gives some concluding remarks and plans for future research.

II. SYSTEM OVERVIEW

This paper recommends Embedded Engineering Learning Platform (E2LP) [5] as a platform for research projects based on FPGA. The E2LP provides an advanced hardware platform that consists of a low cost Spartan-6 Platform FPGA surrounded by a comprehensive collection of peripheral components that can be used to create a complex embedded system. Additionally, software IDE is developed to support usage of the board. In this paper Spartan-6 is used for VHDL configuration, movement control is done over joy buttons and

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, under grant no. TR32030, year 2011-2015

VGA controller is using VGA connector for connection to a VGA monitor.

The central figure of the proposed system implemented on the E2LP is the MicroBlaze - soft 32-bit RISC processor. MicroBlaze [1] is implemented with Harvard memory architecture. Separate address spaces for instruction and data accesses are used. Both instruction and data interfaces of MicroBlaze are 32 bits. There is no separate access to I/O and memory (it uses memory mapped I/O). This system for Block RAM (BRAM) memory accesses uses Local Memory Bus (LMB) and Processor Local Bus (PLB).

Software programming is not described in this paper, but it is done in “C” programing language on MicroBlaze. Every peripheral has driver with functions that are important to them. Some functions were just abstraction of peripheral’s native functionalities, but there were also functions with extra algorithms like object drawing. For accesses to peripherals AXI4-Lite IP Interface (AXI Lite) is used (Fig. 1).

III. Phase I

This time phase of system evolution presents hardware with functionalities which, by themselves, are sufficient for game implementation. That means in this phase hardware is made of a graphics controller and a controller for paddle movement. The rest of the system, implemented in this phase, including scoring, ball and paddles drawing is implemented on MicroBlaze. Everything done here, will serve as a referent point for the phase II.

A. VGA Controller

VGA standard represents an easy way of forming images on the screen. An image is formed by placing successive pixels of the corresponding color, which are always placed from the relative position (0, 0) until the end of the starting line (in the case of 640x480 screen, it is pixel at position (0, 639)), when switching to the next line and so on until the end (in the case of 640x480 screen, it is pixel at position (479, 639)) (Fig. 2).

VGA interface defines 4 digital and 3 analog signals. Digital signals are used for synchronization and positioning. The end of setting a single screen line is signalized by a horizontal synchronization signal (hsync). Vertical synchronization signal (vsync) is signalizing the completion of setting the entire image on the screen. Signals for positioning are indicating which row and column are going to be set on the screen. Analog signals are used to determine the color which is

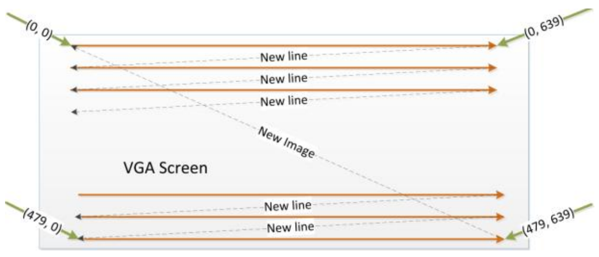

Figure 2. Image Process on VGA Screen

going to be displayed at a given position. The color is formed by combining the illumination intensity of three RGB format components. The intensity of each color is represented by 8 bits; hence each pixel on the screen is represented by 24 bits.

Screen content control relies on modules for text printing, graphics memory and set of registers. Using “pure” VGA controller directly with signals described above is complicated and it is not recommended. Because of that, these three modules were implemented to abstract VGA controller, and give simplified control for screen content.

1) Text Printing: Printing text on the screen is based on addressing a specific symbol from the BRAM memory. The BRAM memory contains predefined values as symbol representation for each symbol from a set of given symbols. Symbols are defined as 8x8 binary values. Giving symbol address, current row and column, module generates screen output value (Fig. 3). This module organizes screen in partitions. One partition is representing 8x8 pixels from the screen. Memory word stores address of a symbol and it maps one screen partition. This means it is only possible to set symbol to fit one of these partitions, not to a random position.

2) Graphics Memory: Graphics memory is used as an extra layer for printing desired values on the screen. The words in this memory are 32 bits wide and represent 32 successive pixels on the screen. Successive locations in memory are mapped to successive positions on the screen. At the end of each line of the screen, the memory locations wrap to the beginning of the next line. Graphics memory is accessed every time when value for the current position on screen is requested (also as text memory). Each of bits indicates which of the two predefined colors will be chosen for current point position. This kind of organization leads to easy screen content control, and everything is executed from MicroBlaze. Also, VGA controller generates interrupts which are processed

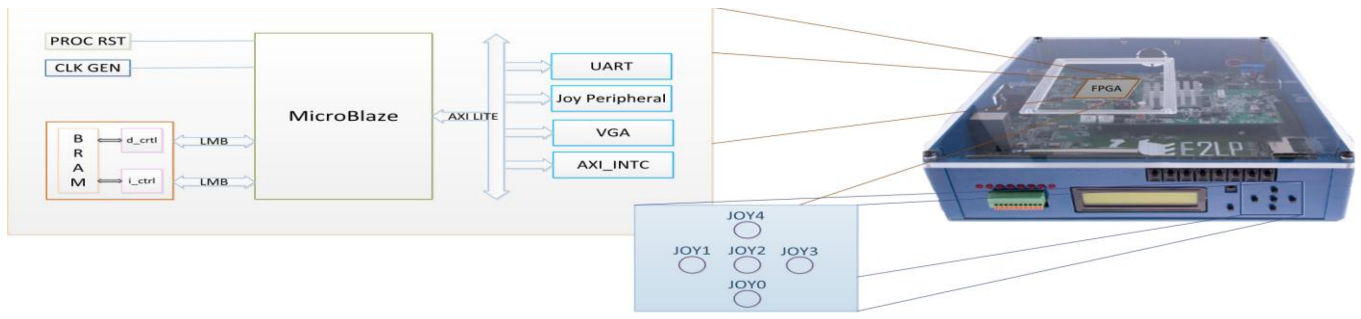

Figure 1. E2LP with MicroBlaze System and Joy Buttons

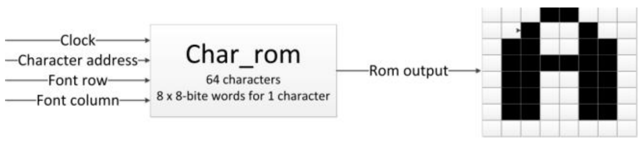

Figure 3. Text Module

by LogiCORE™ IP AXI Interrupt Controller (AXI INTC). These interrupts could be used as the clock for screen content refreshment (Fig. 4).

3) Registers: Some of the values for graphics modules are stored here. They define graphical mode, border line and colors (foreground, background and border). Registers for graphical mode determine whether the memory for both modules forms a picture, or the memory for just one module. Border line refers to the whole screen border and it can be set or not. This module is also accessible from MicroBlaze, and register content can be controlled from there.

B. Paddle Movement Controller (PMC)

The system uses the LogiCORE™ IP AXI Universal Asynchronous Receiver Transmitter Lite (UART Core) [6] to redirect commands from keyboard to the MicroBlaze. UART Core provides interface between UART signals and the Advanced Microcontroller Bus Architecture (AMBA) and also provides a controller interface for asynchronous serial data transfer. UART Core is enabling usage of keyboard buttons, but they must be filtered for reaction on specific buttons. In this case “S” and “K” keyboard buttons are used to move paddles up, and “X” and “M” buttons are used to move paddles down. This process is handled from MicroBlaze.

IV. PHASE II

The first phase concentrated on implementation of the basic Pong game. In the second phase, which is the main contribution of this paper, Pong was reconstructed using MicroBlaze and IP Cores as representative solutions for

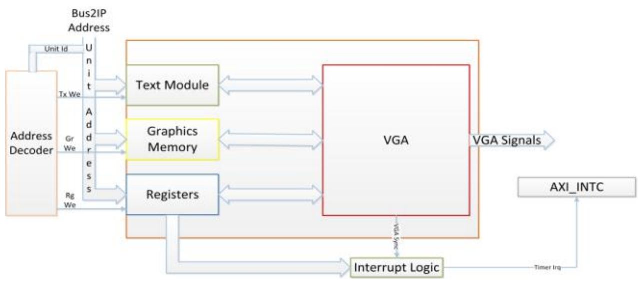

Figure 4. VGA Peripheral – Phase I

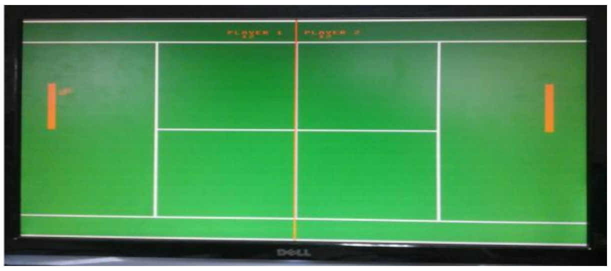

Figure 5. Project Results after Phase I

systems based on FPGA with the goal of accelerating the graphic processing and a broader usage of E2LP platform to make playing more user-friendly. Functionalities are implemented from both software and hardware sides. Phase II upgrades the system (Fig. 6), based on comparison to the similar systems and directly comparing software with hardware from phase I (Fig. 4).

A. Joy Peripheral

Joy Peripheral (Fig. 7) holds PMC functionality and logic that is easy to handle. It controls signals from E2LP Joy buttons (Fig. 1). For purpose of this system implementation two instances of Joy Peripheral are used, controlling signals from buttons JOY1 and JOY4 (for up paddles direction) and JOY0 and JOY3 (for down paddles direction). Paddle control is based on Joy Peripheral interrupts handled from MicroBlaze. Interrupts are triggered at the moment when peripheral state is changed, and this depends on signals coming from JOY buttons.

Three states are defined, and here is an example for one instance:

1) UP: Peripheral state UP is set when button JOY1 is pressed, but JOY0 is not pressed.

2) DOWN: Peripheral state DOWN is set when button JOY0 is pressed, but JOY1 is not pressed.

3) IDLE: Peripheral state IDLE is set in cases which are not defined in 1 and 2.

After interrupt happens, defined state register holds current peripheral’s state. Accessing to this register from MicroBlaze, paddle position on the screen can be updated.

B. Hardware Sprites

In the phase I object drawing is done by setting values that represent objects to graphics memory; every time when object changes its position graphics memory must be updated. This kind of approach has complex processing which must be provided by the MicroBlaze, particularly because memory architecture (32-bit words) for graphics memory is not ideal for setting object of any size to random position. Also, simply setting two objects to the same memory word will result in overwriting the first of these two. Because of this behavior, software side must handle this relatively complex situation.

In early video gaming, hardware sprites were a method of compositing separate bitmaps so that they appear to be part of a single image on a screen. Simple arcade games often work with predefined graphics objects, which fully meet the definition of hardware sprites. It enables much easier screen content control and that is the reason why they are implemented here. In contrast to the full control of an object drawing, which must be provided from the MicroBlaze, this kind of approach dramatically reduces effort on software side. All that is necessary is to fill the memory with sprite bite representation and then control them by setting starting pixel’s position. This process is similar to text printing described in section 3, only it has better resolution defined with 4x4 pixels partitions and it uses blocks to represent one sprite. Block is defined by number of cells in row and column. For example, block for ball sprite contains 4x4 partitions (16x16 pixels). Sprite memory is

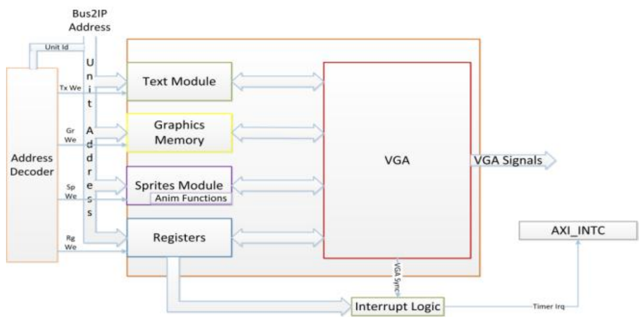

Figure 6. VGA Peripheral – Phase II

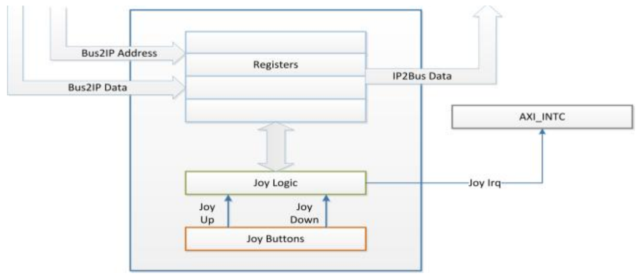

Figure 7. Joy Peripheral

representing sprites on the screen. While writing to this memory, it is necessary to provide starting address from BRAM memory for current sprite and number of rows and columns (cells) occupied with this sprite.

Sprites implemented here are not static. Sprite can be rotated and shifted to create simple animations effect. This procedure uses sprite bite representation and modifies output depending of function demanded.

Supported functions are:

1) *Rotation:* Using original sprite, simulates rotation for 90, 180 or 270 degrees.

2) *Shifting:* Shifts original sprite for the given number of bits.

V. Results

PMC over UART Core, described in section 2 (also proposed by Ke, Eric and Winston [2]) is large overhead for this kind of systems. Paddle movement is controlled with 4 buttons and it is unnecessary to include whole UART Core for that. After comparing PMC from section 2 with Armandas's [3] Nintendo Controllers, it was clear that it is better to have simple peripheral for PMC. Also, comparing to Armandas, including peripheral to MicroBlaze is much easier than creating new peripheral for system proposed by Armandas. Very important fact is that peripheral in that kind of systems must be considered from the beginning. Adding features like this in the middle of project progress can be very complex (or impossible), which is not case with system that proposes this paper.

Graphics memory seems to be sufficient from the beginning, but after increased effort needed on software side it was clear that extra abstraction layer for graphics functionalities is necessary. Hardware Sprites were good solution, and this is confirmed by results of measuring execution time needed for drawing objects in phase I and phase II (Table I). Drawing object, in phase I, depends of object position, so execution time is not always the same. For this case, time in table I represents average value from 1000 samples of the same object drawing. In phase II there are no variations like in phase I; every sample within 1000 lasts equally. Phase II extremely reduced execution time compared to phase I in object drawing.

Extra features are also included to hardware sprites. It is possible to create simple animations, gaining user experience with rotation and shifting functionalities. Approach proposed by Armandas uses multiple sprites to represent different states of an animation (also possible here). Comparing to this approach, there is no need to store manually every image necessary for an animation, but they can be used only for simple animations.

TABLE I. Objects drawing comparison

	Execution time	
	Phase I	***Phase II***
Ball	6804 ns	7 ns
Paddle	7785 ns	7 ns

VI. Conclusions

The advantage of using MicroBlaze as a representative solution is the design flexibility during the progress of the system development. System's modules which require a lot of effort to be done by processor (software part of a system) can be reduced with hardware implementation of certain functionalities. Also, hardware modules of a system can be modified and their evolution brings better experience and easier development to the rest of the system. Choosing platform for research is also important. E2LP has wide range of peripherals and it is possible to use them to improve a system.

In the future research, system will be further improved to become a framework to support simple game generally, not to be bound to Pong game. Rotation will be considered to support any degree set. Also, more functionality for hardware sprites will be considered.

References

[1] Xilinx, Inc., MicroBlaze Processor Reference, Guide UG984 (v2014.1), April 2, 2014

[2] Ke Xu, Eric Li, Winston Chao, "Pac-man final project report", University of Columbia, July 4, 2014

[3] Armandas Jarusauskas, "FPGA based VGA driver and arcade game", University of Sussex, 2009/2010

[4] Pong game, www.ponggame.org (accessed: August 10, 2014)

[5] I. Kastelan, N. Teslic, M. Temerinac: "E2LP: An Embedded Engineering Learning Platform", *IT System Conference 2013,* pp. 1-4

[6] Xilinx, Inc., LogiCORE IP AXI UART Lite v2.0 Product Guide PG142, April 2, 2014

[7] Pong P. Chu, "FPGA prototyping by VHDL examples", John Wiley & Sons, Inc., 2008