# OLYMPUS: High-Quality Demosaicing Engine
# ECE 751 Project Report

SAGNIK BASU sbasu24@wisc.edu
JAGDISH MOHAPATRA nmohapatra@wisc.edu
LAMSHE RAJA lraja@wisc.edu
RAHIMULLAH SHAIK rshaik@wisc.edu

***Abstract*— Digital cameras use Color Filter Array (CFA) where the sensor detects only a particular spectral band for each pixel. The unmeasured spectral bands are estimated from the neighborhood pixels. Hence to reconstruct a full representation of the color image, an interpolation of the unmeasured spectral bands is required.This is known as demosaicing. Though there are many different demosaicing techniques, there is always a trade-off between dynamic power consumed during computation and the quality of the reconstructed image. The proposed scheme is based on the fact that edges have strong luminance components. Hence, we are proposing a gradient-corrected technique by taking the intensity values of critical pixels into consideration. The proposed techniques give a better performance in terms of low dynamic power consumption and improved PSNR when compared to some of the available demosaicing techniques such bi-linear interpolation.**

## I. INTRODUCTION

Image Signal Processors (ISP) are an integral part of all modern smartphones. From [1] , it can be summarized that it consists of several blocks to convert the Bayer pattern from CMOS sensors to a visually perceptive RGB pattern. More research is being done on efficient ISP designs on modern processors [2]. We propose a novel design for one of the most important blocks of the ISP – which is Image Demosaicing [3]. The RGB output pixels computed from the demosaicing block are important for computer vision algorithms being run on an embedded system. Since the output from demosaicing blocks is being fed to noise reduction blocks, we target generating a more accurate demosaicing hardware in order to reduce dynamic power consumption and improve PSNR. By saving dynamic power on the demosaicing block, we are making ISP pipeline more efficient than before.

Prior work on image demosaicing involves various kinds of interpolation algorithms to accurately estimate the RGB pixels at their corresponding pixel position, from the raw bayer pattern. One of the widely used in real hardware is the bi-linear interpolation - which is simple and efficient. While major strides were made on improving the algorithm complexity of demosaicing, unfortunately, only a few could be realized on real hardware due to the severe constraints in performance, power, and area in embedded smartphone processors, where ISPs are more widely used. Thus, the aim of our project was to find a hardware-friendly interpolation algorithm. We found the one proposed by HS Malvar et al., [] to be one such candidate for our project. In this project

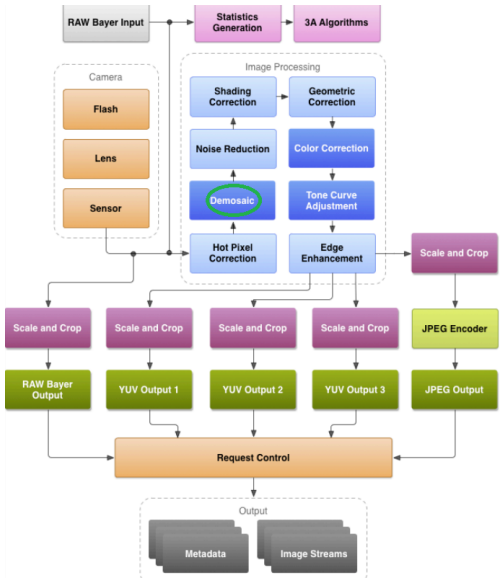


Fig. 1. Blocks of an Image Signal Processor chip

we aim to implement this interpolation scheme in C-Model and ModelSim and evaluate the performance and power implications of the final output on a widely used demosaicing dataset.

Specifically, our paper covers the following considerations of the proposed topic :

1) **Background**: We highlight the relevant prior work and the specific paper that motivated this project. We also explain the working algorithm used for our project.
2) **Methodology**: We explain the software and hardware methodology used for our project.
3) **Results**: We analyze the results of our demosaicing engine implementation.
4) **Related Work**: We summarize other work which exists on demosaicing implementations on hardware.
5) **Contribution**: We highlight the contribution of each member towards this project.

## II. BACKGROUND

First and foremost, we would use [1] to gain understanding of Image Signal Processing blocks and the demosaicing algorithms used. Further, recent work on efficient mapping of

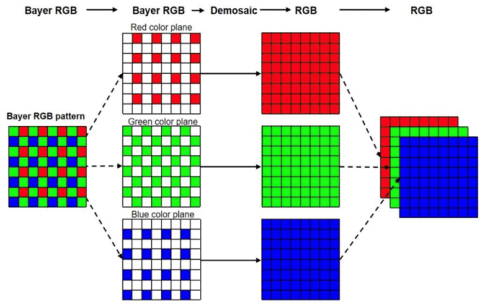


Fig. 2. Image Demosaicing Steps

demosaicing algorithms onto FPGA were done in [4] and [5]. Traditional ISP implementations that use bi-linear interpolation for major computations incur a drain on performance, dynamic power, and efficiency due to inefficient filter size and co-efficient.

Since our primary objective was to improve them while not sacrificing the accuracy or PSNR, we did an extensive literature survey on improvements in linear interpolation. One of the papers which showed potential was developed by HS Malvar et al. [6]. Their overall algorithm was based on the intuition that bi-linear interpolation did not add any edge information on the visual scenes while performing the demosaicing. Hence, adding this edge or gradient information can cause significant improvements in the RGB output quality. We will discuss more about the algorithm implementation in the software section.

While planning to map the same algorithm into hardware, there is a certain number of considerations to make before deciding the micro-architecture. Currently, the industries are using mainly 2 PPC (pixel per clock) and some are using 4 PPC designs while processing image pixels. Now color image pixels can be of varying bits, for example, 8bits, 12 bits, 14 bits, etc. Given the time constraints we had for ECE 751, we decided to choose 1pixel=8bits for our design and we assumed that we would be able to complete it if we consider 8bit image pixels. Having said that, the more the number of bits per pixel, the better is image quality. Mostly, HDR images are comprised of more bits per pixel, where processing a large amount of data into the DDR and e DDR and retrieving them back is going to be the major bottleneck and designers are finding ways to accommodate.

## III. Methodology

### A. Software

From the software perspective , the first task was to complete generating a set of bayer test vectors and golden reference for the demosaicing block. Since raw bayer format is dependent on the CMOS camera vendor, it is crucial to use a widely know dataset in order to reproduce the correct results. We found the Microsoft demosaicing dataset [7] to be the perfect candidate for our purpose. We are using the OpenCV cvtColor() function for generating the ground truth RGB image from the bayer image. Fig. 3 and 4 show a bayer image and the converted RGB image using software demosaicing.

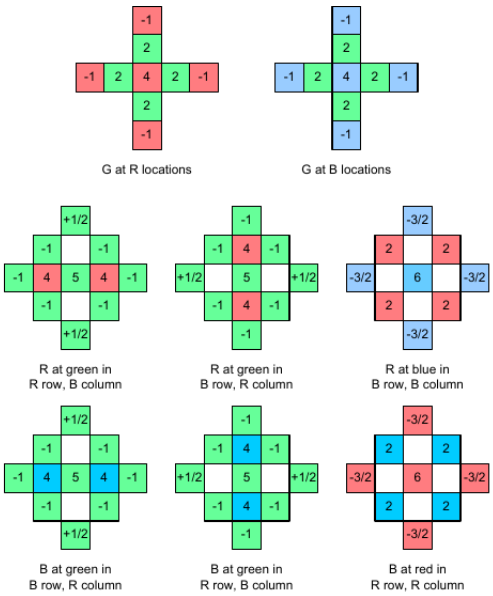


Fig. 3. Interpolation scheme

To implement the C-model of the interpolation scheme we thoroughly studied the steps of the iterative algorithm described in [6]. It involves selecting each center pixel to accurately estimate the corresponding RGB data. In our C-model, the main outer loops will iterate throughout each of the RGB channel dimensions and continue filling them from the bi-linear interpolation scheme.

$$\hat{g}(i,j) = \frac{1}{4}\sum g(i+m, j+n) \tag{1}$$

To each value of the corresponding bi-linear scheme, we will be adding the gradient information multiplied by the gain parameters - which are separate for each channel.

$$\hat{g}(i,j) = \hat{g}_B(i,j) + \alpha\Delta_R(i,j) \tag{2}$$

where delta is the gradient of the red pixel at that location. It is computed as below :

$$\Delta_R(i,j) \triangleq r(i,j) - \frac{1}{4}\sum r(i+m, j+n) \tag{3}$$

Based on the center pixel location, we use conditional branching statements to decide next filter operation. In Figure 4, we discuss one such conditional block. Here the neighbor's array stores the elements of the 5x5 block which is a kind of region of interest of our block. The neighbor-mask array contains the mask values which are required to satisfy the interpolation scheme. Each neighbour-mask element is a binary 0 or 1. Hence, the overall calculation ensures that the denominator of the block is always a power of 2 - which can be easily synthesized in hardware as a shift operation.

One of the crucial reasons for implementing the C-model as a first step is to estimate the gain parameters from the dataset. In equation (2) we can see that the gain parameter

```
if((x & 1) == RedX && (y & 1) == RedY)
{
    /* Center pixel is red */
    OutputRed[i] = Input[i];
    OutputGreen[i] = (2*(Neigh[2][1] + Neigh[1][2]
        + Neigh[3][2] + Neigh[2][3])
        + (NeighPresence[0][2] + NeighPresence[4][2]
        + NeighPresence[2][0] + NeighPresence[2][4])*Neigh[2][2]
        - Neigh[0][2] - Neigh[4][2]
        - Neigh[2][0] - Neigh[2][4])
        / (2*(NeighPresence[2][1] + NeighPresence[1][2]
        + NeighPresence[3][2] + NeighPresence[2][3]));
    OutputBlue[i] = (4*(Neigh[1][1] + Neigh[3][1]
        + Neigh[1][3] + Neigh[3][3]) +
        3*((NeighPresence[0][2] + NeighPresence[4][2]
        + NeighPresence[2][0] + NeighPresence[2][4])*Neigh[2][2]
        - Neigh[0][2] - Neigh[4][2]
        - Neigh[2][0] - Neigh[2][4]))
        / (4*(NeighPresence[1][1] + NeighPresence[3][1]
        + NeighPresence[1][3] + NeighPresence[3][3]));
}
```

Fig. 4. C- Model of conditional block

along with the gradient plays an important role in overall accuracy. While [6] estimates the gain parameters using second order output statistics on the Kodak dataset, we use the same procedure but on a the high quality and more diverse MSR Demosaicing dataset. These co-efficient were then used directly on the Hardware verilog model, which we will discuss next.

### *B. Hardware*

The crucial task from hardware perspective is coming up with novel demosaicing hardware engine architecture, and implement it using verilog HDL.

In our 8bit per pixel 1 ppc design, we micro architected the demosaicing engine by deciding the input and output signals and internal wires. We wrote the RTL for the demosaicing engine in System Verilog, as it provides better measures while writing combinational and sequential blocks. Then we wrote the test-bench to verify our RTL in Verilog and we used Modelsim to simulate our design. While coming to the verification plan, we wrote a comprehensive test-bench with 50 sets of different test cases to measure the efficacy of our design. In each testcase, we fed bayer pixel data to the DUT and the DUT then generates the missing R/G/B pixels for the given load of bayer pixel. We got 100 percent accuracy for all our samples by comparing our results with the results taken for the same image from MATLAB.

In our implementation, we took a 5x5 two-dimensional matrix array named pixel window which stores bayer data upon which the operations are to be performed upon and the depth of each pixel is 8 bits which is a basic color image. Also, we consider two arrays named pixel_row_en and pixel_col_en which are used to select a particular row and a particular column in order to select a specific pixel. We took a 2-bit bus signal bayer center pixel which stores the type of center pixel which is an important information to us as the whole operation is based on bayer center pixel values. When bayer_center_pixel is 0(Blue), 1(Green on Blue), 2(Green on Red), 3(Red). The demosaicing block performs operations and we take 24-bit final output bus which is RGB values of the pixel out of starting from the MSB, 8 bits represent Red value, the following 8 bits represent the Green values and the rest 8 bits represent Blue value. We synthesized our design using Design Vision efficiently. The operation of the demosaicing block is as follows, it first reads pixel matrix and the central pixel type and then we estimate the green at non-green, non-green at non-green, non-green in the same row as green value, non-green in a different row green value. Depending upon the central pixel value, we concatenate the corresponding R, G, B values and store them in our final RGB vector.

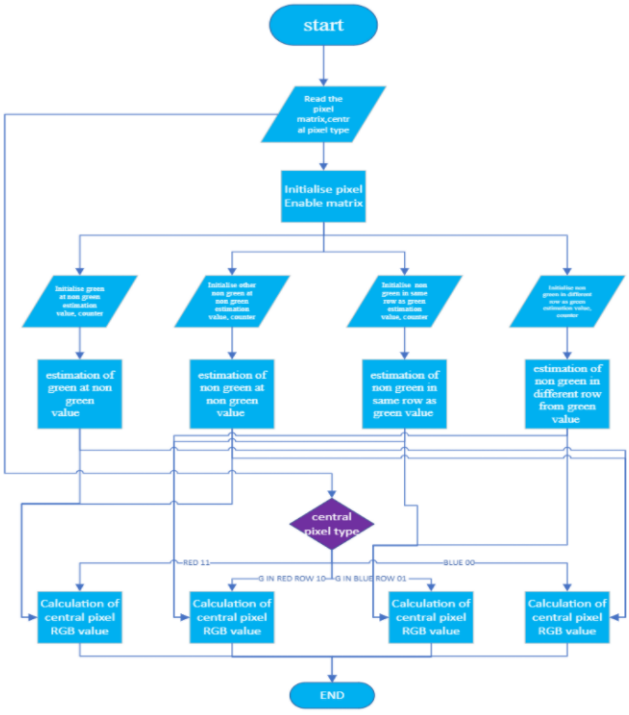


Fig. 5. Hardware block diagram

For the calculation of RGB values for central pixel which may be a red, blue, green in red row, green in blue row pixel, we take the pixel matrix and initialize the pixel enable matrix which is used to select the row or column. Then estimation of green pixel value at non green pixel, Estimation of non-green at non green pixel, estimation of non-green in the same row as green value, estimation of non-green in a different row as green values are carried out as per the proposed algorithm. Hence, we get red, green, blue values for the center pixel from raw data. This ensures that our algorithm is correct and the system uses minimum memory transactions, thus reducing the dynamic power consumption.

Synopsys design compiler is used to synthesize the design and generate the gate-level netlist. Synopsys 32nm educational cell library (saed32) is used. The variations of IC manufacturing processes affect the speed of the transistors and hence it affects the timing of the gates. The operating conditions, which are the PVT corners are chosen in the middle range of the PVT corners. Typical speed NMOS PMOS are chosen. Higher voltages are faster but more power-hungry. Thus, the voltage at which the design runs is set in the medium voltage of 0.85V. Temperature also affects the speed of the transistors. Cold temperatures are fast and

hot temperatures are slow. So, the temperature is set at room temperature of 25 Celsius.

Synopsys needs to know the drive strength of inputs. The drive strength of the input is set with the same strength as if it was driven by a 2x1 NAND cell. Synopsys also needs to know the amount of load the output must drive. The more the output load, the slower it will be. It is set that the output has to drive a 0.1pF load. There are some routing capacitances for internal nodes in the design. These parasitic capacitance's need to be set so that the tool can design the logic fast enough to overcome them. A wire load model is set to estimate the capacitance of the wiring between gates. If nodes are too slow to transition, they will spend too much time in a region with both high Vds and high Vgs. This can damage the transistors and slow them down over time. Hence the maximum transition time is set at 0.1ns.

Map effort refers to the time the synthesizer takes to optimize the design in terms of mapping it to our standard cell library. The medium effort is set for mapping to the standard cell library. Then the design is compiled with an 'Ungroup all' option. The ungroup all flattens the whole hierarchy so that the entire design is compiled as one module. The synthesis script is executed, and the results are generated. The power report gives a detailed view of dynamic power & static power and thereby gives the total power consumption. Similarly. the area report gives a detailed report of the number of cells, cell area, and routing area. Switching activity in a design causes dynamic power consumption, whereas leakage power of transistors contributes to static power consumption.

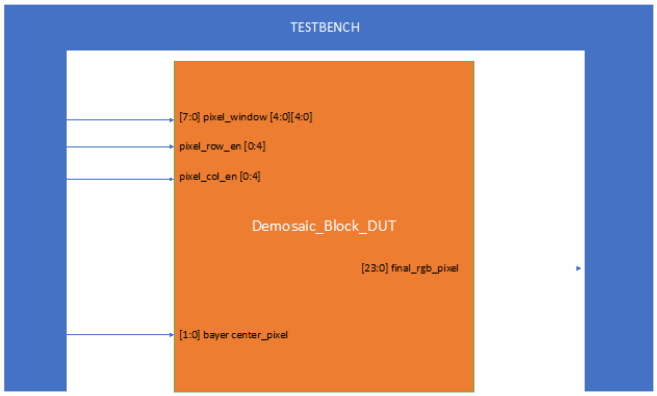


Fig. 6. Hardware Testbench

### C. Testbench

We have finished designing with exhaustive test-cases to measure all the feature of the top hardware module before it goes to FPGA and test the code coverage for the RTL. The input pin description of our testbench :

- pixel window : a 5x5 window with each 8bit-width bayer pixel data
- pixel row en : a Single bit signal with depth 5. Used to select the particular row to start computation
- pixel col en : a Single bit signal with depth 5. Used to select the particular column to start computation
- bayer center pixel : a 2-bit depth signal to check the center pixel type

The output ports of our testbench :

- final rgb pixel : à 24-bit pixel data with R,G,B components each contributes 8bit

## IV. Results

The results generated for the high-quality linear interpolation technique are compared with the synthesized results of the bi-linear interpolation method. From the results, it is evident that the bi-linear interpolation consumes more dynamic power than the high-quality interpolation technique because of the switching activity. Whereas high-quality interpolation consumes a bit more static power than bi-linear interpolation. This is due to the fact that the number of cell counts are more than bilinear-interpolation method because of one extra layer of computations. This extra layer of computations increased the transistor cell counts which contributes some more leakage power.. Since the high-quality linear approach requires more cells for computation, there is an area overhead when compared to bi-linear interpolation. So, there is an obvious power vs area trade-off. Overall, there is a 2 percent increase in area and an 8.9 percent reduction in power consumption.

We were also able to achieve a net improvement in PSNR from 26.27 dB to 29.81 dB. Theoretically, our approach is more accurate since we take into consideration the effect of edges in our algorithm, which the bi-linear interpolation does not. We also wanted to evaluate the structural similarity measure which was also done in [6] as part of their measurements, but due to time constraints were unable to design the metric.

| | **Bilinear (µW)** | **Olympus:High-Quality Linear(µW)** |
|---|---|---|
| Dynamic power | 590 | 302 |
| Static power | 2421 | 2439 |
| Total power | 3011 | 2741 |

Table 1 : Power consumption of Bilinear vs High-Quality linear

| | **Bilinear ($\mu m^2$)** | **Olympus:High-Quality Linear($\mu m^2$)** |
|---|---|---|
| No. of cells | 1786 | 1837 |
| Cell area | 5285 | 5406 |
| Total area | 6408 | 6542 |

Table 2: Area of Bilinear vs High-Quality Linear

## V. Related Work

We can broadly classify the demosaicing techniques prevalent today as two types - learning-based and interpolation-based. Deep learning models have led to many state of the algorithms in image processing task like optical flow

Fig. 7. Bayer image

Fig. 8. RGB Image

estimation , image denoising and classification . Recently, researchers from Microsoft has shown that using large scale RAW dataset from CMOS sensors, it is possible to demosaic the input Bayer data to RGB. While these techniques can be run in inference engines, they are more susceptible to over-fitting and sometimes may not give visually perceptible results.

Interpolation-based techniques like [3] and [1] are more widely used in the industry to design ISP chips. They can be further classified into - iterative and non-iterative. The non-iterative demosaicing algorithms consider the interpolation of edge-direction or local covariance signal information. Iterative techniques, like our project, involves iteratively updating the green, red and blue channels by ensuring the color ration rules are strictly enforced. It has been found in [1] that iterative techniques are more widely adopted and are easily synthesize-able on ASICs and with techniques like caching and pipelining, the overall run-time of the system are more optimized.

## VI. Conclusion

In this section , we consider some improvements that can be incorporated into our project and then summarize our implementations of the Demosaicing engine.

### A. Future Work

Our implementation of the Demosaicing engine improved the PSNR with respect to the Bi-linear approach. However, there is enough room for improving the hardware engine, since not many open research work has been done to synthesize ISP on re-configurable hardware. In the time allotted to the project, we are able to synthesize a working model which gives the accurate output. We believe that by adopting pipe-lining strategies, we can effectively hide the loads/store latency with the computations. A four-stage pipeline can be implemented in the computation-heavy data path in the design so that we can improve the throughput of the design efficiently. Also, some caching techniques can be applied as well while taking the full 5x5 window from the image, because there are a lot of redundant pixels are computed every time. These two improvements can be thoroughly done in the current design.

### B. Summary

Throughout this project we aimed at getting a working synthesize-able model of a high quality demosaicing engine in verilog and more importantly, learn more about the theory that requires in developing Image Signal Processors. Being a student of a graduate embedded systems course, it is highly crucial to understand what it takes to design a subsystem such as ISP under several power and area constraints while also maintaining the quality of the output. Our implementations which include a C-model and a verilog module, are clearly differentiated based on the complexity of the computations of the model. Though we met obstacles in the initial phase of the project, our team was able to successfully generate the net-list of the hardware in time for the final presentation, and was able to perform detailed analysis on the results.

Below, we enumerate the original milestones of our project proposal. We have mentioned in *italics* the status of the item at project submission. For the parts we have completed successfully, we over-struck them.

- **Milestone 1**: Tasks that we plan to complete before the first progress report deadline:
  1) ~~Survey various stages of ISP pipelines and choose which to target for our project.~~ *Completed.*
  2) ~~Do a more comprehensive literature review of recent high-quality demosaicing techniques. Particular attention will be given to analyzing the portability of the algorithm to re-configurable hardware.~~ *Completed.*
  3) ~~Develop a C-model of the implementation and find the gain parameters of the high-quality interpolation scheme on a good dataset.~~ *We are using the Microsoft research demosaicing dataset for our project*
  4) ~~Micro-architected the Demosaicing HW engine. Start design using Verilog~~ *Completed.*
  5) ~~Simulated using Modelsim. Verified the output RGB image with Ground Truth RGB image.~~ *Completed.*
- **Stretch Milestones**: These are the tasks which we hoped to complete, but may be too large to fully realize them given the time constraints.
  1) Parallelize the demosaicing computation. *We can try for more instruction-level parallelism to hide latency on the C-model implementation.*

2) Use approximate computation techniques like approximate multipliers to achieve aggressive power and area reduction. *This is a reasonable next step after adapting achieving golden data matching, since image processing applications are highly error resilient*.

## VII. Contribution

From the starting of the project we have been able to split the task amongst ourselves in sustainable manner. We would meet twice regularly to understand each others progress and solve any critical issues in our work. We were also involved in helping each other debug our work.

- Lamshe: Synthesized the design using Synopsys Design Compiler and generated the gate-level netlist, area, and power reports of high-quality linear interpolation and bi-linear interpolation techniques. She used the tools to accelerate the final synthesis of the demosaicing engine. She tested the design of high-quality linear interpolation technique using ModelSim for different corner cases and gave feedback to the team. She also converted the bayer data into RGB data using MATLAB coding for verifying the correctness of the design output.
- Sagnik: Contributed to the software part of the design process. Wrote the C-model of the interpolation algorithm and was able to successfully develop a tool for PSNR and SSIM measurements. He also experimented with various parts of ISP design to find more optimization techniques in the pipelines. Also, he helped write most of the documentation for the project.
- Jagdish: Micro-architected the Demosaicing hardware engine which involves the I/O mapping of the hardware, internal signal logic, design constraints for 1 PPC(pixel per clock) design, making room for dynamic power improvement by deciding suitable clock frequency, decided the verification methodology and test plan for the design. One of the challenging parts was finding a way to feed the raw Bayer image and getting the accurate RGB pixels to verify with the DUT's output using some known tools/simulators. For this, after a lot of sync up with team members, went ahead to do the MATLAB processing of a raw Bayer image and dumped the distinct R, G, B matrix values into an excel sheet and compared these MATLAB generated RGB values with the DUT's output RGB values to verify the efficacy of our hardware design. Contributed to the verification test plan by adding 11 distinct test cases. Worked closely with Rahim and Lamshe for the System Verilog based RTL design and testbench creation and gave my inputs and timely feedback in each phase of the design during RTL bringup, verification approach and synthesis.
- Rahim: He Micro-architectured the demosaicing block and implemented the RTL design according to the algorithm used in Verilog HDL using ModelSim, Simulated it without any compilation errors along with a comprehensive testbench with sufficient testcases in Modelsim with the help of Jagdish and Lamshe. He also gave valuable inputs and feedback to the team during the MATLAB processing of a raw Bayer image and converting it into RGB data and while synthesizing of our design using Synopsys Design Compiler and generating the gate-level netlist.

### A. Project Code and Tools

The source code used to implement the Demosaicing engine can be downloaded from this Google Drive link here. The verilog implementations is in dut.sv file and the testbench is in dut tb.sv file. The project folder also contains the results of the synthesized verilog code. Most of the research were performed using our personal computing capacity. We used tools like Synopsys Compiler Design, ModelSim and Matlab.

## REFERENCES


[1] S.-H. Choi, J. Cho, Y.-M. Tai, and S.-W. Lee, "Implementation of an image signal processor for reconfigurable processors," 2014 IEEE International Conference on Consumer Electronics (ICCE), pp. 141–142, 2014.

[2] P. Hansen, A. Vilkin, Y. Krustalev, J. Imber, D. Talagala, D. Hanwell, M. Mattina, and P. N. Whatmough, "Isp4ml: The role of image signal processing in efficient deep learning vision systems," 2020 25th International Conference on Pattern Recognition (ICPR), pp. 2438–2445, jan 2021. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICPR48806.2021.9411985

[3] K. Hirakawa and T. Parks, "Adaptive homogeneity-directed demosaicing algorithm," IEEE Transactions on Image Processing, vol. 14, no. 3, pp. 360–369, 2005.

[4] W. Hsu and C.-S. Fuh, "Real-time demosaicking for embedded systems," 2007, pp. 1–2.

[5] A. Karloff and R. Muscedere, "A low-cost, real-time, hardware-based image demosaicking algorithm," 2009, pp. 146–150.

[6] H. Malvar, L. wei He, and R. Cutler, "High-quality linear interpolation for demosaicing of bayer-patterned color images," in 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 3, 2004, pp. iii–485.

[7] D. Khashabi, S. Nowozin, J. Jancsary, and A. W. Fitzgibbon, "Joint demosaicing and denoising via learned nonparametric random fields," IEEE Transactions on Image Processing, vol. 23, no. 12, pp. 4968–4981, 2014.