FPGA Snake Via VGA

Mariam Bekhit
College of Engineering
California State University, Long Beach
Long Beach, United States of America
mariam.bekhit@student.csulb.edu

Pi Oliver
College of Engineering
California State University, Long Beach
Long Beach, United States of America
pi.oliver@student.csulb.edu

Justin Salazar
College of Engineering
California State University, Long Beach
Long Beach, United States of America
justin.salazar@student.csulb.edu

***Abstract*— The purpose of this project is to implement an FPGA snake game that utilizes the Digilent Nexys A7-100T's on-board VGA port, directional pushbuttons, and the seven-segment display. The snake shall be initialized to a random spot on the screen and move in a certain direction until the user pushes a button to change the direction the snake is drawn in. There shall be sprites (objects/apples) that the snake eats as it moves across the screen, and the snake will grow longer as it eats the apples. The score shall be increased by one for each apple eaten. The game shall run indefinitely until either the snake eats itself (intersects with itself) or the snake hits a wall (border of the screen). The score will be tracked on the seven-segment display.**

I. Introduction

This snake game project is meant to combine modules from all areas of learning thus far. It combines both sequential and combinational circuits into one design. The snake game is displayed at a 640x480 resolution using a vertical and horizontal sync generator and a basic clock divider to fit the VGA standard. The Snake game also outputs the score to the seven-segment array module which multiplexes the individual displays to show multiple digits in decimal form. The game is controlled using the on-board directional pushbuttons with a debounce function to prevent unintentional misinputs.

II. Background & Preliminaries

The intention of this project was to demonstrate the use of the VGA port on the Nexys A7-100T and the FPGA itself as a display adapter/graphics processor to play a simple game of Snake, and additionally keep score/time on the onboard 7-segment array for additional functionality. For the game itself to operate in a conventional manner a user may be used to, the implementation requires a variety of modules working cohesively.

III. Implementation

A. SNAKE GRAPHICS

This module utilizes the x and y coordinates of the current pixel being drawn by the VGA Generator and determines the RGB value. The boundaries are drawn in fixed locations however the apple and snakehead may vary depending on the pushbutton input. The apple has its location placed initially at When the coordinates of two objects are equivalent, collision is detected. This module also stores and draws the 16x16 bitmap image of the snake's head, which is visible in Figure 1 below. We also took some inspiration from several alternative implementations of Snake and Pong on an FPGA which we found on the Internet [1]-[3], [9].

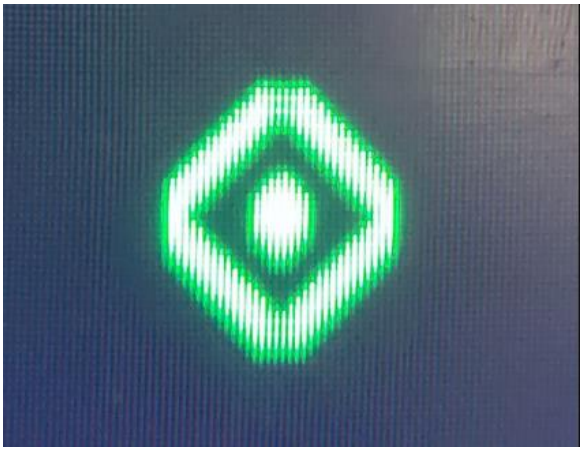


Figure 1: Snake Head Visible on Monitor Display

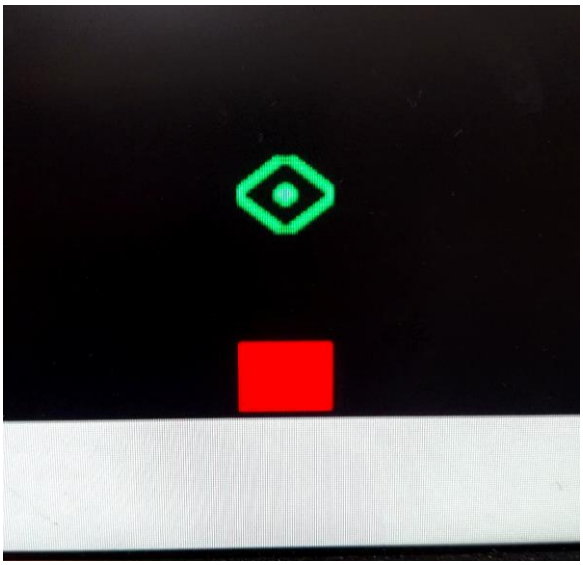
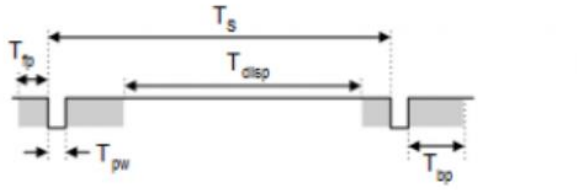


Figure 2: Snake Head Approaching Apple

B. VGA SYNCHRONIZATION GENERATOR

This module is essential in synchronizing the vertical and horizontal display signals according to the Video Electronics Standard Association (VESA) standards for VGA and was by far the most difficult and time-consuming part of the implementation to code and debug, taking over a month to develop before the first successful tests. We closely followed the example given by Pong Chu in Chapter 13 of his book *FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Edition*, making careful modifications in the implementation over the development cycle to suit our signal timing

requirements and desired target resolution [4]. The system clock frequency of the Nexys A7-100T FPGA is 100 MHz, so we implemented a clock divider to create a VGA clock at 25 MHz for the pixel clock as per the requirements as specified in the manufacturer's datasheet and VESA standards for the target resolution: 640x480 pixels resolution with a 60 Hz Vertical Refresh Rate and a 25 MHz Pixel Clock Frequency [5]. We chose to pay attention to and implement the widely adopted VESA standard recommendations for VGA to maintain compatibility of our FPGA game with a wide variety of commonly available VGA monitors in the consumer market, such as older models that end-users might have lying around the house. We also experimented with the VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT) Timing Specifications for 640x480 at 60 Hz and found that the timing specifications for the Nexys A7-100T were *mostly* compliant with the VESA DMT timing specifications; however, Digilent specifies that the Sync Pulse need only be 521 horizontal lines, not 525 as specified by the VESA standards as shown in both Figures 1 and 2 below [6], [7]. We found that either option was able to meet the signal timing requirements of all the monitors we tested the game on.



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 2: Digilent Signal Timings for a 640-Pixel by 480-Row Display Using a 25 MHz Pixel Clock and 60 Hz Vertical Refresh on the Nexys A7-100T [6]

VESA MONITOR TIMING STANDARD

Adopted: n/a **** For Reference Only - Not a VESA Standard ****
Resolution: 640 x 480 at 60 Hz (non-interlaced)
EDID ID: DMT ID: 04h; Std. 2 Byte Code: (31, 40)h; CVT 3 Byte Code: n/a
BIOS Modes: 11h, 12h, 101h, 110h, 111h, & 112h (1, 4, 8, 15, 16, & 24 bpp)
Method: ***** NOT CVT COMPLIANT *****

Detailed Timing Parameters

Timing Name	= **640 x 480 @ 60Hz;**					
Hor Pixels	= **640;**	// Pixels				
Ver Pixels	= **480;**	// Lines				
Hor Frequency	= 31.469;	// kHz	=	31.8 usec	/	line
Ver Frequency	= 59.940;	// Hz	=	16.7 msec	/	frame
Pixel Clock	= **25.175;**	// MHz	=	39.7 nsec		± 0.5%
Character Width	= **8;**	// Pixels	=	317.8 nsec		
Scan Type	= **NONINTERLACED;**			// H Phase	=	2.0 %
Hor Sync Polarity	= **NEGATIVE;**	// HBlank	=	18.0% of HTotal		
Ver Sync Polarity	= **NEGATIVE;**	// VBlank	=	5.5% of VTotal		
Hor Total Time	= 31.778;	// (usec)	=	100 chars	=	800 Pixels
Hor Addr Time	= 25.422;	// (usec)	=	80 chars	=	640 Pixels
Hor Blank Start	= 25.740;	// (usec)	=	81 chars	=	648 Pixels
Hor Blank Time	= 5.720;	// (usec)	=	18 chars	=	144 Pixels
Hor Sync Start	= 26.058;	// (usec)	=	82 chars	=	656 Pixels
// H Right Border	= 0.318;	// (usec)	=	**1** chars	=	8 Pixels
// H Front Porch	= 0.318;	// (usec)	=	**1** chars	=	8 Pixels
Hor Sync Time	= 3.813;	// (usec)	=	**12** chars	=	96 Pixels
// H Back Porch	= 1.589;	// (usec)	=	**5** chars	=	40 Pixels
// H Left Border	= 0.318;	// (usec)	=	**1** chars	=	8 Pixels
Ver Total Time	= 16.683;	// (msec)	=	525 lines		HT – (1.06xHA)
Ver Addr Time	= 15.253;	// (msec)	=	480 lines		= 4.83
Ver Blank Start	= 15.507;	// (msec)	=	488 lines		
Ver Blank Time	= 0.922;	// (msec)	=	29 lines		
Ver Sync Start	= 15.571;	// (msec)	=	490 lines		
// V Bottom Border	= 0.254;	// (msec)	=	**8** lines		
// V Front Porch	= 0.064;	// (msec)	=	**2** lines		
Ver Sync Time	= 0.064;	// (msec)	=	**2** lines		
// V Back Porch	= 0.794;	// (msec)	=	**25** lines		
// V Top Border	= 0.254;	// (msec)	=	**8** lines		

Figure 3: VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT) Timing Specifications for 640x480 at 60 Hz [7]

The Nexys A7 series of FPGAs are only capable of outputting 4-bits each on the Red, Green, and Blue (RGB) channels respectively – allowing us to only use 12-bit color on our FPGAs as seen in Figure 3 below [6]. Digilent uses voltage dividers in parallel to select generate 4-bit color instead of a DAC, which explains the presence of the resistors on each line from the 3 RGB pins which receive digital RGB information from the 12-bit RGB channel, which we routed to our 12-bit RGB signal from our top module in the constraints file we specified for the Nexys A7-100T [6]. A code example of how we mapped our constraints for the VGA assembly is described in Figure 4 below.

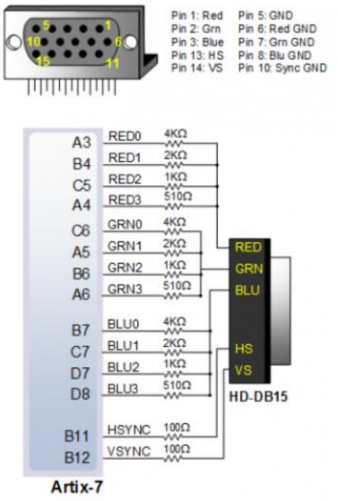


Figure 4: Nexys A7 VGA Interface [5]

```
133 ##VGA Connector
134 set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { rgb[8] }];
135 set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { rgb[9] }];
136 set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { rgb[10] }];
137 set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { rgb[11] }];
138 set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { rgb[0] }];
139 set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { rgb[1] }];
140 set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { rgb[2] }];
141 set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { rgb[3] }];
142 set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { rgb[4] }];
143 set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { rgb[5] }];
144 set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { rgb[6] }];
145 set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { rgb[7] }];
146 set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { hsync }]; #
147 set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { vsync }]; #
```

Figure 5: Constraints Mapped for the Nexys A7-100T VGA Interface

We tested the game via the Nexys A7-100T on three monitors: a Dell ST2010-BLK Flat Pane Monitor, and a Dell monitor and an eMachines monitor [8]. The Dell ST2010-CLK accepted either 521 rows or 525 rows for a sync pulse since the monitor supports automatic multiscan as shown in Figure 5 below, so that monitor was compatible with both timing specifications we tested with it [8]. We were unable to find sufficient documentation on the other two monitors aside from the Dell ST2010-CLK, but the other two monitors worked with our implementation.

Resolution Specifications

Horizontal scan range	30 kHz to 83 kHz (automatic)
Vertical scan range	56 Hz to 76 Hz (automatic)
Maximum preset resolution	1600x900 at 60 Hz

Figure 6: Dell ST2010-BLK Flat Pane Monitor Resolution Specifications [8]

Preset Display Modes

Display Mode	Horizontal Frequency (kHz)	Vertical Frequency (Hz)	Pixel Clock (MHz)	Sync Polarity (Horizontal/Vertical)
VESA, 720 x 400	31.5	70.0	28.3	-/+
VESA, 640 x 480	31.5	60.0	25.2	-/-
VESA, 640 x 480	37.5	75.0	31.5	-/-
VESA, 800 x 600	37.9	60.7	40.0	+/+
VESA, 800 x 600	46.9	75.0	49.5	+/+
VESA, 1024 x 768	48.4	60.0	65.0	-/-
VESA, 1024 x 768	60.0	75.0	78.8	+/+
VESA, 1152 x 864	67.5	75.0	108.0	+/+
VESA, 1280 x 1024	64.0	60.0	108.0	+/+
VESA, 1280 x 1024	80.0	75.0	135.0	+/+
VESA, 1600 x 900	55.5	60.0	97.75	+/-

Figure 7: Dell ST2010-BLK Flat Pane Monitor Preset Display Modes [8]

C. SEVEN-SEGMENT DISPLAY

This module allows the game to display the game's score and playtime to the 7-seg display on the board. The score is placed in the right four digits of the display, allowing a theoretical score of up to "9999" to be displayed. This also applies to the left four digits allowing a maximum game time of "9999" seconds or about 2.8 hours before overflowing to 0 seconds. The display is driven by the system clock at 100 MHz and is divided within the module by 1000 to slow down and properly allow the digits to multiplex and iterate through the individual digits of the input number right to left. Only one digit of the displayed number is illuminated at a time, however the rate at which the module switches between the digits of the input number is fast enough that the human eye perceives it as a static image. The module outputs an 8-bit cathode signal which is tied to the individual segments of the digit and the 8-bit anode signal which is tied to the individual digit itself. Both the anodes and cathodes are active when "low".

D. RANDOM NUMBER GENERATOR

For this module, we took inspiration from the Worcester Polytechnical Institute's ECE 4514 Digital Design II Lecture 6: A Random Number Generator to make a Linear Feedback Shift Register (LFSR) generate pseudorandom numbers [10]. We use two sets of five LFSR modules to provide random coordinates to the "apple", two random numbers are generated using two 5-bit Linear Feedback Shift Registers (LFSR) that each provide a pseudo-random number between 0 and 31. This is then multiplied by 10 with an additional 100 pixel offset for both the x and y-axis, respectively, to ensure that the resulting coordinates are within the play area and visible. A simulation of the pseudo-random output is visible on Figure 8 below with a repeat period of 15 clock cycles.

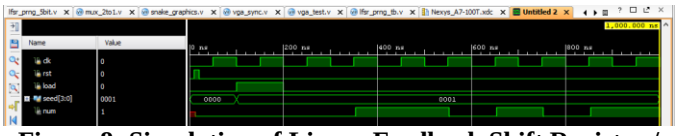


Figure 8: Simulation of Linear Feedback Shift Register / Pseudo-random Number Generator

E. CLOCK DIVIDER [1 SECOND]

This clock divider was necessary to generate a 1 Hz signal from the 100 MHz master clock to increment the game time by one second until reset. The module is initialized with a fixed divider value of 50,000,000 so that it alternates a half second low, half second high for a total period of 1 second. Clock input increments count until reaching divider value and toggling output.

F. TOP MODULE

Top module to instantiate all other modules and provide a basis for the game to operate within. This includes the setting, updating, and resetting of the stat output to the seven-segment display and lastly, the RGB value out of the snake graphics module which is synchronized between the pixel clock and system clock before it is outputted to the display.

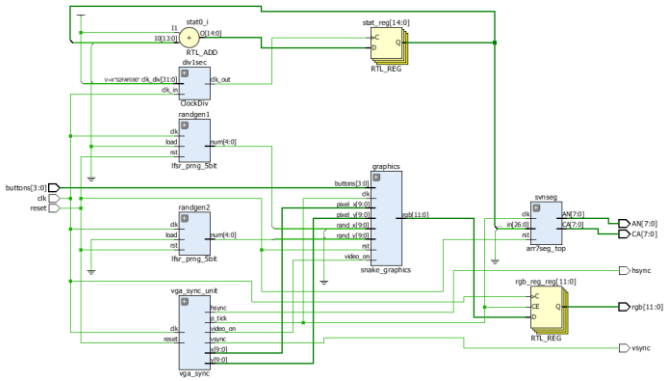


Figure 9: Hierarchical Structure of Snake Game in Verilog

IV. EVALUATION

Several iterations were required to achieve the current functionality of the game. Proper timing between modules and the graphics being displayed was essential, otherwise producing a variety of errors or even preventing the monitor from detecting the board. Initially, the VGA synchronization was tested and verified within the Vivado simulator, and the onboard switches were used to display various RGB values to the screen. We adopted some inspiration from N. V. Satanarayana Murthy's, "Video Graphics Array interfacing through Artix-7" to help give us an idea on how to write a testbench for the VGA module [5]. This was followed by drawing various static shapes to verify proper synchronization between the x and y-axis. The complexity quickly ramped up as more characteristic features were implemented. The head of the snake would not properly collide and reset position within the boundaries on the x-axis, however, the y-axis boundaries functioned as intentioned. The seven-segment display would also display unexpected values while incrementing each second. After seven seconds, the rightmost 4 digits would begin displaying random values due to an overflow error which was resolved. The most difficult component by far would be the generation of the snake's "tail" and having it follow behind the head of the snake. The tail would require a large amount of memory to store the location data of all the bitmapped tail segments within the display area. In addition, getting the tail synchronized properly with the head would take much longer than our allotted amount of time so we decided to not implement it. The current implementation is satisfactory, despite the difficulties encountered during development. Simulations of the VGA sync generator simulation results are shown below in Figures 10-14.

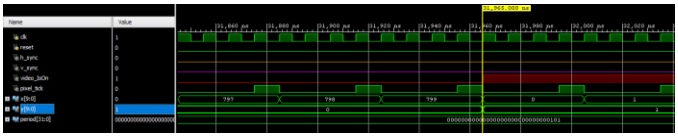


Figure 10: Simulation of VGA Generator - Vertical Sync Front Porch

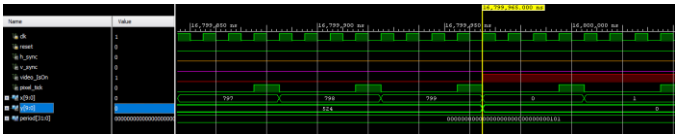


Figure 11: Simulation of VGA Generator - Vertical Sync Back Porch

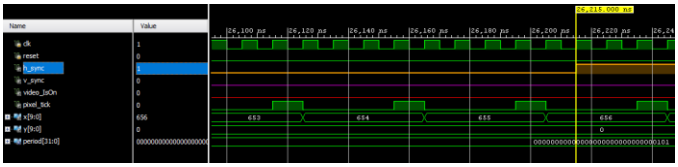


Figure 12: Simulation of VGA Generator - Horizontal Sync Front Porch

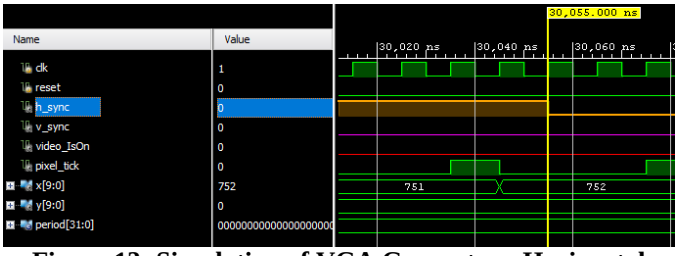


Figure 13: Simulation of VGA Generator - Horizontal Sync Back Porch

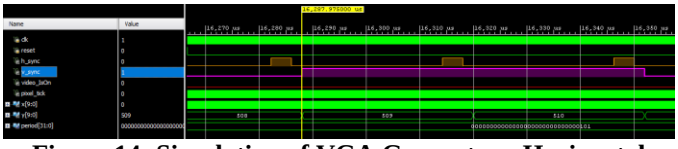


Figure 14: Simulation of VGA Generator - Horizontal Sync Finish

V. CONCLUSION

The complexity of implementing all of the desired features for the game was severely underestimated. Many iterations and many more hours were required to achieve the current level of functionality of the design. Research into the various concepts at play, and referencing other works allowed

it to be possible. Cooperating and relying on each other to implement the various modules required fought the time-induced anxiety of the project's deadline approaching.

REFERENCES

[1] S. S. Krishnajith, *Snake Game on FPGA in Verilog*, *SlideShare* [Online]. Available: https://www.slideshare.net/sskrishnajith/snake-game-on-fpga-in-verilog

[2] J. P. Nicole, *Pong Game*, *fpga4fun* [Online]. Available: https://www.fpga4fun.com/PongGame.html

[3] user3100088user3100088 30533 gold badges88 silver badges1919 bronze badges, et al., *Snake Game Using FPGA (Altera)*, *Stack Overflow* [Online]. 1 Mar. 1962, Available: https://stackoverflow.com/questions/20839209/snake-game-using-fpga-altera

[4] P. P. Chu, "VGA Controller I: Graphic" in *FPGA Prototyping by Verilog Examples Xilinx Spartan-3 Version*. Hoboken, NJ: Wiley, 2008, ch. 13, pp. 309-340.

[5] N. V. Satanarayana Murthy, "Video Graphics Array interfacing through Artix-7", International Research Journal of Engineering and Technology (IRJET), vol. 3, issue 3, pp. 1688-1695 Available: https://www.irjet.net/archives/V3/i3/IRJET-V3I3353.pdf

[6] Digilent, *Nexys A7 Reference Manual*, *Digilent* [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual

[7] Video Electronic Standards Association (VESA), *VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)*, *VESA* [Online]. Available: https://glenwing.github.io/docs/VESA-DMT-1.13.pdf

[8] Dell, *Dell™ ST2010-BLK Flat Panel Monitor User's Guide*, *Dell* [Online]. Available: https://downloads.dell.com/manuals/all-products/esuprt_electronics/esuprt_display/dell-st2010wfp_user's%20guide_en-us.pdf

[9] D. Lovegrove, I. Sweetland, K. Jacobson, R. Alves, *"Snake" on an FPGA*, *Instructables* [Online]. Available: https://www.instructables.com/Snake-on-an-FPGA-Verilog/

[10] P. Schaumont, *ECE 4514 Digital Design II Lecture 6: A Random Number Generator in Verilog*, *Worcester Polytechnical Institute* [Online]. Available: http://rdsl.csit-sun.pub.ro/docs/PROIECTARE%20cu%20FPGA%20CURS/lecture6[1].pdf