CE339 - High Level Digital Design

Assignment 2 – "Snake" Video Game

Akshay Gopinath
Registration Number: 2005614

A report presented for the degree of
Electronic Engineering

School of Computer Science and Electronic Engineering
University of Essex
England
March 25, 2024

CE339 - High Level Digital Design

Assignment 2 – "Snake" Video Game

Akshay Gopinath

ABSTRACT

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This report documents an experiment to design a "Snake" Video Game on hardware using a Hardware Description Language (HDL) called VHDL (Very High Speed Integrated Circuit Hardware Design Language). The target platform is the Digilent Basys3 Board which houses an Artix-7 based FPGA[1]. The VHDL code is synthesised using Xilinx Vivado. The VGA (Video Graphics Array) port on the Basys3 board is used to display the game on a compatible monitor and the player score is shown on the 7-segment display. This report will explain the design in a top-down approach, whilst going into detail on every sub-components. The top level schematic will generate the necessary signals required to correctly display the game and the score, such as the VGA synchronisation signals, RGB (red, green, blue) colour signals and the 7-segment display cathode and anode signals.

2. THE DESIGN

2.1. High Level Overview

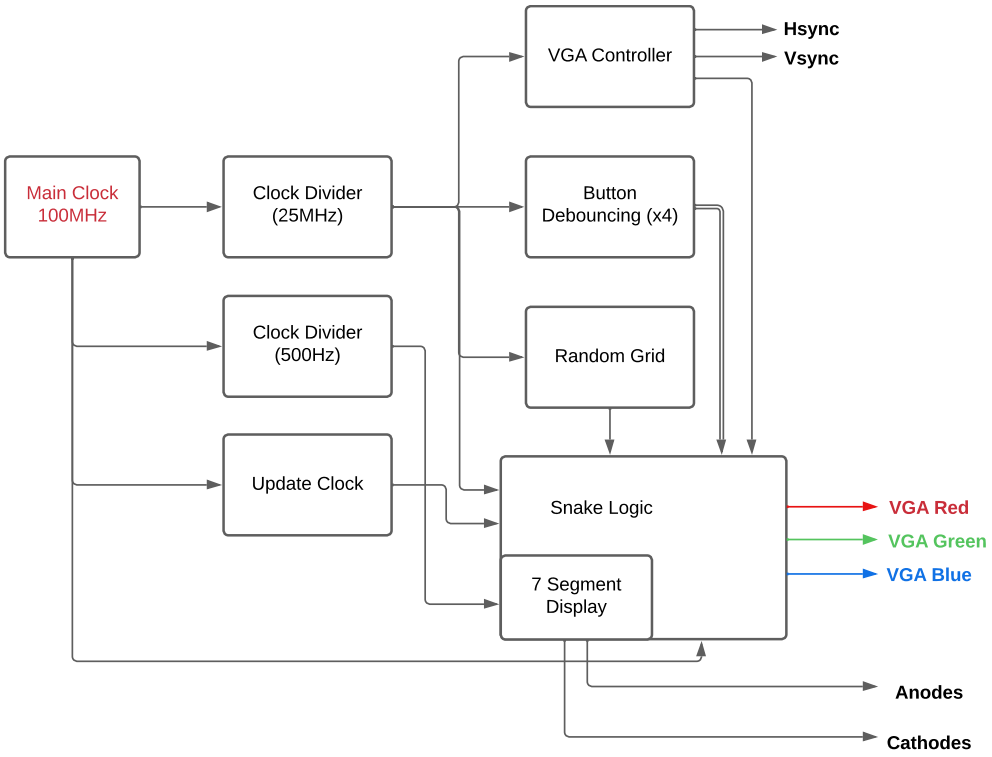


FIG. 1: *High level architecture of the system*

Figure 1 above shows the high level block diagram of the system. The main clock is from the Basys3 board which is at 100MHz frequency. The targer resolution is 640x480. The pixel clock is 25MHz, with a refresh rate of 60Hz. In order to generate the required synchronisation signals (by the VGA Controller module) for these specifications, the VGA controller module needs a 25MHz a clock. Hence the 100MHz master clock is given as input to a clock divider to generate this frequency. The 25MHz clock is also used for button debouncing (to keep it in synchronisation with the graphics being rendered to the screen), and also for the Random Grid module. Another clock divider was instantiated to generate a 500Hz clock. The 500Hz clock is used for the time multiplexed 7-segment display driver, which resides inside the Snake Logic module. The update clock is a module used to generate a pulse at a desired frequency (in this case 25Hz), which is high only for one clock cycle of the master clock. The purpose of this module is to set the update frequency of the game logic, hence the output of the update clock is given as input to the Snake Logic

module. The Random Grid module is used to generate pseudo random locations for the positions of the food in the game. And the output is given to the Snake Module as input. The VGA Controller module generates the horizontal and vertical synchronisation signals for the VGA Port. It also outputs the current x and y count co-ordinate as well as the blanking signal for the Snake Logic module to keep track of the screen position. The Snake Logic block is the main heart of the system. The Snake Logic module is the heart of the entire system, and contains the game main logic, as well as the rendering signals. This module contains many processes to control game elements such as the snake location direction, snake direction, snake size, game state, game levels etc. The module also contains the 7-segment display driver and BCD (Binary Coded Decimal) counters to display the score whilst playing the game. This module outputs the anode and cathode signals for the 7-segment display as well as the red, green and blue signals for the VGA port.

2.2. Detailed Overview

2.2.1. snake

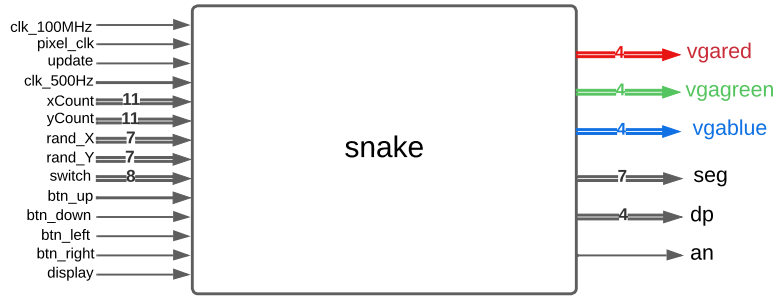


FIG. 2: *Inputs/outputs of snake module*

Figure 2 on the left depicts the inputs and outputs of the `snake` module. This module contains the main logic that controls the snake game, as well as the Read Only Memories (ROMs) that store the bitmapped sprites/graphics. The inputs are master 100MHz clock (for synchronisation), pixel clock (25MHz), update clock (25Hz), x and y counters, random food location, and the debounced buttons. It outputs the red, green, blue signals for VGA and the cathode and anode signals for the 7-segment display.

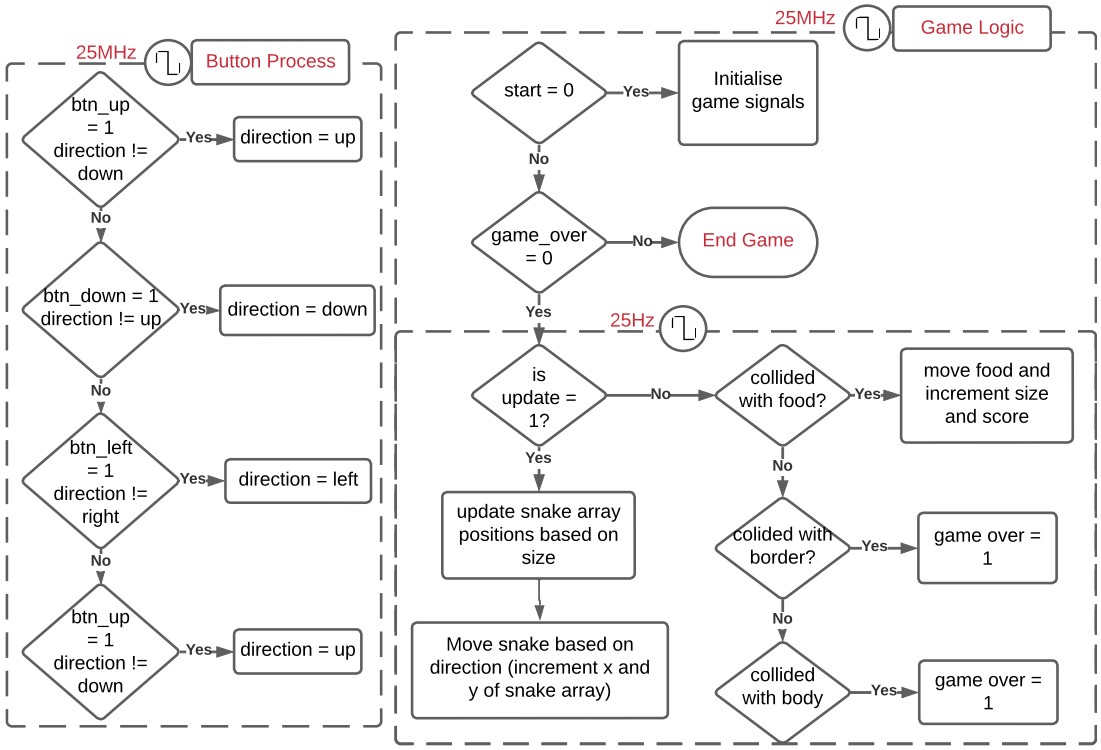


FIG. 3: *Button and game logic processes of the snake module*

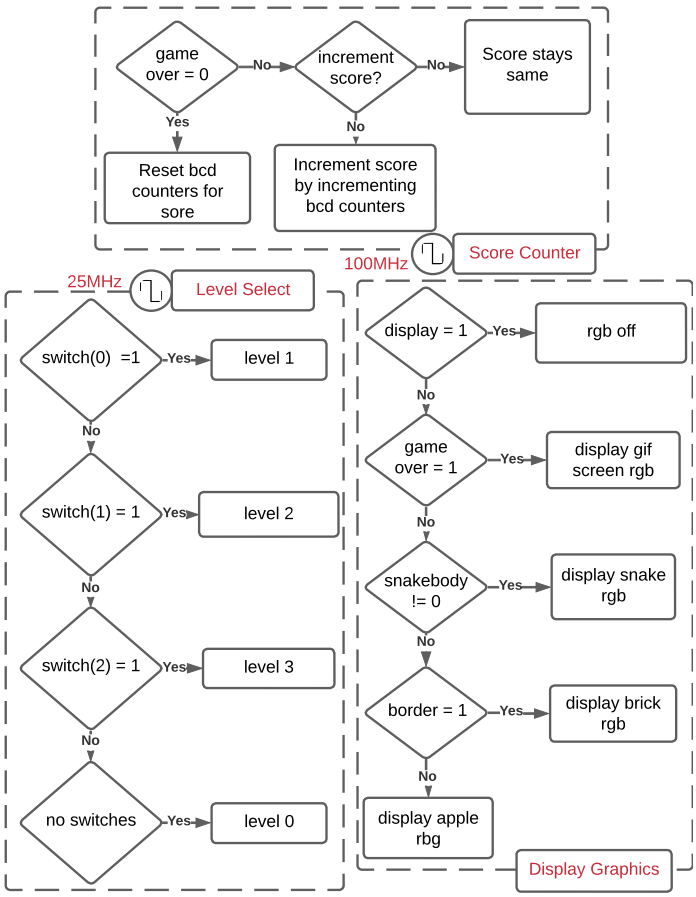


FIG. 4: *Level select and score count process and colour rendering*

The `snake` module is composed of many processes to construct the game logic and behaviour. Figure 3 and Figure 4 shows flow diagrams to represent this. Figure 3 above shows the button/input and the main game logic process. The button process sets the value of the `direction` register based on the button input. The button input is received from the `Debounce` module (which debounces the button presses for reliability). This process updated with the pixel clock of 25MHz. The game logic process handles updating the snake for the movement as well as checking for collisions. If the start signal is zero, the game is idle and all the starting game conditions are set to defaults (such as initial food location). If the game over signal is high, the game will end until the game is reset, and once reset, the game is reset back to default values. This part of this process is updated at 25MHz. The other parts of the process is updated at 25Hz, which is the games update clock. When the update clock is high, the snake position array (for both x and y locations) are updated in a for loop depending on the current snake size. The snake position at the first index (snake head) is updated depending on the current direction (set by the buttons). When the update clock is low, the collision between the border, food and the snake body is checked. If a collision with the food is detected, the snake size is incremented (unless max snake length is reached). Next if the snake head collides with a border or itself, then the game over signal will be set, thus ending the game, until the game is reset. Figure 4 shows three other important processes in this module. The score counting process increments the score during the game when food is collected and when the game ends, the game is reset. BCD counters and a 7-segment display driver is used to achieve this. The level select process is very simple, it selects the level based on the switch input. And the levels are border signals that are generated based on certain conditions that decide where the borders are placed on the screen. Lastly, the colour displaying/rendering logic is a combinational process, the other processes we have seen so far are sequential logic. If the display signal is high (same as blanking, which is active low logic), then the RBG colours are turned off. The rgb colours are selected based on which graphics is to be rendered, in this case, the border graphics, the game over GIF and the snake. Graphics in this game are all bitmapped ROMs, this will be explained later in the report.

2.2.2. **updateClk**

(a) *Flow chart for updateClk*

(b) *Entity block of updateClk*

FIG. 5: *updateClk module diagrams*

Figure 5 above contains two sub figures, Figure 5a and Figure 5b which are the architecture and block diagram of this entity respectively. This module has one input, the master clock and one output, the update clock. This is a generic value where the max count can be configured for a different update clock. The update clock used in this game is set to 25Hz. The architecture counts till the set max value and sets the output high for one clock cycle of the master clock, and then becomes low after.

2.2.3. randomGrid

FIG. 6: *randomGrid entity diagram*

Figure 6 on the left shows the inputs and outputs of the randomGrid module. This module is used to generate the pseudo random x and y locations for the food. The pseudo random generation implementation in this project is very simple, as a simple mathematical operation is done on the current random location to move the food 'unpredictably'.

Code Snippet 4: randomGrid architecture implementation

```
if rising_edge(pixel_clk) then
    rand_X <= ((rand_X + 3) mod 37) + 1; -- set random x and y position
    rand_Y <= ((rand_Y + 3) mod 27) + 1;
end if;
```

Since the implementation is simple, it is shown in Code Snippet 1. A simple arithmetic oprtation is computed on the value read back from the current random x and y value, at the rising egde of the pixel clock. The output of this module is given as input to the `snake` module.

2.2.4. 7-Segment Display

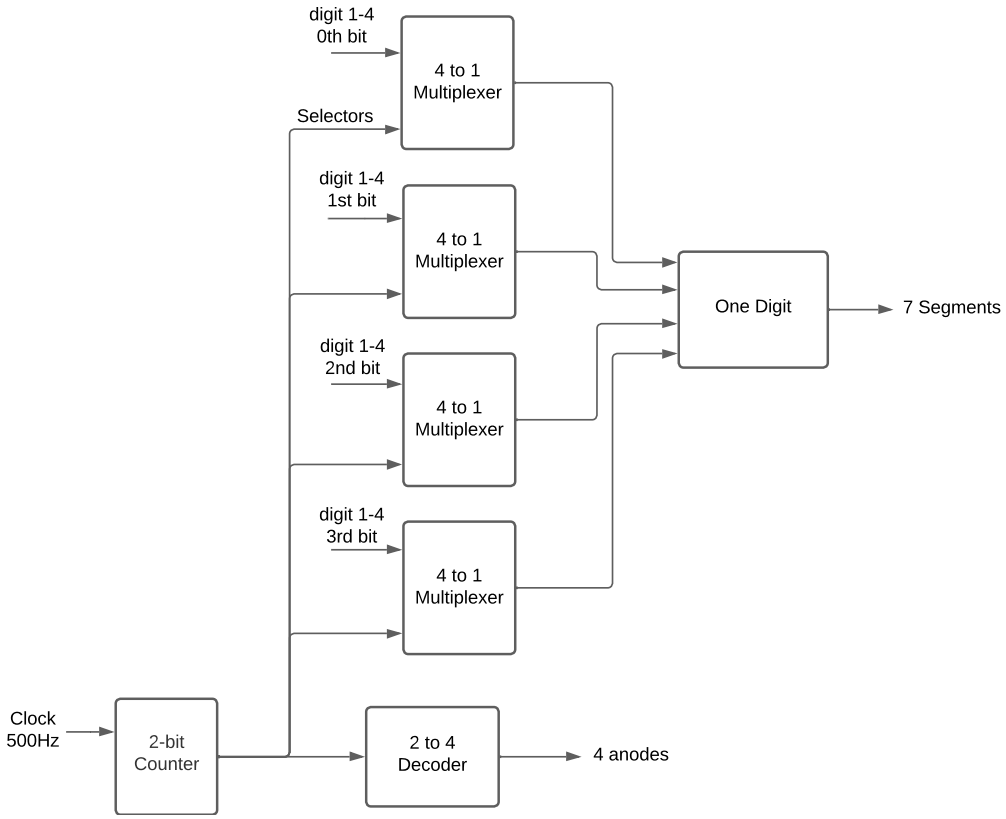


FIG. 7: *Architectural block diagram of the four_digits module*

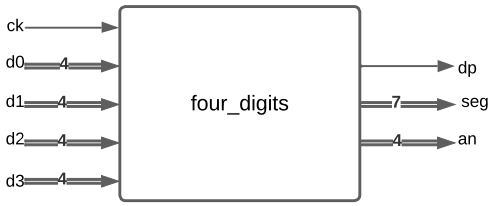


FIG. 8: *four_digits entity diagram*

This section explains the 7-segment display driver (and it's subcomponents) circuit used to display the score. Figure 8 on the left shows the entity inputs and outputs and Figure 7 above shows the high level connection diagram. This module takes in a 500Hz clock, and four inputs which are four bits wide, and generates the required anode and cathode displays. Time division multiplexing is used to display the four digits on the four seven segment displays available on the Basys3 board. The 500Hz clock given to a two bit counter which counts from 0→3, and it's output is fed into a 4-t0-1 multiplexer to select the corresponding bit for that digit. The four multiplexer output is given to a module called `one_digit` which is a 7-segment decoder. The 2-bit counter output is also used by a 2-to-4 decoder to select the correct anode which is currently being displayed. The anodes are swapped at a rate of 500Hz, a refresh rate human eyes cannot visibly see. The 2-bit counter being very simple is implemented using behavioural modelling. Figure 9, 10 and 11 below shows the entity block diagrams of `mux4_1`, `decoder2_4` and `one_digit` respectively, which is used in the circuit from Figure 7.

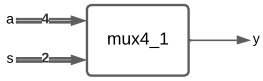


FIG. 9: *mux4_1 entity*

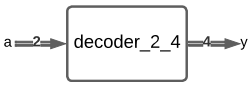


FIG. 10: *decoder_2_4 entity*

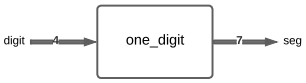
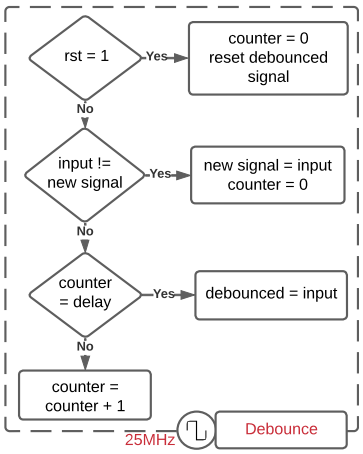


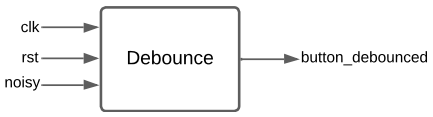
FIG. 11: one_digit entity

All the three modules are very simple. `one_digit` takes in a 4 bit number as input and outputs the corresponding 7 bit output for the 7 segments on the 7-segment display. The `decoder` entity is a simple two-to-four decoder, a certain bit of the 4 bit output is low depending on the input. For example, if the input is 01_2, the output is 1101_2, the 2^{nd} bit is low. And lastly, the 4-to-1 mux routes the corresponding bit of the input `a` depending on the select input `s`, to the output. The `four_digit` module is instantiated inside the `snake` module from Figure 2 and its outputs are propagated outside the `snake` module.

2.2.5. **Debounce**



(a) *Flow chart for Debounce*



(b) *Entity block of Debounce*

FIG. 12: *Debounce module diagrams*

Figure 12a and 12b shows the flow chart and entity diagram for the Debounce module respectively. There are three inputs, clk, rst and noisy (the button input) and one output which is thd debounced press. This module has a configurable generic parameter, `DELAY`. This module is clocked at the pixel clock, 25MHz. The module has a counter that counts till a delay value is reached, and once the count is same as the delay, the button press is debounced, and the counter is reset. The debounced output is input to the `snake` module.

2.2.6. vga_controller_640_60

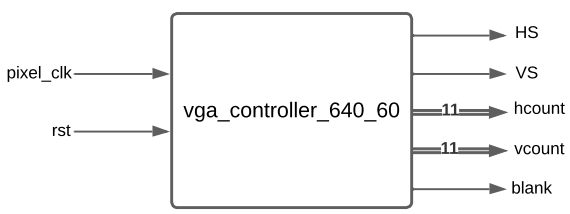


FIG. 13: *VGA Controller module*

Figure 13 on the left shows the entity diagram of the `vga_controller_60_40` module. This module wasn't written by the author and was given by the supervisor. This module has 2 inputs, the 25MHz pixel clock (`pixel_clock`) and reset (`rst`). And outputs the horizontal sync (`HS`), vertical sync `VS`, current horizontal counter value (`hcount`), current vertical counter value (`vcount`) and the blanking signal (`blank`). The horizontal and vertical synchronisation signals are outputted to the VGA port in the top level hierarchy. The vertical counter, horizontal counter and the blanking signals are all given as inputs to the `snake` module from 2 to control the game logic.

2.2.7. nbit_clk_div