

EEL 4930 - System-on-Chip Design

Spring 2023

Homework 1

Submitted By

Mohammad Bin Monjil (50782093)

Shuvagata Saha (58898380)

Hasan Al Shaikh (34715621)

Date: April 16, 2023

Task 1:

We used the Pentalinux tool by Xilinx to build an OS image which can be run on the Zedboard SoC. We boot the built image on the QEMU emulator. We run the “Hello World” application which was named myapp. We modified the image to load this application and then we boot the image on SoC to run the application.

```
of_cfs_init: OK
ALSA device list:
  No soundcards found.
Freeing unused kernel memory: 1024K
Run /init as init process
INIT: version 2.97 booting
Starting udev
udevd[66]: starting version 3.2.9
random: udevd: uninitialized urandom read (16 bytes read)
random: udevd: uninitialized urandom read (16 bytes read)
random: udevd: uninitialized urandom read (16 bytes read)
udevd[67]: starting eudev-3.2.9
urandom_read: 1 callbacks suppressed
random: udevd: uninitialized urandom read (16 bytes read)
random: dd: uninitialized urandom read (512 bytes read)
rcu: INFO: rcu_preempt detected stalls on CPUs/tasks:
rcu: 1-....: (1 GPs behind) idle=086/1/0x40000000 softirq=1606/1607 fqs=56
          (detected by 0, t=6043 jiffies, g=-87, q=3)
random: crng init done
```

Figure 1: Challenge in booting the linux system in Qemu.

We faced a challenge while booting the image. The synchronization mechanism rcu detected stalls on CPU when the VM was using more than 1 processor. We solved this problem by assigning only a single processor to the VM.

Task 2:

We completed Task 2 following the provided high-level block diagram with a input module to read image and send to processing module through a 24 bit int channel, processing module to perform median filtering/edge detection and send the processed image through another 24bit int channel to the output module. The output module is used to retrieve the image.

In our implementation, the processing module first collects all the pixels from the channel, converts the pixel values into a cv:: Mat object, performs median/sobel on the Mat object using opencv functions. The processed image is then converted back into integer values and sent through the channel to output module. The output module then converts the ineteger values back into the Mat array object. There are several signals connected between the module (other than the channels) to synchronize the data transfer between them. Figure 1 and 2 shows the resulting processed image on a sample image.



Figure 2: Sobel edge detection

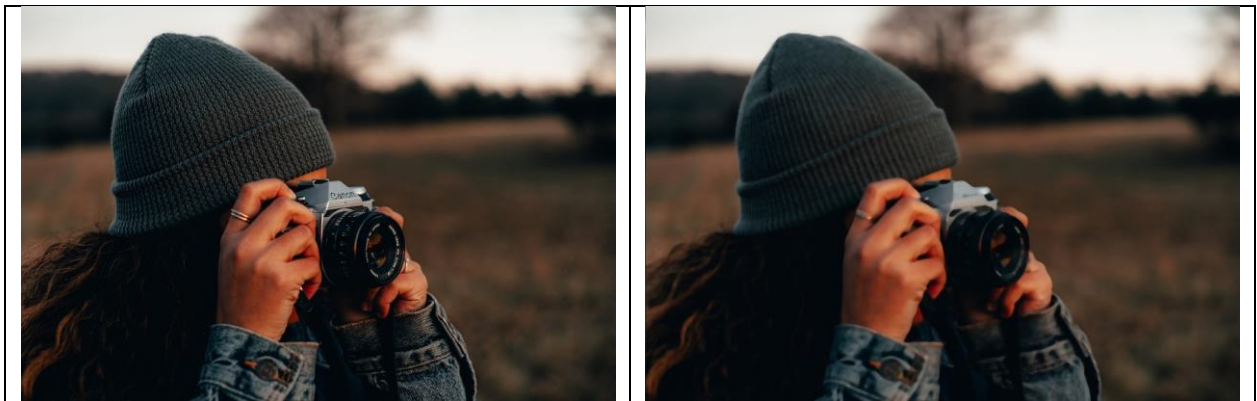


Figure 3: Median Filtering using a kernel size of 7

Challenges:

For the sake of simplicity we initially implemented the above version. The execution time of this implementation is on the higher side. Later, we attempted to implement the project using FIFO concepts to reduce the runtime. The main change was in the processing module. Our pseudocode for the thread of the processing module is given in appendix. Unfortunately, we could not finish debugging the code.

Submissions:

Task 1: Bitstream and .xsa file, linux images and Petalinux project files, video.

Task 2: C++ project files, binaries, readme.txt

Contribution of members:

Mohammad Bin Monjil – Processing Module, Top-level

Shuvagata Saha – Task 1

Hasan Al Shaikh – Input & output module, report

Appendix

Pseudocode of the thread of processing module with FIFO :

```

while(true) {
    if (!done in)
        { fifo.write(pixelin.read()) }
        { if (fifo.full)
            { start_processing = true } }
    if (start_processing)
        { if (first_time)
            { for (i = col+1; i ≥ 0; --i)
                { buffer[i] = fifo.read() }
                first_time = false }
            else { buffer[0] = fifo.read() } }
        mid_point = (col+1)
        for (i = 0; i < 3; ++i)
            for (j = 0; j < 3; ++j)
                { index = mid_point + (i-1)*col + (j-1)
                    if (index < 0 or index > 2*(col+1)-1)
                        window[i,j] = 0
                    else
                        window[i,j] = buffer[index] }
                filtered_window = filter(window)
                send2output(filtered_window, current_pixel)
                current_pixel++
                for (i = 0; i < 2*(col+1)-1; ++i)
                    { temp2 = buffer[i+1]
                        if (i = 0)
                            { buffer[i+1] = buffer[i] }
                        else
                            { buffer[i+1] = temp2 }
                        temp1 = temp2
                    }
                wait() }

```

Reads from channel and stores in the fifo.

buffer = int array [2x(col+1)]
if first time then we insert values from fifo to mid position of the buffer to the beginning. otherwise, values are inserted at the beginning of the array

Collect values from buffer and store in a 3x3 window

// sobel or median
// send to output module
// current pixel number

shift the buffer array by 1 position right