COLOR DETECTION USING FPGA

ES 203: Digital Systems

Indian Institute of Technology Gandhinagar
Dishank Goel (18110052) Kushagra Sharma (18110091)
Pushkar Mujumdar (18110132) Shivam Sahni (18110159)

Objective

1) To count the number of occurrences of circles of different colors in an image.
2) To classify the circles as small, medium or large based on their radius

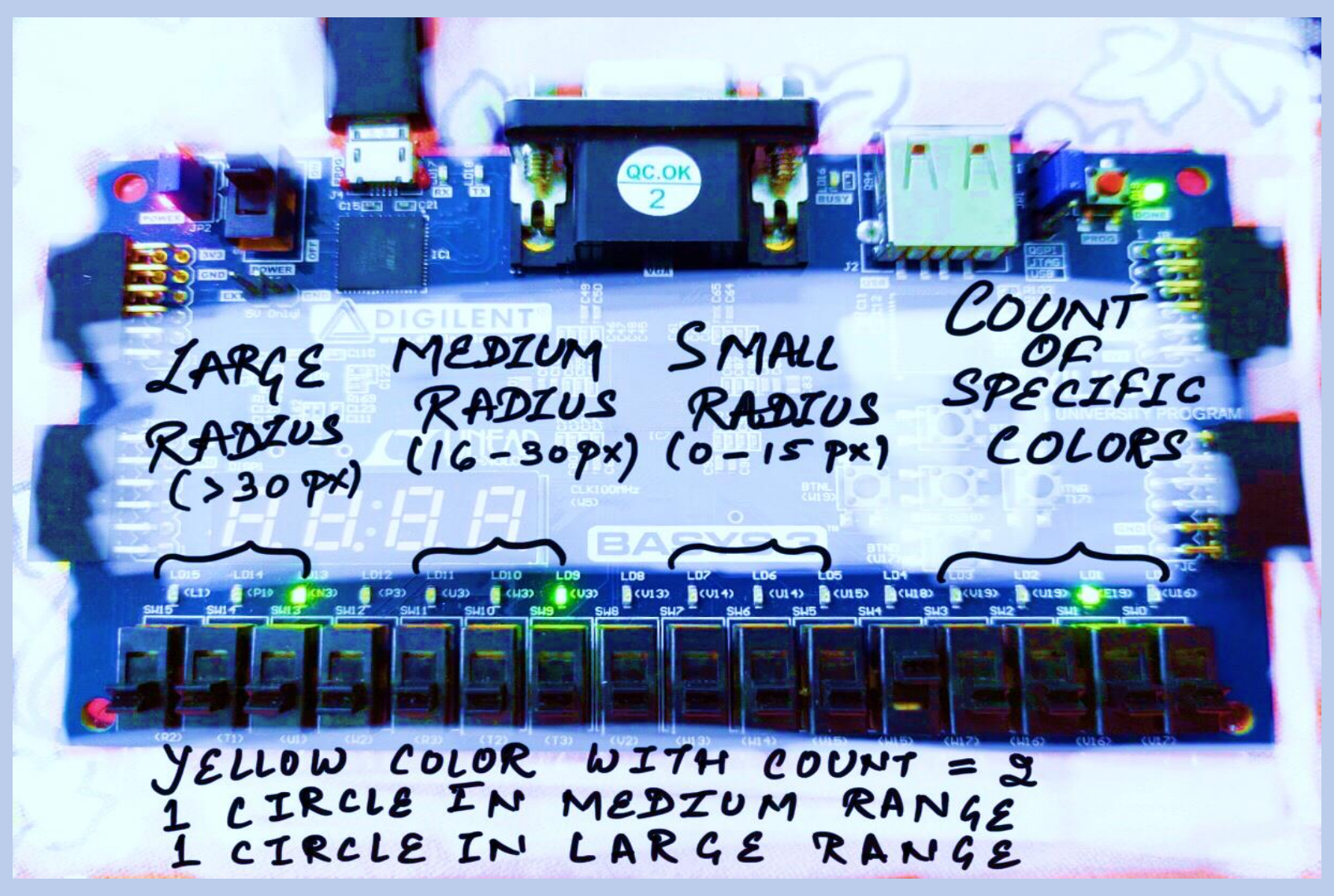
Introduction

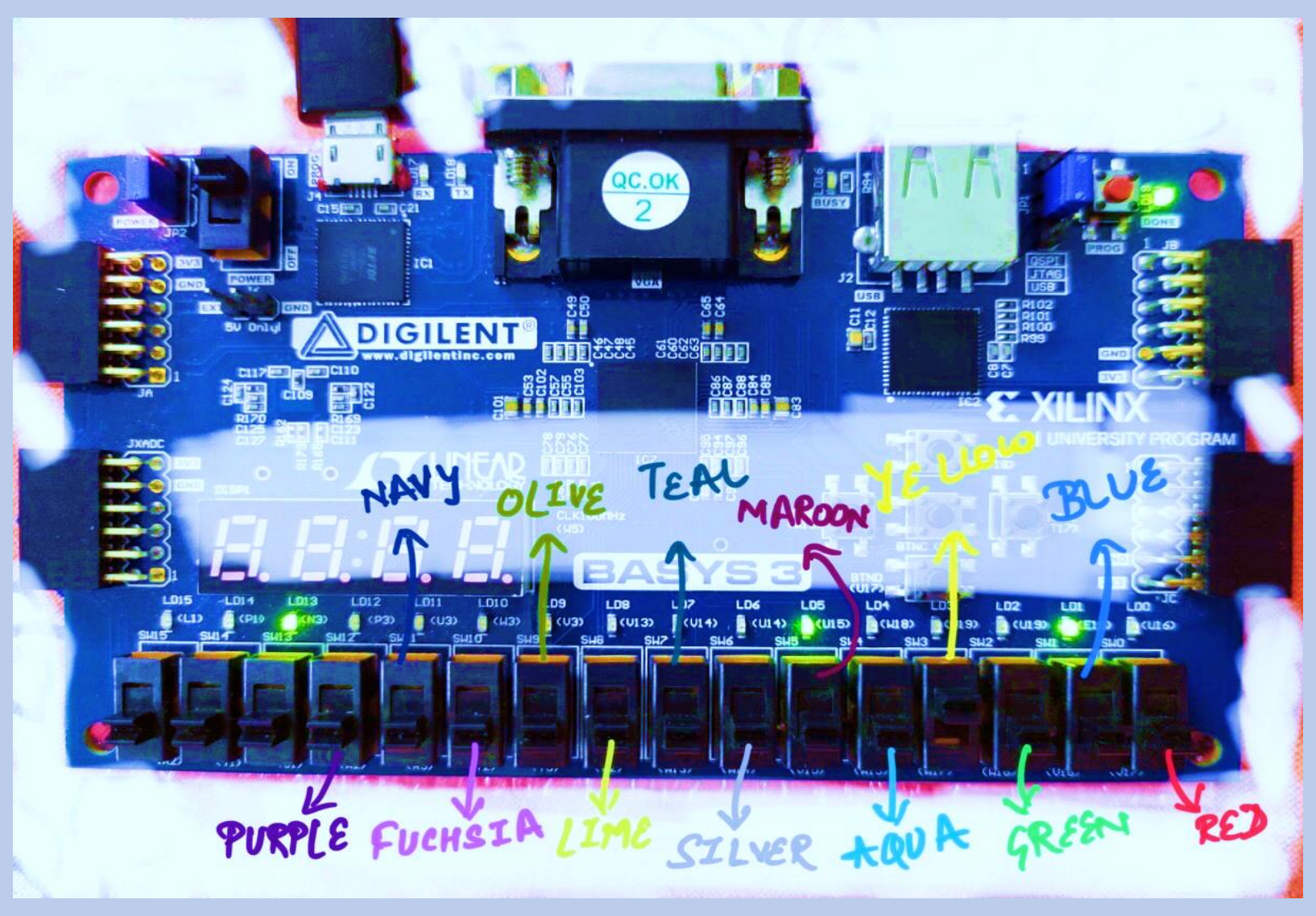
In this project, we built a classifier in Verilog which is able count the occurrences of circles based on their colours. Also, we are classifying the circles based on their sizes by determining their radius.

The general pipeline of our implementation involves pre-processing an image file to its COE format and then loading this into the block ram of the FPGA.

After running the implementation on BASYS3, it is expected that the output LEDs show the count of the occurrence based on their colours along with separate outputs which tell the number of circles belonging to the small, medium and large range. The small range is from 0 to 15 pixels, medium range comprise of 16 to 30 pixels and large range has circles greater than 30 pixels radius.

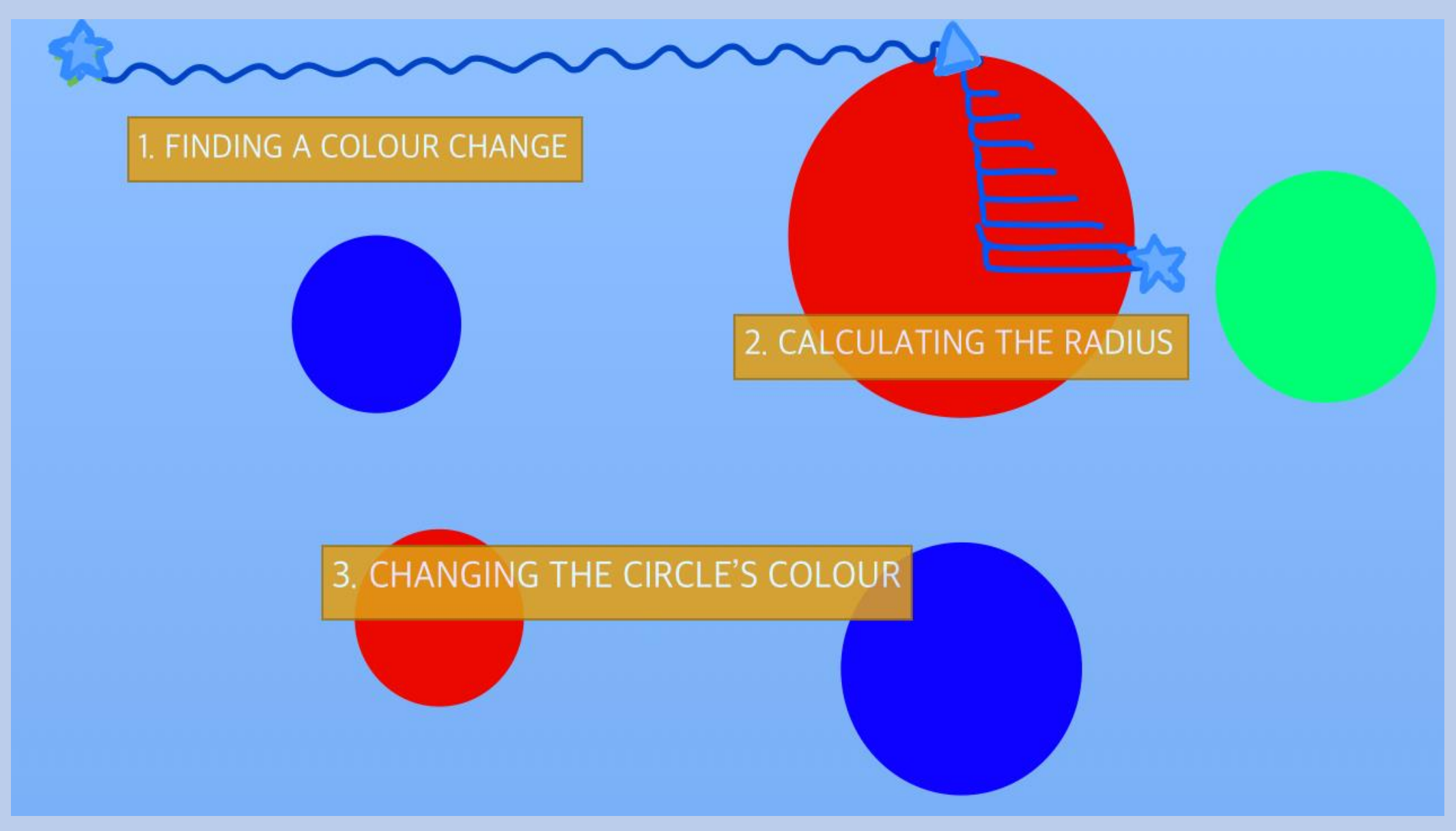
The Basys3 board with its input configured can be seen in the following image. Here, we were able to classify 13 different colors.
The output are mapped accordingly and are visible in the adjacent image.





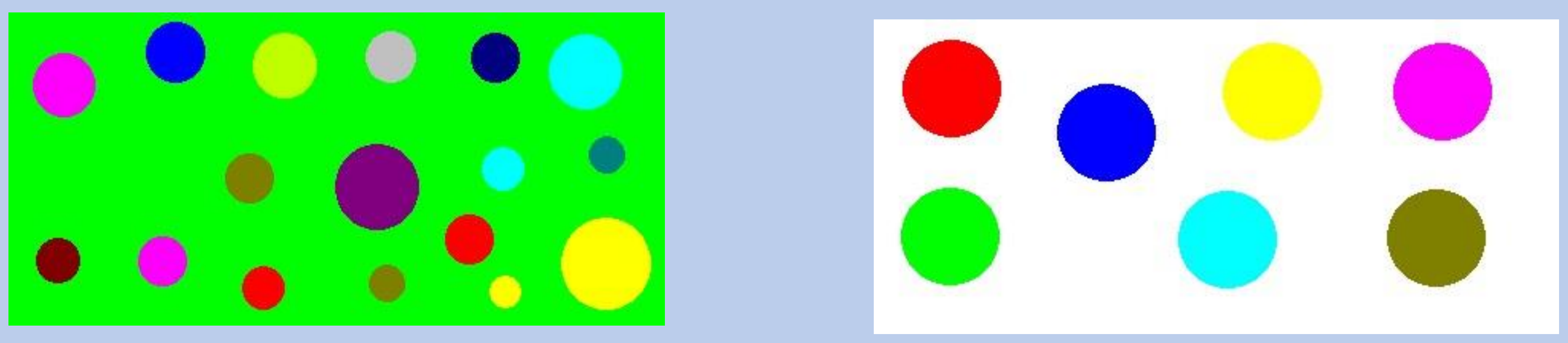
Algorithm and Implementation

- First, the image is converted into its coefficients file (.COE) using python. During this step, the 24-bit true color coding is changed to 8 bit color.
- The coefficients file of the image is then loaded into the Block RAM on the FPGA in continuous memory locations. A single port BRAM is used for this purpose.
- Now, for counting the circles, the image data is traversed row wise to find the first pixel which has a color that is different from the background color of the image. This pixel is the top-most point of some circle. The count of that color is increased by one.
- For detection of radius, we go one pixel down one pixel right at each step until we get a pixel that is different from the color of the top-most pixel of the circle.
- The distance travelled downwards from the top-most is taken as the radius of the circle.

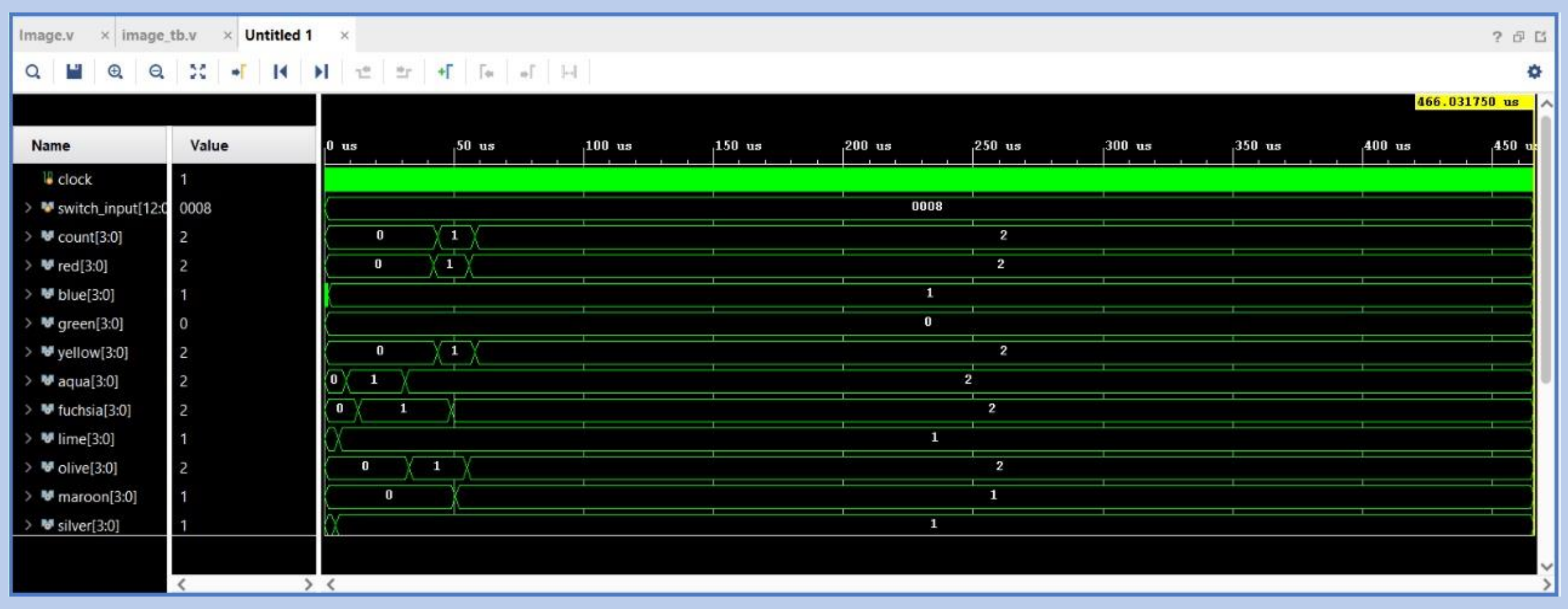

1. FINDING A COLOUR CHANGE
2. CALCULATING THE RADIUS
3. CHANGING THE CIRCLE'S COLOUR

- The count of the appropriate size for the color is increased by one.
- Now that we have detected the circle and we don't want to detect it again. Henceforth, we traverse back to the top-most pixel and start coloring the square whose mid-point of top edge is the top-most point of the circle to the background color.
- Appropriate tolerances are given because the pixelated image does not have a sharp, single pixel top.
- This process continues until the traversing pointer reaches the last pixel of the image.

Results



Few of the images used for testing the Algorithm



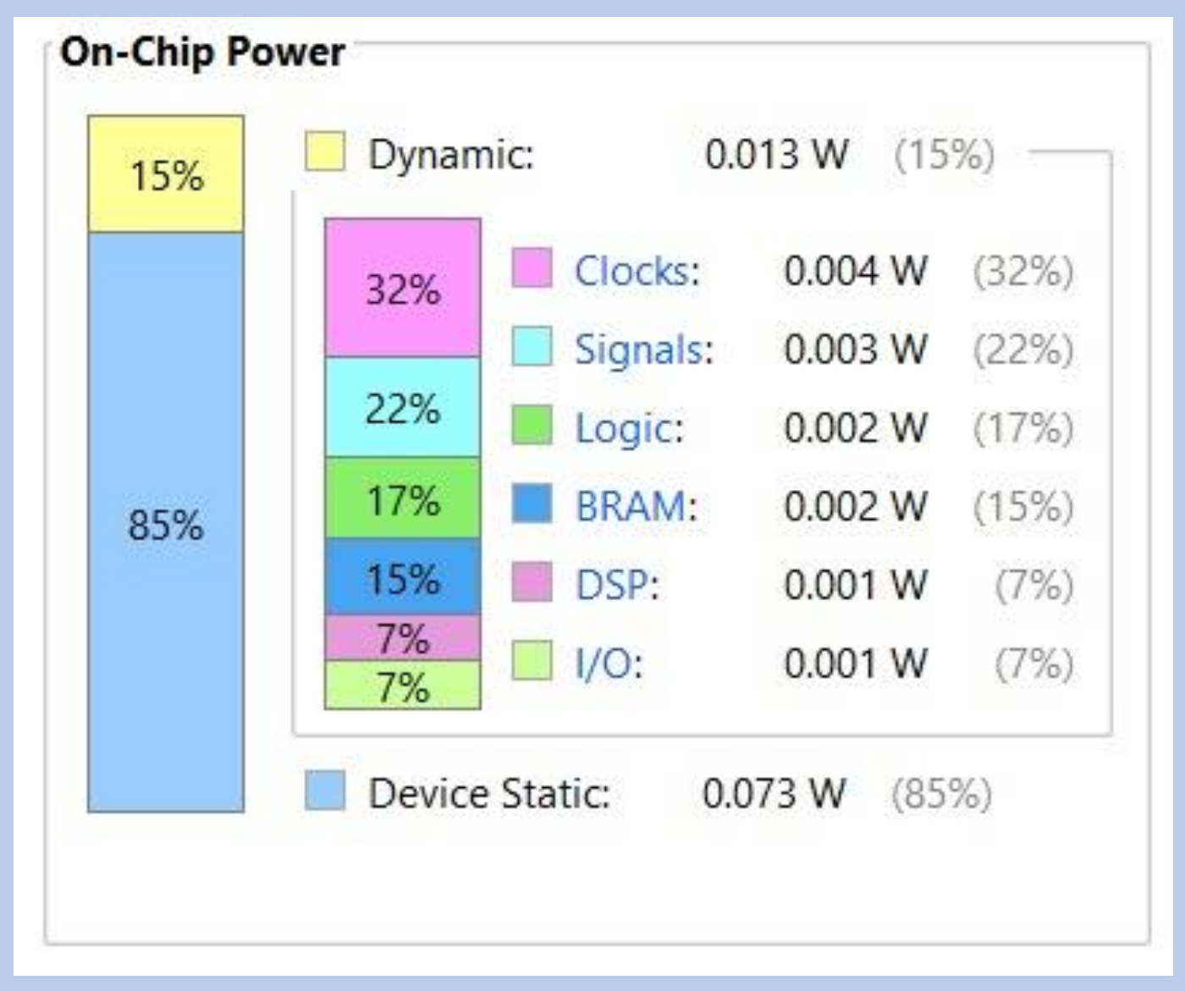
The above waveform shows the output for the first image (green background). The implementation counts the occurrences of circles of different colors. It also accurately classifies circles of each color into three categories of sizes (small, medium and large).

For the first image (with green background) the number of LUTs used was 1194.

The algorithm is implemented in such a way that no pixel of the image is visited more than three times. This is true even for the worst case.

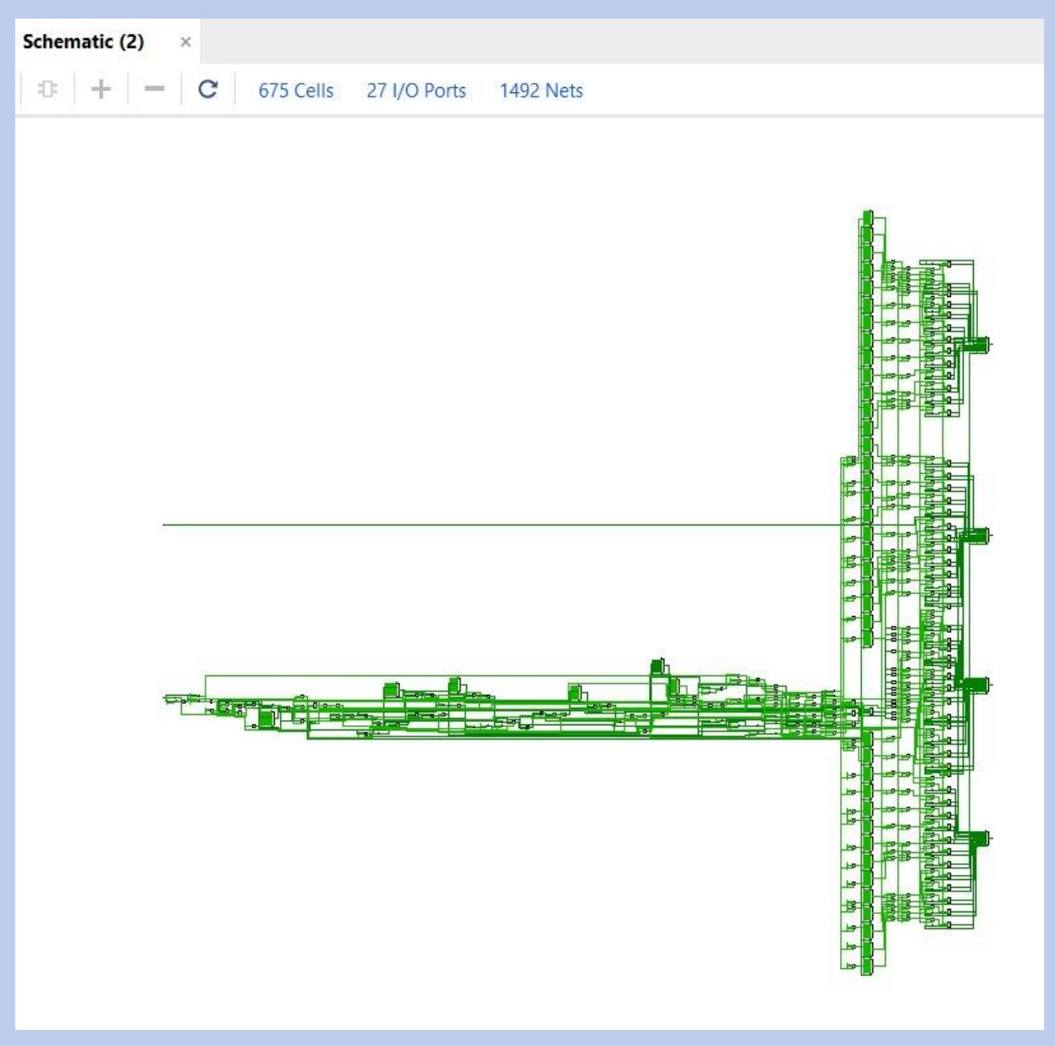
Time Complexity: ***O(bh)***

Where b is the breadth of the image and h is its height



On-Chip Power

Dynamic:	0.013 W	(15%)
Clocks:	0.004 W	(32%)
Signals:	0.003 W	(22%)
Logic:	0.002 W	(17%)
BRAM:	0.002 W	(15%)
DSP:	0.001 W	(7%)
I/O:	0.001 W	(7%)
Device Static:	0.073 W	(85%)



Conclusion

The algorithm that we used for color detection on FPGA was also implemented using Python and the same image was given as input.

Time taken by Python:	0.5 seconds (maximum)
Time taken by FPGA implementation:	0.065 millisecond

We conclude that FPGA implementation is much faster than Python because of the dedicated hardware.

The algorithm used for shape detection, the conventional Hough Transform runs on ***O(n⁴)*** time complexity.
The algorithm that we implemented to detect the location and radius of circle and then count the number of circles of a color runs in ***O(n²)*** time complexity. The indigenous algorithms has its limitations though,

References

- https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5378379
- https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_3/pg058-blk-mem-gen.pdf
- https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/

Scan to check our experiment video