High-Performance Image Processing System Design Using Verilog

Chao-Jia Liu
Electrical & Computer Engineering. The University of Texas at San Antonio-BSE 1.506
San Antonio, TX 78249
chao-jia.liu@my.utsa.edu

Varsha Kothawade
Electrical & Computer Engineering. The University of Texas at San Antonio-BSE 1.506
San Antonio, TX 78249
varsha.kothawade@my.utsa.edu

***Abstract*— This research presents the design and implementation of a high-performance image processing system using Verilog hardware description language. The growing demand for real-time image processing applications, such as computer vision, medical imaging, and augmented reality, necessitates efficient and hardware-accelerated solutions. The design encompasses key image processing algorithms, including but not limited to image filtering, edge detection, and image segmentation, implemented in Verilog. The utilization of parallel processing techniques enables simultaneous execution of multiple images processing tasks, enhancing throughput and reducing processing time. Additionally, the system incorporates optimizations for memory access and data transfer to further improve overall performance. The Verilog-based image processing system is evaluated on popular FPGA platforms, demonstrating its efficiency and suitability for real-time applications. The experimental results showcase the system's ability to handle high-resolution images with low latency, making it well-suited for use in time-sensitive scenarios. Furthermore, the modular design allows for easy customization and adaptation to accommodate a variety of image processing algorithms, providing flexibility for future applications. This research contributes to the field of image processing by providing a scalable and efficient hardware solution that meets the demands of real-time processing, making it applicable to a wide range of domains, from embedded systems to advanced image processing applications. The Verilog implementation serves as a foundation for further exploration and optimization of hardware-accelerated image processing systems, addressing the increasing need for high-performance computing in the field of image processing.**

Keywords*— *Image processing, Memory, MATLAB, Xilinx Vivado, hex conversion, effects on image

I. Introduction

The project aims to design and implement key image processing algorithms, which will be including image filtering such as inverting image from its original color, variation in image brightness and adding threshold to the original image edge detection, and image segmentation, which will store the converted hex values in memory. By successfully implementing image processing algorithms on an VHDL, this project aims to contribute to the advancement of real-time image processing capabilities, opening possibilities for improved performance in critical applications where low latency and high throughput are essential. The outcomes of this research have the potential to impact fields such as medical diagnostics, surveillance systems, and autonomous vehicles, where rapid and efficient image processing is crucial for decision-making processes. The image processing through memory, will successfully convert the original image into various effects. Various effects will include image converting colors into black and white image, increasing the brightness of the image, and adding the threshold to the original image. At last, we will be designing the chip which will give the exact implementation of chip before it goes to the fab. While performing the various operation this project will go through the different applications such a MATLAB and Xilinx vivado.

II. Project Overview

The primary focus of this project is to formulate and implement image processing algorithms, employing a memory-based approach with Verilog to declare a memory area for storing pixel information. This unconventional methodology seeks to offer versatility and adaptability, diverting from the conventional FPGA-centric platforms. The project encompasses several key objectives:

Establishment of a library housing hardware-proven and configurable Verilog-based image processing blocks and utilities, including filters, edge detectors, color converters, and image resizers. These components facilitate the prototyping and simulation of image processing algorithms without necessitating the writing of Hardware Description Language (HDL) code.

Exploration of trade-offs within different levels of parallelism, resource utilization, and performance for pixel and component processing. Techniques such as loop unrolling, pipelining, and dataflow optimization will be employed within the Verilog-based memory approach.

Generation of synthesizable VHDL or Verilog code directly from MATLAB or Simulink models of image processing algorithms. Methods and frameworks like HDL Coder, Simulink HDL Coder, and Xilinx System Generator will be utilized for this purpose.

Demonstration of image processing applications across various domains, including stereo vision, fog rectification, blob analysis, and contrast enhancement, using FPGA boards and cameras within the Verilog-based memory approach.

Furthermore, the project will delve into future directions and open problems in memory-based image processing research, exploring areas such as software integration through hardware, the amalgamation of machine learning and deep learning techniques, the evolution of domain-specific languages and compilers, and the evaluation of energy efficiency and reliability.

III. Project design

In this section, we delve into the design and implementation steps of the project, with a focus on a memory-based approach utilizing Verilog for image processing. The methodology spans three main phases: prototyping and simulation, code generation, and implementation and testing.

The image processing journey commences with a PNG/JPG/JPEG file with dimensions 768 X 512. Initially, the input image undergoes conversion into a hardware-compatible format through MATLAB. This involves transforming the original image pixel into a hex file format using Simulink.

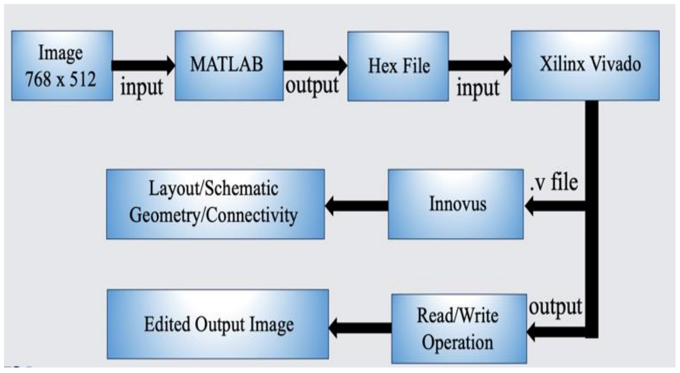


Fig. 1 Image processing workflow

In the second phase, synthesizable Verilog code is generated from MATLAB or Simulink models of image processing algorithms. Techniques like HDL Coder and Simulink HDL Coder, along with verification tools such as HDL Verifier and Simulink Verification and Validation, ensure functional equivalence and correctness of the generated code. Coding standards and best practices are implemented to enhance code readability and maintainability.

The project leverages a library comprising hardware-proven and configurable blocks capable of executing common image processing operations, including filtering, edge detection, color conversion, and image resizing. This library supports various image formats like RGB, YCbCr, and grayscale, and different data types such as fixed-point and floating-point. Techniques like pipelining and dataflow optimization enhance parallelism and throughput. Tools like Xilinx Vivado HLS and Innovus Implementation System software estimate resource utilization and performance across diverse architectures and devices. Comparative analysis guides the selection of optimal design parameters for each algorithm.

In the Third phase, the project focuses on the implementation and testing of image processing algorithms without the utilization of dedicated platforms. This approach involves employing Verilog to declare a memory area for image pixel information. The project employs tools like Xilinx Vivado Logic Analyzer and Innovus Implementation System software for monitoring and debugging the image processing system. Performance, power consumption, and reliability are meticulously measured and evaluated in this environment.

IV. Tools and Methodology

The "Tools and Methodology" section details the tools employed in implementing the project and outlines the input procedures, reflecting on the output. The project design, centered around "image processing using Verilog," prominently revolves around three pivotal tools: MATLAB, Xilinx Vivado, and Innovus Implementation System software. The actual project is focused on a memory-based approach, leveraging Verilog to declare a memory area for storing image pixel information.

In the Verilog code, we describe the three key steps: uploading the image, reading the input, and writing data to achieve the desired effects. The third step involves specifying parameters to operate in different effects. Upon the completion of the Verilog code, the edited version of the input image is derived. A notable example of an effect is the Invert effect, as illustrated below. The practical project encompasses a total of four operations, including inverting the image, adding brightness, adjusting the image threshold, and converting the image to black and white.

Below are the execution steps involved in this project.

Fig. 2 Input image (768x512) to MATLAB

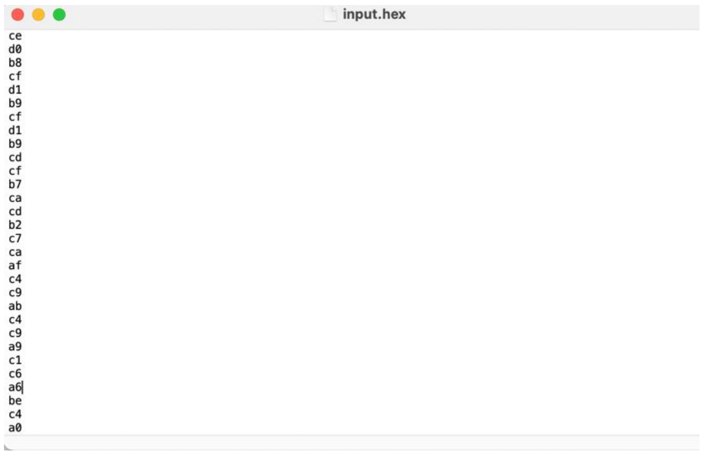

input.hex
```
ce
d0
b8
cf
d1
b9
cf
d1
b9
cd
cf
b7
ca
cd
b2
c7
ca
af
c4
c9
ab
c4
c9
a9
c1
c6
a6
be
c4
a0
```

Fig. 3 Input to the Xilinx Vivado

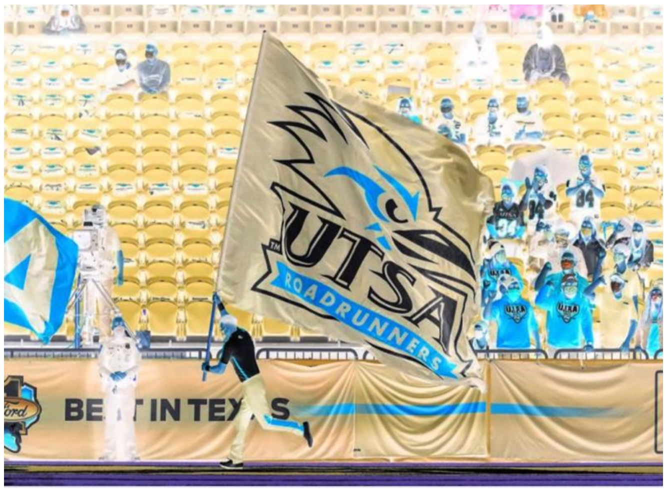


Fig. 4 Output from Xilinx Vivado (operation – Invert)

V. Result s and discussion

In this Image Processing through Verilog project, the original image undergoes initial processing in MATLAB, where it is converted into a hexadecimal file to facilitate subsequent Verilog-based image processing. The image then traverses the Verilog code, where key operations, such as Brightness Addition, Invert, Threshold, and Black and White Conversion, are applied. The Verilog code execution takes place through Innovus software to formulate the chip design. Through Innovus, we achieve geometry design capture, chip layout generation, connectivity verification, and timing analysis.

This section provides an exhaustive examination of results derived from our Verilog-based image processing project. Key operations, including Brightness Addition, Invert, Threshold, and Black and White Conversion, are thoroughly discussed. The Brightness Addition operation employs a Verilog code framework where a sign bit acts as a selector, orchestrating pixel color adjustments for heightened luminosity. RGB color conversion techniques are seamlessly integrated into the Verilog implementation to ensure effective handling of diverse color channels. The subsequent Invert operation involves a Verilog approach to color inversion, effecting an inverse transformation of each pixel's color composition. For the Threshold operation, the Verilog code incorporates threshold logic and RGB color conversion methodologies. This operation classifies pixel values, resulting in a binary output where pixels of similar color are uniformly rendered either white or black. The Verilog implementation of Black and White Conversion strategically transforms the image to monochrome through the integration of monochrome logic and RGB to monochrome conversion.

This section provides an exhaustive examination of results derived from our Verilog-based image processing project. Key operations, including Brightness Addition, Invert, Threshold, and Black and White Conversion, are thoroughly discussed. The Brightness Addition operation employs a Verilog code framework where a sign bit acts as a selector, orchestrating pixel color adjustments for heightened luminosity. RGB color conversion techniques are seamlessly integrated into the Verilog implementation to ensure effective handling of diverse color channels. The subsequent Invert operation involves a Verilog approach to color inversion, effecting an inverse transformation of each pixel's color composition. For the Threshold operation, the Verilog code incorporates threshold logic and RGB color conversion methodologies. This operation classifies pixel values, resulting in a binary output where pixels of similar color are uniformly rendered either white or black. The Verilog implementation of Black and White Conversion strategically transforms the image to monochrome through the integration of monochrome logic and RGB to monochrome conversion.

Notably, RGB color conversion techniques are seamlessly integrated into the Verilog implementation to ensure effective handling of diverse color channels. The subsequent Invert operation entails a Verilog approach to color inversion, effecting an inverse transformation of each pixel's color composition. For the Threshold operation, the Verilog code is designed to incorporate threshold logic and RGB color conversion methodologies. This operation classifies pixel values, resulting in a binary output where pixels of similar color are uniformly rendered either white or black. The Black and White Conversion, implemented in Verilog, strategically converts the image to monochrome through the integration of monochrome logic and RGB to monochrome conversion.

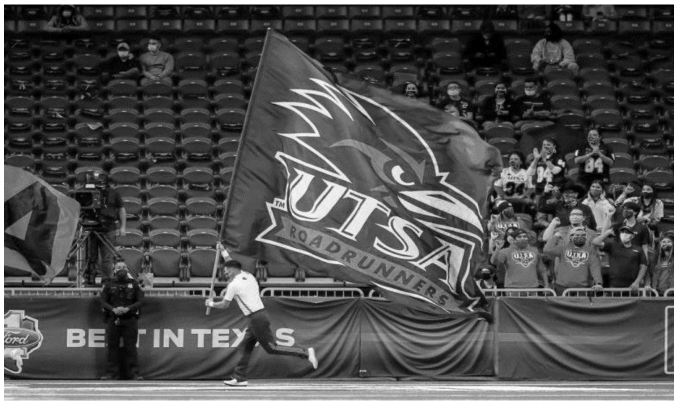
Performance simulators are employed for race-driven evaluations using pre-generated traces, focusing on the timing of instruction execution rather than semantic correctness. The Verilog implementation of Black and White Conversion transforms the image to monochrome, showcasing a streamlined integration of monochrome logic and RGB to monochrome conversion. For example, RGB (128, 128, 128) becomes RGB (255, 255, 255), illustrating the simplicity and effectiveness of the operation.

The Brightness Addition operation, realized in Verilog, introduces a sign bit as a selector, orchestrating pixel color adjustments to enhance luminosity. For instance, the operation transforms RGB (0, 0, 0) to RGB (255, 255, 255), highlighting the efficacy of our design.

In the Invert operation, our Verilog code skillfully achieves color inversion, exemplified by the transformation of RGB (255, 0, 0) (red) to RGB (0, 255, 255) (cyan). The Verilog code encapsulates the intricacies of color channel manipulation without delving into specific coding intricacies.

The Threshold operation utilizes threshold logic within the Verilog code, binary classifying pixel values. Although detailed code insights are omitted, an example such as setting a threshold to convert RGB (128, 128, 128) to RGB (255, 255, 255) illustrates the functionality.

Transitioning from Vivado output to Innovus results, layout analysis indicates a commendable density of 49.9%. Furthermore, timing analysis reveals a total CPU time of 2.5 seconds, a total real-time of 4 seconds, and a memory usage of approximately 1200 megabytes. Rigorous geometry and connectivity analyses affirm the structural integrity of our image processing Verilog design, revealing zero violations and warnings.



Black and White image

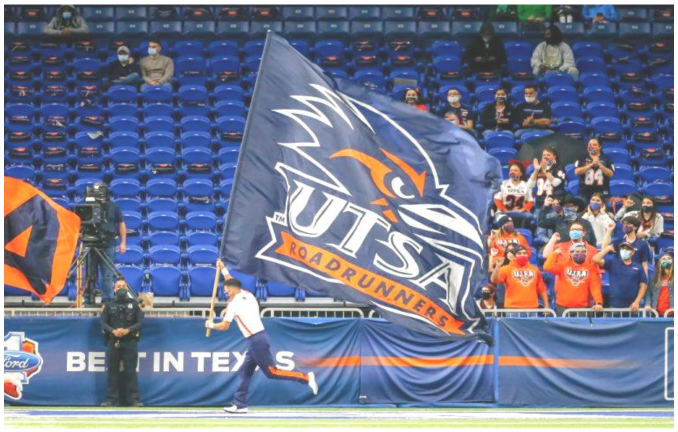
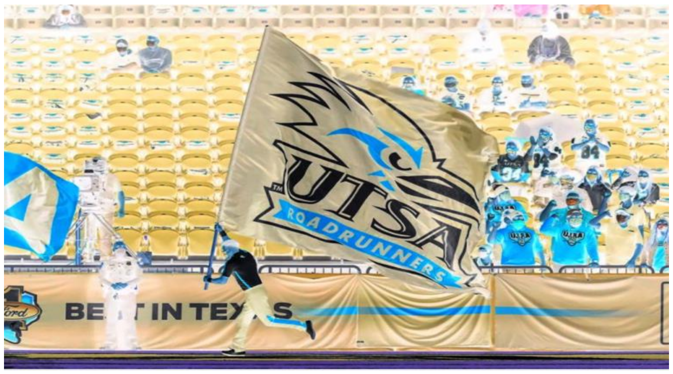
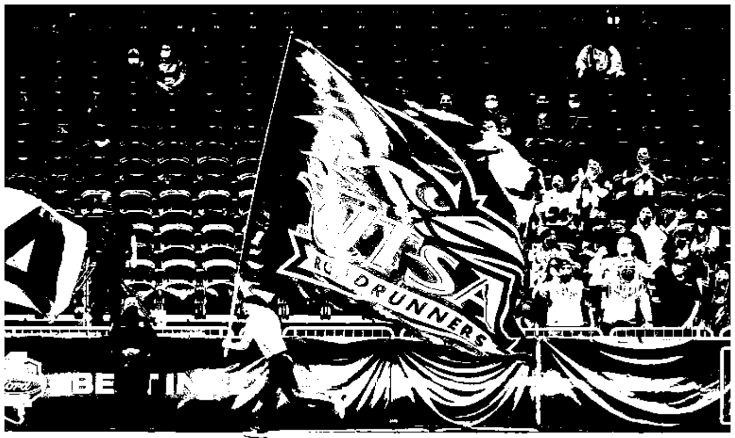


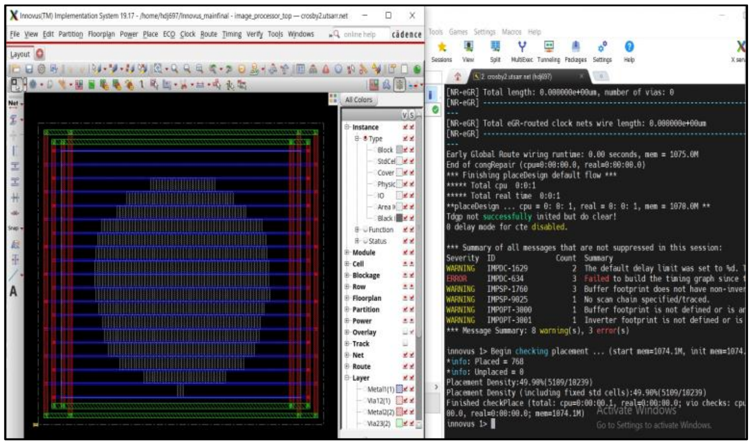
Image brightness



Invert operation of Image

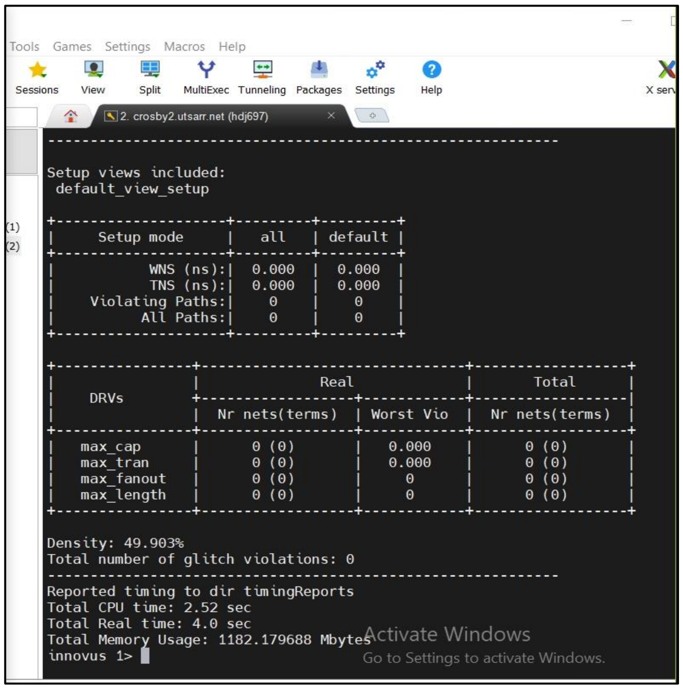


Threshold Operation



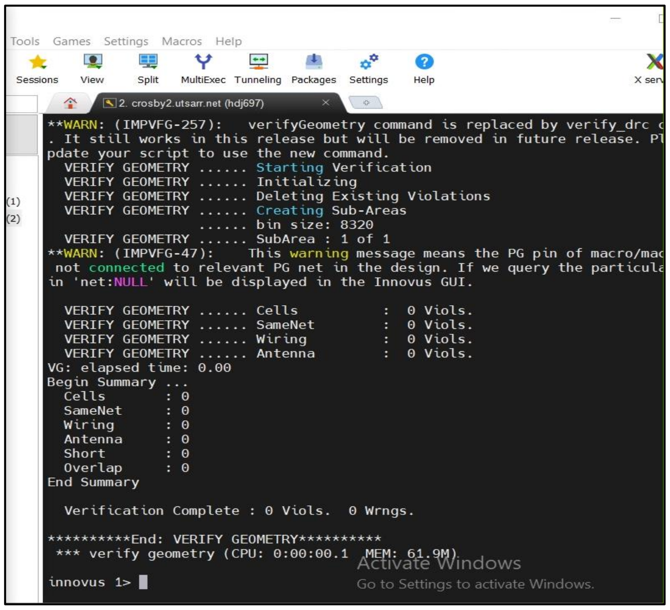
```
[NR-eGR] Total length: 0.000000e+00um, number of vias: 0
[NR-eGR] ------------------------------------------------------------
---
[NR-eGR] Total eGR-routed clock nets wire length: 0.000000e+00um
[NR-eGR] ------------------------------------------------------------
---
Early Global Route wiring runtime: 0.00 seconds, mem = 1075.0M
End of congRepair (cpu=0:00:00.0, real=0:00:00.0)
*** Finishing placeDesign default flow ***
***** Total cpu  0:0:1
***** Total real time  0:0:1
**placeDesign ... cpu = 0: 0: 1, real = 0: 0: 1, mem = 1070.0M **
Tdgp not successfully inited but do clear!
0 delay mode for cte disabled.

*** Summary of all messages that are not suppressed in this session:
Severity  ID              Count  Summary
WARNING   IMPDC-1629          2  The default delay limit was set to %d. T
ERROR     IMPDC-634           3  Failed to build the timing graph since t
WARNING   IMPSP-1760          3  Buffer footprint does not have non-inver
WARNING   IMPSP-9025          1  No scan chain specified/traced.
WARNING   IMPOPT-3000         1  Buffer footprint is not defined or is ar
WARNING   IMPOPT-3001         1  Inverter footprint is not defined or is
*** Message Summary: 8 warning(s), 3 error(s)

innovus 1> Begin checking placement ... (start mem=1074.1M, init mem=1074.
*info: Placed = 768
*info: Unplaced = 0
Placement Density:49.90%(5109/10239)
Placement Density (including fixed std cells):49.90%(5109/10239)
Finished checkPlace (total: cpu=0:00:00.1, real=0:00:00.0; vio checks: cpu
00.0, real=0:00:00.0; mem=1074.1M)
innovus 1> 
```

Layout design



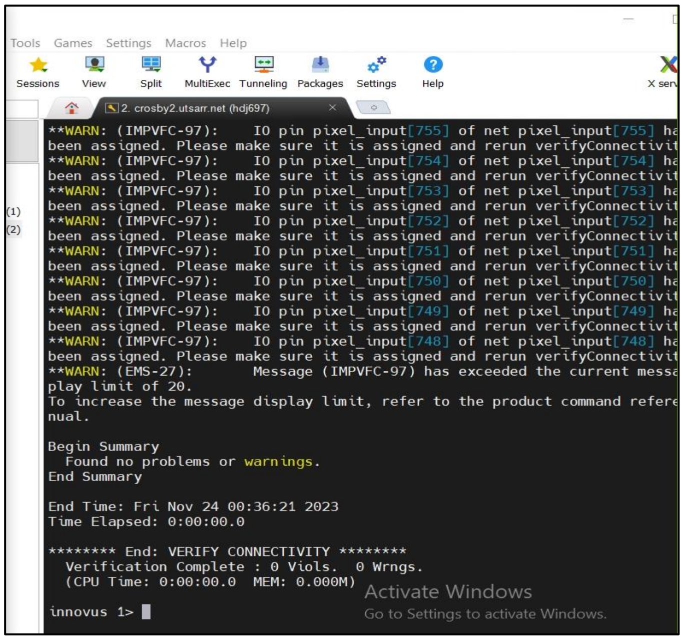
```
------------------------------------------------------------

Setup views included:
 default_view_setup

+--------------------+---------+---------+
|     Setup mode     |   all   | default |
+--------------------+---------+---------+
|           WNS (ns):|  0.000  |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |
|          All Paths:|    0    |    0    |
+--------------------+---------+---------+

+----------------+--------------------------------+------------------+
|                |              Real              |       Total      |
|    DRVs        +----------------+---------------+------------------|
|                | Nr nets(terms) |  Worst Vio    |  Nr nets(terms)  |
+----------------+----------------+---------------+------------------+
|   max_cap      |     0 (0)      |    0.000      |      0 (0)       |
|   max_tran     |     0 (0)      |    0.000      |      0 (0)       |
|   max_fanout   |     0 (0)      |      0        |      0 (0)       |
|   max_length   |     0 (0)      |      0        |      0 (0)       |
+----------------+----------------+---------------+------------------+

Density: 49.903%
Total number of glitch violations: 0
------------------------------------------------------------
Reported timing to dir timingReports
Total CPU time: 2.52 sec
Total Real time: 4.0 sec
Total Memory Usage: 1182.179688 Mbytes
innovus 1> 
```

Timing Analysis


Tools Games Settings Macros Help
Sessions View Split MultiExec Tunneling Packages Settings Help X serv
2. crosby2.utsarr.net (hdj697)

```
**WARN: (IMPVFG-257):   verifyGeometry command is replaced by verify_drc c
. It still works in this release but will be removed in future release. Pl
pdate your script to use the new command.
  VERIFY GEOMETRY ...... Starting Verification
  VERIFY GEOMETRY ...... Initializing
  VERIFY GEOMETRY ...... Deleting Existing Violations
  VERIFY GEOMETRY ...... Creating Sub-Areas
                  ...... bin size: 8320
  VERIFY GEOMETRY ...... SubArea : 1 of 1
**WARN: (IMPVFG-47):    This warning message means the PG pin of macro/mac
 not connected to relevant PG net in the design. If we query the particula
in 'net:NULL' will be displayed in the Innovus GUI.

  VERIFY GEOMETRY ...... Cells          :  0 Viols.
  VERIFY GEOMETRY ...... SameNet        :  0 Viols.
  VERIFY GEOMETRY ...... Wiring         :  0 Viols.
  VERIFY GEOMETRY ...... Antenna        :  0 Viols.
VG: elapsed time: 0.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:00.1  MEM: 61.9M)

innovus 1>
```

Geometry Evaluation


Tools Games Settings Macros Help
Sessions View Split MultiExec Tunneling Packages Settings Help X serv
2. crosby2.utsarr.net (hdj697)

```
**WARN: (IMPVFC-97):    IO pin pixel_input[755] of net pixel_input[755] ha
been assigned. Please make sure it is assigned and rerun verifyConnectivit
**WARN: (IMPVFC-97):    IO pin pixel_input[754] of net pixel_input[754] ha
been assigned. Please make sure it is assigned and rerun verifyConnectivit
**WARN: (IMPVFC-97):    IO pin pixel_input[753] of net pixel_input[753] ha
been assigned. Please make sure it is assigned and rerun verifyConnectivit
**WARN: (IMPVFC-97):    IO pin pixel_input[752] of net pixel_input[752] ha
been assigned. Please make sure it is assigned and rerun verifyConnectivit
**WARN: (IMPVFC-97):    IO pin pixel_input[751] of net pixel_input[751] ha
been assigned. Please make sure it is assigned and rerun verifyConnectivit
**WARN: (IMPVFC-97):    IO pin pixel_input[750] of net pixel_input[750] ha
been assigned. Please make sure it is assigned and rerun verifyConnectivit
**WARN: (IMPVFC-97):    IO pin pixel_input[749] of net pixel_input[749] ha
been assigned. Please make sure it is assigned and rerun verifyConnectivit
**WARN: (IMPVFC-97):    IO pin pixel_input[748] of net pixel_input[748] ha
been assigned. Please make sure it is assigned and rerun verifyConnectivit
**WARN: (EMS-27):       Message (IMPVFC-97) has exceeded the current messa
play limit of 20.
To increase the message display limit, refer to the product command refere
nual.

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Fri Nov 24 00:36:21 2023
Time Elapsed: 0:00:00.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.0  MEM: 0.000M)

innovus 1>
```

Verify Connectivity

VI. Conclusion

In conclusion, we have successfully implemented an image processing unit using Verilog, providing the capability to transform the original image into various effects, such as heightened brightness, black and white conversion, threshold application, and the Invert operation. The final chip design was accomplished using Cadence Innovus. Employing Verilog for image processing presents distinct advantages, including efficient memory-based processing, configurability, and customization. However, it is important to note that this approach demands a solid foundation in hardware knowledge, potentially posing a barrier for software developers. Verilog, well-suited for algorithms with consistent data flow, may encounter challenges with more iterative algorithms that demand complex data management. Despite these considerations, the Verilog-based approach offers a flexible and efficient solution for image processing tasks.

VII. Future work

With existing to the brightness, Invert, Threshold operation and black & white operation this project will be extending in future to the different effects such as, flipping image vertically or horizontally and the mechanism of uploading multiple images with multiple effects simultaneously.

Reference

[1] M. I. AlAli, K. M. Mhaidat and I. A. Aljarrah, "Implementing image processing algorithms in FPGA hardware," *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, Amman, Jordan, 2013, pp. 1-5, doi: 10.1109/AEECT.2013.6716446.

[2] Donald G. Bailey, "Image Processing," in *Design for Embedded Image Processing on FPGAs*, IEEE, 2011, pp.1-19, doi: 10.1002/9780470828519.ch1.

[3] Donald G. Bailey, "Field Programmable Gate Arrays," in *Design for Embedded Image Processing on FPGAs*, IEEE, 2011, pp.21-52, doi: 10.1002/9780470828519.ch2.

[4] Mathwork Vision HDL Histogram [Online]. Available:

[5] https://www.mathworks.com/help/visionhdl/ug/histogram-equalization.html

[6] AcceleratedImageProcessingonFPGAs

[7] D. P. Andini, G. Sugiarta, F. Isdaryani, F. M. Hafidh, “Image Processing Implementation on the Field Programmable Gate Array”, in *Proceedings of the 2nd International Seminar of Science and Applied Technology (ISSAT)*, Advances in Engineering Research, volume 207. 2021.

[8] Johnston, C. T., K. T. Gribbon, and D. G. Bailey. "Implementing image processing algorithms on FPGAs." In *Proceedings of the Eleventh Electronics New Zealand Conference*, ENZCon’04, pp. 118-123. 200