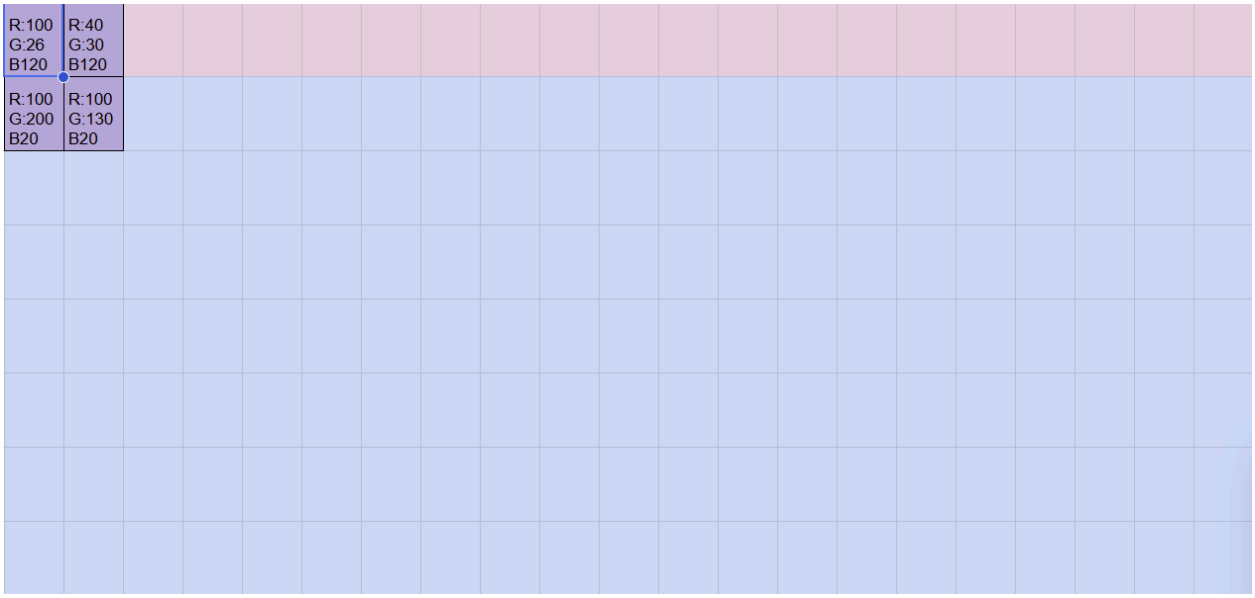
Median Pixel Filter

The median pixel filter applies a 2×2 kernel kernel to an image of IMAGE_LEN and IMAGE_HEIGHT. Since a 2×2 kernel covers an even number of pixels (4 total), there is no single middle number. The function typically calculates the median by **sorting** the 4 pixel values and taking the average of the two middle values. This is done for each of the 3 pixel channels (red, green, blue) separately. The resulting median pixel channels should then combined back into a pixel be output.

The convolution will start in the top left corner of the image. It then move one pixel to the right until it reaches the right edge of the image. The filter will then drop down a row and move all the way to the left edge. This will continue until the filter passes over every pixel in the image.

First Convolution Example



The diagram above shows the state of the module when it can do its first convolution. The module has received the entire top row of data and the two left most pixels in the second from top row.

The convolution operation will happen as follows:

1. Split the extract each of the pixel's color channels
    a. Red: 100, 40, 100, 100
    b. Green: 26, 30, 200, 130
    c. Blue: 120, 120, 21, 20
2. Find the median value of each channel
    a. Red: 100
    b. Green: 80
    c. Blue: 70.5 → rounds up to 71
3. Recombine into a single pixel

    Pixel = {Red: 100, Green: 80, Blue: 71}
4. Output new pixel and set pixel_valid_if_o.valid = 1

Module I/O

The module has following header:

```
module median_filter #(
    parameter int IMAGE_LEN    = 1080,
    parameter int IMAGE_HEIGHT = 720
) (
    input  logic                clk,
    input  logic                rst,
    input  logic                start_i,
           pixel_valid_if.slave  pixel_valid_if_i,
    output logic                 done_o,
           pixel_valid_if.master pixel_valid_if_o
);
```

- clk
  - Posedge clock signal
- rst
  - Posedge synchronous reset signal
- start_i
  - Asserted at least one clock cycle after rst is de-asserted and at least one clock cycle before the module receives valid data
- pixel_valid_if_i
  - pixel
    - Input pixel
  - valid
    - Valid flag. If this is 1, the input pixel should be used. If this is 0, the input pixel should be ignored. This flag can drop to 0 any point during operation
- done_o
  - Asserted for one clock cycle after all pixels have been processes and output
- pixel_valid_if_o
  - pixel
    - Output pixel
  - valid
    - Valid flag. If this is 1, the output pixel should be used. If this is 0, it should be ignored

Other Files

The directory also includes a median_pixel_pkg.sv and pixel_valid_if.sv. When creating your design, you will likely want to add to the median_pixel_pkg.sv. It

might also be a good idea to create other modules and use available modules in the rtl_utils folder to improve the readability of your code. **You shouldn't need to modify pixel_valid_if.sv.**

Other Info/Requirements

- Try to stick to the style guide
- Only write synthesizable System Verilog
- Avoid pasting in code from Chatgpt or another LLM. This will lead to complex code that is difficult to debug
- I will eventually do code reviews on your modules. Please make your own branch on Github and upload your code there
- You will eventually be making testbenches for your code in Python. More on this later