# The Design Document for CSEE 4840 Embedded System Design

Fatima Dantsoho (fd2508) & Michael  Lanzano (mml2238)

CameraControl

Spring 2022

## System Overview

In this application, real time video image processing on the FPGA, scene irradiance data collected by an OV7670 CMOS will be streamed to the FPGA target board where it will be processed by convolution filters and finally output as a signal to a VGA display. The software will generate parameters for the convolution kernels. The user will interact with software with the keyboard. Avalon buses will be used for transferring video streams from the camera and to the VGA display. Both filtered and unfiltered video will be shown on screen. Filter settings will be displayed below.

The system will perform 3 main operations:

1. Interface with camera
2. Apply a filter to the video stream
3. Display the video on VGA output

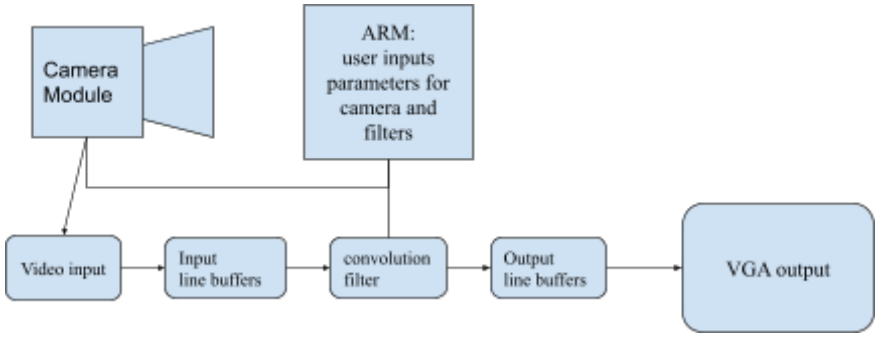


Figure 1: System Block Diagram

## Interface with camera

1. OV7670 image sensor
2. SCCB camera interface
3. 8 bit data bus connected via GPIO
4. We will work in YUV 4:2:2 color space. This format provides 16 bits per pixel with full resolution luminance (y) and half resolution color (u, v).

## The Filter Kernel

Because convolution is the sum of n x n products in the neighborhood of a pixel, a filter whose kernel is n x n pixels must have n rows of data to operate on. For a VGA image in YUV, a single row will take 16 bits * 640 pixels/row = 10,240 bits. This is slightly larger than a single

block of M10K memory so we will need to look out for potential address misalignment issues. Should this become an issue, one possible solution would be to store only half a row per M10K block. The cyclone V chip on our DE1 appears to have 397 blocks of embedded memory, so even for a large 15x15 filter kernel we would only be using 10% of the resource if we used 2 blocks per row.

A filter kernel is an n x n array of coefficients that are used to calculate the sum of products in the neighborhood of the target pixel. A very common and useful type of image filter is the gaussian. A 2D gaussian is an n x n matrix whose elements take the value of the gaussian probability distribution as a function of its distance from the center of the kernel. Because the values represent probabilities, all values are in the range from 0 - 1 and sum to 1. Calculated directly, the values of these coefficients are floating point values, which poses an issue for hardware implementation. Fortunately fixed point approximations come close to the true value as the bits used in the fixed point approximation and kernel size increase. The way we calculate a fixed point approximation of a floating point number is as follows: First we select an appropriate resolution in fractional base 2 units. For example, if our floating point number is ⅓ and our resolution is in 1/256 our approximation of ⅓ is $43*2^{-8}$. In this case we do our sum of products using 16 bit integers and then right shift the 16 bit result by 8 bits and store the 8 LSBs in our 8 bit target register.

We would like to implement a generalized image filter with kernel size, and bit depth set in hardware and have the coefficients determined in software and passed to hardware with ioctl calls to be stored in memory. 8 signed bits should be sufficient for each. So if we take 15 x 15 as the upper bound for kernel size, that would require just over 1k bits of embedded memory.


Figure 2: Effects of a Sobel filter (3x3 kernel size)

## Convolution Process

Before we perform our n x n convolution we must wait until n rows of pixel data have been stored in our M10K line buffers. This will create a base latency of 8 x 2 x 640 x n cycles.

We declare an n x n block of 16 bit registers to hold n x n pixel values copied from our line buffers. This is facilitated by a pointer to each buffer. Each clock cycle values in the block register are passed to the left and the right column of the register takes values from the line buffer 1 and 2. The lower right value is the new pixel value from the camera module.

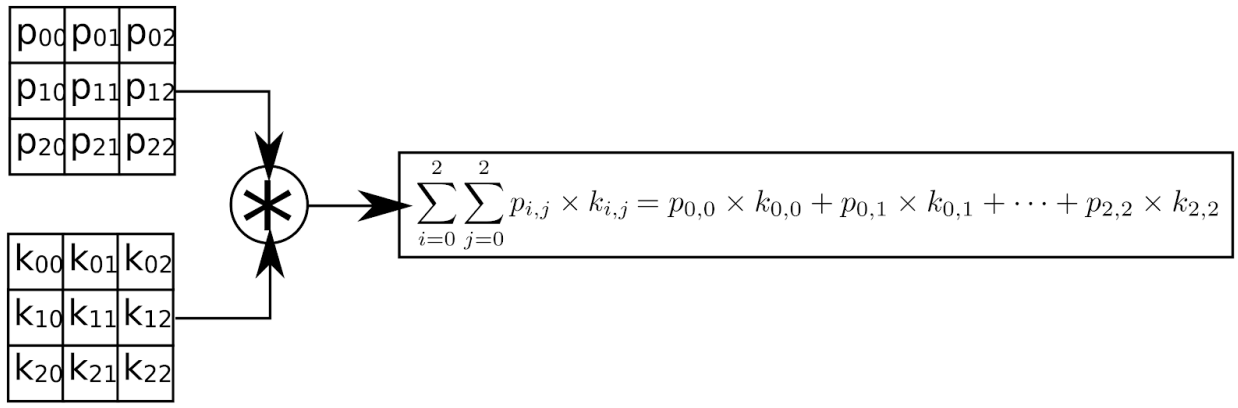


Figure 3: Convolution operation for a 3x3 kernel

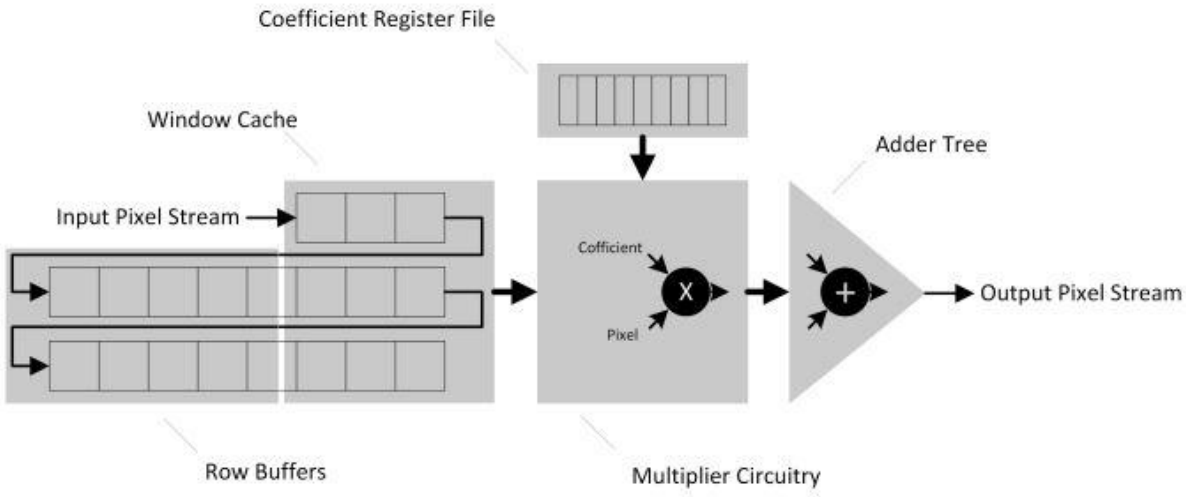


Figure 4: Image filter block diagram

Once our block register is populated with pixel data we perform the convolution operation on the pixel corresponding to the center of the kernel by taking the sum of products of luminance (y) values for each of the n x n pixels in the neighborhood with the corresponding fixed point values in our kernel register. For this job we will either leverage the on board DSPs (asking the megawizard for a multiply accumulated resource) or create a multiplier module to execute in parallel. Next we add the nxn products. We will implement a cascading architecture that will take $\log_2(n)$ +1 cycles to pass the column data through the multiplier and the first stage adders. The sum of the column sums will be computed in the second stage adders after the block register has advanced to the right in parallel with the column calculations (first stage adders) for the next pixel. Convolution should take $\log_2(n)$ + 1 cycles to complete with additional latency of $\log_2(n)$.

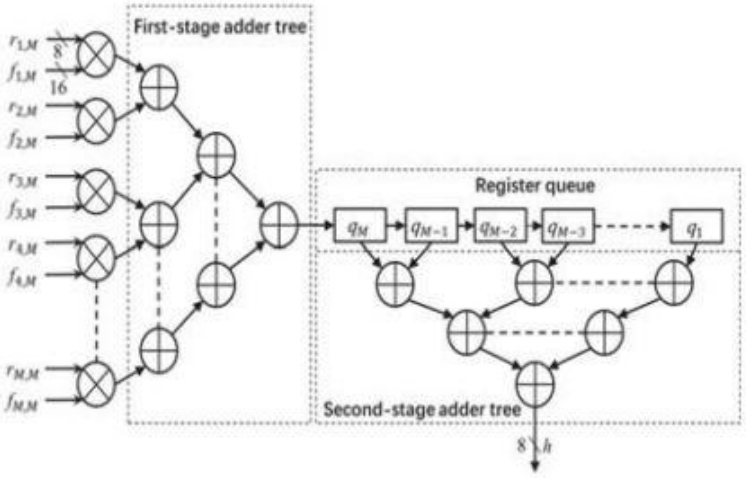


Figure 5: Convolution module.
Multiplication of data and coefficients are shown for *M*th column

## VGA Output

In our labs we output to VGA by writing RGB data to a framebuffer that corresponded to pixel locations on screen. In our application we have avoided writing an entire frame to memory to reduce memory overhead and time delays associated with reading and writing to SDRAM. We would like to try to use line buffers for VGA output to keep output delays to a minimum but it may be more practical to store the entire image to a frame buffer. In any case we will need to convert YUV to RGB prior to VGA output. Alternatively we could use the luminance value (y) for R,G & B and output a filtered gray scale image to screen. This may be a sensible option as edge detection is understandable in grayscale. Furthermore we would be able to store a single channel of our image in M10K memory but not full RGB. 8 bits * 480 cols * 640 rows = 2.5 Mb. The DE1-SoC has 3.9 Mb of M10K.

## Software

Software will start and stop the camera and control image capture parameters such as exposure and color balance. Software will also provide filter kernel parameters to the convolution module. The software layer will consist of a device driver for the FPGA camera top module and an executable user space program.

## Milestones

Milestone 1 (April 10th):

Output raw camera data to a VGA display

Milestone 2 (April 20th):

Implementation of convolution module

Milestone 3 (April 30th):

A. Finish software application for user interface
B. Testing and Debugging

## References


- https://medium.datadriveninvestor.com/understanding-edge-detection-sobel-operator-2aada303b900
- https://community.element14.com/technologies/fpga-group/b/blog/posts/gradient-filter-implementation-on-fpga-part-2-implementing-gradient-filter
- Efficient FPGA Implementation of Automatic Nuclei Detection in Histopathology Images
- High Throughput 2D Spatial Image Filters on FPGAs