Verilog HDL Design

Project Report

Bregelem Circle Drawing Algorithm
Implementation on FPGA
Using VGA Out

<<Names of Project Team Members>>
<<One Name per Line>>
<<Line 3>>
<<Line 4 – replace the text on these lines>>

April , 20

Abstract

We implemented Bresenham Circle algorithm which is a hardware (or software!) friendly algorithm to draw circles with arbitrary center (fixed center is used) and radius on the screen.

The algorithm is quite efficient: it contains no multiplication or division. Because of its simplicity and efficiency, the Bresenham Circle Algorithm can be found in many software graphics libraries, and in graphics chips. We implemented it on FPGA using Verilog HDL to write the algorithm and then displaying it on LCD using VGA out.

1 TABLE OF CONTENTS

2 INTRODUCTION

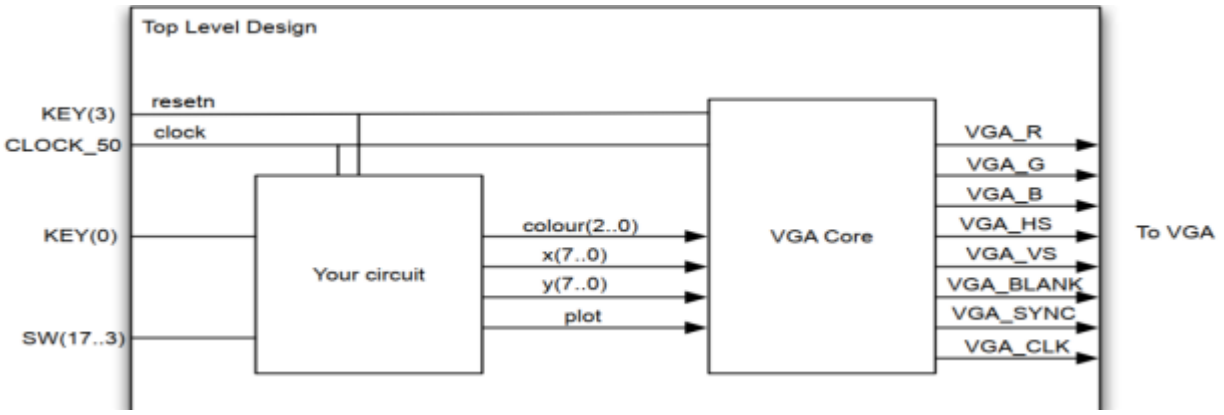
A hardware to draw circles on the VGA screen is designed using Verilog HDL language and implemented Bresenham Circle Drawing Algorithm on FPGA. At first glance, you might think that drawing circles is really boring but turns out though, it is one of the most fundamental operations in graphics technology. And because it's so fundamental, circle-drawing (along with line drawing and other basic geometric shapes) is often implemented with dedicated hardware. Anytime you look at a screen with any sort of rendered graphics, be it your desktop PC, smart phone, smart TV, Xbox, or even futuristic virtual reality technologies like the Oculus Rift or Microsoft HoloLens, you will see hardware implemented shape-drawing in action. Bresenham Algorithm is very efficient and simple to implement

3 DESIGN OBJECTIVES

Main objective of this design is to get familiar with VGA controller and VGA out on LCD and implementing Bresenham Algorithm on FPGA using Verilog HDL.

4 FUNCTIONAL DESCRIPTION

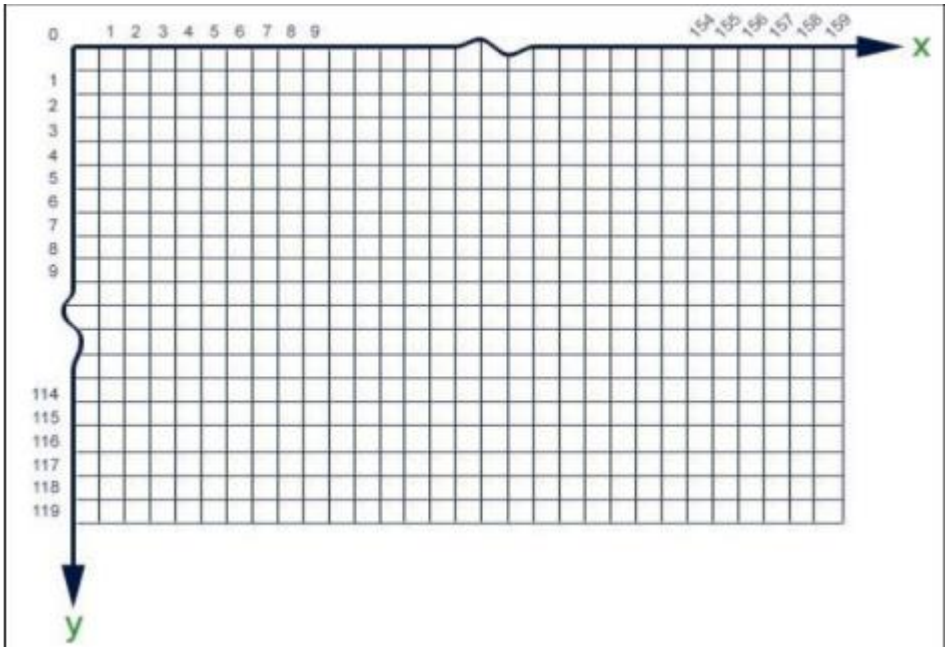
Bresenham algorithm to set X and Y values to respective values to draw circle was implemented and then given to VGA adapter component used which is being taken from University of Toronto's webpage made by a group of students there. This is common practice in industrial design – taking predesigned components that are either purchased or written by another group and incorporating them into your own design.Top Level Model of which is given below :



4.1 VGA (LCD) SCREEN :

We can picture the VGA screen as a grid of pixels. The X/Y position (0,0) is located on the top-left corner and (159,119) pixel located at the bottom-right corner. The role of the VGA Adapter core is to

continuously draw the same thing on the screen at the monitor's refresh rate, e.g. 60 Hz. To do this, it has an internal memory that stores the colour of each pixel. Your circuit will write pixel colours to the VGA Adapter core. n order to save on the limited memory on DE-1 SOC board, the VGA Adapter core has been setup to display a grid of 160x120 pixels.



4.2 VGA ADAPTER CORE :

To set the colour of a pixel, you first drive the VGA Adapter Core's X, Y and COLOUR inputs with the pixels' x coordinate, y coordinate, and desired color value, respectively. You then raise the PLOT input to high. You must keep these values driven until the next rising clock edge. At the next rising clock edge, the pixel colour is accepted by the VGA Adapter core's memory. Then, starting on the next screen redraw, the pixel will take on the new colour. In the following timing diagram (from the UofT Website), two pixels are changed: one at (15, 62) and the other at (109,12). As you can see, the first pixel drawn is green (rgb = 010) and is placed at (15, 62). The second is a yellow pixel at (109, 12). It is important to note that, at most, one pixel can be changed on each cycle. If you want to change the colour of m pixels, you need m clock cycles.

Inputs:

Resetn	Active low reset signal (does not reset the screen buffer). Digital circuits with state elements should always contain a reset.
Clock	Clock signal. The VGA Adapter core must be fed with a 50MHz clock to function correctly.
Colour (2 down to 0)	Pixel colour (3 bits). Sets the colour of the pixel to be drawn. The three bits indicate the presence of Red, Green and Blue components for a total of 8 colour combinations.
x (7 down to 0)	X coordinate of pixel to be drawn (8 bits) – supported values $0 \leq x < 160$.
y (6 down to 0)	Y coordinate of pixel to be drawn (7 bits) – supported values $0 \leq x < 120$.
Plot	Active high plot signal. Raise this signal to cause the pixel at (x, y) to be set to the specified colour on the next rising clock edge.

Outputs :

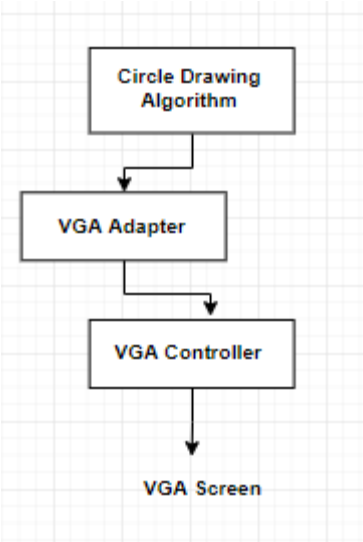
VGA_CLK	VGA clock signal.
VGA_R (9 down to 0) **VGA_G (9 down to 0)** **VGA_B (9 down to 0)**	Red, Green, Blue components of display (10 bits). These signals are connected to the Digital-to-Analog Converter (DAC) on the DE2 board (or whichever you have) before transmitting to the monitor.
VGA_HS **VGA_VS** **VGA_SYNC** **VGA_BLANK**	VGA control signals.

4.3 BRESENHAM CIRCLE DRAWING ALGORITHM IMPLEMENTED:

```
function circle_bresenham ( xc , yc , r )
        x := 0
        y := r
        d := 3-2*r
        loop
                if x > y exit loop

                setPixel (xc + x, yc + y)
                setPixel (xc - x, yc + y)
                setPixel (xc + x, yc - y)
                setPixel (xc - x, yc - y)
                setPixel (xc + y, yc + x)
                setPixel (xc - y, yc + x)
                setPixel (xc + y, yc - x)
                setPixel (xc - y, yc - x)

                x := x + 1

                if d < 0 then
                        d := d + (4 * x) + 6
                else
                        d := d + 4 * (x – y) + 10
                        y := y – 1
                end if-else
        end loop
end function
```

5 BEHAVIORAL AND RTL DESCRIPTION

Top module implements a state machine and then instantiates the VGA_adapter module to drive different display on VGA screen.



5.1 Top Module (Circle Drawing Algorithm):

Inputs :

CLOCK_50	50 MHz clock input
SW[9:3]	For radius Input of the Circle to be displayed(1-59) values greater than 59 are hardwired to 59 to avoid any logical errors
SW[2:0]	Different 8 colors inputs of the circle
KEY[0]	Reset Button which clears the screen to BLACK
KEY[1]	Str (which draws circles on pressing)

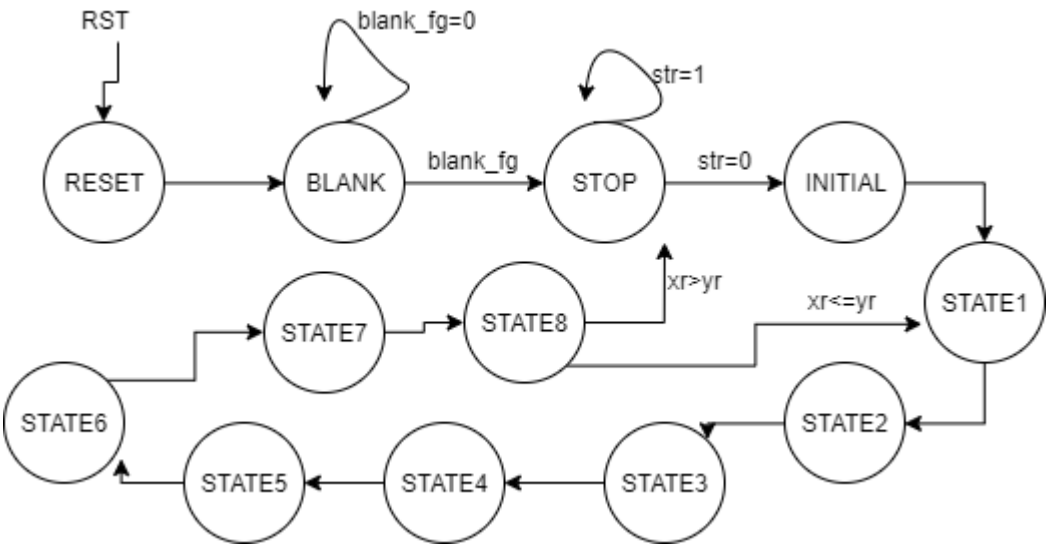
Intermediate Outputs :

These outputs are then used to drive VGA_adapter.

X coordinates	X axis Pixel which needs to be ON
Y coordinates	Y axis Pixel which need to be ON

Color	Color of the Pixel

5.2 State Diagram :



5.3 Explanation :

Our Design takes radius and color of the circle using Switch inputs on the FPGA board and wait for the KEY[1] to be pressed to display respective radius and color circle on the VGA out. A little complex state machine is being implemented which has 10 states according to the algorithm and performs different checks in different states to perform the algorithm which can simply be understood from Verilog Code.

6 Initial Versions :

6.1 For VGA_adapter Testing (VGA_check) :

To understand working of VGA adapter and its implementation on DE-SOC1 board and to clear all the queries interfacing it with LCD screen we tested this version of the code which simply takes position of the pixel from SW[8:4] for Y axis and SW[3:0] for X axis and uses KEY[1] to plot it on VGA screen.

6.2 To Implement Reset Logic (reset_FSM):

As VGA_adapter reset only refreshes the screen and doesn't clear the screen so we had to design a FSM to implement screen clearing logic. For which we used different we simply implemented this pseudo code :

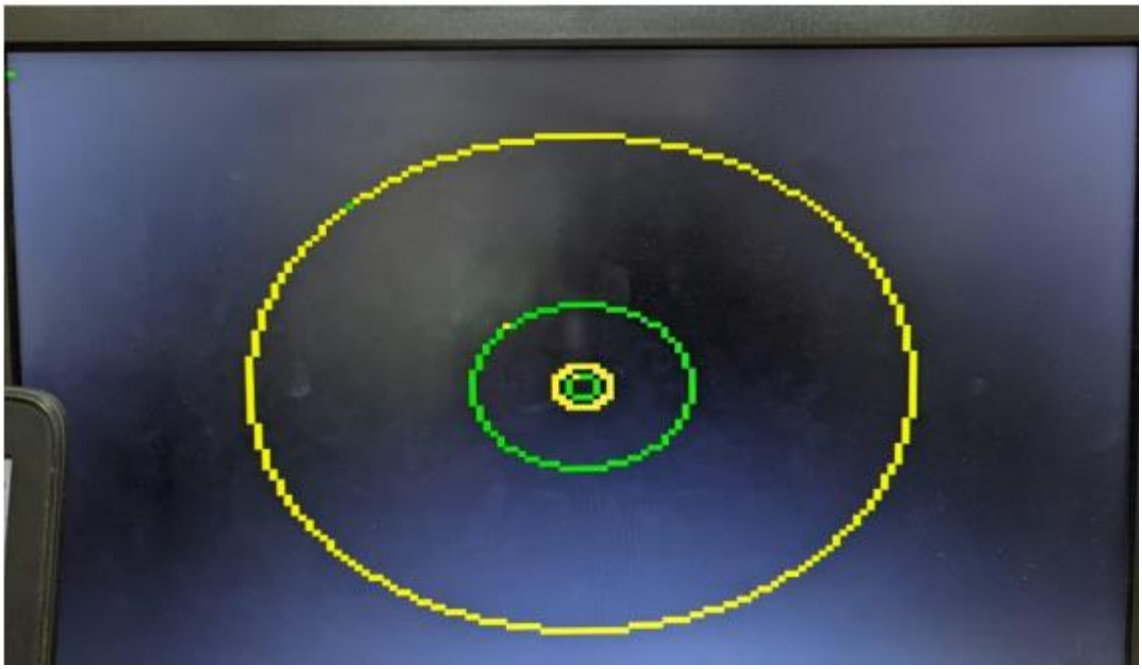
```
for y = 0 to 119 {
        for x = 0 to 159 {
                set pixel (x, y) to colour ( y mod 8)
        }
}
```

7 Design Verification

This was a algorithm based design and algorithm was tested by seeing outputs on the VGA display as if the algorithm was implemented correctly or not.

8 Analysis of Results

After doing all the implementation we uploaded the code on the FPGA and resolve all the issues we faced to get this output on the VGA screen which perfectly implements the logic algorithm.



9 SUMMARY AND CONCLUSIONS

After doing this all we can conclude that it was a great exercise to understand the importance of such algorithms to be implemented on actual hardware which can help us speed up the process and also got very familiar with designing real world implemented circuits using Verilog HDL and then using FPGA to implement it. So much to learn afterall.

10 REFERENCES

The VGA Adapter core was created at the University of Toronto for a course similar to ours. The following describes enough for you to use the core; more details can be found on University of Toronto's web page: https://www.eecg.utoronto.ca/~jayar/ece241_07F/vga/