AN EFFICIENT FPGA OVERLAY FOR COLOR TRANSFORMATION FUNCTION USING HIGH LEVEL SYNTHESIS

Minal Deshmukh¹, Prasad Khandekar² and Nishikant Sadafale³

¹Vishwakarma Institute of Information Technology, Pune, India
²Dr. Viswanath Karad MIT World Peace University, Pune, India
³Dr. Viswanath Karad MIT World Peace University, Pune, India

¹minal.deshmukh@viit.ac.in, ²prasad.khandekar@mitwpu.edu.in, ³nishikant.sadafale@gmail.com

Abstract

Image Processing is a significantly desirable in commercial, industrial, and medical applications. Processor based architectures are inappropriate for real time applications as Image processing algorithms are quite intensive in terms of computations. To reduce latency and limitation in performance due to limited amount of memory and fixed clock frequency for synthesis in processor-based architecture, FPGA can be used in smart devices for implementing real time image processing applications. To increase speed of real time image processing custom overlays (Hardware Library of programmable logic circuit) can be designed to run on FPGA fabric. The IP core generated by the HLS (High Level Synthesis) can be implemented on a reconfigurable platform which allows effective utilization of channel bandwidth and storage. In this paper we have presented FPGA overlay design for color transformation function using Xilinx’s python productivity board PYNQ-Z2 to get benefit in performance over a traditional processor. Performance comparison of custom overlay on FPGA and Processor based platform shows FPGA execution yields minimum computation time.

Keywords: FPGA, high level synthesis (HLS), image processing, overlay.

1. Introduction

In most of the image processing applications, images are processed separately and then the data is fed to the actual application. This method proves helpful for non-real time applications where the latency does not play a vital role. The real time applications, such as lane tracking for driver assist, machine vision, HD surveillance, image compression etc., need the data to be processed in real time for decision making. The latency of the algorithm does play a vital role in these applications. The standard image processing algorithms implemented on a general-purpose CPU introduce enough latency due to which the throughput of the real time applications may get affected.

Solution to this problem is to implement the algorithm on a hardware platform. Implementation on hardware creates a dedicated block to perform the specified task. This results in reducing the CPU overhead and the latency. The functions implemented on hardware accelerate the overall execution of the algorithm. These hardware accelerators are usually implemented on a FPGA. The hardware in our study contains programming logic equivalent to Xilinx Artix 7 series FPGA and a dual-core ARM 9 processor with Python programming through Jupiter notebook web server. There are several techniques to implement a design on FPGA, one of those techniques is synthesis in Hardware Description Language (HDL). Although HDL code synthesizes an efficient hardware, but user needs to manage the timing constraints, area reduction, pipelining etc. manually. The alternate solution to HDL is High Level Synthesis (HLS), which takes input as C/C++ and converts it into HDL. Using HLS, the user needs to focus on functionality of the
architecture, other operations such as, timing, clock gating, area reduction, FSM encoding, etc., are automatically managed by HLS, which results in ease of use and allows efficient coding to accelerate higher productivity and high performance in RTL design [2].

In this study, we have implemented the color transformation function in HLS and created its custom IP core. Usually, color transformation is a pre-processing block of image processing algorithm, where the input image needs to be transformed from one color space to another color space like RGB to YCbCr or RGB to Gray etc.

The rest of the paper is organized as follows. Section 2 discusses overlay-hardware library framework, while Section 3 provides related survey for color transformation function. Section 4 comprises of implementation of color transformation function using HLS. Performance parameters are discussed in section 5, finally we discuss our conclusions in Section 6.

2. Overlay: Hardware Library

Overlay is hardware library of programmable logic circuit in the form of bit stream. Various overlays for PYNQ board are available and we can also design custom overlay in Xilinx Vivado design suite to accelerate overall performance of software application. Figure 1 below shows basic steps involved in overlay generation comprises of two parts High Level Synthesis and Vivado Design. Bitstream generated can be imported in python as hardware library.

![Figure 1. Framework for overlay generation](image)

A software programmer can use an overlay similar to a standard library functions used in high level language to call default functions. An analogous technique is used to load Overlays in FPGA dynamically, as and when needed. In this study custom overlay has been created by using Vivado HLS and Vivado integrator [3]. Custom overlay for color transformation functions is loaded from Python on demand. Custom overlay permits user to make a use of custom hardware in the FPGA without having detail understanding of ASIC style CAD tools.

3. Color Transformation

In image processing and communication color space transformation has turn out to be a most effective scheme as lesser data permits developers to do more compute intensive procedures in shorter time. Processing an image in the RGB color space, with a set of RGB values for each pixel is not practical since it requires more computation time. A mechanism for conversion between color space is must have feature in real time embedded / IoT applications.

3.1. RGB to YCbCr Color Transformation

For the applications in which color data of the image is important, RGB image is transformed into YCbCr, CMY, YUV etc. The
Fig. 2 shows YCbCr transformation transforms the RGB image into luminance and chrominance constituents. The Y component (luminance) is more sensitive to the human eye than the Cb and Cr components (chrominance). The foremost benefit of transforming the image from RGB color space to the YCbCr color space is the effect of luminance can be removed during image dealing out. Hence, the RGB image is transformed into YCbCr in JPEG compression. [11] The conversion of RGB image to YCbCr domain is given in formula below [6].

\[
\begin{bmatrix}
Y \\
Cb \\
Cr
\end{bmatrix} = \begin{bmatrix}
0.299000 & 0.587000 & 0.114010 \\
-0.168736 & -0.331264 & 0.500002 \\
0.500000 & -0.418688 & -0.081312
\end{bmatrix}
\]

Equation 1.

![Figure 2. RGB to YCbCr transformation](image)

This paper presents color transformation functions, implemented in HLS using high level language C++ and synthesized into a custom IP.

### 3.2. RGB to Gray Color Transformation

For many applications of image processing color information is irrelevant such as edge detection algorithms. The use of GRAYSCALE image removes the irrelevant color information from the image as well as it reduces the complexity of the overall algorithm. The Fig. 3 shows the RGB image GRAYSCALE image transformation technique.

![Figure 3. RGB to gray transformation](image)

For high level synthesis, color transformation function is written in C++ language. The algorithm 1 describes the flow of color transformation function.

### 4. HLS Implementation

#### 4.1. Implementation of Color Transform Functions

<table>
<thead>
<tr>
<th>Algorithm 1 Color Transformation Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require: I/O streaming interface</td>
</tr>
<tr>
<td>Ensure: Data transmission on input stream</td>
</tr>
<tr>
<td>while Input buffer not empty do</td>
</tr>
<tr>
<td>Multiply I/P data with YCbCr/GRAYSCALE matrix</td>
</tr>
<tr>
<td>Transfer result on output stream</td>
</tr>
<tr>
<td>if last pixel then</td>
</tr>
<tr>
<td>BREAK</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>CONTINUE</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end while</td>
</tr>
</tbody>
</table>

Algorithm 1.

#### 4.2. Library Functions
In this study, the color transformation functions RGB to YCbCr and RGB to Gray are implemented in C++ using the Xilinx Vivado HLS 2019.2 software tool [2]. HLS provides C and C++ libraries to permit greater efficiency and throughput in RTL scheme. Apart from that, the tool also consists of dedicated HLS libraries supporting math functions, stream interfaces, video functions, DSP blocks etc.

4.3. Simulation of Color Transformation Functions

The Vivado High-Level Synthesis (HLS) compiler offers a programming environment alike to those existing in general purpose processing unit compilers. The significant modification is that Vivado HLS compiles the C/C++ code into an enhanced RTL microarchitecture, while general purpose processor-built compilers produce assembly code to be executed on a fixed, GHz rate, processor. The simulation of RGB to YCbCr and RGB to Gray function written in high level language (C++) is done with the help of test bench. A color image is provided as input to the test bench and to get a color transformed image of the source image as shown in Fig 6 and 7.

After getting successful results of test bench in the C++ simulation, the function is synthesized. The synthesis process converts the C++ language code into Hardware Description Language (HDL). The code is synthesized for both the HDLs viz. Verilog and VHDL. This HDL code is wrapped and exported into RTL to generate the IP core.

4.4. Device used for Prototyping-PYNQ-Z2

PYNQ structure comprises of- a) Python Libraries to regulate programmable logic b) Jupyter notebook interface c) Predefined PYNQ overlays and d) preinstalled python packages. The PYNQ-Z2 board is intended to be used with an open-source framework that empowers embedded programmers to use features and the capabilities of Xilinx ZYNQ [3] to get benefit in performance over a traditional processor. The device used for prototyping belongs to the Xilinx zynq-7000 SoC family.

The color transformation function is synthesized in HLS for the device PYNQ Z-2 (xc7z020-clg400-1) SoC development board. The SoC is built on dual-core Cortex-A9 processor along with a programming logic fabric equivalent to Xilinx Artix-7 FPGA. The Processing System (PS) communicates with the Programming Logic (PL) through interfaces. These interfaces are based on ARMs AMBA AXI4 interface. Fig. 4 shows the interface between PS and PL.

![Figure 4. PS-PL interface in PYNQ Z-2](image)

The design in Fig. 5 shows color transformation model both RGB to Gray and RGB to YCbCr comprise of three important blocks: ZYNQ processing system which includes operating system peripheral interconnect of ARM A9 processor, AXI4 interconnect can be integrated in to any software running on CPU and used to communicate with the FPGA device with help of AXI memory interconnect and AXI peripheral interconnect and Custom IP core designed in Vivado HLS which can be imported in python interface of PYNQ framework. These color transformation models can be designed separately in Vivado Design Suite.
5. Results and Discussions

5.1. Test Image Results

Hardware simulation outputs for RGB to GRAY image conversion function is shown in Fig. 6. The image was tested for the RGB to GRAY conversion function in HLS. The image on the left-hand side is the original image fed to the HLS function. The image on right hand side is the result of the HLS function after gray scale conversion.

Figure 6. Hardware simulation output for RGB to Gray Conversion function

Hardware simulation outputs for RGB to YCbCr image conversion function is shown in Fig. 7 was tested for the RGB to YCbCr conversion function in HLS. The image on the left-hand side is the original image fed to the HLS function. The image on right hand side is the result of the HLS function after YCbCr conversion.

Figure 7. Hardware simulation output for the RGB to YCbCr Conversion function

5.2. Resource Utilization Report

The optimization setting is for maximum clock speed. Table I and Table II details the resource requirements of RGB to YCbCr and RGB to GRAY color conversion functions respectively. The design resource utilization reports are generated by the HLS tool. Only 3 DSP blocks were used by the design whereas, almost the whole function was implemented in flip-flops and LUTs.

Table 1. FPGA resources used for RGB to YCbCr

<table>
<thead>
<tr>
<th>Name</th>
<th>Used</th>
<th>Available</th>
<th>%Utilization</th>
</tr>
</thead>
</table>

Copyright © Authors

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731
5.3. Performance Comparison

The compute intensive color transformation function is designed and synthesized in Vivado HLS and the respective IP’s are implemented on the PYNQ Z-2 board and the execution timing is observed on FPGA, CPU with OpenCV function and CPU without OpenCV function. Table III and IV details an important attribute of study implemented for custom overlay design is the FPGA execution time is less in both color transformations which is necessity of real time image processing applications. PYNQ overlays are created in Vivado IP integrator, makes embedded design easy. Software developers can use Python environment to import custom overlay without requiring to design a color transformation overlay, which permits overlays designed in programmable logic to be controlled by python running in Processing system. This is comparable to software libraries made by skillful designers which can be used by software designers employed at the application level [3].

<table>
<thead>
<tr>
<th>Name</th>
<th>Used</th>
<th>Available</th>
<th>% Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>1882</td>
<td>106400</td>
<td>1</td>
</tr>
<tr>
<td>LUT</td>
<td>2689</td>
<td>53200</td>
<td>5</td>
</tr>
<tr>
<td>DSP48E</td>
<td>13</td>
<td>220</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2. FPGA resources used for RGB to gray

<table>
<thead>
<tr>
<th>Platform</th>
<th>Execution Time (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP on FPGA</td>
<td>0.00387</td>
</tr>
<tr>
<td>CPU (w/o OpenCV function)</td>
<td>10.3836</td>
</tr>
<tr>
<td>CPU (w/ OpenCV function)</td>
<td>0.004917</td>
</tr>
</tbody>
</table>

Table 3. Performance comparison for RGB to gray

<table>
<thead>
<tr>
<th>Platform</th>
<th>Execution Time (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP on FPGA</td>
<td>0.00377</td>
</tr>
<tr>
<td>CPU (w/o OpenCV function)</td>
<td>0.1031</td>
</tr>
<tr>
<td>CPU (w/ OpenCV function)</td>
<td>0.00788</td>
</tr>
</tbody>
</table>

Table 4. Performance comparison for RGB to YChCr

6. Conclusion

This paper presents state of art FPGA custom overlay for color transformation block using HLS to accelerate the overall performance in the real time image processing. Successful conversion of C++ code to HDL code is achieved using HLS with minimum utilization of resources and achieved lowest execution time for custom IP on FPGA. The designs were implemented on PYNQ-Z2 board. This feature gives the designer more flexibility to use custom hardware in the programmable logic without having to use ASIC style tools to balance the accuracy, speed, and power in portable devices. Custom Overlay can be loaded to the FPGA as required alike software default library. This methodology can be followed to implement more complex algorithms on FPGA using HLS.

References


