Embedded Systems
CSEE W4840

Design Document

Hardware implementation of connected component labelling

Avinash Nair ASN2129
Jerry Barona JAB2397
Manushree Gangwar MG3631

Spring 2016

Table of Contents

Preface

This document describes the components and procedures that constitute of the project. It mainly consists of two parts involving the two interacting interfaces (Linux and FPGA). The entities and algorithms within each interfaces are also described. For deeper understanding of the procedures, flowcharts have been added.

Linux

Image Storage

The images that would be used for connected component identification will be captured beforehand and will be stored in the SoCkit board memory. The images would be read from the files with the help of OpenCV library functions. Each image is then processed to identify and label connected components.

Main Program

This program will be responsible for reading the images stored in the memory, dividing the pixel data into blocks of size MxN, transmitting the pixel information in each of the blocks to the FPGA for processing, retrieving the processed pixel data and recombining it into a single image.

Preconditioning data for Image Processing

Padding

The first stage of processing involves padding the image with a single pixel wide border. This is done in order to make the algorithm on the FPGA simpler by not having to handle edge cases.

Block Processing

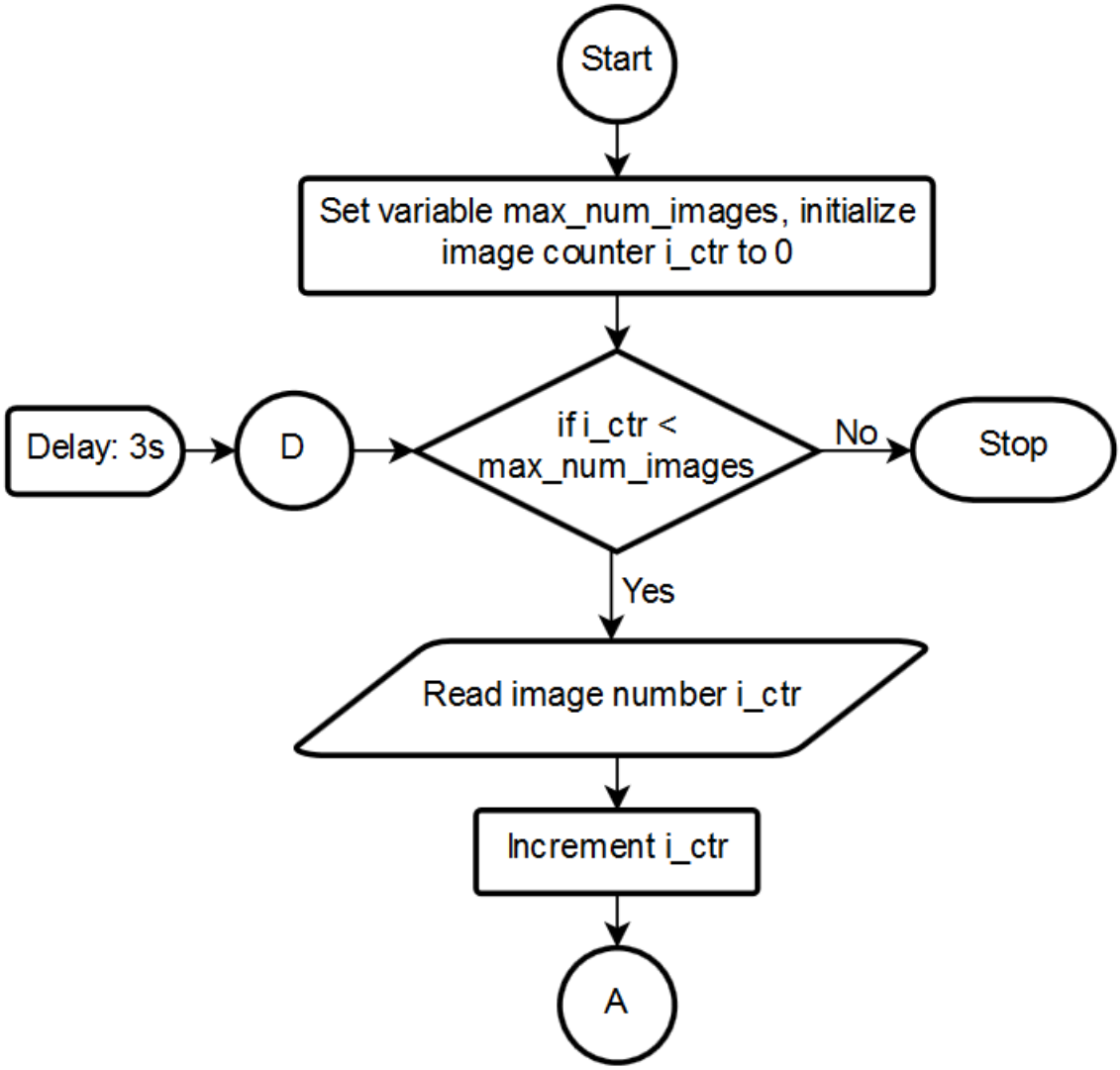
We do not have enough memory on the FPGA to process the entire image in a single shot. Instead, we need to divide the input image into overlapping blocks of size B. The overlap between adjacent blocks would be at-most 1 row and 1 column. The overlapping allows us to avoid dealing with boundary cases separately and reduces the number of duplicate (equivalent) labels that get generated.

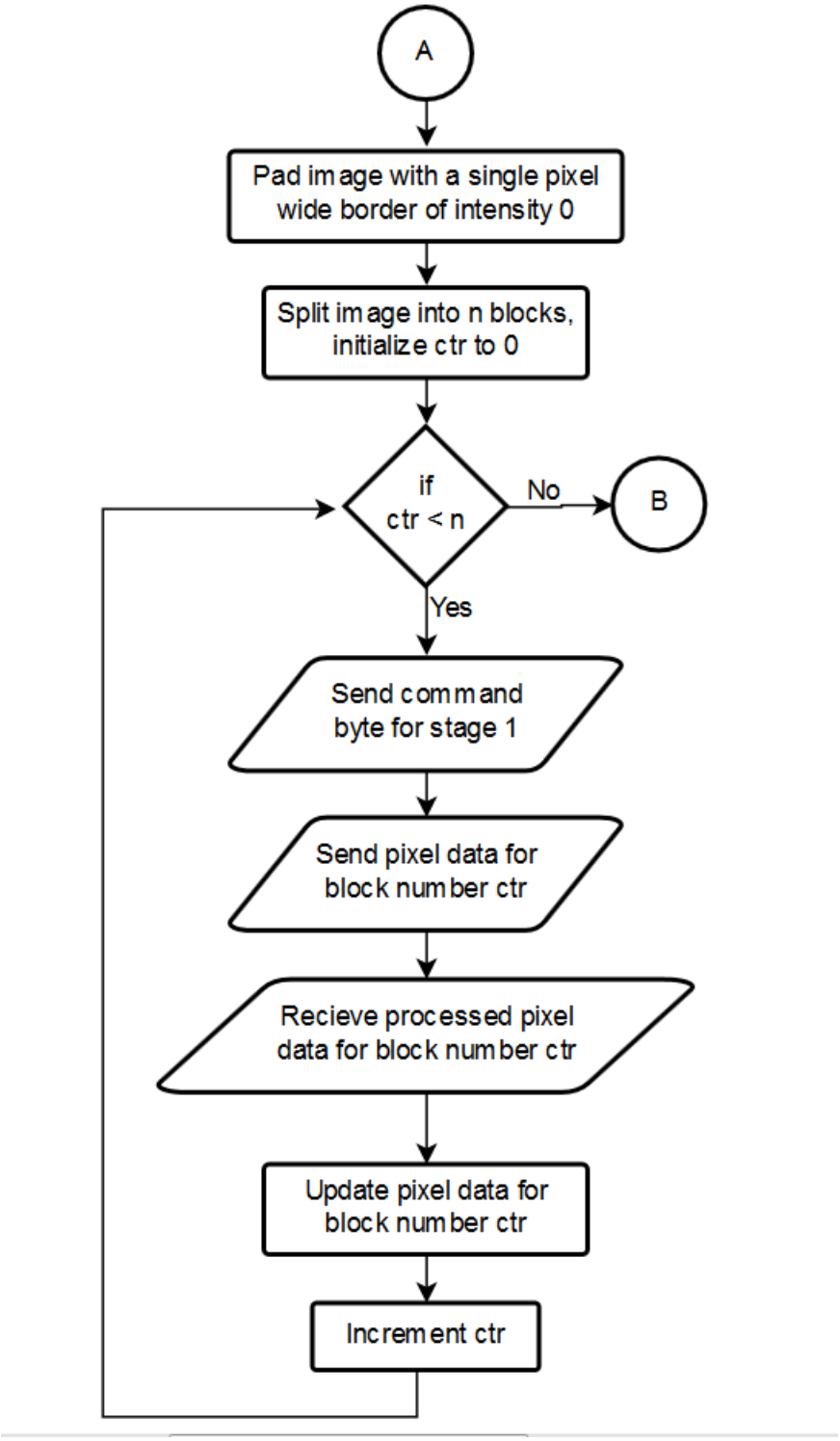
Each block of size B then needs to be sent to the FPGA for processing. The processing happens in three stages. In the first stage, we do a raster scan of each block, threshold each grayscale pixel and assign it a label if it's a foreground pixel. If the algorithm detects an equivalence between two labels it stores this information in a LUT. The second stage, we do another raster scan of the block to resolve any ambiguities in labeling with the help of the LUT created in the first stage. The third stage, is used to display the results on a monitor.

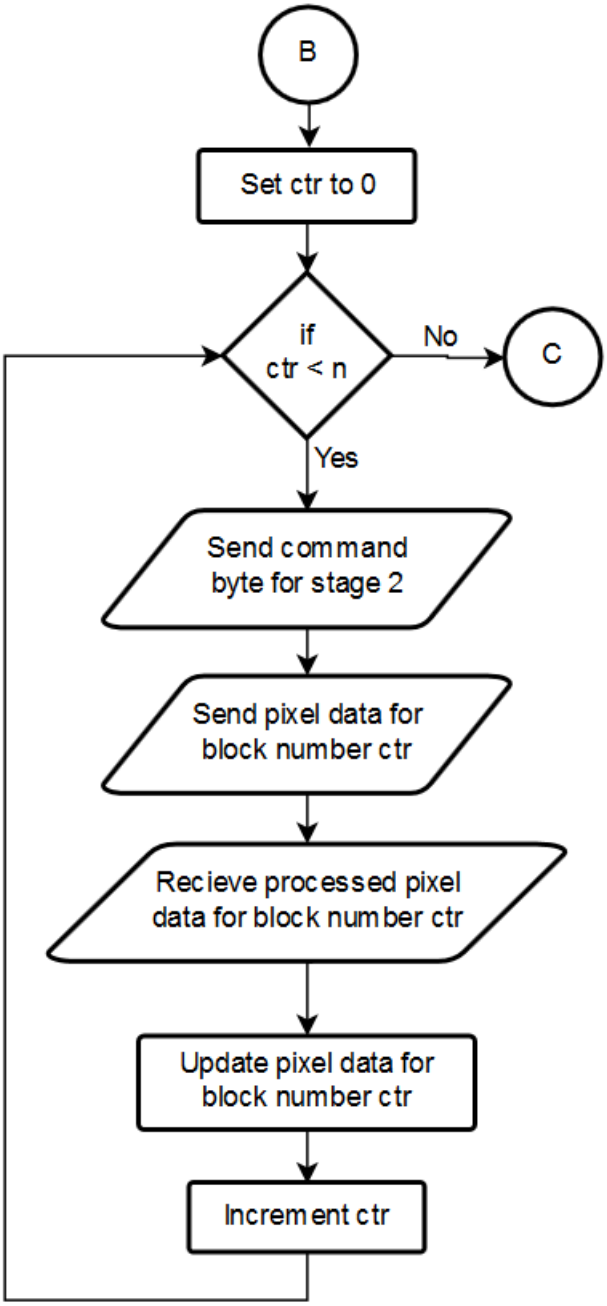
The transfer of pixel data corresponding to each block happens in two steps. We first send a command bit to the FPGA, through the device driver, to tell it which stage of processing needs to be done on the data that it receives. This is then followed by the pixel data of the block that needs to be processed.

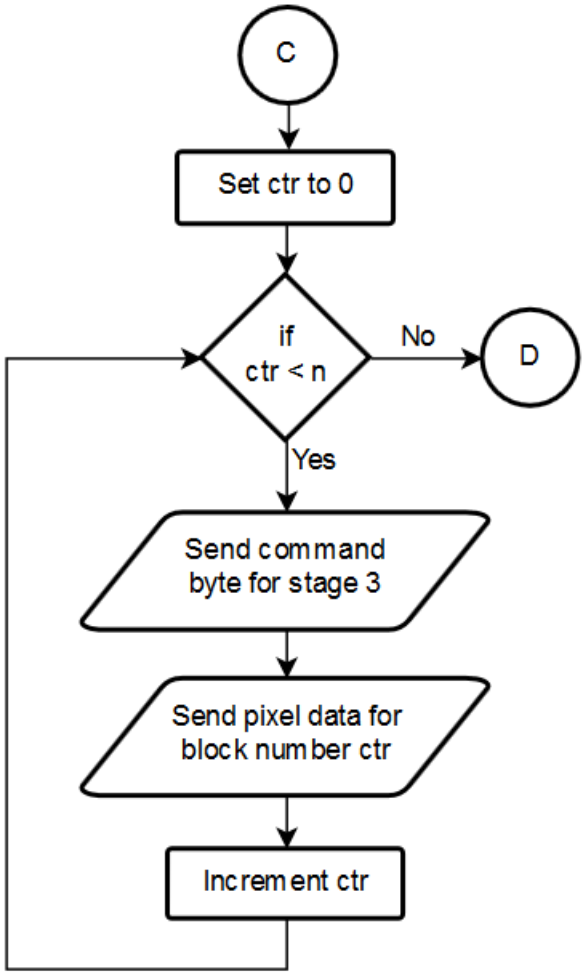
Once the data is transferred, the program waits till it receives the processed data. The original image's pixel values are replaced with the processed pixel values before sending the next block of data.

The following is the flowchart of the program:


Start
Set variable max_num_images, initialize image counter i_ctr to 0
if i_ctr < max_num_images
No
Stop
Yes
Delay: 3s
D
Read image number i_ctr
Increment i_ctr
A


A
Pad image with a single pixel wide border of intensity 0
Split image into n blocks, initialize ctr to 0
if ctr < n
No
B
Yes
Send command byte for stage 1
Send pixel data for block number ctr
Recieve processed pixel data for block number ctr
Update pixel data for block number ctr
Increment ctr


B
Set ctr to 0
if
ctr < n
No
C
Yes
Send command
byte for stage 2
Send pixel data for
block number ctr
Recieve processed pixel
data for block number ctr
Update pixel data for
block number ctr
Increment ctr



Driver implementation

A device driver compiled into the kernel will be implemented. The driver will allow the copying to and from the user interface (C code that manages the blocks of pixel information split from the original input image) and the kernel space, which subsequently will be copied or read to a memory location in the FPGA, implemented by RAM.

FPGA

RAM implementation

In order to store the pixel information in the FPGA memory location which the FPGA will process, a RAM will be implemented. Each memory address will hold a byte-size memory

location. The total RAM capacity will be "B bytes" equivalent to the size of a single block of pixels plus one control byte which will be transmitted at the beginning of each block from the C code. Therefore, every block pixel will be stored in individual addresses each.

Control Byte and States

Because the size of the input image will largely exceed the memory capacity of the FPGA, the input image will be split in blocks of identical size which will be processed one at a time by the system. After a block is processed, the resulting block will be sent back to the C code interface before the next block is read to process.

There are different types of processes that will be executed on each block each time. The control byte is meant to inform the system what type of operation it must perform on each pixel block read from the driver. The operations are designated as "states" in which the system will remain until the control byte from the next block is read. There will be 3 major states that will be included in the control byte and that the system will handle:

- The first looping through the block which will label each pixel either as "foreground" or "background" upon comparison to a previously established threshold.
- After the labeling is completed for all blocks, the blocks will be looped through again to unify the labels upon the detection of connected components.
- In the third state, the system will display the blocks on a visual peripheral.

Connected component labelling on FPGA

Connected component labelling assigns each connected component with a unique label. The following steps are performed in labelling an image block:

1. Chose a threshold T to convert the input grayscale image into a binary image.

2. Compare the value of each pixel with T.
    a. If I(x,y) > T then I(x,y)=1, foreground pixel
    b. Else I(x,y)=0, background pixel
3. Define a 3-pixel neighborhood.

N2	N3
N1	P

4. Scan the image in blocks of size MxN.
5. In the first scan, in case of foreground pixel / I(x,y) = 1:
    a. If none of its neighbors are labelled, then assign a new label to P.
    b. If only one of its neighbor has a label, then assign the same label to P.
    c. If two or all neighbors have the same label, then assign the same label to P.
    d. If neighborhood pixels have different values, then copy the value in N1 to P and create an entry in the equivalence table for labels in neighborhood pixels.
6. Repeat step 5 for all foreground pixels.
7. In the second scan, find the smallest label for each set of equivalences and replace each label in the image with the corresponding smallest label.

First scan

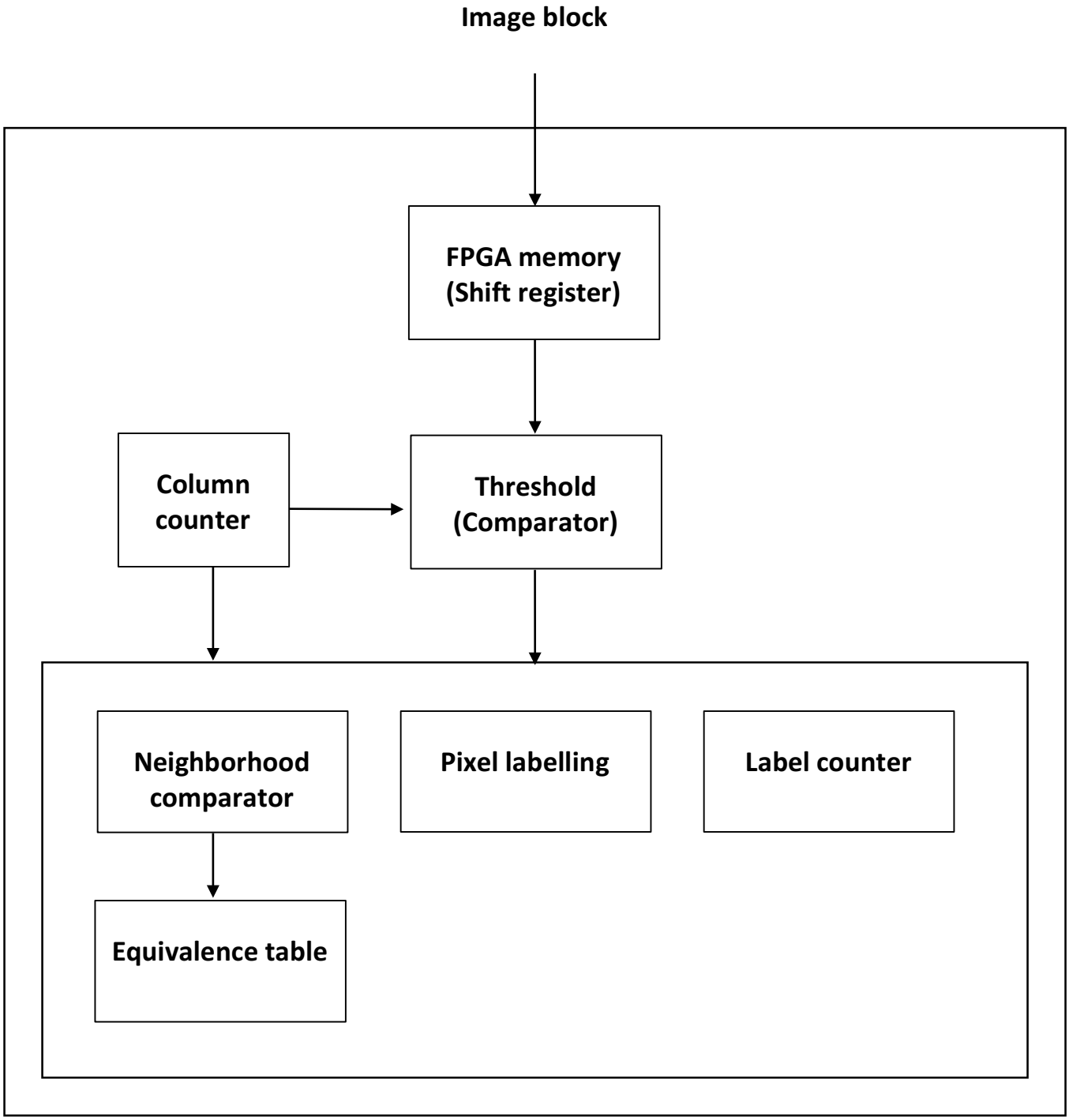
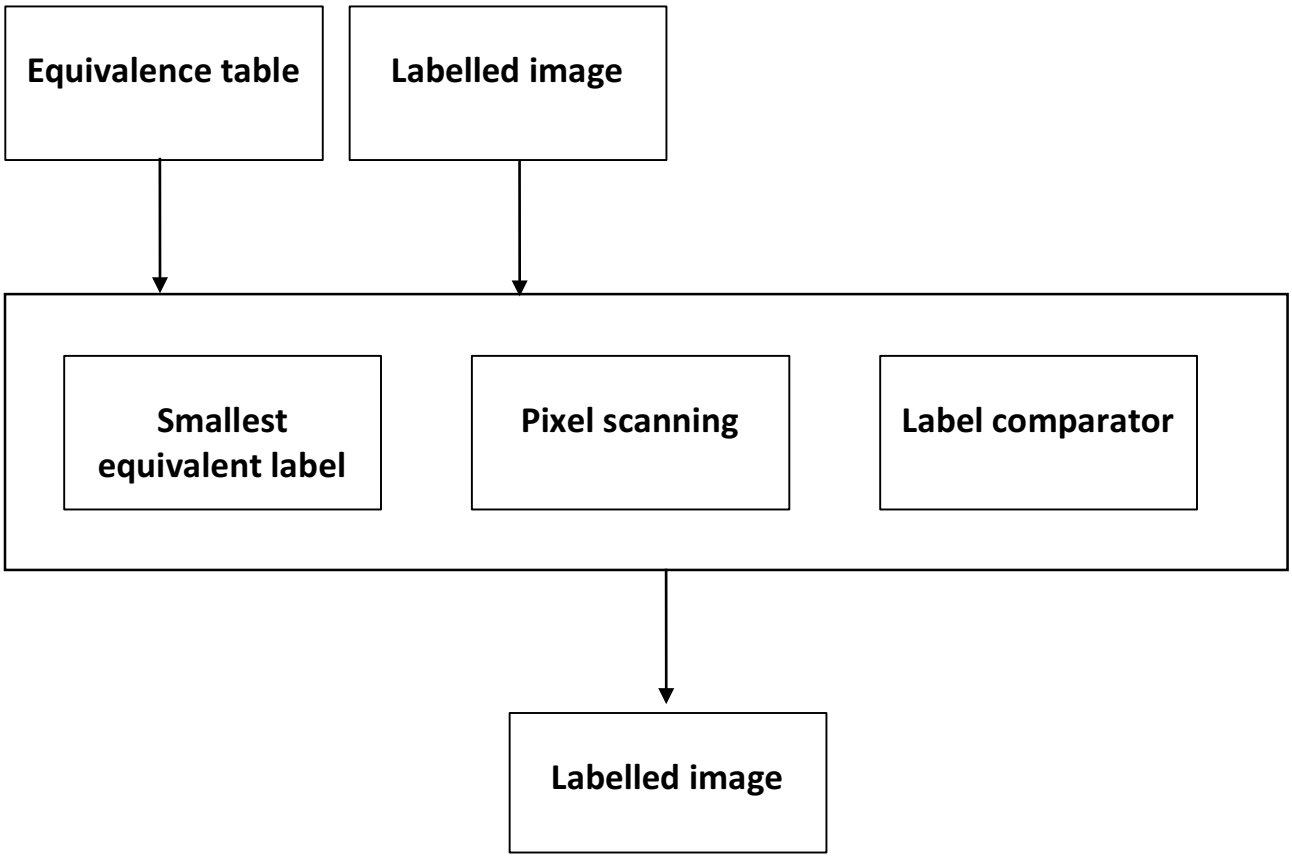


Image block
FPGA memory
(Shift register)
Column
counter
Threshold
(Comparator)
Neighborhood
comparator
Pixel labelling
Label counter
Equivalence table

Second scan


Equivalence table
Labelled image
Smallest equivalent label
Pixel scanning
Label comparator
Labelled image