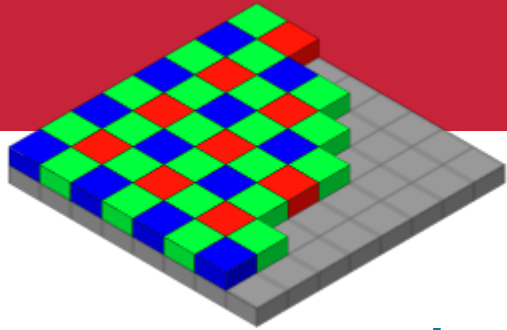


# Demosaic RTL ISP Design



[\*wikipedia]

Dr. Maikon Nascimento

[maikon@ualberta.ca](mailto:maikon@ualberta.ca)

<https://github.com/maikonadams>

<http://scientistengineer.blogspot.com/>

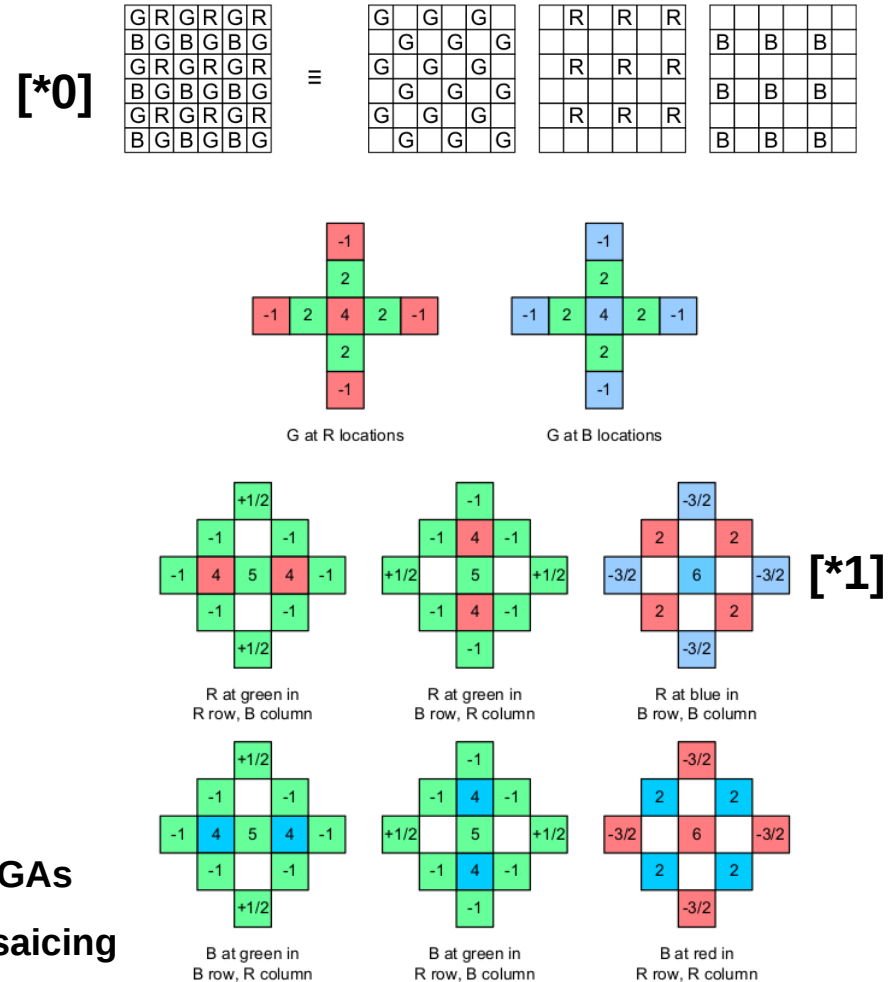
<https://www.linkedin.com/in/maikonadams/>

# Agenda

- Introduction
- Workflow
- Octave Model
- RTL Design
- Results
- Conclusion and Future Work

# Introduction

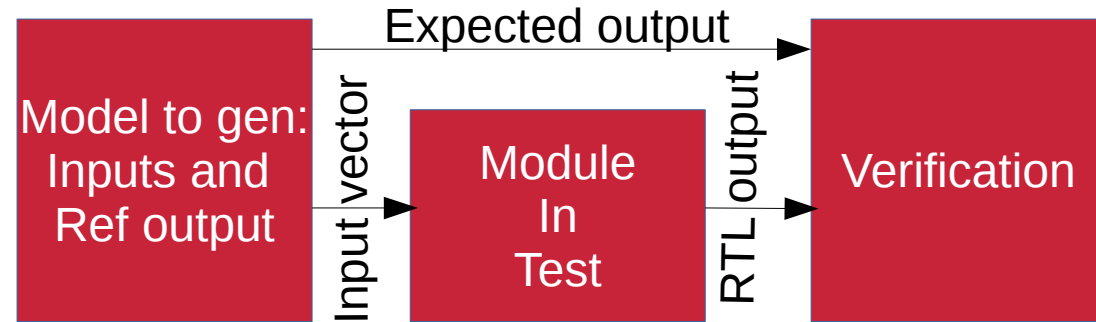
- An imaging sensor with a matrix of pixels/sensors of any common digital camera is actually composed by a pattern of pixels in blocks of 2x2, where 2 diagonally opposed are green and the other two are red and blue. Which resembles a mosaic like illustrates in the picture above.
- The Colored filters extracted from [\*1] are the interpolation filters that uses 2 colors to interpolate and fill up the matrix gaps.
- Those filters will work together with muxes to be selected according to the pixel position and CFA order, the filters also are repeated.



**[\*0]** Donald Bailey, Design for Embedded Image Processing on FPGAs

**[\*1]** Malvar and Cutler, High-Quality Linear Interpolation for Demosaicing of Bayer-Patterned Color Images

# Workflow



- The basic framework is illustrated in the figure where a model is coded in Octave (a free Matlab),
- The Octave model has a true-colour image to evaluate the algorithm and also generate the inputs for the RTL and the expected output that the RTL is compared against.
- True-colour images also can provide a reference for the overall algorithm performance using PSNR to compare. One of the images is shown here.
- In the RTL we then compare the output of the design with the output of the Octave Model which is a hardware friendly version of the algorithm.



# Octave Model

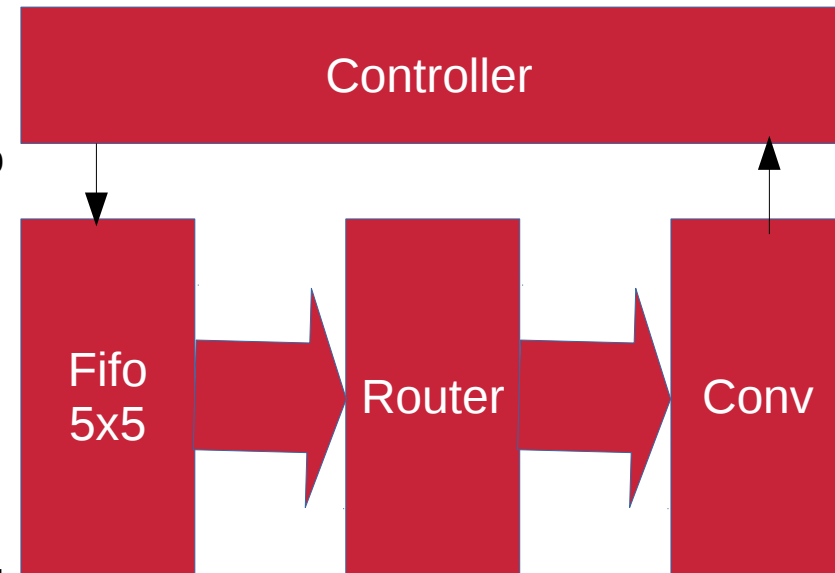
- Octave was chosen because of the similarity of Matlab and all the functions, and packages needed. But Matlab itself could be used, or Python, or any language for quick algorithm coding that also have libraries for writing and reading files.
- The gray image on top is the result where the true-RGB image is converted to a mosaiced Bayer pattern simulating a raw image from the camera and being converted to an input to the RTL design.
- For this algorithm a header with a few parameters were included for quick modification of the testing conditions such as image dimensions and pixel width.
- Also this model will run a hardware friendly version of the Demosaicing algorithm itself with floor operations for truncation of words when re-quantization is needed, also limiting the words size based on the RTL bus size like: uint8 or uint16.
- Another important aspect is to decided what to do with the border which for this first version is padded with 0.
- The code is on github and here I show some snaps.



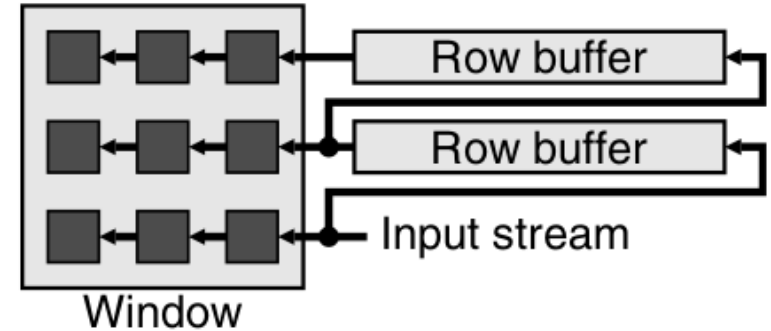
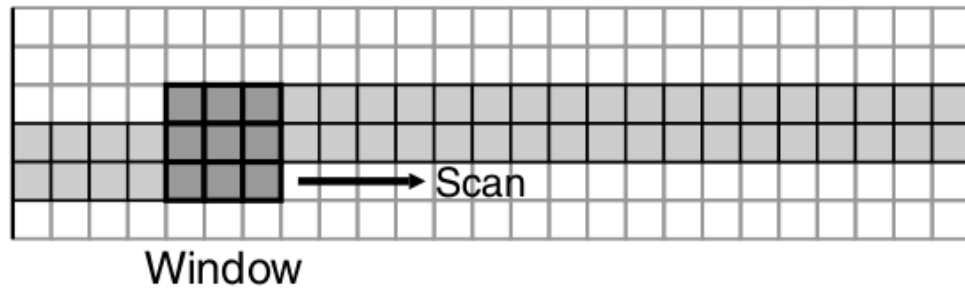
```
sorter_bin.m generate_mosaic_image.m sim_demosaicing_simple.m
function generate_mosaic_image(varargin)
1
2
3   p = inputParser();
4
5   p.addParamValue('input_file_location', './../data/truecolorimages/ '); %jpg
6   p.addParamValue('input_image_file', 'kodim07.png');
7   p.addParamValue('ISP_pixel_size', 8);
8   p.addParamValue('G_IMG_WIDTH', 768); %768
9   p.addParamValue('G_IMG_HEIGHT', 512); %512
10  p.addParamValue('G_PIXEL_WIDTH', 8);
11  p.addParamValue('num_of_frames', 1);
12
13  p.addParamValue('save_location', './../rtlslimlib/isp/demosaicing/input_data');
14  p.addParamValue('save_input_image', 'input_image');
15  p.addParamValue('save_expected_output', 'expected_output_image');
16
17  p.parse(varargin{:});
18
19  disp('-----Starting of Generating Files----- ');
20
21  %Apply filters N reconstruct the image
22  %shape = 'full';
23  shape = 'same';
24
25  sv_image_tmp_G_at_G = conv2(iv_image, G_at_G_coef, shape); uint16(sv_image_tmp_G_at_G); %g0
26  sv_image_tmp_G_at_R = floor(conv2(iv_image, G_at_R_coef, shape)/8); uint16(sv_image_tmp_G_at_R); %g1
27  sv_image_tmp_G_at_B = floor(conv2(iv_image, G_at_B_coef, shape)/8); uint16(sv_image_tmp_G_at_B); %g2
28
29  rgb_image(:, :, 1) = sv_image_R;
30  rgb_image(:, :, 2) = sv_image_G;
31  rgb_image(:, :, 3) = sv_image_B;
32
33  rgb_image(1:2, :, 1:3) = 0;
34  rgb_image(G_IMG_HEIGHT - 1:G_IMG_HEIGHT, :, 1:3) = 0;
35
36  rgb_image(:, 1:2, 1:3) = 0;
37  rgb_image(:, G_IMG_WIDTH - 1:G_IMG_WIDTH, 1:3) = 0;
38
```

# RTL Design

- The basic RTL architecture is shown in the figure, where the controller manager interfaces such as AXIS4, pixel addresses, valid and ready signals transmission.
- Fifo5x5 is responsible for the converting the stream of pixels into a 5x5 mask keeping the shape of the block.
- Router manages conditionals that might change the design based on the pixel position.
- And finally conv realizes the convolution, clip and clamp the pixels.
- The initial tool used for this is ModelSim, but could be any simulation software. For debuding testbenches are also included and code to read and write files, which comes from the octave model.



# RTL Design - FIFO

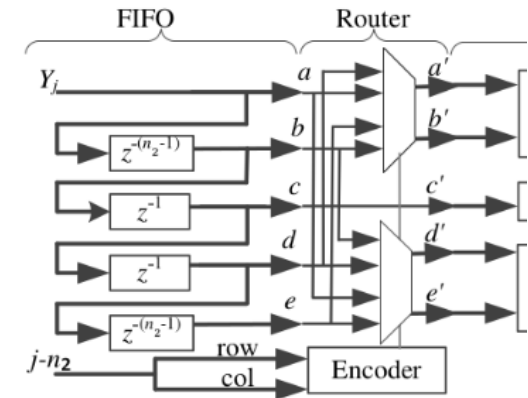


[\*0] Donald Bailey, Design for Embedded Image Processing on FPGAs

- The image shows a 3x3 window/mask to buffering 2 rows of the frame, we use similar approach but for 5x5 mask;
- The fifo will buffer 4 rows and will use shift registers to shape the 5x5 grid;
- From the stream of coming pixels, this Fifo5 will output 25 pixels on parallel to the next module, the router.
- This module is expected to consume more blocks of memory.

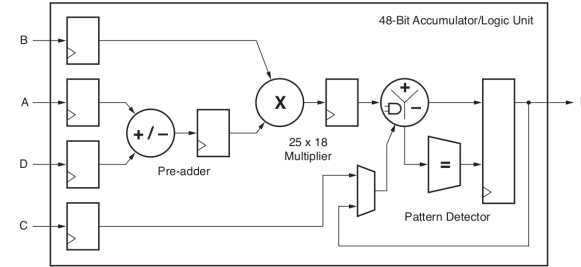
# RTL Design - Router

- The Fig. Is not from this design but represents the concept where the shape of filter mask may change according to the pixel position, which is why the router is needed.
- For this first version the conditional is very simple and when the pixel is on the border or 1<sup>st</sup> and 2<sup>nd</sup> rows or cols the output is 0 padding, which makes the calculation made only for the center and it is easy to model.
- A second version will modify the filter mask to adapt it for the border and corner of the image.

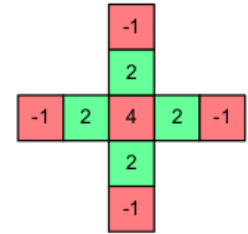




# RTL Design - Convolution



**[\*2] 7 Series DSP48E1 Slice User Guide (UG479)**



G at R locations

```

-- |XX|XX|12|XX|XX
-- |XX|11|10|09|XX
-- |08|07|06|05|04
-- |XX|03|02|01|XX
-- |XX|XX|00|XX|XX
----- *2

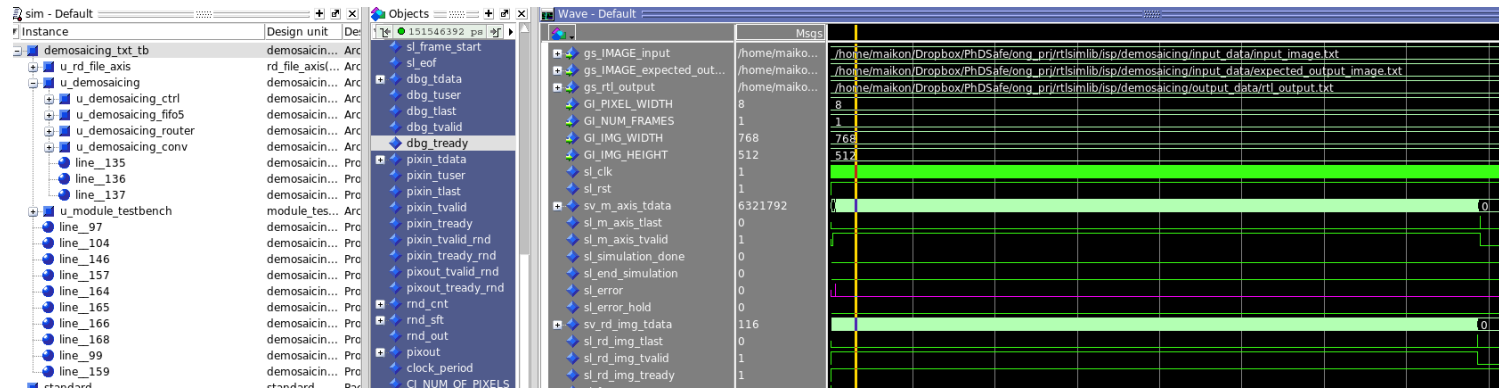
constant CI_lb_Quant          : integer := 2;
type t_Coef is array (12 downto 0) of integer;

----- coefficients are quantized already

constant CI_COEF_GatRnB      : t_Coef :=
    (-2,
     0, 4, 0,
    -2, 4, 8, 4, -2,
     0, 4, 0,
     -2);
    
```

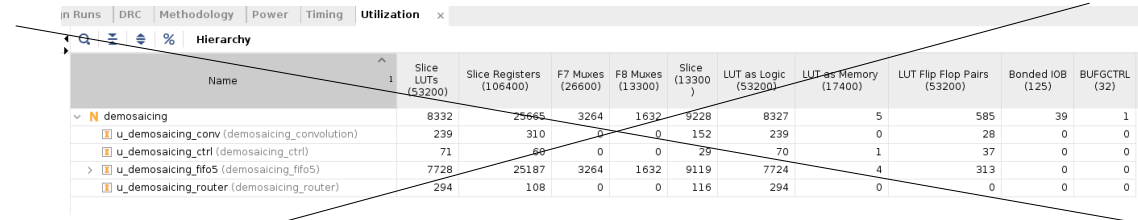
- The Fig. On top shows the basic DSP block present at Zynq platform or 7 Series FPGAs of Xilinx, which is a decision the design have to make to use them instead of logic. In my first design I did not use them and later I changed the coding style to infer the DSP tiles.
- Because there are fractional numbers in the coefficients, a fixed point conversion is needed, and in the coding style I also keep the shape of the coefficients to be more clear for code reading.
- A third approach of implementation is to use open the operations with sum and bus shifting which is possible due to the nature of the coefficients numbers present.

# Results



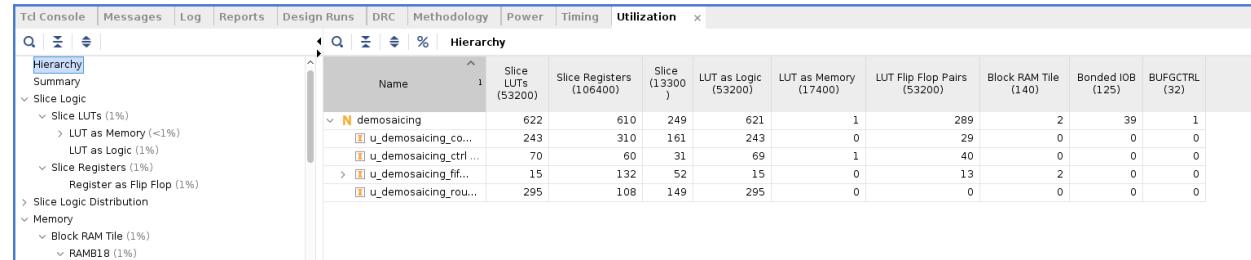
- ModelSim was used for the basic development and functional validation. The image on top illustrates the project where text files were created by Octave with the input vector and expected output, then image dimensions and basic signals for verification like valid and error are there to show any error when comparing with Octave. In pink shows 0 errors so the design has bit accuracy.
- In terms of FPGA performance and resource utilization we have the report from Vivado.
- The RTL and Octave can be found at:  
<https://github.com/maikonadams/fpgaip/tree/master/demosaicing>

# Results



This screenshot shows a Vivado Utilization report for a design named 'demosaicing'. The report is displayed in a table format with columns for various resources and their utilization counts. The resources include Slice LUTs, Slice Registers, F7 Muxes, F8 Muxes, Slice (13300), LUT as Logic, LUT as Memory, LUT Flip Flop Pairs, Bonded IOB, and BUFGCTRL. The utilization counts are as follows:

Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	LUT Flip Flop Pairs (53200)	Bonded IOB (125)	BUFGCTRL (32)
<b>N demosaicing</b>	8332	25665	3264	1632	9228	8327	5	585	39	1
u_demosaicing_conv (demosaicing_convolution)	239	310	0	0	152	239	0	28	0	0
u_demosaicing_ctrl (demosaicing_ctrl)	71	68	0	0	29	70	1	37	0	0
u_demosaicing_fifo5 (demosaicing_fifo5)	7728	25187	3264	1632	9119	7724	4	313	0	0
u_demosaicing_router (demosaicing_router)	294	108	0	0	116	294	0	0	0	0



This screenshot shows a Vivado Utilization report for a design named 'demosaicing'. The report is displayed in a table format with columns for various resources and their utilization counts. The resources include Slice LUTs, Slice Registers, Slice (13300), LUT as Logic, LUT as Memory, LUT Flip Flop Pairs, Block RAM Tile, Bonded IOB, and BUFGCTRL. The utilization counts are as follows:

Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	LUT Flip Flop Pairs (53200)	Block RAM Tile (140)	Bonded IOB (125)	BUFGCTRL (32)
<b>N demosaicing</b>	622	610	249	621	1	289	2	39	1
u_demosaicing_co...	243	310	161	243	0	29	0	0	0
u_demosaicing_ctrl ...	70	60	31	69	1	40	0	0	0
u_demosaicing_ff...	15	132	52	15	0	13	2	0	0
u_demosaicing_rou...	295	108	149	295	0	0	0	0	0

- Here just to show a simple coding style decision can penalize the design by misleading Vivado. This case the memory was not being inferred using blocks of RAM shown in the first picture.
- The max frequency so far is 150 MHz for xc7z020clg400-1.

# Conclusion and Future Work

- I presented my usual methodology to develop ISP RTL design using Octave, Modelsim, and Vivado. The system uses real data and aims to be a professional design to be implemented at any FPGA interfacing an Imaging sensor.
- My design achieved 0 bit error or bit accuracy against the Octave Model, comparisons were made in the RTL project.
- It is still missing a demo in a live system which is the next step using DMA, AXIS4 interface, and embedded Linux to test the design with the real images that I have.
- Also the Max. Frequency can be improved with pipeline more the design in the convolution module. The critical path is there and vivado has shown 7 or 8 levels of logic. My future work I am going to show how to debug and work with Vivado to increase max frequency.