REAL TIME VIDEO STITCHING IMPLEMENTATION ON A ZYNQ FPGA SOC

By

Dhimiter Qendri,
B.Eng , University Of Ontario Institute of Technology 2017

A Major Research Project

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the program of

Electrical and Computer Engineering

Declaration of Authorship

I hereby declare that I am the sole author of this MRP. This is a true copy of the MRP, including any required final revisions.

I authorize Ryerson University to lend this MRP to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this MRP by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my MRP may be made electronically available to the public.

REAL TIME VIDEO STITCHING IMPLEMENTATION ON A ZYNQ FPGA SOC

DHIMITER QENDRI

Master of Engineering

Electrical and Computer Engineering
Ryerson University

2019

Abstract

This project details the design and implementation of an image processing pipeline that targets real time video-stitching for semi-panoramic video synthesis. The scope of the project includes the analysis of possible approaches, selection of processing algorithms and procedures, design of experimental hardware set-up (including the schematic capture design of a custom catadioptric panoramic imaging system) and firmware/software development of the vision processing system components. The goal of the project is to develop a frame-stitching IP module as well as an efficient video registration algorithm capable for synthesis of a semi-panoramic video-stream at 30 frames-per-second (fps) rate with minimal FPGA resource utilization. The developed components have been validated in hardware. Finally, a number of hybrid architectures that make use of the synergy between the CPU and FPGA section of the ZYNQ SoC have been investigated and prototyped as alternatives to a complete hardware solution.

Keyword: Video stitching, Panoramic vision, FPGA, SoC, vision system, registration

Table of Contents

Contents

List of Tables

List of Figures

List of Acronyms

ASIC	Application Specific Integrated Circuit
ADAS	Advanced Driver Assistance System
APU	Application Processor Unit
AXI	Advanced eXtensible Interface
BRAM	Block Random Access Memory
DAC	Digital Analog Converter
DOG	Difference of Gaussians
DDR	Double Data Rate
EMIO	External Multiplexed Input /Output
CMOS	Complementary Metal-Oxide-Semiconductor
FIFO	First in First Out
FPGA	Field Programmable Gate Array
FMC	FPGA Mezzanine Connector
FOV	Field of View
FSBL	First Stage Boot loader
HDL	Hardware description Language
HLS	High Level Synthesis
I2C	Inter Integrated Circuit
ILA	Integrated Logic Analyzer

I/O	Input / Output
JTAG	Joint Test Access Group
LDO	Linear Drop-Out Regulator
LED	Light Emitting Diode
LSB	Least Significant Bits
LUT	Look Up Table
MIO	Multiplexed Input / Output
MPU	Micro Processor Unit
MP	Mega Pixels
HDMI	High-Definition Multimedia Interface
SIFT	Scale Invariant Feature Transform
SURF	Speeded up robust features
SPI	Serial Peripheral Interface
SoC	System on Chip
PCB	Printed Circuit Board
PS	Processing System
PL	Programable Logic
PWM	Pulse Width Modulation
PHY	Physical Layer Interface
PMU	Power Management Unit

TRM	Technical Reference Manual
RANSAC	Random sample consensus
VGA	Video Graphic Array
VDMA	Video Direct Memory Access
XADC	Xilinx Analog to Digital Converter

Chapter 1
Panoramic vision

Machine vision is the capability of embedded systems with image sensors to extract useful information from acquired image parameters such as depth, color, 3D shape and geometric information in general. To extract the maximum amount of information from a scene, the vision system needs to have a wide field of view hence the need for panoramic imaging systems. This project focuses on the implementation and design of a real time video stitching system with semi-panoramic imaging capabilities.

1 Introduction

1.1 Objective

The main objective of this project is to explore the technical problems and find an efficient implementation of run time video image stitching from multiple camera sensors. The goal of the project is to implement on the fly merging of video streams from separate video sensors at a high frame rate in order to allow for real time tracking. The main challenges addressed in the implementation stage are the schematic capture design of a custom hardware platform with hyper-stereoscopic panoramic video capabilities, the design of custom HDL IP that implements run time video frame stitching as well as the design of IP modules used for automatic image registration which includes both rectification and seam identification.

1.2 Motivation

Panoramic image stitching has several commercial applications in diverse fields such as medical imaging, astrophotography, architecture, robotics, industrial inspection, advance driver assistance systems (ADAS) and of course in panoramic photography and film. Panoramic image mosaicking has already become a default feature on current mobile OS camera software.

Wide angle semi-panorama creation is based on image stitching. Image stitching is the process of taking a number of images from different perspectives, transforming them so that the same objects on all the images align with each other and then overlapping the images in such a way as to form a panorama based on the number of different perspectives of the same scene. The main

issue faced when implementing image stitching is how to effectively remove the misalignments due to parallax which results in visible image feature duplication as well as aligning the adjacent images in such a manner so that there is no visible seam along the boundary between the two images.

Vision processing systems work with specific frame rates subject to requirements of response time. The response time is defined as the time interval to react to an external stimulus. The response time is directly linked with the camera sensor frame rate which is an intrinsic property of the sensor. The camera sensors used on this project allow the frame rate to be changed based on the selected resolution.

The requirements for the reaction time depend directly on system constrains such as the platform moving speed, the reaction time of any transducer interfaced with the vision system and the execution time of the image processing algorithms. To maximize the reaction time, the algorithm execution time has to be kept to a minimum.

The vision system pipeline makes use of CMOS camera sensors that incorporate a DSP image processing engine. These sensors are provided either with external optics or with integrated optics. The camera lens determines the optical properties of the sensors. The lens purpose is to focus the lights that is reflected or emitted from the object under the FOV. This light forms the image in the camera sensor. The lens size determines the FOV and the working distance.

The main limitation of current imaging systems is that most camera sensors have a very limited field of view which is typically 55-65 degrees. While there exist special optical imaging platforms with much wider field of view such setups are expensive and suffer from astigmatism. This incurs deformations on the acquired images which have to be fixed either by additional optics or by adding further steps in the image processing pipeline thereby decreasing reaction time.

To solve the issue of limited FOV as well as minimize reaction time one needs to implement an image stitching algorithm that executes fairly fast with regards to system reaction time in order to meet real time requirements. In general, this requires making use of hardware implementations which can parallelize calculations to speed up execution delays.

1.3 Tasks

The difference between a video imaging system and a vison system is that the former contains no image processing. Vision systems on the other hand extract information from the acquired image frames and find sense from them. This in effect requires integrating an algorithm in the image processing pipeline. The main task of this project is the implementation of a vision image processing pipeline capable of run time stitching of video frames at 30 frames per second.

The primary task of the project was the implementation of a custom IP that receives the individual video streams from each camera and implements run time video stream stitching.

The second task was the schematic capture design of a custom hardware platform centered on the ZYNQ 7020 SOC. The platform main novelty is the implementation of a custom catadioptric system consisting of four separate cameras together with a mirror assembly forming a hyper-stereoscopic imaging system.

The third project task was the investigation and implementation of automatic image registration algorithm using a mixed approach by using a hybrid intensity and features-based approach. The main challenge of this task was the development of the algorithms for identification of correspondence points between adjacent image features. The registration process consists of image rectification and seam identification. For each of these steps a number of custom HDL IP modules were implemented. The simulation of each module as well as the Verilog code are included in Appendix A. The last task was the analysis of the developed modules.

1.4 Panoramic Imaging

The imaging systems are generally classified as active or passive. Passive imaging systems make use only of passive optical sensors and elements such as mirrors and lenses. Active imaging systems on the other side make use of active elements such as projectors, collimated light sources and active light sources. Active stereo system includes structured light systems which make use of a single projector and camera or time of flight systems make use of a

collimated light source such as a laser and a monocular camera. The imaging systems in this project fall under the passive systems since no use is made of any active light sources.

There exists a multitude of methods for creating panoramic images. These systems can be categorized on the number of imaging sensors they use; such as monocular cameras, stereoscopic cameras, hyper-stereoscopic cameras and so on.

Panoramic cameras in themselves are classified in two main categories, dioptric and catadioptric. Dioptric imaging systems use only refractive elements such as lenses. This includes rotating cameras, fish-eye based cameras and camera clusters. Catadioptric systems make use of single curved mirrors or flat mirrors paired with multiple cameras.

Current smartphone monocular camera systems support the creation of panoramic images by implementing on the fly software stitching of the acquired image. This requires that the user manually moves the camera in the direction of the panorama. A similar setup can be automated by using a motorized approach to manual movement. This approach however reduces the reaction time as well as the system reliability. In addition, it adds a host of other issues such as the need for image stabilization due to motion jitter.

As such preference is given to a catadioptric static imaging platforms that make use of multiple video sensors and reflective elements. Integration of mirrors in the optical setup increases the limited field of view of the camera sensors. If placed strategically, mirror elements can also simplify the algorithmic complexity of the system.

Recovering spatial information from a single 2D image is in general an ill-posed problem mathematically. To aid in the recovery of depth information stereo systems make use of the differences in between image key-point features on the same view in order to infer a depth disparity map. Inferring depth from a monocular camera is challenging so multiple camera sensor are needed.

Passive panoramic stereo systems use two monocular cameras to infer a depth map from difference in parallax. The setup of a stereo machine vision includes two cameras that are separated by a distance d called the baseline of the system. The projection of the same view on

two different surfaces allows the underlying algorithm to estimate the depth of the objects in the field of view. The baseline defines the depth resolution achieved by the system.

1.5 Project Organization

The remainder of this project report is organized as follows. Chapter II shows an exposition of the most common methods and techniques used in video stitching. A general survey of past image stitching implementations on FPGA platforms is included. The aim of this chapter is to select the most efficient approach for the selected hardware platform subject to the engineering constraints after analyzing the existing approaches. This section also presents an overview of the main milestones of the project as well as the original work contribution.

Chapter III is focused on the algorithm presentation and explanation. The first part of Chapter III presents an implementation of a software centric approach for image stitching. The main deficiencies of this approach are analyzed and identified with respect to the requirements and constraints. In the second part the architecture of the proposed HW image stitching algorithm is presented, together with block diagrams. This section also gives a description of the designed hardware platform.

Chapter IV describes the experimental setup and the specific implementation details on the selected FPGA SOC. Furthermore, a detailed explanation of each module is described together with simulation results. The experimental results together with the demonstration and comparison of algorithm with prior approaches are shown in Chapter V. This chapter also shows timing diagrams for each of the custom HDL modules. Finally, in Chapter VI, a summary of the main results from the project is provided together with potential avenues that require further optimization.

Chapter 2
Related works

This chapter presents the main theory behind semi-panoramic vision covering the main techniques in use as well as a short survey of the field with applications to FPGA systems.

2 Panoramic Imaging

A panoramic image can be synthesized by stitching together multiple images. The images can be taken by a single rotating camera sensor, multiple camera sensors or special panoramic cameras.

In the most general case, when two images of the same scene are taken from two different cameras placed at arbitrary angles with respect to each other the pixel corresponding to the object 3D reference point gets mapped to different image coordinates. The pixel coordinate can be mathematically described as a combination of a Euclidean transform and a perspective transform.

The experimental setup designed for this project contains three cameras on a flat PCB spaced a distance d apart from each other. The differences in parallax with respect to each camera do not include changes in observation angle. For cameras placed on a flat plane such as the PCB, there is no projective transform so the setup is simplified. This simplifies the compositing surface since it's a simple plane and no projective 3D camera rotation transformations are needed so perspective transform can be removed from the calculations. In turn this simplifies the homography matrix.

Image stitching is the process of taking a number of images from different perspectives, transforming them so the features on all the images align with each other and then merging the images in such a way as to form a panorama based on the number of different perspectives of the same scene.

The main problem is the identification of the corresponding features on two adjacent images as well as transforming one image with respect to the adjacent one so that

There are a number of steps required to obtain a panoramic image from individual images. The cameras have to be line locked so that the same frame is acquired at the same moment from all

different cameras. The central procedure to image stitching is image registration. This is the process of aligning the images so that stitching can be performed.

There are a number of approaches to image registration which can broadly be categorized as direct that is intensity based or feature based [1].

Direct methods make use of an operator for image alignment. One example of direct methods are wavelet transform based techniques which consider the multiresolution representation of each image on different frequency sub-bands [18]. Feature based approaches are dominant in image stitching techniques. All feature-based techniques make use of key-points. The key-point is defined as a sparse set of features which is locally present on all the image perspectives. Key-point matching is the procedure of finding a correspondence between these features from image to image.

The homography matrix models the appropriate geometrical transformations to each respective image that relates the location of a specific pixel coordinates between different image perspectives. In this chapter a quick review of the two main methods is going to be covered. Then a number of recent and previous works that focus on hardware implementations of image stitching.

2.1 Problem Analysis

To stitch two images taken from different perspectives one has to determine the corresponding features that both images share. Then one of the images has to be transformed so that the perspective matches with the image that it will be stitched with. The two images then have to be positioned so that the overlapping areas are fully removed and the seam between the two images does not have any visible visual artifacts. To accomplish the above step a rectification procedure has to be applied so that the same feature pixel on both images is at the same level. Even if the rectification procedure is perfect and the images are aligned correctly, one can still observe visual artifacts along the seam if the two cameras are focused differently.

Estimating the seam where the two images start to share object features can applying multiband seam blending algorithms is one method of removing these visual artifacts. There are a number of seam estimation algorithms. Algorithms that optimize the pixels based on energy functions such as the color gradient along the seam. The naïve seam blending method uses the median value and the average of the overlapping pixels. Another method is to use center weighting or use multi band pixel blending. Automatic seam identification can be done independently from rectification and it can also be integrated as a precursor step to determining the epi-polar line.

Rectifying the two images basically requires determining a couple of corresponding pixels on both images. This can be accomplished by matching intensity features so that the two brightest pixels on the rights side of image A, correspond to the two leftmost side brightest pixels on image B.

These methods are classified as direct methods. The main disadvantage of intensity-based methods is that they do not offer the required accuracy due to differences in lighting. As such a naïve implementation will return many false positives which make the procedure inadequate.

The next approach is to determine features known as key-points. A key-point is defined as a sparse set of features which is locally present on all the image perspectives. This requires pre-processing of each image or at least sections of images in order to identify these descriptors.

A survey of the most common feature-based image stitching algorithms is given in the section below. The main disadvantage of feature-based techniques is the computational complexity of such approaches.

What can be observed by both aforementioned techniques is that there needs to be a balance between computational complexity of the applied algorithms and the accuracy and execution time that is required for the task at hand.

Another problem with cameras is lens distortion. This involves barrel distortion where the pixel coordinates are displaced away from the center of pincushion distortion where the pixel coordinates are displaced toward the center of the image. The approach taken in this thesis is to avoid any warping transformations and simply shift the images along the x and y directions relative to each other until they are fully aligned.

2.2 Publications

In [2] Popovic et al present an FPGA implementation of image blending using a hemispherical polydioptric system. The work mainly focuses on comparison of the blending techniques. Results show that the nearest neighbor technique results in visible artifacts. The linear blending technique is partially better at removing the visual artifacts. The best performance is obtained by applying a restricted Gaussian blending which applies a Gaussian filter only near the seam area.

In 2015 Disney Research [4] demonstrated an algorithm that uses local warping that allows for robust stitching with minimal parallax artifacts. This method allows for spatiotemporally stable panoramic video stitching. In [16] Lu et al. perform image warping using an efficient implementation of line buffers.

In [5] Shieh et al, present a video stitching algorithm that makes use of the OpenCV framework as well as applying the Difference of Gaussians method (DoG) to construct a scale space that shows the extreme points. After identifying the image descriptors and applying affiant transformation the stitched images are obtained.

In [7] Kawanishi et al make use of a six-camera setup coupled with a hexagonal pyramidal mirror in order to acquire stereo views. The camera system makes use of Tsai's method in order to restore the radial distortion of each camera image. In [8] Nayar presents a catadioptric omnidirectional camera system while in [13] and [14] the authors show applications of catadioptric system to robotic platforms.

In [12] Kar-han Tan et al present a panoramic camera that makes use of a mirror pyramid. The novelty of this approach is the mirror pyramid which forms a virtual camera with a very wide field of view.

2.3 Patents

There are plenty of patents that cover motion-based image stitching. While the patents are not open with regards to the algorithmic techniques, they still provide information on the applied methods. In [3] Apple shows how to implement motion-based image stitching. In addition, a number of fabless silicon providers also offer custom image stitching IP. The implementation details behind these IP cores however are not open. As an example, in [9] Omintek demonstrates an IP capable of warping the individual frames to allow seamless stitching. The IP is cable of dealing with a maximum of up to 8 different video sources.

2.4 Feature based methods

Fundamental to the procedure of image stitching in feature-based approaches is the detection of key-points. Image key-points are image features that contain particular criteria such as being invariant to image scaling, image rotation, 3D camera viewpoint, illumination of noise. The special point about these features is that they are invariant under transformations hence they are known as local invariant descriptors. Key-point matching is the procedure of finding a correspondence between these features from image to image. A number of different algorithms have been proposed for determining image key points.

Some of the most used techniques are a) Harris Corner Detector b) SIFT (Scale Invariant Feature Transform) c) Speeded up Robust features. It should be noted that SIFT and SURF algorithms are patented [24].

Feature based approaches for image stitching share the same algorithmic steps. The main differences reside in the types of algorithms that can be implemented in each step.

The primary step is the feature pointe detection. The next step is compiling the feature point descriptors. The third step is feature point matching. The fourth step is the calculation of the homography matrix. In general, this uses an algorithm that leverages the feature vectors to obtain a homography matrix. The homography matrix models the appropriate geometrical transformations to each respective image that relates the location of a specific pixel coordinates

between different image perspectives. It is used to apply transformations to the image so that the features in different images are aligned on the same plane.

The fifth step is implementation of image stitching by joining the warped images obtained from the previous step. The last step is image blending in order to remove any visual artifacts that are present. The next phase is matching the key-points on images taken from different perspectives.

The sixth step is applying the homography matrix to each image apart from the reference image and obtaining the warped images. The fifth step is rectifying the images so that each image is aligned along the same epi-polar line. This will require moving the images along the vertical direction until they are aligned.

The seventh step is merging the images so that on the generated panorama all the features in the different perspectives are closely aligned in such a manner so that no overlap or visual artifacts can be observed.

There are a number of methods used to obtain feature point descriptors. Corners, blobs, edges and ridges can be used as features. A quick summary of the most common methods is shown below.

2.4.1 SIFT

The Scale Invariant Feature Transform (SIFT) algorithm is invariant to image scale and rotational transformations. This algorithm finds the scale space extremums and localizes the key-point in an image frame. The algorithm is quite robust against changes in noise or image distortions. An FPGA implementation is shown in [19].

After finding a set of key-points, a region is selected around each key-point. The next step is to compute the gradient magnitudes and orientations around each key-point. From the computed gradients a histogram is created with each sample weighted by the gradient magnitude.

The rotational invariance of each descriptor is obtained by rotating the descriptor coordinates and the histogram orientation relatively to the dominant orientation. The key-points are then matched to a database of key-points by using the nearest neighbor criterion based on the Euclidean distance of the vector.

Four random points are selected and the Homography matrix is computed based on the solution of the linear equations. Then the image points are projected on the image plan of image I2 using the computed homography matrix Hi. The distance of inliers is then calculated based on a selected threshold.

These calculations are repeated k times by selecting the best homography matrix with the maximum number on inliers. An algorithm used for this computation is the RANSAC algorithms. An FPGA implementation of RANSAC is given in [20].RANSAC stands for Random Sampling with Consensus. The image stitching stage computes the homography transformation between two adjacent frames I_j and I_k. After transforming Image I_j then the images I_j and I_k are blended together. The main issue is to find the optimal boundary between the regions of overlapped pixels between the two adjacent images that reduce the visual artifact.

2.4.2 SURF

The main problem with SIFT is that it's fairly slow. Speeded Up Robust Features (SURF) is another patented local feature descriptor that speeds up the detection of features [25].

SURF makes use of blob detectors to determine image key-points. The main disadvantage of SIFT and SURF is the significant memory requirements and the fact they these algorithms are under patents. SURF has been implemented on FPGA platforms [23], [24].

2.4.3 Harris Corner detector

The Harris corner detector is a mathematical procedure used to determine corresponding features between two different images. The elements are determined from patches of fixed size. The task essentially is to find the best similar patches on two frames that are taken from a different perspective. An efficient implementation is shown in [10].

The Harris Corner detector leverages the use of corners which are defined as junctures of contours. Corners are excellent feature key-points since they are very stable from changes of viewpoints. They can be recognized easily by looking at intensity values of a small patch.

Any shift of the patch location should yield a change in appearance. The Harris corner detector is given by Equation 2-1 below:

$$E(u,v) = \sum_{x,y}^{\infty} (w(x,y)[I(x+u, y+u) - I(x,y)]^2) \quad (Eq\ 2-1)$$

Where W is the window function and the second term represent the difference between a shifted version of Intensity and the original value.

The second term will be zero for constant patches while for distinctive patches this will be larger so it can be seen that E(u,v) will be large for corners. The Harris detector can be framed as a matrix multiplication operation between the windowing function and the components of the intensity gradient.

For small shifts using a bilinear approximation one can obtain the form shown in equation 2-2:

$$E(u,v) \approx [u,v] M \begin{bmatrix} u \\ v \end{bmatrix} \quad (Eq\ 2-2)$$

The matrix M is a 2x2 matrix computed from image derivatives as shown in equation 2-3:

$$\mathrm{M} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} (Eq\ 2-3)$$

The gradients I_x and I_y are computed using the Sobel operator. In practice the windowing function is Gaussian with sigma = 1. In algorithmic form the Harris corner detector is composed of the following steps:

a) The first step is the computation of the gradients. This requires computing the x and y derivatives of the image which is implemented by convolutions between the 3x3 pixel

window and the vertical and horizontal gradient mask. In this step the input image is converted into an x and y derivative map as given in equation 2-4 below:

$$I_x = G_\sigma^x * I \qquad I_x = G_\sigma^y * I \quad (Eq\ 2-4)$$

b) The second step is Gaussian smoothing. This is done by leveraging the gradient images computed on the previous steps. Computing the products of the derivative at every pixel as shown in equation set 2-5 below.

$$\begin{aligned} I_{x2} &= I_x * I_x \\ I_{y2} &= I_y * I_y \quad (Eq\ 2-5) \\ I_{xy} &= I_x * I_y \end{aligned}$$

c) The third step is computing the Harris measure which serves as an indicator of which pixel may be a corner pixel. This requires computing the sums of the products of the derivatives at each pixel as shown in equation 2-6.

$$\begin{aligned} S_{x2} &= G_{\sigma'} * I_{x2} \\ S_{y2} &= G_{\sigma'} * I_{y2} \quad (Eq\ 2-6) \\ S_{xy} &= G_{\sigma'} * I_{xy} \end{aligned}$$

For each pixel a Harris measure matrix H(x,y) is defined as in equation 2-7.

$$H(x,y) = \begin{bmatrix} S_{x2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y2}(x,y) \end{bmatrix} (Eq\ 2-7)$$

d) The fourth step is the application of thresholding. The Harris Corner Detector uses the determinant and trace to find a factor R which represents the "corner score" for each pixel as given in equation 2-8.

$$\mathrm{R} = Det(H) - k\big(Trace(H)\big)^2 \ (Eq\ 2-8)$$

e) The last step is the application of non-maximum suppression criterion. Based on the threshold of the computed R value one can compute the non-max suppression.

Implementations of Harris corner detector on FPGA platforms can be found in [1] and [2]. The main disadvantage of the algorithm is that it's computationally expensive due to its sequential nature. Since it is not known beforehand where the corners may be located on a captured image the algorithms need to run. On the other side this algorithm offers the most accurate detection of key-points even when images are noisy.

2.5 Feature based stitching algorithms

Feature based algorithms for synthesis of panoramic image from individual frames require a number of sequential steps. First the cameras have to be line locked so that the same frame is acquired at the same moment from all different cameras. The second step is to identify features in these images known as key-points. The special point about these features is that they are invariant under transformations hence they are known as local invariant descriptors. The next step is to match the key-points on images taken from different perspectives. The third step is to use an algorithm that leverages the obtained feature vectors to obtain a homography matrix. The homography matrix is used to apply transformations to the image so that the features in different images are aligned on the same plane. The fourth step is applying the homography matrix to each image apart from the reference image and obtaining the warped images.

The fifth step is rectifying the images so that each image is aligned along the same epi-polar line. This will require moving the images along the vertical direction until they are aligned.

The sixth step is to merge the images so that the generated panorama all the features in the different perspectives are closely aligned in such a manner so that no overlap or visual artifacts can be observed.

2.6 Design decisions

The main requirement for the implementation of frame stitching in this project is to meet reaction time constrains and real time operation.

The main issue with feature-based techniques is that they take considerable resources and execution time. Resource usage is directly related with power consumption which does not bode well for mobile implementations. On the other side execution time would violate the real time operation requirements of the system.

One idea is to pursue a mixed approach where one can calculate feature only on specific regions to decrease the computational complexity. Imagine both images are partitioned into tiles as shown in the Figure 2-1 below.

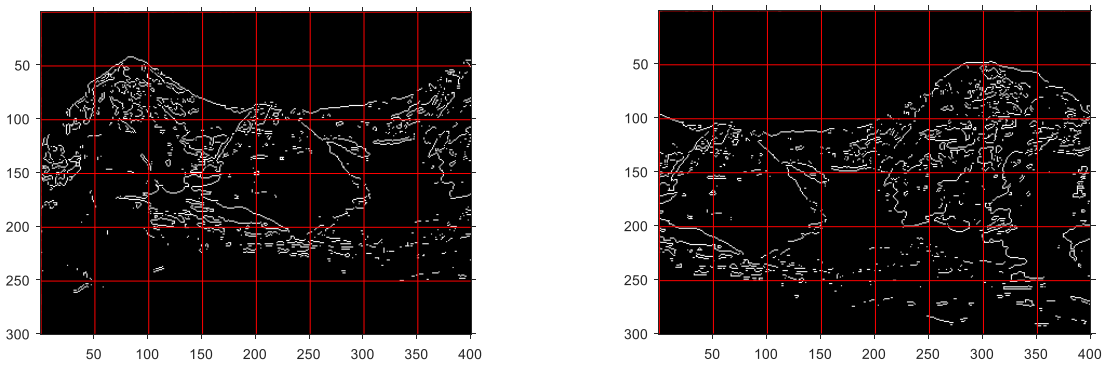


Figure 2-1 Algorithm only operating on tiles

The algorithm can then run only on specific tiles on both images. To avoid implementing feature-based algorithms such as SIFT, SIRF or equivalents one can borrow steps from the feature-based approach and then use intensity-based algorithm to perform the process.

As an example, one can apply an energy gradient filter to specific adjacent tiles of each image. Then a key-point search can be performed only on these tiles. Based on the detected highest energy gradient, key-points one can then perform rectification.

This removes the need for direct intensity-based key-point search which is suspect to erroneous results due to lighting conditions. On the other side this also removes the need to rely on resource intensive algorithms such as Harris corner detector and represents a middle ground that uses the least amount of resources while still operating within the real time requirements.

Chapter 3
Proposed approach

3 Proposed Approach

3.1 ZYNQ SOC Overview

The lab prototyping system is based around a Zedboard development board which uses a ZYNQ 7010 SoC. The ZYNQ SoC couples a dual core ARM Cortex A9 denoted as the Processing System (PS) with the FPGA subsystem known as the Programming Logic (PL) in a single die. The ARM Cortex A-9 CPU includes 256 KB of On Chip Memory (OCM), including a number of peripherals as well as external memory interfaces such as DDR3L, NOR flash and SD-card.

The PS has a number of multiplexed I/O denoted as MIO and grouped in banks that start with a 50x designation. Some of the MIO can be routed to the PL I/O side via EMIO. The MIO are used to connect the ARM core to external peripheral and devices such as a) Serial Flash, b) NAND Flash c) UART d) I2C sensors e) USB PHY OTG f) Gigabit Ethernet PHY g) SD card.

The location of all these peripherals is pre-determined by the I/O map and specified in the ZYNQ TRM. The PL section itself provides a lot of I/O that are grouped in Banks denoted as BANKx where x is any of (13,34,35, 55).

The processor on the PS section always boots first and can be used independently from the PL section. The PL can be configured as part of the boot process using the First Stage Bootloader (FSBL) or it can be configured later from the PS. The PL supports partial reconfiguration.

3.2 Proposed Hardware

The lab hardware uses three OVM7690 cameras which interface with the ZYNQ via the FMC connector. The spacing of the cameras is close to 8 cm. This spacing was dictated by the size of the mirrors mounted between the cameras. The distance however represents a problem due to changes in parallax between adjacent cameras. In addition, this spacing incurs changes in intensity. A custom hardware platform was designed to overcome these limitations of the current hardware. The idea was to place the cameras much closer and use smaller mirrors.

Since the Zedboard is a relatively big development board it was required to design a custom solution that incorporated all the required hardware while taking the least amount of space.

The hardware is designed around a ZYNQ XC7020 in a 10CLG484 package. This is the same SOC package that is located on the Zedboard. The block diagram in Figure 3-1 shows the main schematic design blocks of the system.

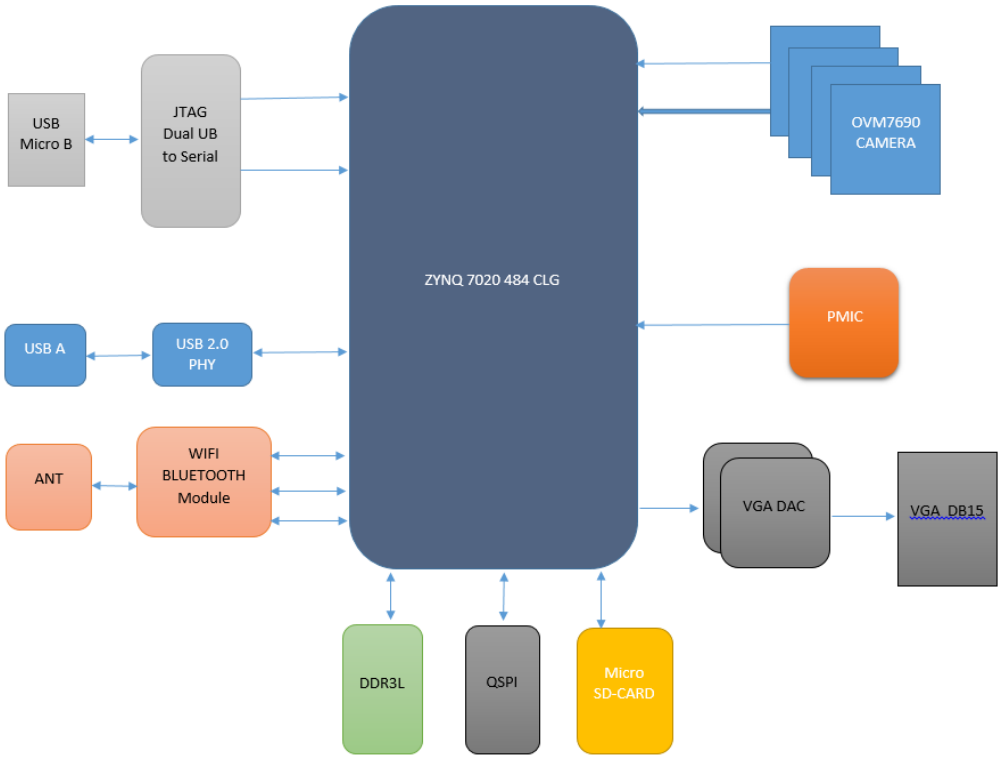


Figure 3-1 Proposed block diagram of the custom hardware platform

Bank 0 of the SOC contains the JTAG programming signals connected to the PS side, the XADC pins and the programming indicator pins. All JTAG signals use pull-up resistor and transient suppressing diodes. Bank 13 of the ZYNQ on the PL side is equipped with a number of pins which can be used to interface with differential signals. Both Banks 13 and 33 are used to interface with the external VGA DAC peripheral. Bank 34 and 35 of the ZYNQ are used to interface with the OVM7690 cameras. To simplify the hardware the voltage level for these banks

was set to 2.5V since this is within the operating limits of the camera I/O pins which specify from 1.8V to 2.8V.

Bank 502 is used to interface the DDR3L RAM with the PS side. Bank 500/501 contains user pins as well as the bootstrap pins. The bootstrap pins are sampled during bootup and allows the ZYNQ to boot from Quad SPI flash (QSPI) , SD card or JTAG. SD card memory is connected to SDIO 0, pins 40-47.

A 33.33 MHz crystal oscillator is used to source the PS Clock. A 24 MHz clock is used to source the USB OTG PHY. A USB3320C PHY is used to add USB OTG to the system.

The I2C peripheral are used to configure the cameras and the PMU chips. Both I2C1 and I2C2 pins use 0ohm jumpers between each SDA and SCL line. Both lines then are pull up to VDDIO by 10 K resistors.

Two push-buttons are included, one is the PS reset push-button. The other is a PS switch. There are 2 external reset pins. PS_SRTS_B pin for soft reset. Bootstrap pins are not sampled and device is aware of previous status. PS_POR_B pin for reset. Bootstrap pins are sampled and BootROM runs from the APU. There are 7 boot mode strapping pins using MIO pins. Each of these pins are connected using 20K pull-ups or pull-downs. Pins [4:0] are used to select the BOOT mode.

Pins [1:0] VMODE are used to select the I/O voltage levels for the MIO voltage Banks. VMODE[0] controls MIO pins 15:0 while VMODE[1] controls MIO pins 53:16. A pull-up resistor selects the LVCMOS18 while pulldown selects LVCMOS25 which is compatible with LVCMOS33.

The PS section makes use of a DDR3L memory bank that is composed of one 16-bit width chip. The chip has a size of 2 GB and operates at a frequency of 533 MHz The DDR3L SDRAM is connected to the PS section via BANK 502. The termination resistors and a termination voltage LDO are also included as part of the schematic design.

The power management unit is composed of two highly integrated chipsets from Infineon specifically designed for ZYNQ SOC's. The power sequencing of the various PS and PL power rails is done automatically and can be configured via the I2C bus.

3.3 External peripherals

The rest of the design blocks describe the external peripheral of the system.

3.3.1 Camera sensors.

The camera sensors make use of 4 OVM7690 cameras that are interfaced directly with the SOC. Configuration of the cameras is done directly from ZYNQ using an I2C switch. The camera has two power domains. The analog rail domain is fed by a 2.8V LDO with a current capacity of 200mA. The digital domain can run from 1.7V to 3V. Since the ZYNQ banks already run at 1.8V the camera I/O are also set to operate at 1.8V. Each camera is fed AVDD rail is connected to the filtered output of a single 2.8V LDO. The OVM7690 is controlled by an SCCB bus, which is functionally equivalent to the I2C bus. Since all cameras have the same I2C address an I2C switch is used to allows independent programming of each camera.

3.3.2 WIFI BLE module

An WIFI / BLE combo module is used to allow the ZYNQ SOC to send images wirelessly via WIFI. The chip interfaces via the SDIO protocol with the ZYNQ. Configuration is done using a serial port with flow control. The schematic of this peripheral is comprised of the analog frontend with the antenna matching circuit and the digital section comprising the IC and two LED's used for activity notification.

3.3.3 VGA DAC

Two VGA DAC with an 8-bit resolution were used. The DAC's are connected to DB15 VGA connectors. The decision to use VGA as opposed to HDMI was made simply on cost.

3.3.4 USB

A USB PHY was added allowing the ZYNQ to interface will external USB 2.0 peripherals.

3.3.5 JTAG and Serial Port.

The ZYNQ SOC is programmed from an FTDI2232 JTAG programmer. Port A of the FTDI chipset is configured as a JTAG programmer while port C is configured as a serial port. Port B and D are left un-used. The FTDI2232 is connected to an external EEPROM which is used to store configuration settings.

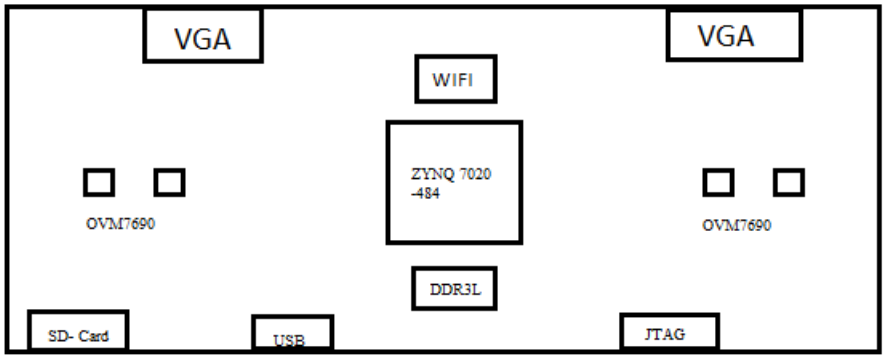


Figure 3-2 Proposed placement for designed hardware

The part placement for the PCB layout is suggested to follow the diagram shown in Figure 3-2 above.

3.4 System BOOT

The ZYNQ SOC follows a multi stage boot. The primary stage is the BOOT ROM that is not user accessible. The BOOT ROM reads the bootstrap MODE pins to determine from which memory device to boot, then it determines if the boot is secure or not and performs system cleanup and initialization. BOOTROM is executed from internal ROM code memory.

It copies the FSBL from external NOR flash memory to the internal SRAM. When the boot code is not available in SPI NOR FLASH the BOOTROM tries to boot from the JTAG. The next step is to jump to the FSBL which is executed from the on-chip SRAM.

3.5 Experimental Setup

The most basic setup consists of two video camera sensors denoted as C1 and C2 placed a distance d apart as given in Figure 3-3. In the most general case the cameras occupy a position in 3D space which defines their angular relation with respect to one another. The projection of a 3D point in each camera plane corresponds to different points M1 and M2. Rectification consists of finding a common line through E1 and E2 called the epipole. Points R1 and R2 are the focal places of the camera and line R1R2 is called the baseline. When the baseline is parallel to the X axis in an XY coordinate system epipolar lines are horizontal and parallel with respect to each other.

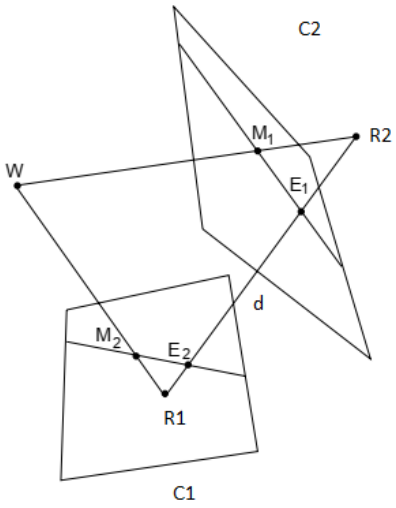


Figure 3-3 General case for epipolar projection

To simplify the analysis from the general case, the setup dictates that both cameras are placed on a flat surface as shown in Figure 3-4. The distance d is typically in the range of 10-15 cm. Provided that the separation distance falls within the FOV of the camera sensor one can observe that the images taken by the two cameras at any time share some common parts.

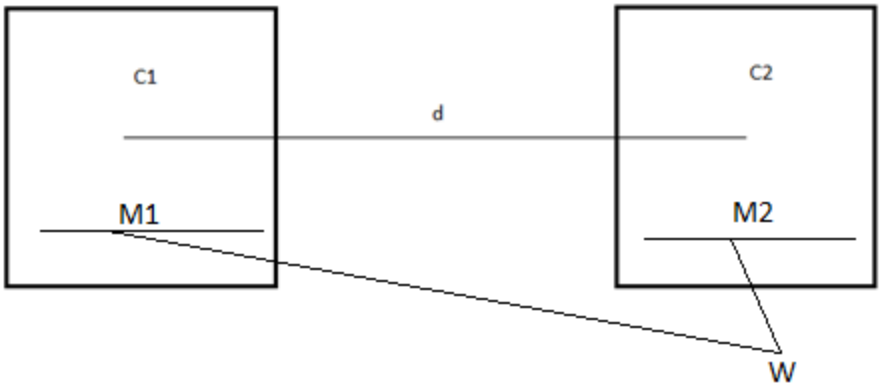


Figure 3-4 Simplified case of epipolar projection for lab setup

Assuming camera C1 is positioned to the left and camera C2 is positioned to the right one can observe that the right side of camera C1 shares similar image features with the lefts side of camera C2. In panoramic video stitching, the problem is how to combine the two images in a single image so that the overlapping sections are merged in a single video frame. In addition, to properly implement rectification the same point must also have the same vertical coordinate in both projections.

Due to parallax, the location of the same feature as seen from the two cameras is different. The advantage offered by parallax is that the overlapping visual field is used to extract scene depth perception. Since the angle of projection of the same object is different for each of the two cameras, a number of image features such as intensity of pixels, light exposures and object alignment are also different.

In this project the main aim is to stitch the images without addressing issues like pixel intensity and light exposure and camera synchronization. The goal of the project is to implement image stitching so that the synthesized panoramic image does not exhibit any visual artifact.

The experimental setup designed in the lab contains three cameras on a flat PCB spaced 10 cm apart from each other as shown in Figure 3-5. The parallax is defined as the angle between the two cameras FOV.

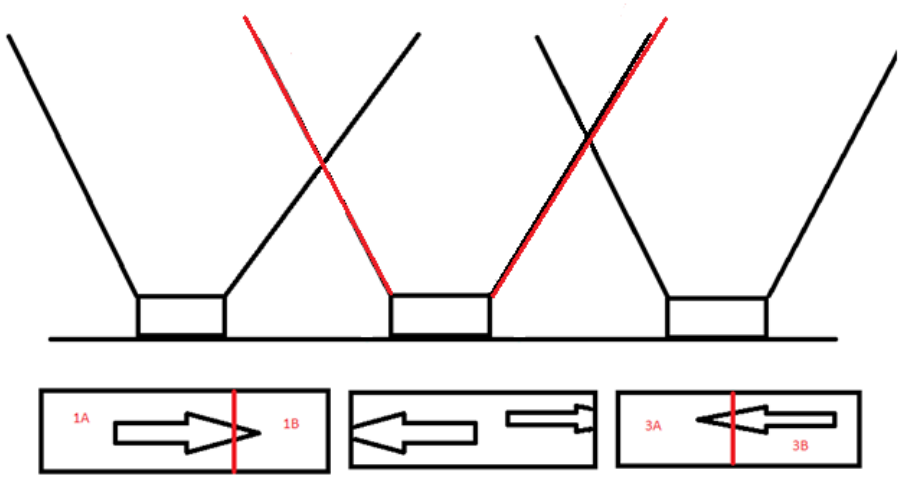


Figure 3-5 Catadioptric optical system for panoramic imaging

The sketch in Figure 3.5 presents the main idea behind the proposed catadioptric system. Three camera sensors are located on a planar surface equally spaced among each other. All cameras on the above setup are precisely positioned with respect to one another. The cameras operate at the same pixel clock and are sourced from clocks which are in phase with each other.

Between each camera there is a mirror slanted at an angle α with respect to the normal. The angle is chosen in such a way as to minimize the overlapping features between the adjacent images but not completely remove them. The overlapping visual fields from each camera are used to extract features for stitching the images as well as extracting depth information.

In the most general case, when two images of the same object are taken from two different cameras placed at arbitrary angles with respect to each other, the pixel corresponding to the object 3D point gets mapped to different image coordinates. The pixel coordinate can be mathematically described as a combination of a Euclidean transform and a perspective transform. For cameras placed on a flat plane such as the PCB, there is no projective transform since the homography matrix is much simples compared to the general case which includes rotations so the setup is simplified. This simplifies the compositing surface since it's a simple plane and no perspective transform, i.e projective 3D camera rotation transformations are needed when implementing the algorithm. The differences in parallax with respect to each camera do not include changes in observation angle.

Another problem with cameras is lens distortion. This involves barrel distortion where the pixel coordinates are displaced away from the center or pincushion distortion where the pixel coordinates are displaced toward the center of the image. The approach taken in this thesis is to avoid any warping transformations and simply shift the images along the x and y directions relative to each other until they are fully aligned.

The task of the proposed image stitching algorithm is to create a panoramic image based on the above setup. To do so the image stitching IP has to discard part of sides images and join the remaining part with the central frame.

To implement the above task, the image stitching IP needs to have a-priori knowledge of the index location of the seam. This is the line that demarcates the overlapping features on the side images with the parts that need to be flipped from the part that needs to be discarded. This requires that the column index of the seam is provided as input to the module.

In addition, the left and right images need to be rectified with respect to the center image. All images must be rectified so that the same feature overlapping two images shows on the same scan line. The rectification procedure mitigates the vertical shift that is induced on the camera's sensors due to misalignments and from perspective changes.

3.6 Proposed Algorithm

The proposed image registration techniques for implementing image stitching are based on a hybrid approach making use of efficiently calculating image features by filtering the frames and using intensity changes to localize them. Due to the presence of mirrors, it is suggested to decouple the registration process in two separate steps. The first step is the determination of the seam index location. The second step is the rectification process which aligns the adjacent frames in the vertical position with respect to one another. While the above steps can be merged into one single module which outputs the x and y coordinates of a single feature shared between two adjacent images, the current work follows a decoupled approach. FPGA image rectification has been implemented in various forms as in [24], [25], [26], [27].

3.7 Seam index identification

The optimal boundary location for overlapping two adjacent image frames is defined as the seam. For cameras placed on a flat horizontal surface we assume that the seam is a vertical line that demarcates overlapping features between the two adjacent images. In the most general case, the cameras are undergoing relative motion so the seam index has to be recalculated to account for small differences due to vibration at periodic intervals. Image registration algorithm implemented in FPGA are shown in [17], [21], [22].

Other approaches include dynamic programming, graph cuts and Dijkstra shortest path algorithm. All these approaches however require significant resource utilization.

The proposed method implements seam index identification automatically on the FPGA. The idea is to implement an IP module that provides column indices that corresponds to the seam location on the corner frames. In addition, the seam column index of the right and the left frame is provided as an input to the image stitching IP. This allows the IP to be dynamically configured at runtime by using an AXI-Lite Wrapper around the image stitching IP. The PS can then write to the respective registers and update the seam index of each of the cameras.

One option that requires minimal resource utilization is to place an LED between the two cameras, mounted on the mirror. The LED serves as a key-point by being the brightest pixel in the FOV of each camera. Key-point detector modules are used to determine the brightest key-point on each frame. In theory, the column index of the feature should correspond to the seam index. The output of the key-point detector is the line and column of the key-point. This can be used for both rectification and seam coordinates estimation. In practice the implementation of this techniques has multiple shortcomings.

First, the light source should be mounted in the field of view of two cameras. Since a mirror is placed between the two cameras one needs to either mount the LED on the corner of the mirror so that it is still visible in both frames or on a horizontal bar that does not impede the FOV of either camera.

Second, while the LED intensity can be regulated by using a PWM, using a single key-point in practice proves inadequate since different image features on the two frames may have greater intensity than the LED. Third, the LED may obscure visual features.

Nevertheless, an HDL IP was implemented that searches for the brightest key-point in a frame area and returns the key-point column coordinates. A testbench and the IP code are shown in the Appendix A.

A second option for automatic seam column index identification is to pursue a pixel energy-based approach. An energy map can be implemented using different methods such as the gradient magnitude or entropy. The pixel energy map is given by equation 3-1:

$$e(x,y) = \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right| \quad (Eq\ 3-1)$$

Each pixel is assigned an energy measure as given by the gradient operator. This can be any of the well-known filters such as Robert, Prewitt, Sobel, Laplacian coupled with erosion or dilation of the obtained features or using a more sophisticated edge detector such as the Canny edge detector. An efficient Sobel implementation is shown in warping.

Another approach to gradient energy is to compute the map of histogram of oriented gradients for each image area adjacent to each other and locate the indices of the highest energy. To implement this method, one need to add a streaming gradient operator at the output of two adjacent frames and search for the highest energy pixel on the output.

3.8 Automatic seam detection algorithm

To automatically determine the seam column index, the following algorithm is proposed to be implemented in the PL as a custom IP. Assuming two cameras C1 and C2, the first column of the C2 camera frame is taken as the reference template. From prior knowledge from mirror positioning we know that the overlap between camera C1 and C2 is 20-30% starting from rightmost side of frame C1 as shown in Figure 3-6.

The proposed algorithm steps are as follows:

1. Compute energy gradient of columns x to FRAME_WIDTH for frame C1.

2. Compute energy gradient of column 1-2 for frame C2.
3. Calculate Euclidean norm between all column pixels of template and column i of C1
4. Calculate sum for each step i above and save it in memory
5. Implement a state machine that searches for the minimum value for each of the columns i to FRAME_WIDTH.

Theoretically the entry for a match would be zero assuming the frames are vertically rectified. In practice one picks the histogram entry with the smallest value assuming a minor shift in the y direction. The bin index with the lowest value is the seam index where the frames from A1 and B1 start overlapping. Figure 3-7 shows how the two image matrices are stitched together assuming they are already rectified.

64	66	110	132	**97**	**205**
157	239	245	69	**132**	**220**
55	66	20	136	**99**	**87**
159	172	3	31	**159**	**219**

97	**205**	176	179	101	58
132	**220**	92	47	111	56
99	**87**	37	118	87	186
159	**219**	107	13	42	221

Figure 3-6 Two sample image matrices (1 left, 2 right) with overlapping columns

64	66	110	132	**97**	**205**	176	179	101	58
157	239	245	69	**132**	**220**	92	47	111	56
55	66	20	136	**99**	**87**	37	118	87	186
159	172	3	31	**159**	**219**	107	13	42	221

Figure 3-7 Stitched images

33	31	-13	-35	**0**	-108
-25	-107	-113	63	**0**	-88
44	33	79	-37	**0**	12
0	-13	156	128	**0**	-60
33	31	-13	-35	**0**	-108

Figure 3-8 Calculated bins for each operation

Figure 3-8 above shows that the second entry has the lowest value hence it is identified as the seam index. The index of the seam is identified as I = IM_W – 2. This gives the index of the seam with respect to the image column itself.

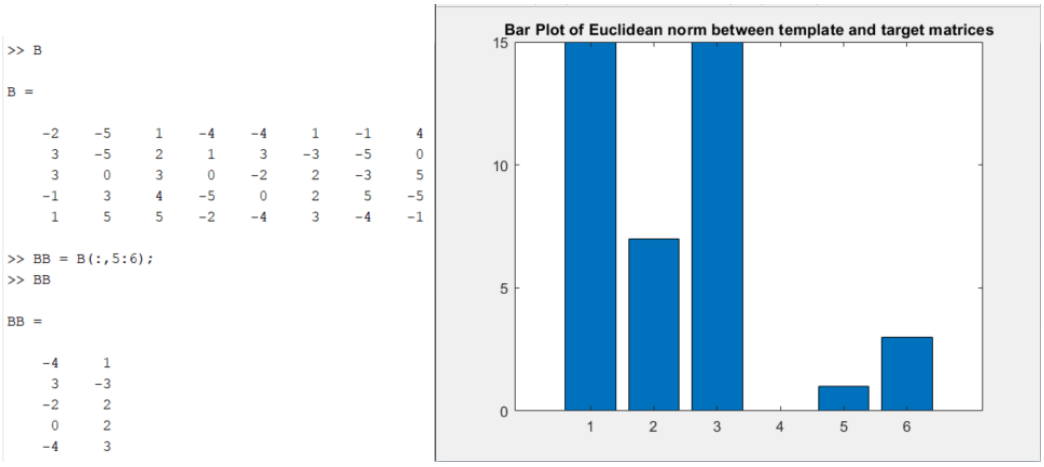


Figure 3-9 Determining seam index by Euclidean norm

A simulation was implemented in MATLAB. Matrix B and BB are used as the target and template matrices respectively as given in Figure 3-9. The bar plot is a histogram representing the index of the column where the seam appears. The simulation above assumes that the images are fully rectified so that there is not discrepancy in the vertical direction.

3.9 Rectification

The lab hardware consists of three cameras soldered on the PCB. Each camera carries intrinsic distortions due to minor differences in lens geometry. In addition, there are extrinsic distortions arising from small mis-alignments from the PCB assembly. FPGA image rectification for stereo setups has been shown previously in [28].

This results in a mismatch of the epi-polar lines which in practice results in adjacent image features which are shifted vertically when transitioning from one camera view to the adjacent as shown in Figure 3-10 Aligning the epipolar lines needs to be implemented automatically during runtime at periodic intervals to account for motion induced artifacts.

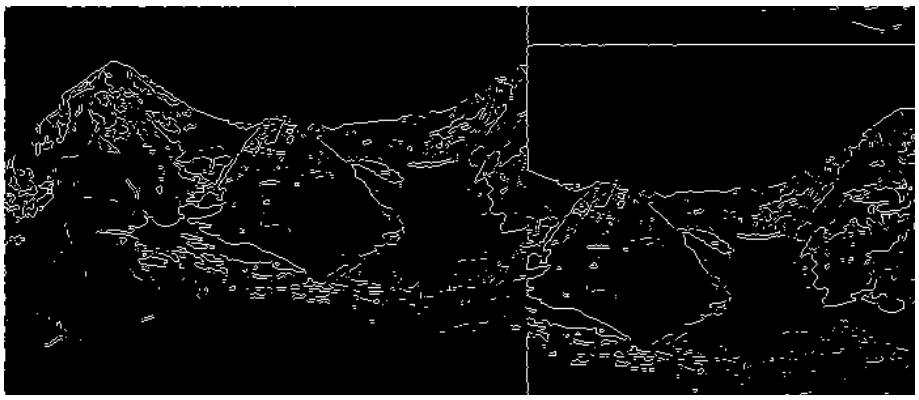


Figure 3-10 Unrectified Sobel filtered adjacent frames

The algorithm needs to determine a key-point feature and align the images such that the feature shows up on the same vertical dimension on the overlapped areas. The main idea behind the online rectification is to determine the coordinates and height of the key-point on two adjacent frames and shift the frame coordinates vertically automatically so that the epipolar lines are aligned along the y direction.

The procedure requires implementing a module that outputs the row coordinates of the key-point feature as well as the height of the key-point feature for both left, right and center images. The left and right frames are then shifted vertically to match with the center frame.

As seen on the diagram above the same image is shown on the left and center cameras as well as on the right and center camera. Two tasks need to be implemented to properly stitch the reflected images. First one needs to rectify the image so that the same features show up on the same scan line. This requires vertically translating the left and the right image with respect to the features on the center image.

Second one needs to determine the seam column index of the left and the right cameras where the image features overlap. This index is calculated separately for the left and the right cam. The image rectification task is considered independently from the seam index identification.

The IP assumes a camera assembly with three sensors as seen in the lab. In a camera assembly with four video sensors, the column index is calculated with respect the adjacent cameras as seen from left to right.

3.10 Proposed firmware modules

The main architecture of the image stitching system is shown on the block diagram below. The current system is designed for three cameras. The main firmware modules are:

a) Camera input capture module
b) Line buffer module
c) Sobel filter
d) Image stitching IP
e) Seam identification IP
f) Rectification IP
g) VGA controller IP

The VGA controller IP will take the input from a BRAM and interface with an external DAC. The DAC has a resolution of 12 bits with format RGB444 so the image format has to be converted from RGB565 to RGB444 by pruning the (Least Significant Bits) LSB.

3.10.1 Camera controller

The camera controller was prototyped using an OV7670 camera module. The main difference compared to the OVM7690 used in the lab is the lack of integrated optics since the sensors come with external lenses. The rest of the camera memory map and functionality is similar to the

OVM7690 since they belong to the same family. Description of the camera firmware is given in the next chapter.

3.10.2 Line Buffer

The pixel stream from the camera is sent sequentially so each of the pixel is received in a serial fashion. To make use of multiple pixels and apply image processing operations to the received data one needs to store multiple pixels temporarily. A line buffer is the mechanism that implement this. For a 3x3 filter, the module creates 2-line buffers that store up to IMG_LEN pixels, where IMG_LEN is the image length. In addition, it also stores a 3-element buffer. Upon start of the pixel stream, the 3x3 windows starts to slide once the pixel index gets incremented.

3.10.3 Sobel Filter

The line buffer is a sub-module used as the first stage when applying a 3x3 convolutional filter. To accommodate the small amount of BRAM used on the Minized development the filters were implemented as streaming modules. The Sobel filter uses the masks shown below.

3.10.4 Seam identification IP

The proposed algorithm is to store the calculated energy gradient obtained from the Sobel filter in two BRAM units. BRAM one will store the last third of image 1 while BRAM 2 will store the energy gradient of the first two columns of the adjacent image. The seam identification IP will read the two BRAM and calculate a Euclidean distance between the first columns of the second image and the columns of the first frame. Foreach column the sum of the Euclidean distances will be calculated and stored in an array. A state machine will continuously update the index of the column with the shortest Euclidean distance. The seam identification module takes as arbitrary inputs the start column index of the first image.

3.10.5 Rectification IP

The next module is the rectification IP. The idea is to compare the pixel energy density of the overlapping columns of the two frames in order to determine if they are shifted vertically. This IP module requires that the result of the seam identification IP specifying the column index is already available. This informs the module of the start column index where the two images

overlap. The modules perform a linear search for the brightest pixel along the index column of the seam on image one and a linear search of the Sobel filtered column of Image two.

3.10.6 Image Stitching

The next step is to use the image stitching to merge the mirrored images. The module takes as input the frame images from all three cameras. Each camera frame is stored on a separate BRAM. The IP also takes as input the seam column index for frame one and seam column index for frame three. It also takes as input the rectifier output index for image 1 and image 3. This allows the module to stitch all three images in a panoramic image by stopping the row read at the seam index and vertically shifting both sides images with respect to image 2 which is the center image.

3.11 OpenCV Implementation

OpenCV is a high-level image processing library which also contains a number of algorithms used for feature-based image stitching. To make use of the patented algorithms one has to install the contribution package by issuing:

pip install opencv-python==3.3.0.10 opencv-contrib-python==3.3.0.10

A panorama image class was created. This class has a number of methods. First one has to detect the features and key-points. To do so the images are converted from BGR color scheme to gray scale images. A list of descriptors is then created using the SIFT method. Then a dictionary pair of keypoints and features is created. The key-points are converted to floating point numbers. The compute_Homography() method makes use of the RANSAC method explained before to compute the homography matrix using the determined points. The details of the methods are shown in reference [5]. The method takes a threshold value as argument which is used to represent the error according to equation 3-2:

$$\|dstPoints - convertPointsHomogenous(H * srcPoints)\| > ransacReprojThreshold \quad (Eq\ 3-2$$

The determined key-points are matched using the method called matchKeypoints(). This method creates a list of valid matches and if the points from images A and B are larger than fours it starts creating the homography matrix.

The main method of the class is the image_sticth() method. This function takes as input both images that are to be stitched, a constant known as Lowe's ratio and the threshold. The Lowe's ratio is used to perform a nearest neighbor ratio test to determine pixel similarity.

After detecting the features and key-points the next step is to obtain the valid key-point matches. One the homography matrix is calculated based on the returned values, it is passed to a function which warps the perspective of the images and joins them together in a single panoramic image.

3.12 OpenCV testing

The above program was implemented in Python leveraging a third party open-source script and the OpenCV library. The execution time of the script depends directly on the size of the images. The Python program is given in Appendix B. The program reads two input images as shown in Figure 3-11 and outputs a key image as given in Figure 3-12 and the final stitched image as shown in Figure 3-13.

Figure 3-11 Two different perspective images of the same scene

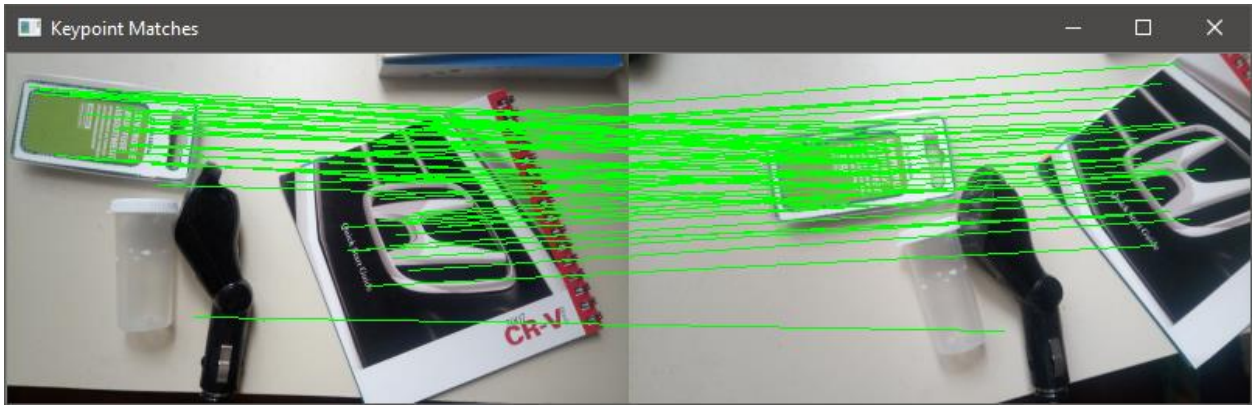


Figure 3-12 Matching features with SIFT

Figure 3-13 Synthesized panoramic image

Table 3-1 below shows the running times and the maximum achievable frame per second for varying resolutions. It should be noted that running times are not 100% identical in different systems due to differences in cache size and CPU speed.

Table 3-1 Frame stitching execution time

Resolution	**Time**
320x480	0.42 sec
640x480	0x56 sec

2304x4096	0.88 sec

It is clear from the table above that the SIFT algorithm is very slow for real time panoramic image stitching. While there have been attempts to port the SIFT algorithm for panoramic image stitching to a FPGA platform as explained in [6] this algorithm takes a significant amount of resources. Since the aim of this work is to find and implement a resource efficient implementation of image stitching [29] it was decided not to pursue this implementation further.

Chapter 4
Implementation

4 Implementation

4.1 Hardware setup

The vision processing hardware setup and the developed IP modules were prototyped on a Minized ZYNQ SoC development platform. The Minized uses a XC7Z007S SOC which couples a single core ARM A9 MPU with a 7-th series architecture FPGA which has equivalent functionality with Artix-7 FPGA's. The programmable logic (PL) section of this SoC has 23K Logic cells and 1.8 Mb of BRAM.

The initial task of the project was the implementation of an image pipeline using a VGA resolution camera based around the OV7670 CameraChip module from Omnivision. The acquired images are then directly piped to a VGA IP module which drives an external VGA DAC connected to the Minized via two PMOD connectors. The developed OV7670 HDL IP modules were ported from VHDL to Verilog based on work done from [11]. The complete prototyping hardware environment is shown in Figure 4-1 below:

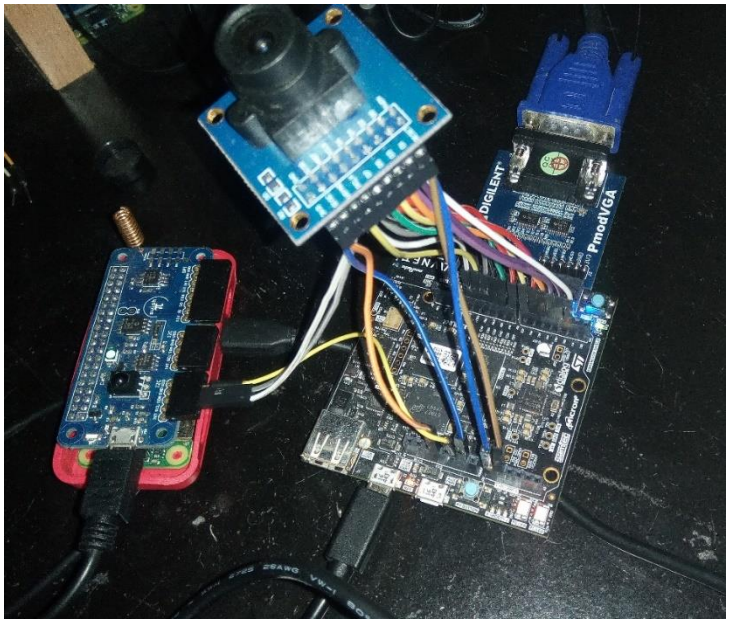


Figure 4-1 Vision processing hardware setup

Apart from the SoC development board, the two main hardware components are the camera module and the VGA interface. The implementation of the camera interface firmware and hardware IP modules constituted a major portion of the work on this project.

4.2 Camera module

The OV7670 camera module is a low-cost camera sensor with a parallel interface and external optical lenses as shown in Figure 4-2. The camera supports VGA resolution up to 0.3 MP.

Figure 4-2 OV7670 camera module

The camera supports multiple resolutions with support for up to 60 frames per second for sub-VGA resolutions as shown in Table 4-1. To accommodate the constrained amount of BRAM on the selected FPGA SoC all the experiments were performed using QVGA resolution. The advantage of this choice being that one can test vision processing algorithms at 60 fps.

Table 4-1 OV7670 Camera sensor resolutions and FPS

Format	Resolution	Frame per second
VGA	640x480	30 fps
QVGA	320x240	60 fps
CIF	350 x 240	60 fps
QCIF	176 x 144	60 fps

From a functional perspective, interfacing the OV7670 camera sensor with the PL requires implementing the pixel data capture logic as well as a module responsible for the camera configuration.

4.2.1 Camera Configuration

The configuration module is responsible for sending the configuration commands to the camera. This is done via the Serial Camera Configuration Bus (SCCB). The SCCB is a two-wire interface which is mostly compatible with the I2C protocol, the main difference being that the logic levels are LVTTL compatible. The SIOC pin outputs a clock which operates at a typical frequency of 100KHz -400KHz. The configuration data is sent via the SIOD pin. Configuration of the camera by the SCCB bus can be implemented in multiple ways. One can either implement the SCCB as a state machine in the PL section, use an AXI IIC IP core, use the embedded I2C peripheral on the PS section or implement the protocol in firmware by using a bit banging technique using AXI GPIO and the EMIO pins. Another flexible approach is to use the I2C detect suite from a Linux environment in conjunction with a PL IIC peripheral. The last option requires deployment of a Linux OS on the MPU as well as a custom device tree.

Camera configuration via an SCCB HDL module does not allow for a flexible configuration since changing the setting will require recompiling the design. This is needed in order to dynamically modify the camera settings during testing of the frame stitching algorithm. The next option is to use an IIC core in the PL to control the camera via I2C like commands from the PS section. This has the disadvantage of consuming LUTS from the fabric.

The second option is to leverage the built in I2C peripheral of the PS. The ARM side of ZYNQ contains two embedded I2C peripherals. Ultimately, this was the approach followed since it allows making use of the least amount of resources. The only pins that have external pull-ups on the dev-board are the SDA and SCL pins.

The complete pinout of the camera sensor is shown on Table 4-2 below.

Table 4-2 OV7670 Camera pinout interface

PIN	Direction	Functionality
VDD	INPUT	3.3 V Power Supply
GND	INPUT	Digital GND
SIOC	OUTPUT	Two-wire Serial Interface clock
SIOD	INOUT	Two-wire Serial Interface data
VSYNC	OUTPUT	Active High, Valid frame, Vertical Sync denoting active frame
HREF	OUTPUT	Active High, Line/Data valid Indicates active pixels
PCLK	OUTPUT	Pixel clock output from Sensor
XCLK	INPUT	Master Camera Clock
D[7:0]	OUTPUT	Data bus
RESET	INPUT	Active HIGH, Reset
PWDN	INPUT	Power Down Active Low

As seen on the pinout table the camera has two additional control pins apart from the SCCB bus pins. The RESET pin is active low and should be held high during operation. The power down (PWDN) pin is active high and can be left un-connected or tied low to remove the camera out of idle mode. Next is the XCLK pin which must be fed from a clock source with a frequency in the 12-25 MHz range. A very important note is that the SCCB bus won't acknowledge requests if the camera is not fed with this clock signal.

This clock source can be either generated on the FPGA using a PLL from a clocking wizard or from an external crystal. The last pin is the pixel clock PCLK signal which is output directly from the external sensor. This is a free running clock which operates in sync with the HSYNC signal. The functional block diagram of the camera is shown on Figure 4-3 below.

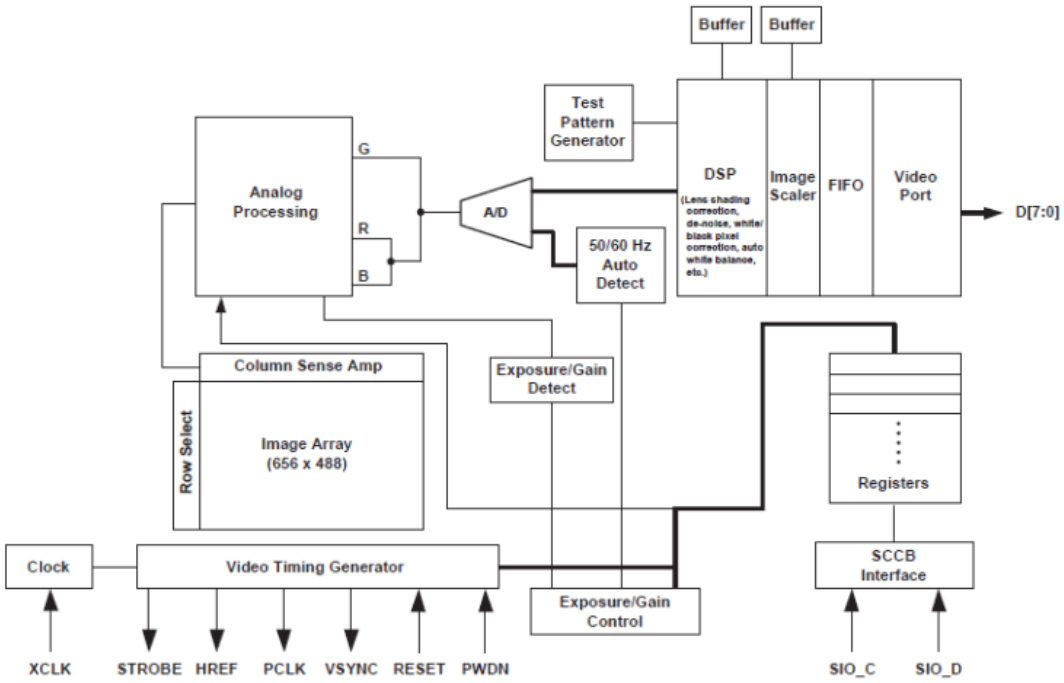


Figure 4-3 Functional Block diagram of the camera sensor

The OV7670 uses an 8-bit parallel bus interface in conjunction with synchronization strobes VSYNC and HSYNC which denote active row and active frame respectively.

4.2.2 Pixel parsing

The data capture module is responsible for capturing the pixel data from the parallel bus interface at the rising edge of the pixel clock. The pixel transfer output starts with an HREF transition. The first pixel takes 3 cycles while the rest of the pixels take 2 cycles as shown in Figure 4-4. The parsed data is encapsulated in a single 16-bit pixel.

The OV7670 camera supports multiple image formats such as Raw Bayer, Processed Raw Bayer, YUV mode, YCbCr 4:2:2, RGB555, RGB565, RGB444 and GRB. The pixel parsing code is implemented to decode only RGB565. Each pixel has 16 bits of data with green color being encoded in 6 bits while red and blue take 5 bits respectively as shown in Tables 4-3 and 4-4.

Table 4-3 First byte during first cycle

DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
R4	R3	R2	R1	R0	G5	G4	G3

Table 4-4 Second byte during first cycle

DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
G2	G1	G0	B4	B3	B2	B1	B0

The OV7670 contains a DSP engine inside the chip which support some basic image processing. The camera allows mirroring the image by respectively flipping the scan order. To mirror the image horizontally, bit [5] of register MVFP (address 0x1E) has to be set to 1. This functionality is very important for the image stitching IP in order to avoid flipping the received pixels along the Y axis on the FPGA.

The main problem with the OV7670 camera is that the datasheet contains inaccurate and absent information so a complete register value set needs an NDA from the supplier. The current register configuration settings were taken from the open-source Linux driver of the camera.

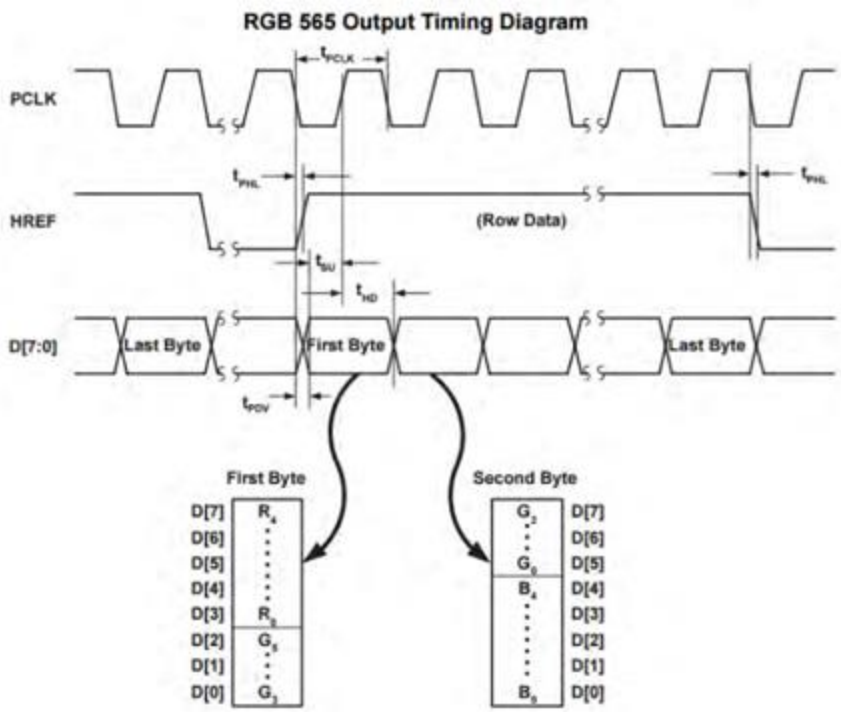


Figure 4-4 Data transmission of a single pixel RGB565 pixel data

The HSYNC signals a new frame; default values for this signal are active high. VSYNC signals a new frame, pixels are read only when VSYNC is low. The default setting for VSYNC is active high.

4.3 VGA Module

An external R2R DAC VGA PMOD module was used to interface the Minized development platform with a VGA display. The PMOD connector interface is given in Figure 4-5. The DAC has a resolution of 12 bits with 4 bits for each of the red, green and blue pixel bits respectively. To use the RGB565 pixel format with the DAC requires pruning each pixel to an RGB444 format. The VGA controller generates the HSYNC and VSYNC timing pulses that determines the row and frame refresh rate of the image. When the HSYNC signal is high the pixel values are displayed on the current row on the monitor.

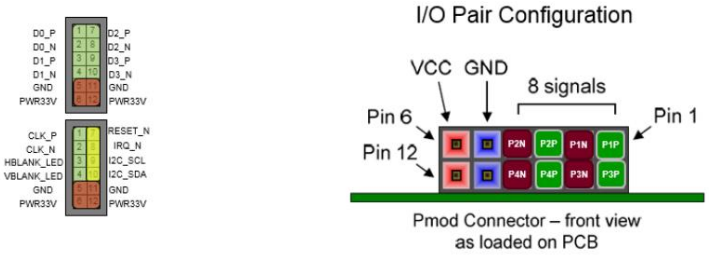


Figure 4-5 PMOD VGA interface connector

Tables 4.5 and 4.6 show the pinout of the DAC with respect to the dual PMOD connector used on the Minized. A custom XDC constraint file was generated to interface the DAC. The architecture of the VGA IP module relies on the generation of the synchronization signals shown in Figure 4.6.

Table 4-5 J4 PMOD on Minized

Minized GPIO	Pinout	Pin Number	Function
K13	D2_P	1	Red0
L13	D2_N	2	Red1
N13	D3_P	3	Red2
N14	D3_N	4	Red3
L15	D0_P	7	Blue0
M15	D0_N	8	Blue1
L14	D1_P	9	Blue2
M14	D1_N	10	Blue3

During this operation from the first row to last row the VSYNC signal is high. This signal then stays low for two-pixel clock cycles until the next frame start to display. The pinout for the VGA peripheral is shown on the table below.

Table 4-6 J5 PMOD on Minized

Minized GPIO	Pinout	Pin Number	Function
P13	D0_P	1	Green0
R14	D0_N	2	Green1
N11	D1_P	3	Green2
N12	D1_N	4	Green3
P15	D2_P	7	HS
R15	D2_N	8	VS
R12	D3_P	9	NC
R13	D3_N	10	NC

Two free running counters were used to determine the VSYNC and HSYNC pulses. The two counters also determine the current address for the video RAM. In the setup above this is the address of PORT B of the dual port BRAM. The VGA module was implemented to allow for VGA, QVGA and QQVGA resolutions. The industry standard timing for VGA resolution running at 60Hz is given by the Tables below. The constants specify the front porch, sync pulse and back porch for both horizontal and vertical signaling. The IP uses the following parameters and clock. The VGA resolution parameters are given in Table 4-7 below while the VGA timing diagram is given in Figure 4-6.

Table 4-7 VGA resolution parameters

SCREEN REFRESH RATE	60 HZ
VERTICAL REFRESH	31.46875 kHz
PIXEL FREQ.	25.175 MHz

SCANLINE PART	PIXELS	TIME [µS]		
VISIBLE AREA	640	25.422045680238	480	15.253227408143
FRONT PORCH	16	0.63555114200596	10	0.31777557100298
SYNC PULSE	96	3.8133068520357	2	0.063555114200596
BACK PORCH	48	1.9066534260179	33	1.0486593843098
WHOLE LINE	800	31.777557100298	525	16.683217477656

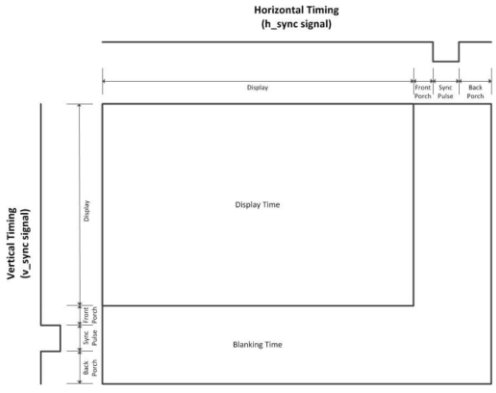


Figure 4-6 VGA timing diagram

The VGA module is fed from a 25.175 MHz clock allowing a 60Hz refresh rate. The clock is sourced from a clocking wizard which in turn is sourced from the FCLK_CLK0 output of the ZYNQ SOC clocked at 50MHz.

The pixel clock frequency is calculated as given by equation 4-1:

$$\begin{aligned} Pixel\ Clock &= Frame\ Clock\ Cycle * Frame\ rate \quad (Eq\ 4-1) \\ &= Line\ Per\ Clk\ Cycle * Lines\ Per\ Frame * Frame\ rate \\ &= 525 * 800 * 60\ fps = 25\ MHz \end{aligned}$$

The DAC takes the 12-bit digital signal and converts it into an analog signal. Since an R2R DAC with 12bit resolution only 4096 possible colors can be displayed. The VGA physical layer is an analog protocol so the output signals from the DAC are 0-0.7V analog signals. 0V represents black while 0.7V represent white. The DAC uses a DB15 connector to interface with the display.

The VGA IP module was configured to output QVGA on a VGA screen by keeping the same timing constants for a 640x480 resolution but modifying the output window.

A simple test was performed by using the ZYNQ block diagram shown in Figure 4-7 which outputs a standard test-pattern on the screen. An Integrated Logic Analyzer was added to verify the timing of the protocol.

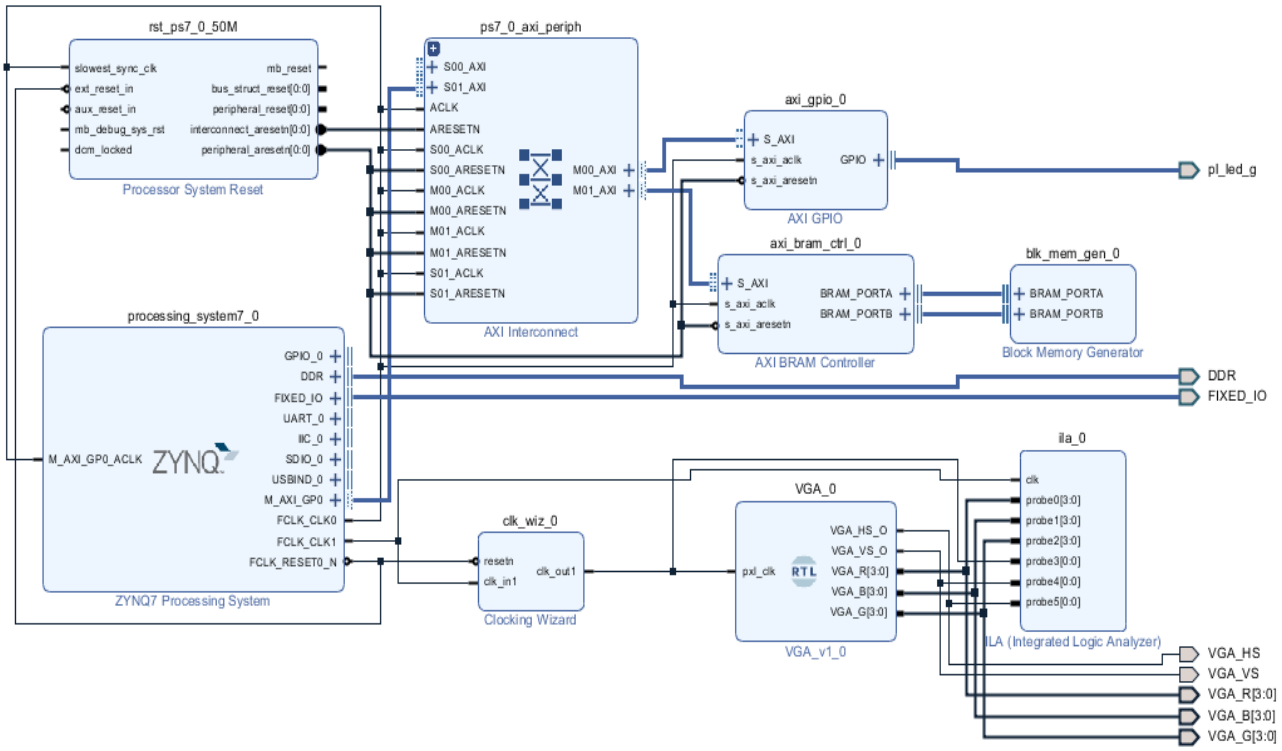


Figure 4-7 VGA setup with a test pattern and ILA

4.4 Camera Testing

To test the image pipeline a block diagram was implemented where the camera would stream data to an output display as shown in Figure 4-8. The OV7670 capture module is interfaced directly with a dual port BRAM. Port A of the BRAM is clocked from the pixel clock PCLK.

On every incoming pixel the address is incremented. Port B of BRAM is clocked by the VGA module which consumes the generated pixel stream. An Integrated Logic Analyzer (ILA) IP was used to capture and verify the camera data as shown in Figure 4-9. The same procedure was also used to verify the operation of the VGA module as shown in Figure 4-10.

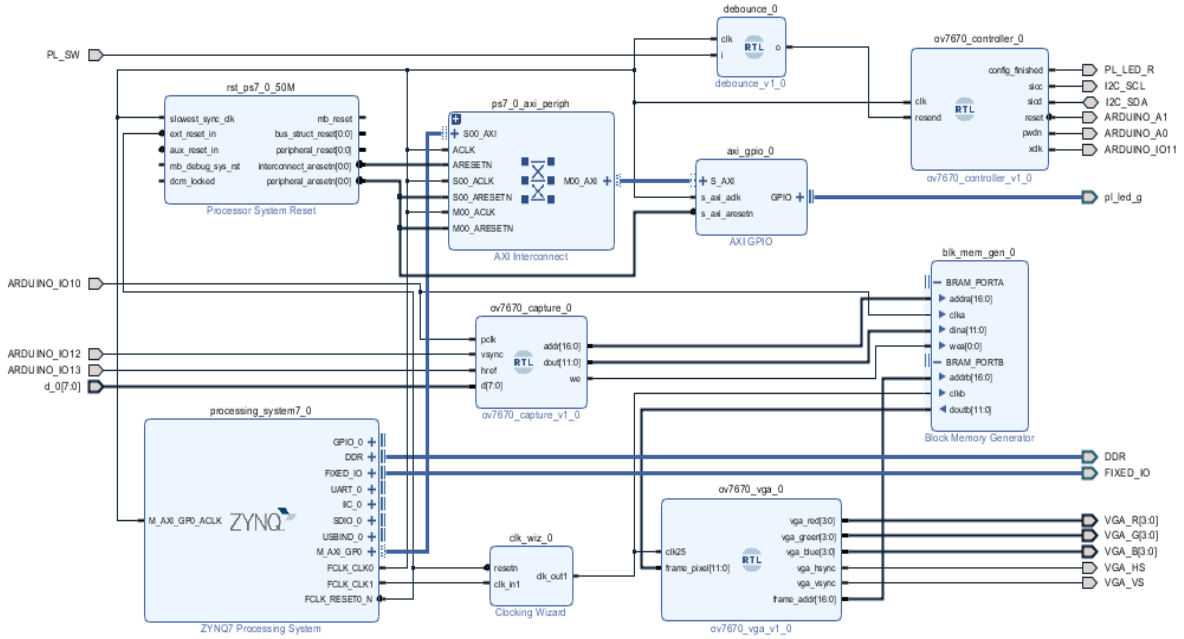


Figure 4-8 OV7670 Camera to VGA setup

Since the VGA module operates at 25 MHz another method is to implement a streaming architecture that does not require storing the pixels. The only constraint to do that is to operate both the VGA and image capture module at the same clock frequency.

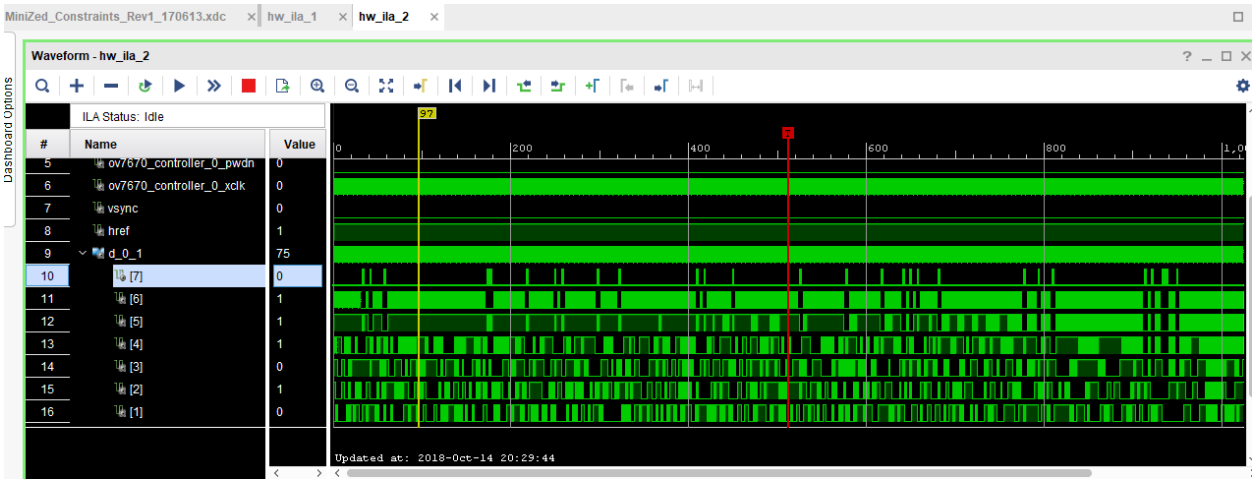


Figure 4-9 Logic analyzer data for OV7670 camera input capture module

In programmable logic hardware images are represented as a stream of pixels coming in a raster scan order (top left to bottom right). The pixel clock should be the same as the VGA operating frequency in order not to overflow the BRAM.

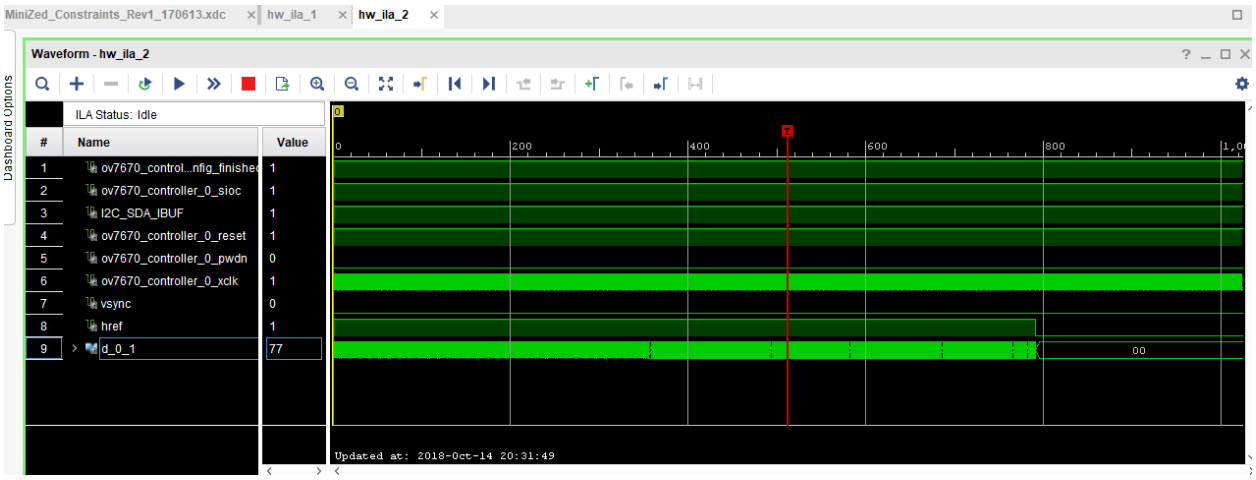


Figure 4-10 Logic analyzer output for VGA module operation.

The OV7670 is capable of outputting VGA, QVGA, QQVGA and CIF sized images. The Minized development platforms contains a XC7Z007S chip which has only 1.8Mb of BRAM. This is equivalent to fifty 30Kb Blocks, which in itself is insufficient to store a full VGA resolution frame. Since the XC7Z007S SoC does not have enough BRAM to store a full-size VGA image so the settings were modified to allow storage of a full sized QVGA image. VGA has a resolution of 640 (H) by 480 (V) = 30720 pixels.

One VGA frame contains 30720 pixels. Assuming pixel format of RGB (444), this implies 12 bits per pixel. The RAM will have an input data width of 12 bits and an input data depth of 307200 which is the same as the number of pixels for one full frame.

The address register length for the BRAM is computed by equation 4-2:

$$Address\ Length = \frac{ln(307200)}{ln(2)} = \lceil 18.2 \rceil = 19 \quad (Eq\ 4-2)$$

The total memory needed for 1 VGA frame is computed as 307200*12 bit = 3686400 bits as given in equation 4-3.

$$\frac{3686400}{8} = \frac{460800}{1024} = 450\ \text{kB}\ (\text{Eq}\ 4-3)$$

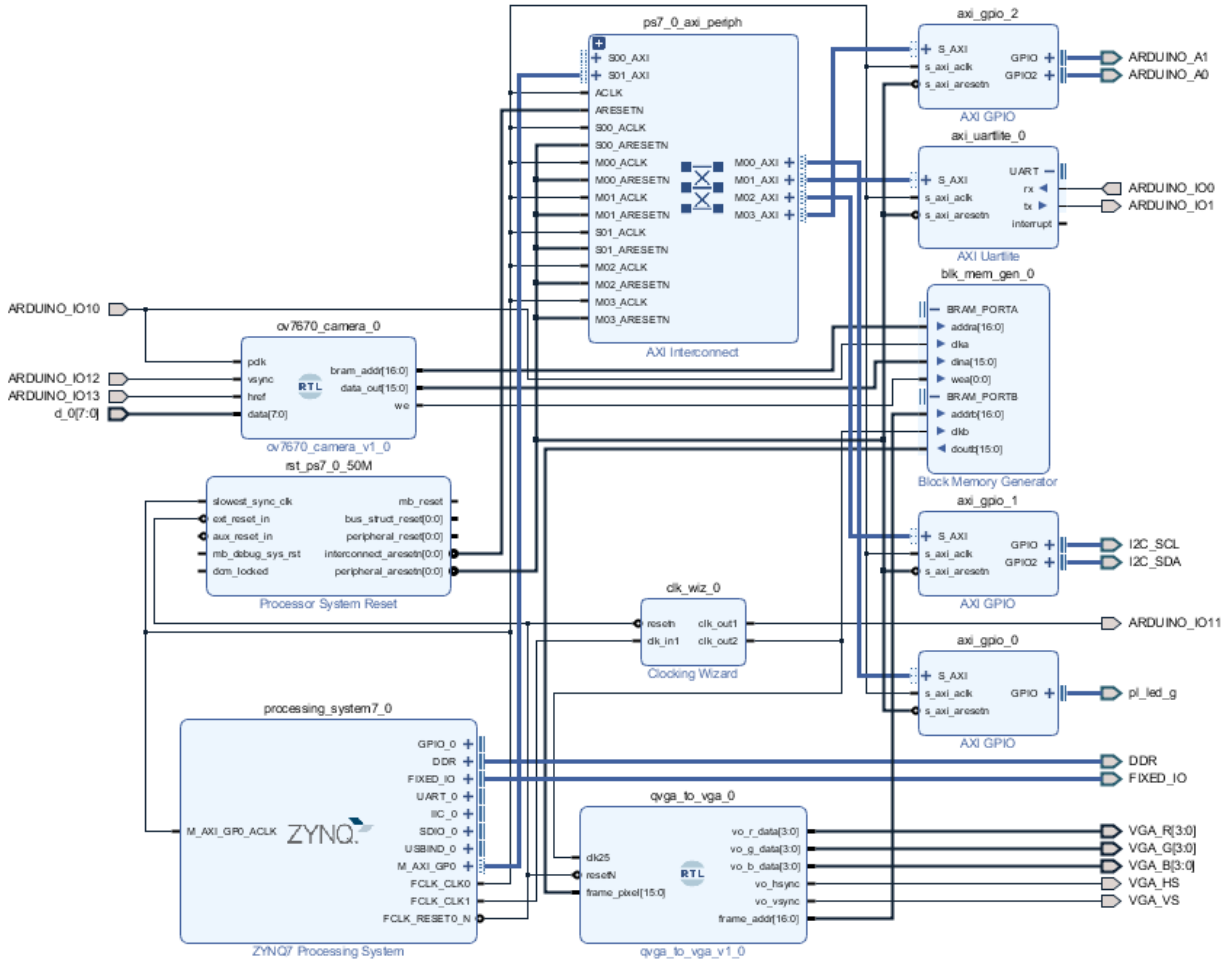


Figure 4-11 OV7670 Camera to QVGA with soft IIC

The OV7670 camera has a test pattern which is used to test the output as given in Figure 4-12. During troubleshooting it was identified that the QVGA timing parameters were incorrect, resulting in a shifted video test pattern. Revising the interface resulted in fixing the test pattern test.

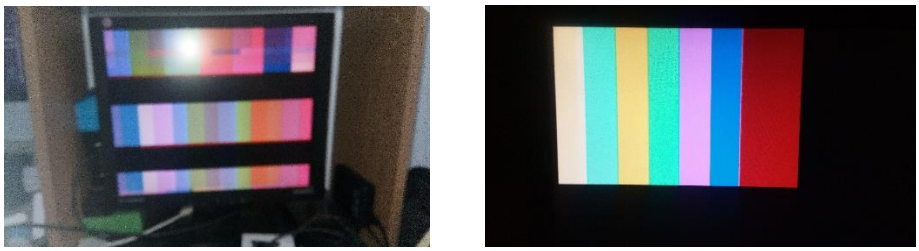


Figure 4-12 Testing the camera interface.

The AXI GPIO were used to bit-bang the I2C protocol as shown in Figure 4-11. After testing the camera with the test pattern, the configuration settings were changed from the default version via a Linux tool. The i2cdetect utilities suite were used to configure the camera with the proper setting in VGA mode.

The results shown in Figures 4-13 and 4-14 show the initial and after the configuration capture modes of the camera after the control sequence was updated.

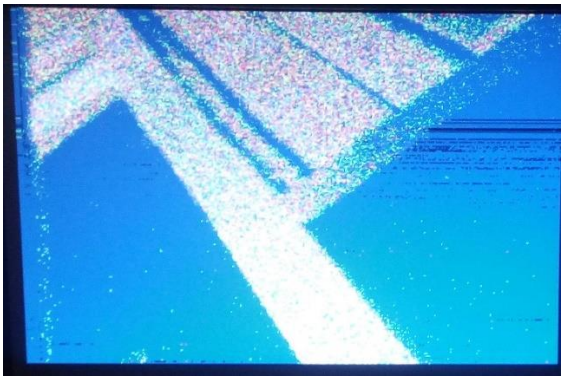


Figure 4-13 OV7670 camera capture settings

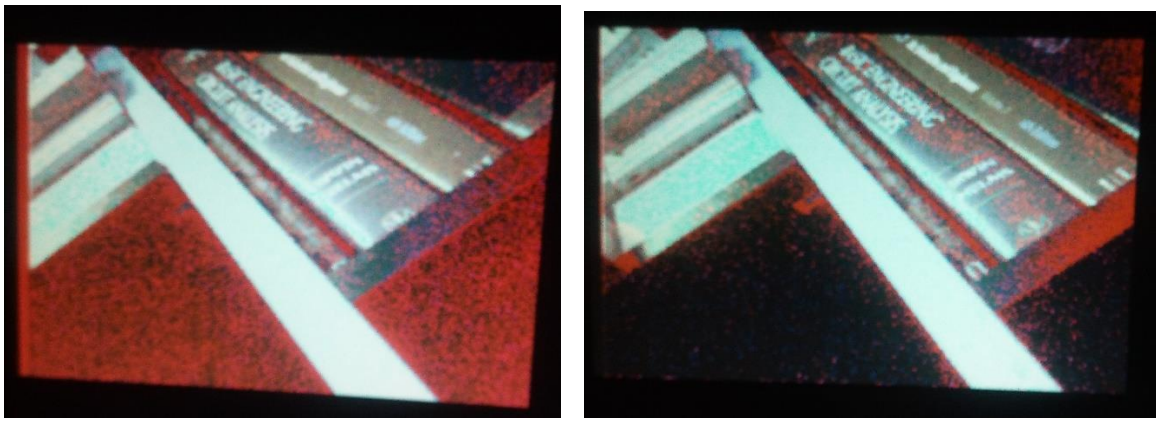


Figure 4-14 Changing the gamma, brightness and saturation settings.

4.5 PS implementation

Image stitching can also be implemented in the PS section of the SOC. To do so requires transferring the acquired pixels from the camera sensor to the SDRAM so that the ARM A9 CPU can access it. After processing on the PS section, the image can then be written back to the VGA display via VDMA.

This implementation requires a number of Xilinx data-mover IP's which are platform specific. The main disadvantage of such an approach is the large resource utilization which incurs a penalty with regards to power usage as well the fact that such a design is not platform agnostic compared to a pure HDL approach.

A test platform was implemented by using the VDMA IP core to transfer a test pattern from a video source to the VGA display as shown in Figures 4-15 and 4-16. This implementation requires that all the custom modules adopt the Xilinx AXI stream interface.

To make use of the OV7670 IP one has to revise the interface so that the streamed data conform to this format. Since the modifications to the IP are non-trivial this approach was abandoned in favor of moving the complete vision processing pipeline on the PL section.

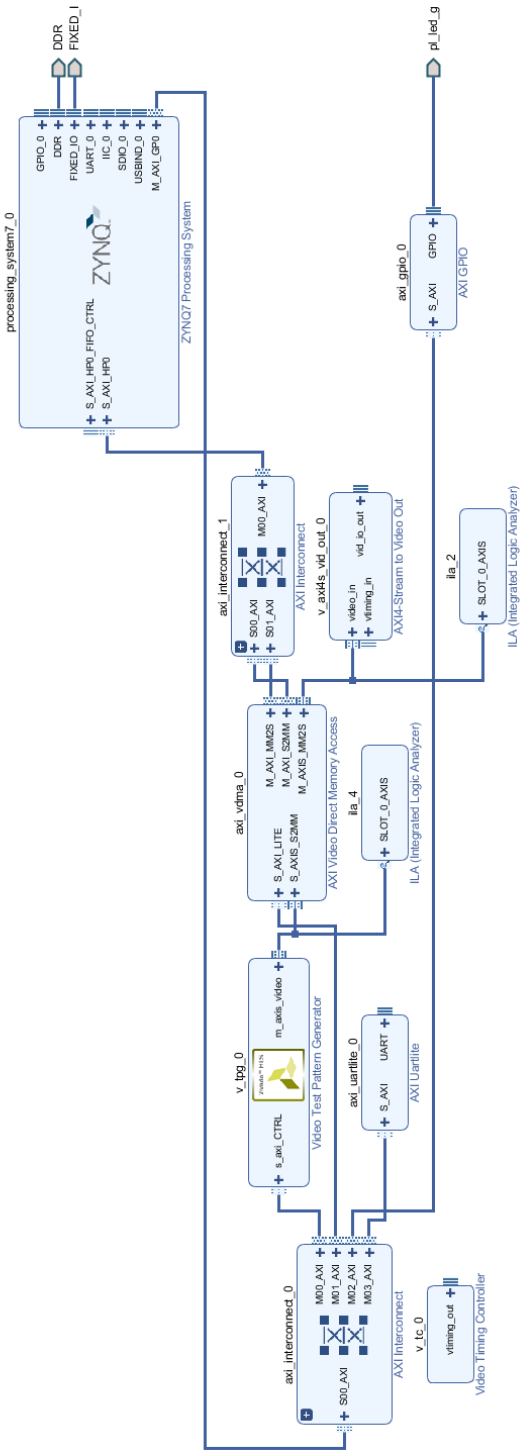


Figure 4-15 Simplified version of the VDMA system.

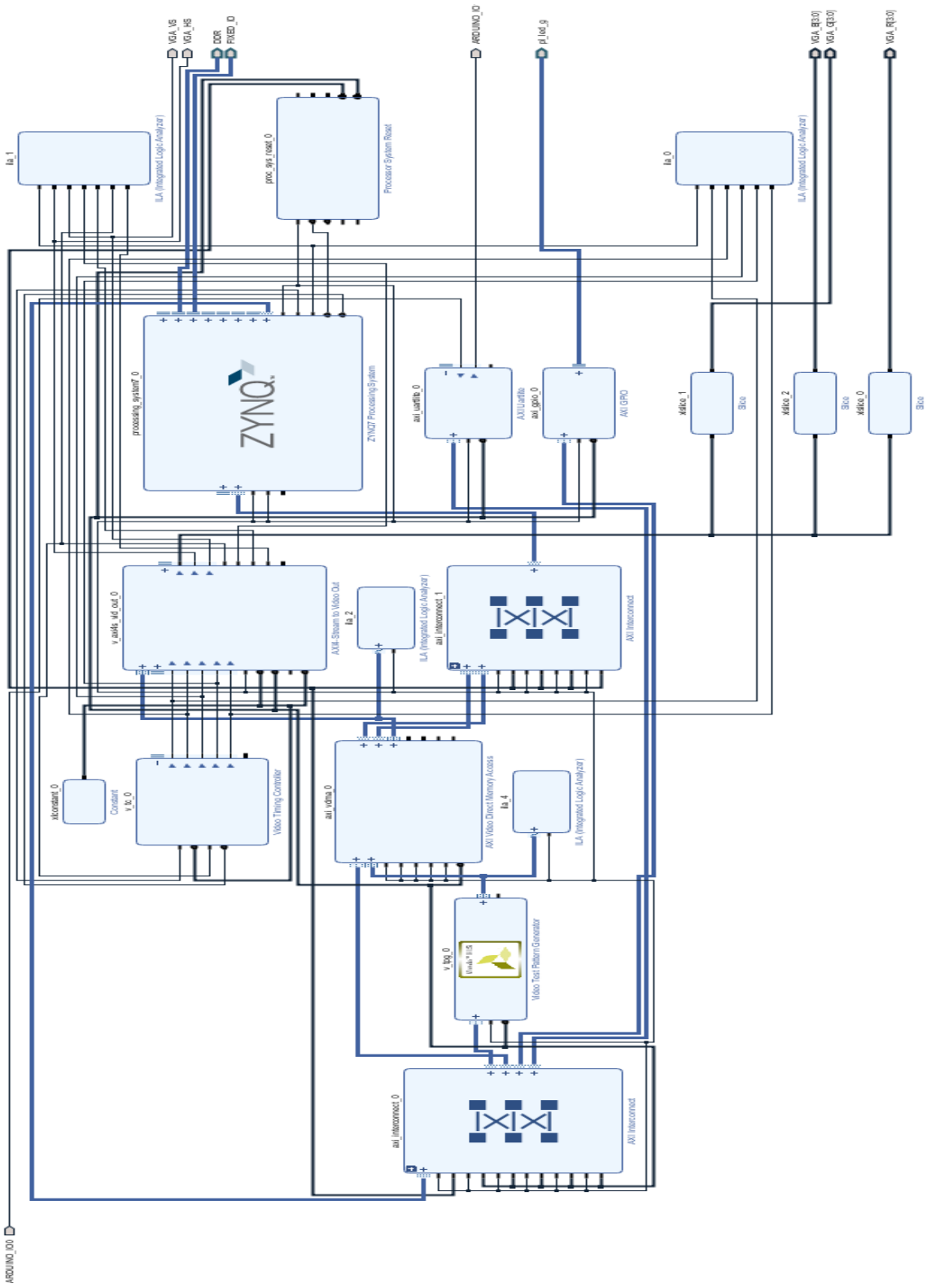


Figure 4-16 Testing VDMA to VGA with video test pattern generator

4.6 Image stitching IP

The image stitching IP was designed based on the engineering requirements from the lab. The IP was developed to be part of another image processing pipeline. The requirements are specified in the IP block diagram below in Figure 4-17.

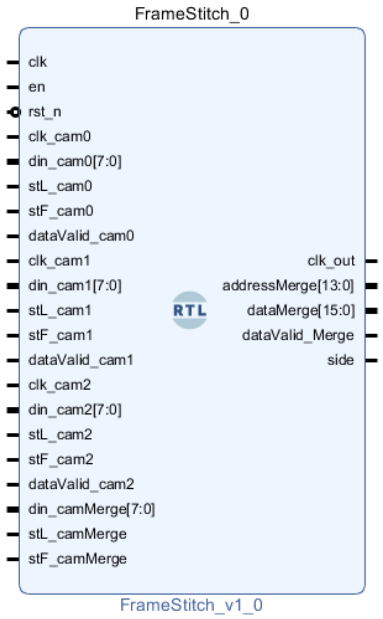


Figure 4-17 Image stitching module

The output of each camera sensor is stored on three separate simple dual port RAM, one for each camera. Port A of each simple dual port BRAM connected to each camera is fed by the pixel clock (PCLK) signal sourced directly from the camera. This clock runs at a frequency f1 = 25MHz allowing for 30-60 fps depending on the resolution. The BRAM can store the images captured directly from the camera or it can store filtered version of the images. Each camera sensor BRAM can store a full frame image at color QVGA resolution.

The write enable signal (WE) of port A for each DP RAM is fed by the data valid signal. The start of line (HSYNC) and start of frame signals (VSYNC) for each camera are fed to an address generator tied to each dual port image BRAM. The RAM addresses as explained above were generated from the PCLK, HSYNC and VSYNC inputs to the main module.

Port B is used by the image stitching Finite State Machine (FSM) to merge the consecutive image lines from all three cameras as given in Figure 4-18. The merged image is then stored on an output simple dual port BRAM. The top-level interface of the camera image stitching IP is shown below.

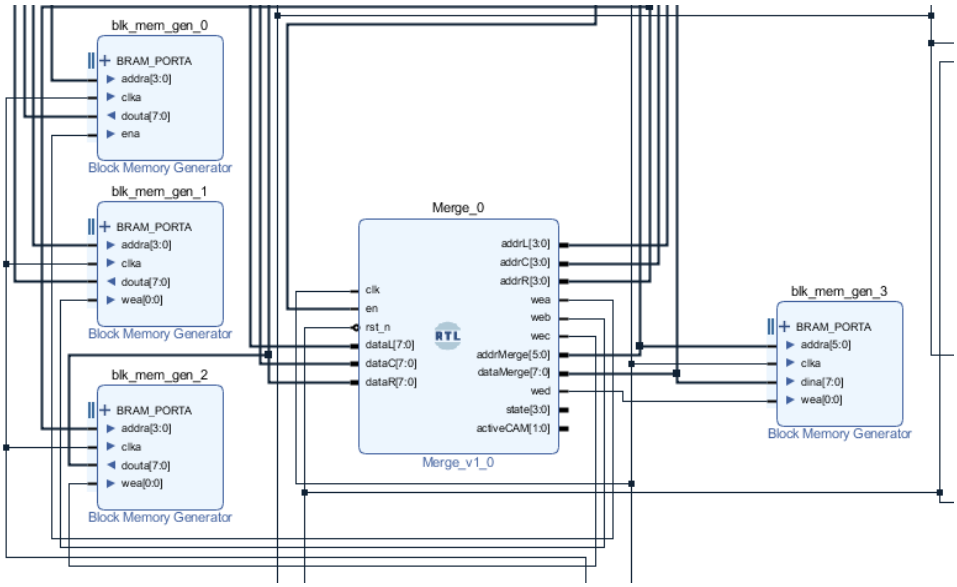


Figure 4-18 Test setup for image stitching IP.

As explained above the presence of the mirrors between each of the three cameras on the system allows for overlapped and mirrored visual fields. The OV7670 can be configured to mirror the image via the SCCB bus. This simplifies the IP since there is no need to mirror the pixels output stream.

The task of the image stitching IP is to read the pixels for all three separate dual port BRAM and merge their data into a single output BRAM so that each consecutive row of all three independent cameras are stitched together in a single output image.

The side frame views should be partitioned due to the presence of mirrors between the center and left cameras as well as the right and center camera. One side of the frame is discarded while the other side is flipped along the Y axis. The frame rows are then copied consecutively on an output BRAM.

The controller task is to read the port B of each dual port BRAM and write to PORT A of the output dual port BRAM. Port B of the output dual port BRAM is connected to the VGA circuit for display.

For each input camera, there are two counters which track the current line and column pixel index. Based on the current line and column pixel counters the output address of the merged dual port BRAM is calculated.

The address of the output dual port BRAM is the sum of the current index of all three input BRAM addresses. The read process occurs sequentially, meaning after the first row of BRAM1 is read the controller proceeds to read row one of the second BRAM and so on until all the rows are read in a consecutive fashion following a round robin scheme. To accommodate this the operating frequency of controller is more than three times as fast as the sensor PCLK. This allows the controller to read all lines while meeting the timing constrains imposed by the running pixel clock for each camera.

The controller operates as a finite state machine which cycles between read states for the input BRAM and write state for the output BRAM. Table 4-8 describes the input and output of the FSM controller.

Table 4-8 Image stitching IP interface

Signal Name	Direction	Width	Description
CLK_IN	IN	1	Input clock to the image stitching IP
RST_N	IN	1	Active Low Reset
EN	IN	1	Active HIGH enable input
WE1	OUT	1	Write enable for BRAM1
WE2	OUT	1	Write enable for BRAM2
WE3	OUT	1	Write enable for BRAM3
WE4	OUT	1	Write enable for BRAM4
Addr1	OUT	16	Address Bus for BRAM1
Addr2	OUT	16	Address Bus for BRAM2

Addr3	OUT	16	Address Bus for BRAM3
Addrmerge4	OUT	17	Address Bus for Output BRAM
DataOut1	IN	8	Data Bus input from BRAM1
DataOut2	IN	8	Data Bus input from BRAM2
DataOut3	IN	8	Data Bus input from BRAM3
DataInMerge4	OUT	8	Data Bus output to output BRAM
ActiveCam	OUT	1	Active Camera signal
state	OUT	3	State machine state
CLK_OUT	OUT	1	Clock output to output BRAM

To accommodate for reading all three input camera buffers, the FSM runs at a higher frequency so the clock input of all three dual port input RAM as well as the output RAM which contains the merged frames are not the same as the PCLK clock which feeds port A of each dual port RAM.

The difference in clock speeds is such as to account for

a) the delay due to the latency of the BRAM read state and

b) the delay due to the latency incurred by having to read all 3 BRAM

c) the delay needed to account for the time it takes to fill all three buffers such that no underflow read scenario occurs.

4.7 Automatic seam registration

The block diagram of the algorithm for automatically determining the seam between the two adjacent frames is shown below in Figure 4-19. The IP runs externally to the image stitching module and

outputs the column index of the detected seam which is used during the registration process.

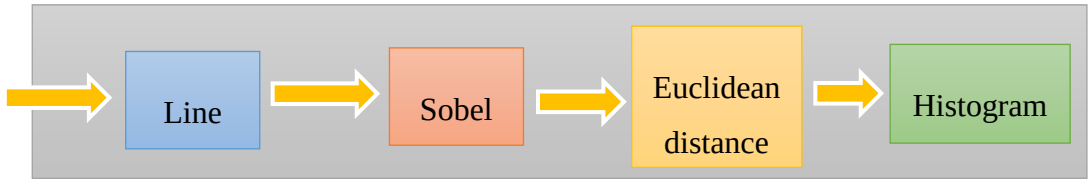


Figure 4-19 Block diagram for automatic seam registration

4.7.1 Sobel filter

There are a number of edge detection methods such as Sobel detector, Canny edge detector, Roberts detector, Prewitt detector. To apply these convolutional filters, a mask is applied to a rectangular pixel neighborhood. The selected size can 3x3, 5x5, 7x7 and so on. This requires that the specific pixel region is available even as incoming pixels are streamed in.

The module that implements this is the line buffer. The output of the line buffer is passed to the Sobel core which implements convolution with the filter mask. The final output is a filtered pixel stream which is stored on a BRAM. A line buffer for a 3x3 window is shown below in Figure 4-20. This module is connected directly to the pixel stream before the data is passed to the convolutional Sobel kernel.

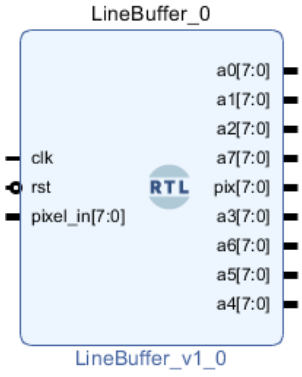


Figure 4-20 Line buffer interface

The Sobel filter is used to produce a gradient energy map of the pixels. The gradients are calculated using the following two kernel masks and equations as given by equations 4-4, 4-5 and 4-6.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (Eq\ 4.4)$$

$$G_x = ((p_2 - p_0) + (p_5 - p_3) + (p_8 - p_6) \quad (Eq\ 4.5)$$

$$G_y = ((p_0 - p_6) + (p_1 - p_7) + (p_2 - p_8) \quad (Eq\ 4.6)$$

The next step is the computation of the Euclidean distance between the filtered first column's pixels of image B and filtered column pixels of image A. This is given by the equation 4-7:

$$E(i,j) = \left| pix^{A}_{(i,j)} - pix^{B}_{(i,j)} \right| \quad (Eq\ 4.7)$$

The third step is the histogram calculation which sums the absolute value of all the difference value for each column. The output of the seam identification module is the column index of the seam between the two images.

4.8 Key-point detector

The module in Figure 4-21 shows the block diagram of the key-point detector. It is implemented as a state machine that searches the BRAM that stores a gradient section of the image for the point with the highest energy. The key-point detector is used as a submodule during the rectification process which occurs after the seam registration procedure.

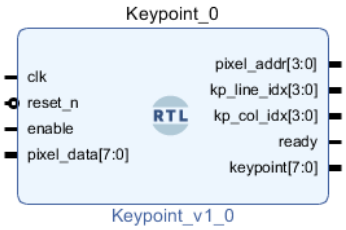


Figure 4-21 Key-point detector interface

Test-benches were implemented for all the IP modules developed in this chapter. The next chapter presents an analysis of the developed modules from a resource and timing perspective.

Chapter 5
Analysis

5 Analysis of results

5.1 Performance analysis

Implementation of image stitching on FPGA platforms is subject to multiple constrains which can be categorized as timing constrains, synthesis constrains and placement constrains.

Timing constrains that relate to individual and global clock paths, setup times for each input and clock periods. Synthesis constrains which relate to the FPGA resources such as LUT , BRAM, DSP element utilization. Placement constrains for every type of logic element such as I/Os clock logic, BRAM, DSP's IOB and global buffers.

From a timing perspective the main requirement is to have a system that allows real time object tracking at 60 frame per second (FPS). From a synthesis perspective the selected FPGA platform needs to have enough logic elements to accommodate the algorithm to be implemented.

The placement constrains directly affect the power budget of the logic implementation. Resource utilization and placement constrains are directly linked to the power budget.

5.2 Image Stitching IP

The majority of the work was focused on the image stitching IP controller. The block diagram interface of the image stitching IP is shown on figure 5.5 shows the top-level interface of the IP. The interface was part of the requirements specifications.

The next step was to implement the submodules. The pixel stream from each camera is stored on a BRAM. The datavalid output of each cam serves as the write enable for the Port A of the BRAM. The pixel clock sources the clock input of the BRAM. The address of each BRAM is calculated based on the VSYNC and HSYNC outputs which are represented by the stL and stF signals for each camera sensor.

Three BRAM's one for each camera are instantiated inside the module. Port A of each camera BRAM interfaces with the input signals and address generator module.

The main module of the image stitching IP is the frame stitching module whose interface is shown in Figure 5-1. This module is implemented as an FSM that controls the read and write cycles between the input and output BRAM units. Since the pixel clock determines the camera frame rate, the frame stitching IP should operate at a higher frequency in order to cope with the data being produced.

To operate at 30fps, each camera needs a 20 MHz XLKC master clock. The frame stitching IP operates at 61 MHz which is more than three times the individual pixel clock in order to account for the delays incurred from BRAM access. This is still below 100MHz.

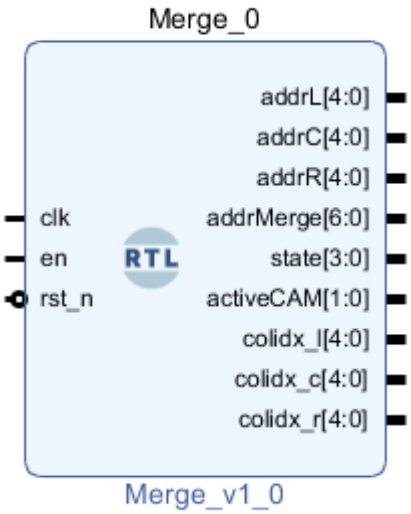


Figure 5-1 Frame stitching core interface

The simulation for a 3x3 image is shown on the test bench in Figure 5-2. The columns indices are accessed sequentially for each consecutive rows.

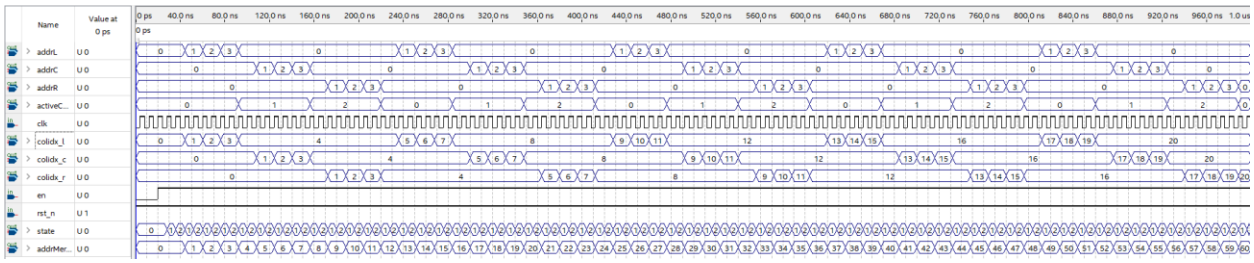


Figure 5-2 Simulation of core stitching IP

The frame stitching IP engine resource utilization was taken from the Vivado report as shown in Figure 5-3. The state machine is implemented using one hot encoding.

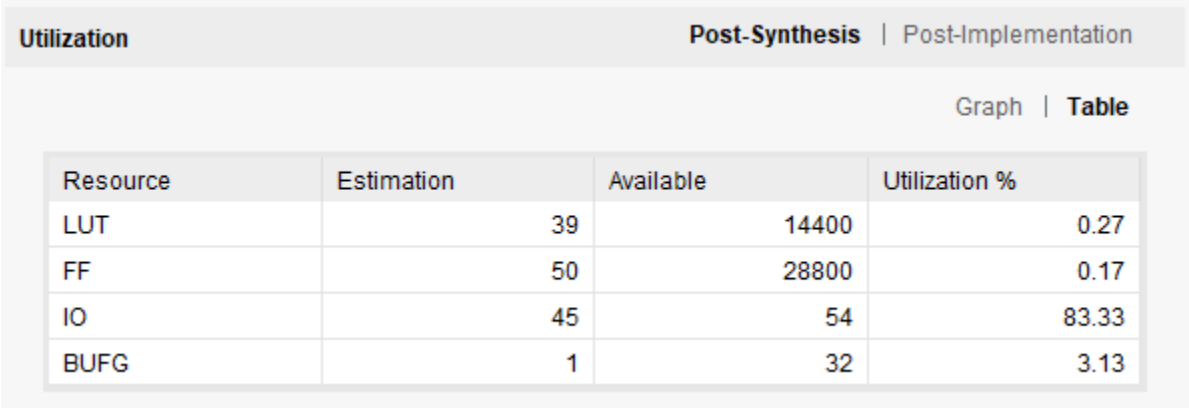
Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	62	14400	0.43
FF	28	28800	0.10
IO	46	54	85.19
BUFG	1	32	3.13

Figure 5-3 Resource utilization of frame stitching core IP

5.3 VGA

The VGA HDL module consumes very little resources due to the efficient implementation of the timing synchronization circuit. The report utilization is shown in Figure 5-4.



Utilization Post-Synthesis | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	39	14400	0.27
FF	50	28800	0.17
IO	45	54	83.33
BUFG	1	32	3.13

Figure 5-4 Resource Utilization of VGA IP

5.4 Seam identification HDL modules

5.4.1 Line buffer

A test bench was written for the 3x3 window line buffer as shown in Figure 5-5. An image with a line length of 16 pixel was used. Each pixel is 8 bits. The two test results below show the results from the test-bench.

The line-buffer module takes as inputs a clock, an active high reset pin, an input pixel stream and has as outputs a 3 by 3 window. The pixel length and the line buffer length are fully parametrizable.

Once the reset line is asserted low, the module starts to process the incoming pixel stream.

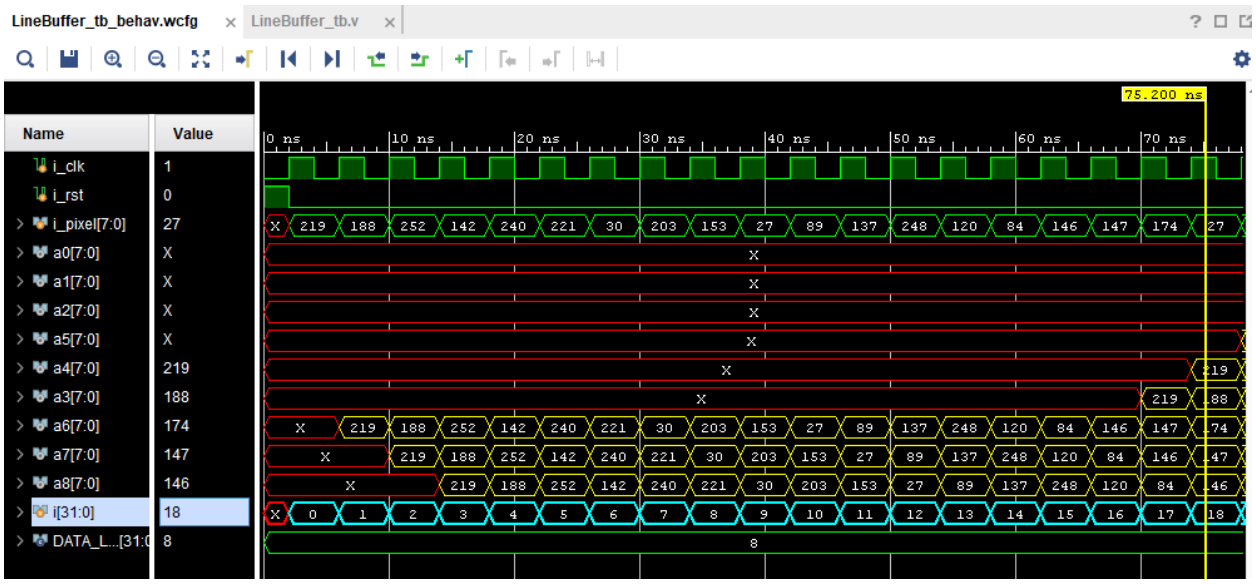


Figure 5-5 Simulation of line buffer module

As can be seen on Figure 5-6 below the first 3x3 window is obtained after the first 19 pixels have already streamed in. With the arrival of the 20-th pixel the 3x3 window slides 1 pixel to the right.

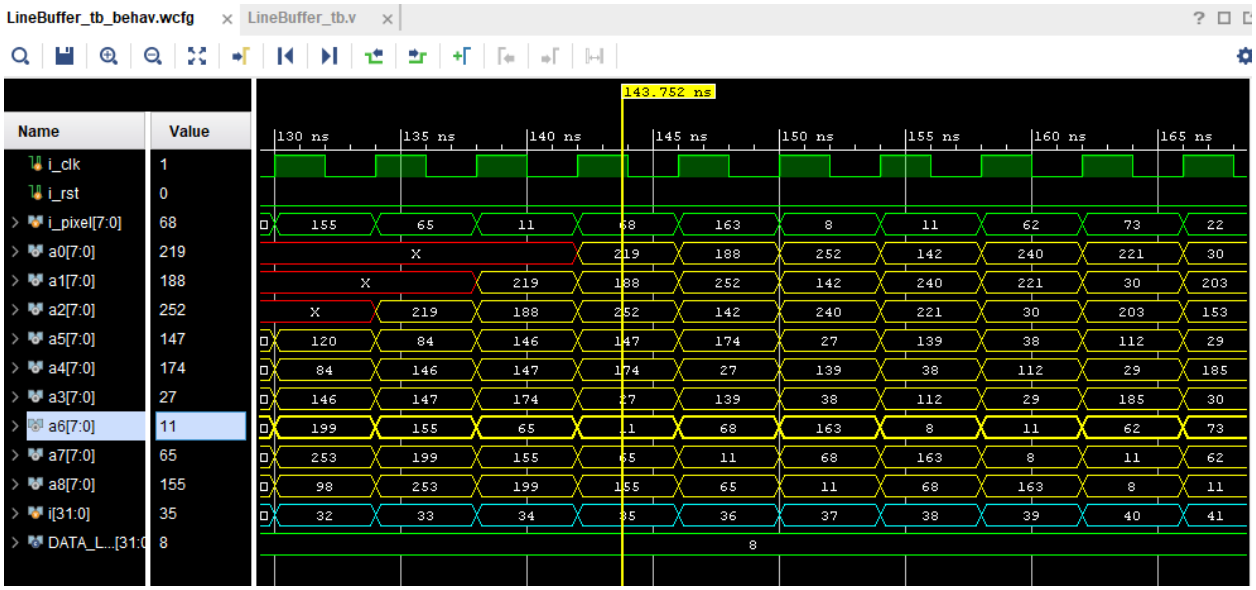


Figure 5-6 Simulation of line buffer module (continued)

Utilization | Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	16	14400	0.11
LUTRAM	16	6000	0.27
FF	80	28800	0.28
IO	81	54	150.00
BUFG	1	32	3.13

Figure 5-7 Resource utilization of the synthesized line buffer.

The report utilization from Vivado is given in Figure 5-7. The IO utilization is not taken into account since the inputs are internal to the FPGA. The resource utilization of a line buffer module depends on the image line length. Extrapolating for a VGA resolution image takes around 640 LUTRAM.

5.4.2 Sobel Filter

The utilization of the Sobel filter module is shown on Figure 5-8 below.

Utilization | Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	73	14400	0.51
IO	72	54	133.33

Figure 5-8 Resource utilization of Sobel.

An HLS implementation of the Sobel filter was also tested in hardware. This implementation requires significantly more resources compared to the pure Verilog HDL version since it infers BRAM for all inputs.

5.5 Key-point detector

The setup in Figure 5-9 shows the module that were used to test the basic implementation of the key-point detector. This was simulated on hardware on Vivado using the ILA to monitor the data.

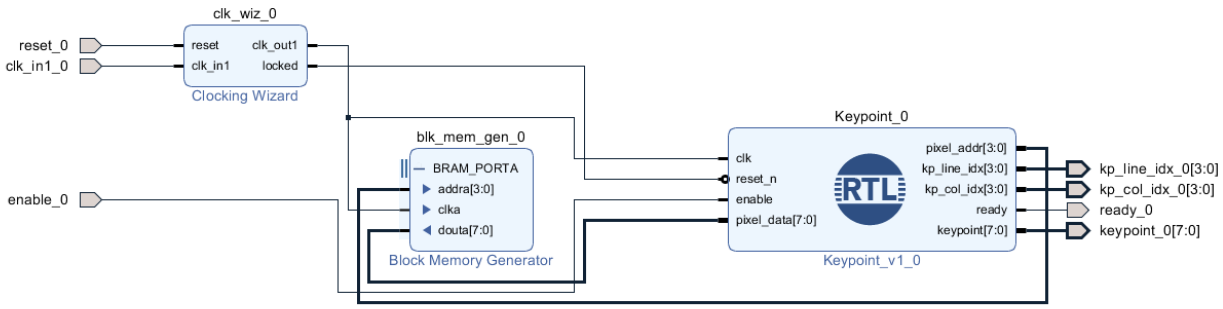


Figure 5-9 Test setup for basic key-point detector

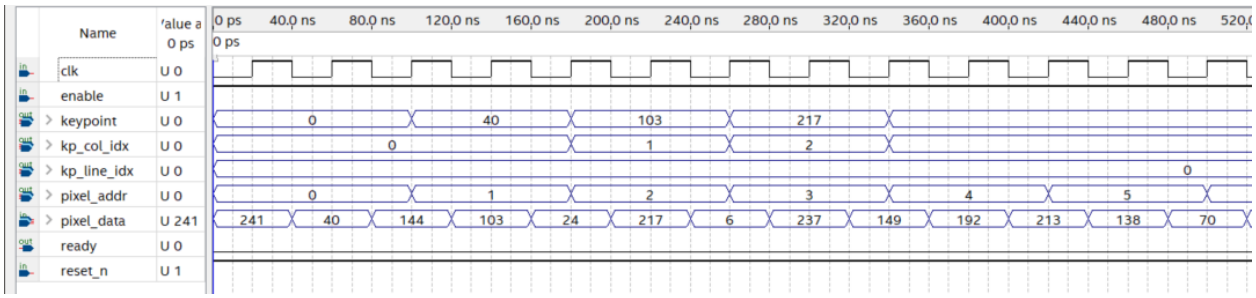


Figure 5-10 Test bench results

The test-bench in Figure 5-10 and the resource usage in Figure 5-11 conclude the analysis of the key-point detector IP.

Utilization | Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	46	14400	0.32
FF	38	28800	0.13
IO	32	54	59.26
BUFG	1	32	3.13

Figure 5-11 Resource usage for Key point detector

5.6 Comparison with OpenCV stitching

The developed IP can operate at more than 160 MHz which allows for run time operation of frame stitching from all three cameras at 30 fps. In comparison with the OpenCV implementation running on a single core CPU this is significantly faster. In addition, implementation of the image registration algorithm also supports run time operation, contrary to the feature-based approaches.

5.6.1 Comparison with streaming method

The main disadvantage of the current implementation is the need to store one complete frame for each camera. This implementation does not allow using low end ZYNQ SOC which do not have enough BRAM. The solution is to implement on-the fly image stitching by storing only the regions where they may be image feature overlap. Typically, this would be the last 50 columns of image A and the first 5 columns of image B. This will require revising the frame stitching interface as well as the image stitching IP.

Chapter 6
Summary

6. Project summary

This project focused on the implementation of an image stitching algorithm on a ZYNQ FPGA SoC. The main contributions of the projects were the schematic design of a custom hardware platform for a hyper-stereoscopic catadioptric vison system and the design of HDL IP modules for image stitching from multiple video sensors as well as the proposal of an algorithm for automatic image rectification using low resource utilization.

The primary objective of the project was the implementation of run time video frame stitching on an FPGA SoC from multiple camera sensors. Demand for real time panoramic imaging vision systems requires that the frame stitching stage from separate video sensors should be accomplished with a very short execution time to satisfy the system reaction time requirements.

Initial work on the project started by evaluating current image stitching algorithms for run time panoramic video mosaicking. The results showed that current CPU implementation cannot satisfy real time requirements so the focus was to move the algorithm on hardware. Image registration is the precursor step that need to occur before image stitching.

The main steps during the image registration procedure identified in frame stitching are the rectification procedure and the seam blending stage. Rectification is the process of bringing each frame along the same epipolar line so that there is no discernible vertical shift among adjacent frames. While seam blending requires determining the column index where the features from adjacent images start to overlap. The seam identification procedure is based on the calculation of a histogram that uses the Euclidean distance between the first column of the first image and the adjacent images in order to determine the overlapping column index. The output of the seam ID module is then passed to the frame stitching IP together with the rectification module.

Both the seam identification module and the rectification module are augmented with an AXI lite interface which allows reading and writing the output data from the PS section. This also allows manually rectifying the images from the PS section.

The implemented frame stitching IP is split into multiple functional modules. The main IP also leverages the smart functionality of the camera sensors to flip the images horizontally.

To maintain low resource utilization the proposed algorithm followed a direct approach which compares key- points of the energy gradients on adjacent images. The IP module then searches for corresponding highest energy gradients on adjacent image patches.

All IP modules were implemented and their respective testbenches were implemented in Vivado 2018.2. using Verilog 2001 HDL.

The initial challenge encountered during the project was the lack of access to an FPGA image processing platform so all of the firmware and hardware prototypes had to be prototyped outside of the lab on a low-cost development platform. An image processing platform was prototyped on a Minized ZYNQ development platform. A significant amount of time was dedicated to implementing the camera firmware. Due to the limited amount of BRAM available on the selected FPGA the firmware was designed with a reduced resolution. The majority of the effort was focused on the implementation of the schematic design and the image stitching IP.

The schematic capture for the custom hardware was implemented on Altium18 based on the requirements identified during the analysis stage. The 484-pin package of the ZYNQ 7020 was used. This allows for plenty of spare I/O for future expansion. The current design uses a highly integrated PMU, however if the peripheral power budget is updated the PMU will have to be upgraded. The schematics of the custom developed board are included in Appendix C.

Validation of the implemented IP's were done by using the integrated logic analyzer on the ZYNQ as well as writing test benches. Comparison with the OpenCV implementation shows that the frame stitching IP is able to run at 120 MHz which allows run time operation at 30 frames per second for VGA resolution.

The main issue with the current implementation is the excessive use of internal BRAM on the image stitching IP. All the sub-modules of the rectification IP module were implemented and tested. The seam stitching algorithm was not fully implemented so the input to the image merging IP are entered manually during compilation as constants or written via the AXI lite peripheral.

In summary the main goal of the project was implemented. Original contributions include the design of a custom hardware platform, design of an image stitching IP and the design of an automatic seam estimation algorithm used for automatic image registration.

References

[1] Richard Szeliski. Image alignment and stitching: A tutorial. Foundations and Trends in Computer Graphics and Vision Vol. 2, No 1 (2006) 1–104, 2006 R. Szeliski

[2] Image Blending in a High Frame Rate FPGA-based Multi-Camera System, Popovic, V., Seyid, K., Akin, A. et al. J Sign Process Syst (2014) 76: 169

[3] L. Juan and G. Oubong, "SURF applied in panorama image stitching," *2010 2nd International Conference on Image Processing Theory, Tools and Applications*, Paris, 2010, pp. 495-499. doi: 10.1109/IPTA.2010.5586723

[4] Panoramic Video from Unstructured Camera Arrays," *Disney Research*. [Online]. Available: https://www.disneyresearch.com/publication/panoramic-video-from-unstructured-camera-arrays/. [Accessed: 16-Dec-2018].

[5] J.-Y. Shieh, Y.-C. Liao, G.-J. Lioao, and Y.-T. Liou, "Dynamic Image Stitching for Panoramic Video", *IJETI*, vol. 4, no. 4, pp. 260-268, Oct. 2014.

[6] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao and W. Feng, "An architecture of optimized SIFT feature detection for an FPGA implementation of an image matcher," 2009 International Conference on Field-Programmable Technology, Sydney, NSW, 2009, pp. 30-37.

[7] T. Kawanishi, K. Yamazawa, H. Iwasa, H. Takemura and N. Yokoya, "Generation of high-resolution stereo panoramic images by omnidirectional imaging sensor using hexagonal pyramidal mirrors," *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, Brisbane, Queensland, Australia, 1998, pp. 485-489 vol.1.

[8] S. K. Nayar, "Catadioptric omnidirectional camera," *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Juan, Puerto Rico, USA, 1997, pp. 482-488.

[9] Lobo, Simon. "Video Stitch." *Omnitek*, Omnitek, 27 July 2018, www.omnitek.tv/image-stitch-subsystem.

[10] T. L. Chao and K. H. Wong, "An efficient FPGA implementation of the Harris corner feature detector," *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*, Tokyo, 2015, pp. 89-93.

[11] OV7670 camera - Hamsterworks Wiki!", *Hamsterworks.co.nz*, 2019. [Online]. Available: http://hamsterworks.co.nz/mediawiki/index.php/OV7670_camera. [Accessed: 1- Dec- 2018].

[12] Kar-Han Tan, H. Hua and N. Ahuja, "Multiview panoramic cameras using a mirror pyramid," *Proceedings of the IEEE Workshop on Omnidirectional Vision 2002. Held in conjunction with ECCV'02*, Copenhagen, Denmark, 2002, pp. 87-93.

[13] D. Cho, J. Park, Y. Tai and I. Kweon, "Asymmetric stereo with catadioptric lens: High quality image generation for intelligent robot," *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Xi'an, 2016, pp. 240-242.

[14] J. Kim, K. Yoon, J. Kim and I. Kweon, "Visual SLAM by Single-Camera Catadioptric Stereo," *2006 SICE-ICASE International Joint Conference*, Busan, 2006, pp. 2005-2009.

[15] S. K. Mohapatra, B. R. Swain and S. K. Mahapatra, "Optimized approach of sobel edge detection technique using Xilinx system generator," *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, Coimbatore, 2015, pp. 338-341.

[16] Yufeng Lu, Xiaohua Luo, Yimu Wang and L. Claesen, "Line buffer reduction for LUT-based real-time image inverse warping," *2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)*, Vancouver, BC, 2016, pp. 1-4.

[17] A. H. Nguyen, M. Pickering and A. Lambert, "The FPGA implementation of an image registration algorithm using binary images," *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, Cancun, 2014, pp. 1-4.

[18] G. Mamatha, V. Sumalatha and M. V. Lakshmaiah, "FPGA implementation of satellite image fusion using wavelet substitution method," *2015 Science and Information Conference (SAI)*, London, 2015, pp. 1155-1159.

[19] K. Agrawal and S. R. Chowdhury, "FPGA based accelerated orientation calculation in SIFT using luts," *2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, Visakhapatnam, 2013, pp. 225-227.

[20] I. Zouhir, L. Abdelhai, S. Samir, N. Abdelkrim and H. Mustapha, "FPGA implementation of the RANSAC based image mosaicing algorithm using the Nios II softcore," *2016 International*

Conference on Systems, Signals and Image Processing (IWSSIP), Bratislava, 2016, pp. 1-4.
[21] M. Sen, Y. Hemaraj, S. S. Bhattacharyya and R. Shekhar, "Reconfigurable image registration on FPGA platforms," *2006 IEEE Biomedical Circuits and Systems Conference*, London, 2006, pp. 154-157.

[22] M. Huang, O. Serres, T. El-Ghazawi and G. Newby, "Parameterized hardware design on reconfigurable computers: An image registration case study," *2009 5th Southern Conference on Programmable Logic (SPL)*, Sao Carlos, 2009, pp. 71-76.

[23] Y. Do and Y. Jeong, "A new area efficient SURF hardware structure and its application to Object tracking," *2013 IEEE International Conference of IEEE Region 10 (TENCON 2013)*, Xi'an, 2013, pp. 1-4.

[24] Y. Zhang, Y. Huang, J. Han and X. Zeng, "FPGA-based efficient implementation of SURF algorithm," *2017 IEEE 12th International Conference on ASIC (ASICON)*, Guiyang, 2017, pp. 315-318.

[25] J. Mun and J. Kim, "Real-time FPGA rectification implementation combined with stereo camera," *2015 International Symposium on Consumer Electronics (ISCE)*, Madrid, 2015, pp. 1-2.

[26] J. G. P. Rodrigues and J. C. Ferreira, "FPGA-based rectification of stereo images," *2010 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Edinburgh, 2010, pp. 199-206.

[27] A. Akin, I. Baz, L. M. Gaemperle, A. Schmid and Y. Leblebici, "Compressed look-up-table based real-time rectification hardware," *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, Istanbul, 2013, pp. 272-277.

[28] M. Pohl, M. Schaeferling, G. Kiefer, P. Petrow, E. Woitzel and F. Papenfub, "An efficient and scalable architecture for real-time distortion removal and rectification of live camera images," *2012 International Conference on Reconfigurable Computing and FPGAs*, Cancun, 2012, pp. 1-7.