DIGITAL SYSTEMS DESIGN

VGA CONTROLLER

-DOCUMENTATION-

Coordinating Teacher: Timar Mihai Students:

Hategan Diana & Fechete David

Group: 30413

Table of Contents

[Chapter 1: Introduction 3](#_Toc104457461)

[1. Requirement 3](#_Toc104457462)

[2. General features 3](#_Toc104457463)

[Chapter 2: Design 4](#_Toc104457464)

[1. Black Box 4](#_Toc104457465)

[2. Components 5](#_Toc104457466)

[1. Basic Components 5](#_Toc104457467)

[2. Main Module 9](#_Toc104457468)

# Chapter 1: Introduction

## Requirement

The requirement is to implement a VGA Controller using an FPGA board. There must be 4 different images (square, circle, triangle, vertical lines) displayed on the screen which colors must be changed using the switches on the FPGA board. The VGA Controller must also be able to move the objects on the screen along the X and Y axis using other buttons.

## General features

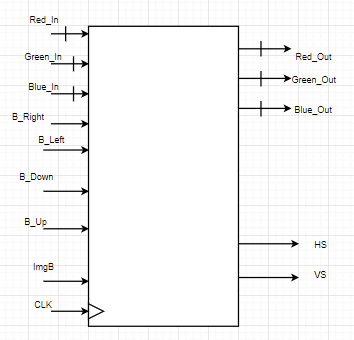
VGA stands for Video Graphics Array. Initially, it refers specifically to the display hardware first introduced with an IBM computer in 1987.

The implementation consists of 2 timing signals (Horizontal and Vertical Sync) and 3 busses that represent the colors Red, Green and Blue.

The project was implemented using a NEXYS-4 FPGA Board and with a resolution of 640 x 480 pixels.

# Chapter 2: Design

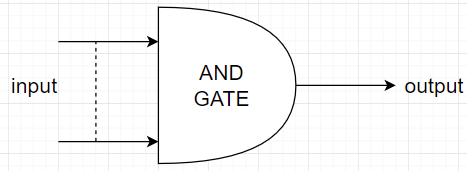
## Black Box



## Components

### Basic Components

#### AND Gate



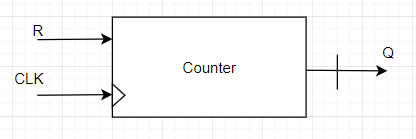
Generic parameter: **N**

Input: **input**

Output: **output**

This component receives a variable number of inputs and implements a basic and gate.

#### Counter



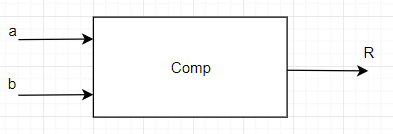
Generic parameter: **bits**

Inputs: **R, CLK**

Output: **Q**

This is a basic counter that increases the output Q with one unit at each rising edge of the clock signal. It also features an asynchronous reset.

#### Comparator



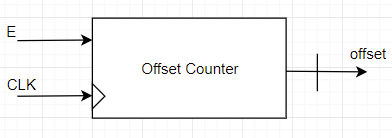
Generic parameter: **N**

Inputs: **a, b**

Output: **R**

This comparator returns the value ‘1’ if **a** is strictly lower than **b**, otherwise ‘0’.

#### Offset Counter



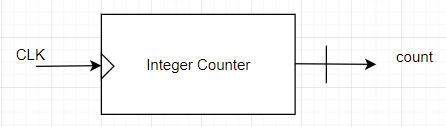
Generic parameter: **number**

Inputs: **E, CLK**

Output: **offset**

This counter has a clock enable feature and also it has an integer type output. If the input enable (E) is active then the counter increases the output value by one unit on every rising edge of the clock, if not, the counter doesn’t work. When offset reaches its limit the counter resets on the next clock cycle.

#### Integer Counter



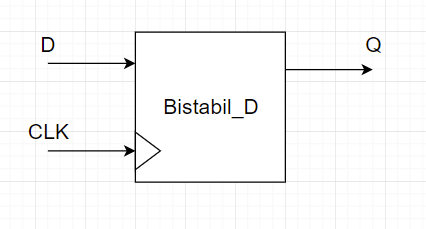
Generic parameters: **max**

Input: **clk**

Output: **count**

It is similar with a normal counter but has as output an integer.

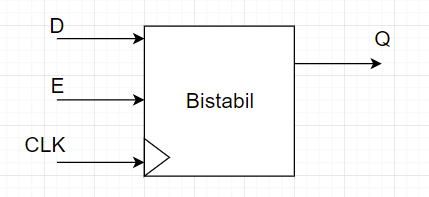
#### D Flip Flop



Input: **D**, **CLK**

Output: **Q**

#### D Flip Flop for Integers



Generic parameter: **number**

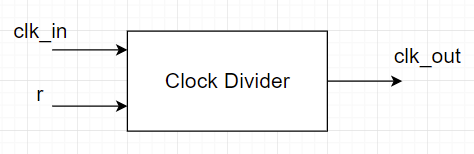
Input: **D,E,CLK**

Output: **Q**

This component is a D Flip Flop for integers. This means that input D and output Q are integers. It has a similar behavior as a D Flip Flop, but it features an enable signal.

### Main Module

#### Clock Divider

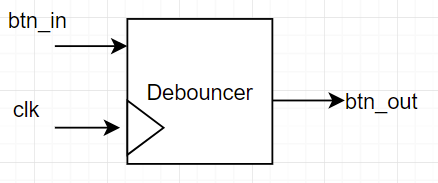


Inputs: **clk\_in, r**

Output: **clk\_out**

This module divides the pixel clock. For a NEXYS-4 FPGA board we have a 100 MHz clock frequency and for our resolution (640x480) we need a frequency of 25MHz. To obtain this frequency we will use a 2-bit counter. Each computation happening in the counter splits the clock in 2 so to get a frequency of 25MHz we will take the most significant bit.

#### Debouncer



Generic parameter: **nr\_bistabile**

Input: **btn\_in, clk**

Output: **btn\_out**

The debouncer component takes an input signal from a button and generates a clean output for digital circuits. In this way the circuit will respond to only one pulse generation.

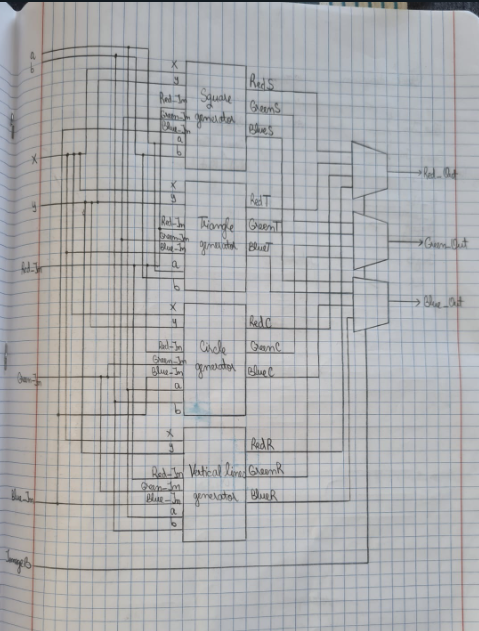
#### Debounce unit

Inputs: **clock, imgB, leftBtn, rightBtn, upBtn, downBtn**

Outputs: **img, left, right, up, down**

This module represent a series of debouncers, one for each button of the interface.

#### Image Generator



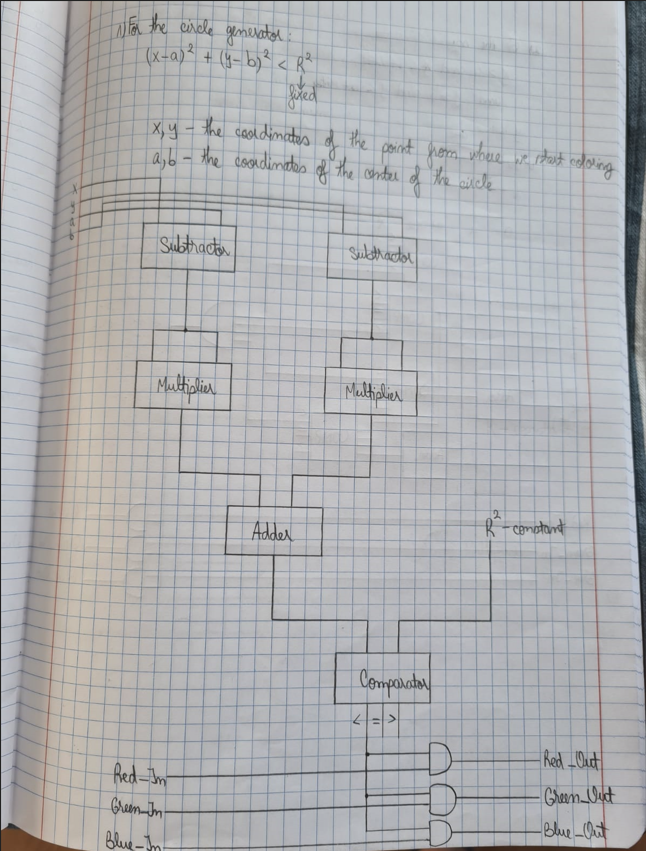
Inputs: **ImgE, x, y, img, a, b, Red\_In, Green\_In, Blue\_In**

Outputs: **Red\_Out, Green\_Out, Blue\_Out**

The image generator uses mathematical expressions to determine which pixels should be colored. The module features a size constant which determines the size of each image

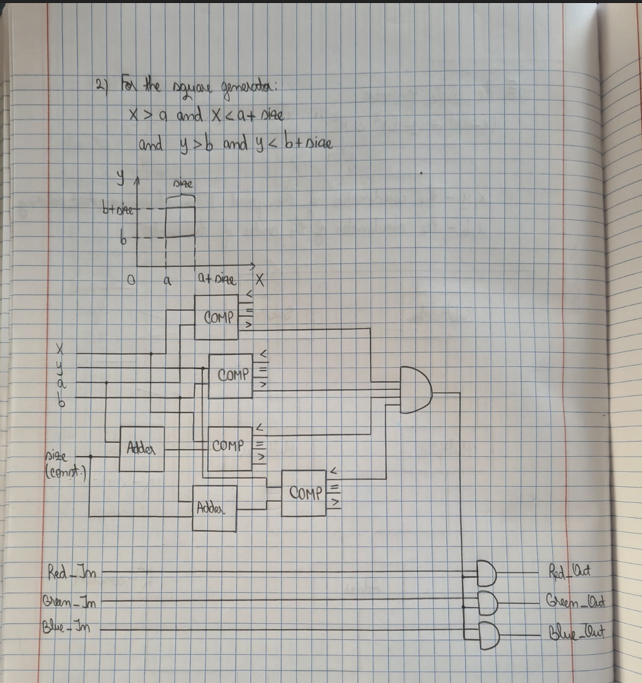
The project features the following images :

1. Circle

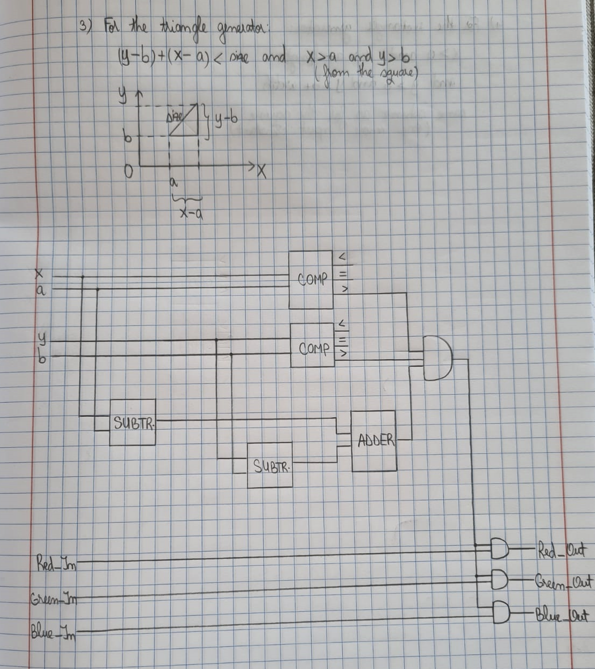


To make sure the circle doesn’t go off the screen because the center is in the top left corner we used a circle\_disp constant.

1. Square

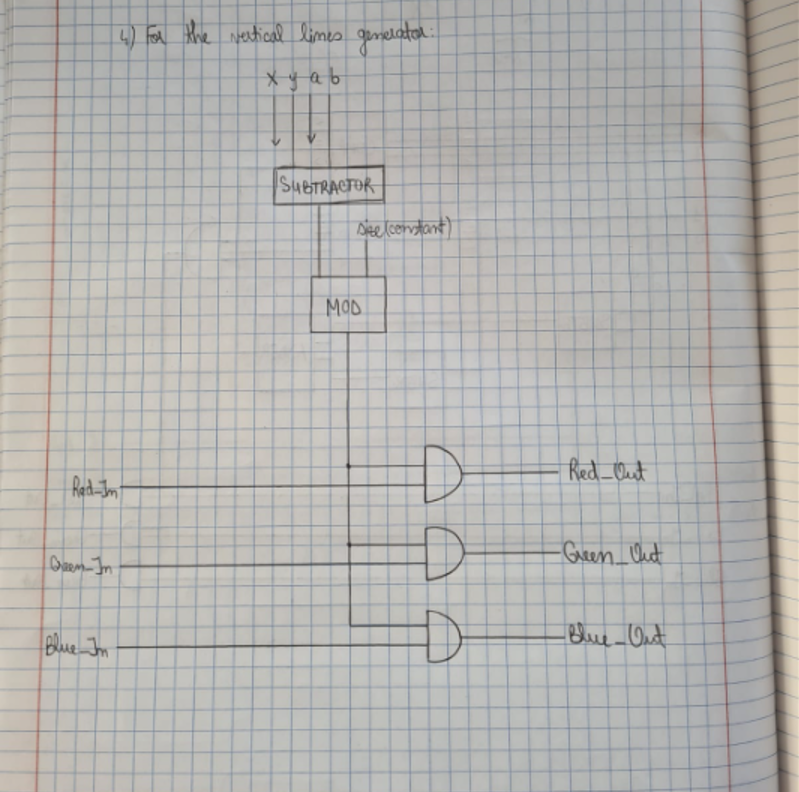


1. Triangle

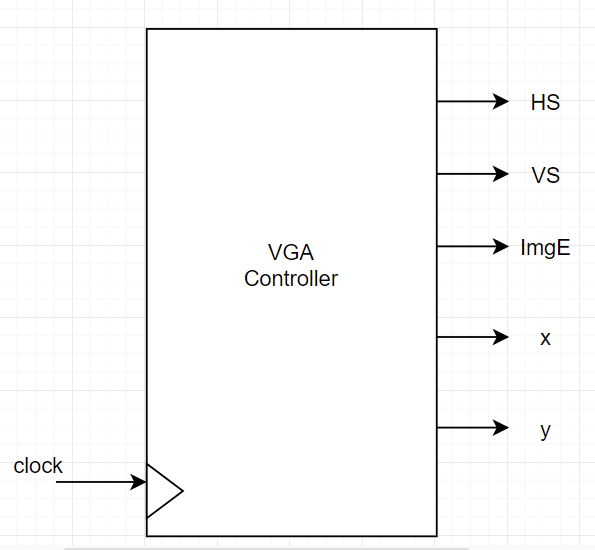


1. Vertical lines

(y-b) mod size = 0



#### VGA Controller



Input: **clock**Outputs: **HS, VS, ImgE, x, y**

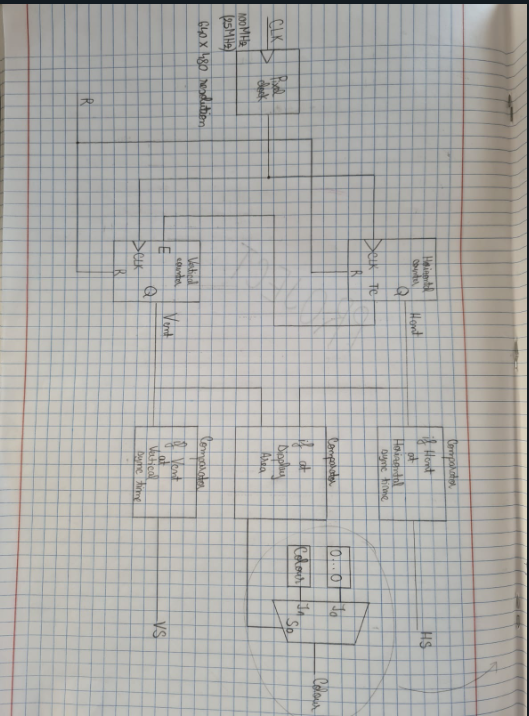
We use a flip flop to store the value of the column, and we can only store it if it’s smaller than h\_display (640 pixels constant = nr of pixels in a row of the screen). Using the value of h\_count we can also determine HS output by the following equations :

H\_first\_part = h\_display + h\_front

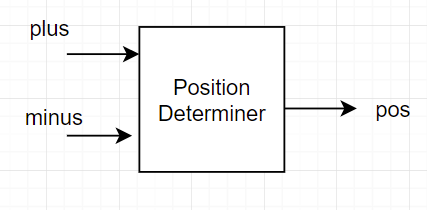
H\_last\_part = h\_display + h\_front + h\_sync\_pulse

We have a similar logic for the vertical sync as well. We can only store v\_count if the current row value is lower than v\_display (480 pixels).

The enable signal (ImgE) has the value ‘1’ to represent the active time of the controller (when the image is showing on the screen) and ‘0’ for the blank period.



#### Position Determiner



Generic parameter: **max**

Input: **plus, minus**

Output: **pos**

It determines the position for the x and y coordinates of the origin of each image. For this, we need 2 counters, each for the 2 opposing directions.