Development of algorithms and architectures for driving assistance in adverse weather conditions using FPGAs

Diego Botero-Galeano

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *Institute National de Sciences Appliquées (INSA)*
Discipline ou spécialité : *Systèmes Embarqués et Robotique*

Présentée et soutenue par *Diego Andrés BOTERO GALEANO*
Le *5 décembre 2012*

Titre : Development of algorithms and architectures for driving assistance in adverse weather conditions using FPGAs

Directeurs de Thèse :
Michel DEVY
Jean-Louis BOIZARD

JURY
Président :
Jean-Yves FOURNIOLS

Rapporteurs :
François BERRY
Ricardo CARMONA

Examinateurs :
Johel MITERAN
Jonathan PIAT
Richard KLEIHORST

Ecole doctorale : *Ecole Doctorale Systèmes (EDSYS)*
Unité de recherche : *Laboratoire d'Analyse et d'Architecture de Systèmes (LAAS)*

Résumé

En raison de l'augmentation du volume et de la complexité des systèmes de transport, de nouveaux systèmes avancés d'assistance à la conduite (ADAS) sont étudiés dans de nombreuses entreprises, laboratoires et universités. Ces systèmes comprennent des algorithmes avec des techniques qui ont été étudiés au cours des dernières décennies, comme la localisation et cartographie simultanées (SLAM), détection d'obstacles, la vision stéréoscopique, etc. Grâce aux progrès de l'électronique, de la robotique et de plusieurs autres domaines, de nouveaux systèmes embarqués sont développés pour garantir la sécurité des utilisateurs de ces systèmes critiques.

Pour la plupart de ces systèmes, une faible consommation d'énergie ainsi qu'une taille réduite sont nécessaires. Cela crée la contrainte d'exécuter les algorithmes sur les systèmes embarqués avec des ressources limitées. Dans la plupart des algorithmes, en particulier pour la vision par ordinateur, une grande quantité de données doivent être traitées à des fréquences élevées, ce qui exige des ressources informatiques importantes. Un FPGA satisfait cette exigence, son architecture parallèle combinée à sa faible consommation d'énergie et la souplesse pour les programmer permet de développer et d'exécuter des algorithmes plus efficacement que sur d'autres plateformes de traitement.

Les composants virtuels développés dans cette thèse ont été utilisés dans trois différents projets: PICASSO (vision stéréoscopique), COMMROB (détection d'obstacles à partir d'une système multi-caméra) et SART (Système d'Aide au Roulage tous Temps).

Abstract

Due to the increase of traffic volume and complexity of new transport systems, new Advanced Driver Assistance Systems (ADAS) are a subject of research of many companies, laboratories and universities. These systems include algorithms with techniques that have been studied during the last decades like Simultaneous Localization and Mapping (SLAM), obstacle detection, stereo vision, etc. Thanks to the advances in electronics, robotics and other domains, new embedded systems are being developed to guarantee the safety of the users of these critical systems.

For most of these systems a low power consumption as well as reduced size is required. It creates the constraint of execute the algorithms in embedded devices with limited resources. In most of algorithms, moreover for computer vision ones, a big amount of data must be processed at high frequencies, this amount of data demands strong computing resources. FPGAs satisfy this requirement; its parallel architecture combined with its low power consumption and flexibility allows developing and executing some algorithms more efficiently than any other processing platforms.

In this thesis different embedded computer vision architectures intended to be used in ADAS using FPGAs are presented such as:

- We present the implementation of a distortion correction architecture operating at 100 Hz in two cameras simultaneously. The correction module allows also to rectify two images for implementation of stereo vision.
- Obstacle detection algorithms based on Inverse Perspective Mapping (IPM) and classification based on Color/Texture attributes are presented. The IPM transform is based in the perspective effect of a scene perceived from two different points of view. Moreover results of the detection algorithms from color/texture attributes applied on a multi-cameras system, are fused in an occupancy grid.
- An accelerator to apply homographies on images, is presented; this accelerator can be used for different applications like the generation of Bird's eye view or Side view.
- Multispectral vision is studied using both infrared images and color ones. Synthetic images are generated from information acquired from visible and infrared sources to provide a visual aid to the driver. Image enhancement specific for infrared images is also implemented and evaluated, based on the Contrast Limited Adaptive Histogram Equalization (CLAHE).
- An embedded SLAM algorithm is presented with different hardware accelerators (point detection, landmark tracking, active search, correlation, matrix operations).

All the algorithms were simulated, implemented and verified using as target FPGAs. The validation was done using development kits. A custom board integrating all the presented algorithms is presented.

Virtual components developed in this thesis were used in three different projects: PICASSO (stereo vision), COMMROB (obstacle detection from a multi-cameras system) and SART (multispectral vision).

Contents

List of Figures

List of Tables

Glossary

AHEBM Adaptive Histogram Equalization Based Method

ACC Adaptive Cruise Control

ADAS Advanced Driver Assistance Systems

AMBA Advanced Microcontroller Bus Architecture

APU Auxiliary Processor Unit

ASIC Application-Specific Integrated Circuit

AXI Advanced eXtensible Interface

BFM Bus Functional Model

BRIEF Binary Robust Independent Elementary Features

BSP Board Support Package

CC Cross-Correlation

CDF Cumulative Distribution Function

CLAHE Contrast Limited Adaptive Histogram Equalization

CLB Configurable Logic Blocks

COMMROB COMMunication with and among ROBots

DA Driving Assistance

DMA Direct Memory Access

DDR Double Data Rate

DRC Dynamic Range Compression

DSP Digital Signal Processor

DUT Device Under Test

EDK Embedded Development Kit

EKF Extended Kalman Filter

ELF Executable Loader File

ETACS External and Taxi Aid Camera System

FOV Field Of View

FIFO First In First Out

FMC FPGA Mezzanine Connector

FOV Field Of View

FPGA Field Programmable Gate Array

FPS Frames Per Second

FPU Floating Point Unit

GPP General Purpose Processor

GPS Global Positioning System

GPOS General Purpose Operating System

GPU Graphic Processing Unit

HDL Hardware Description Language

HIL Hardware In the Loop

HDR High Dynamic Range

HE Histogram Equalization

HEBM Histogram Equalization Based Method

HLS High Level Synthesis

HPC High Performance Connector

IDP Inverse Depth Parametrization

IHM Interface Human Machine

IMAPCAR Integrated Memory Array Processor CAR

IMU Inertial Measurement Unit

IPM Inverse Perspective Mapping

IR Infrared

LAAS Laboratoire d'Analyse et d'Architecture des Systmes

LDF Logique De Fusion

LIDAR Light Detection And Ranging

LPC Low Performance Connector

LUT Look Up Table

lwIP Light Weight IP

LWIR Long-Wavelength Infrared

MIPM Mono Inverse Perspective Mapping

MIG Memory Interface Generator

MPMC Multi-Port Memory Controller

MWIR Mid-Wavelength Infrared

NIR Near Infrared

NTSB National Transportation Safety Board

NUC Non-Uniformity Correction

NPI Native Port Interface

OS Operating System

PC Personal Computer

PCB Printed Circuit Board

PICASSO Plateforme dIntgration de Cameras multiSenOrielles

PLB Processor Local Bus

PPC PowerPC

QDR Quad Data Rate

RADAR RAdio Detection And Ranging

RAP Robotique, Action et Perception

RGBT Red, Green, Blue, Thermal

ROI Region Of Interest

RTOS Real Time Operating System

RT-SLAM Real Time-Simultaneous Localization and Mapping

SAD Sum Absolute Differences

SART Systme d' Aide au Roulage Tout temps

SDK Software Development Kit

SDRAM Synchronous Dynamic Random Access Memory

SIPM Stereo Inverse Perspective Mapping

SLAM Simultaneous Localization and Mapping

SNR Signal to Noise Ratio

SRAM Static Random Access Memory

SWIR Short-Wavelength Infrared

TCP Transmission Control Protocol ;

TCAS Traffic Colision Allerting System

UCF User Constraint File

UDP User Datagram Protocol

UMBM Unsharp-Masking Based Methods

VIS Visible

VFBC Video Frame Buffer Controller

VHDL VHSIC hardware description language

VHSIC Very High Speed Integrated Circuit

VLWIR Very Long-Wavelength Infrared

ZNCC Zero-mean Normalized Cross Correlation

1

Introduction

Embedded systems are everywhere and are part of our life. Thanks to the fast progress in technology, more complex systems have been developed to satisfy many needs of human beings. Some mass markets make the technology on embedded systems progress faster and faster, like smart phones, video games or Internet devices. It is well known that the greater number of computers are now integrated in embedded systems, comparing to laptops or desktops.

Some convergences make new technologies developed for one specific market, adopted for another one. Let us cite the Kinect product, introduced on the mass market as a device for video games, but which is more and more used in robotics applications as a low-cost 3D camera. The Kinect massive distribution illustrates also the importance of sensing, and moreover vision applications, which impulse technology transfers from research laboratories to companies: let us cite also the spectacular diffusion on smart phones or on cameras, of algorithms developed in vision labs some years ago, like the Viola&Jones face detection algorithm or the stitching methods, used now by everybody in the world.

One domain where technological improvements are now a survival conditions for many companies, concerns automotive or transportation industry. Advanced Driver Assistance Systems (ADAS) have been studied from twenty years in different applications involving vehicles, trains, aircrafts, helicopters...; these developments are justified by safety considerations, not only for marketing ones!!

Nowadays, numerous ADAS exist in the marketplace; many of them make use of advanced sensing technologies, like cameras, GPS, IMU, active sensors (Radar and LIDARs), etc. The processing of sensory data acquired by these devices, requires powerful, but cheap and low-energy computation systems. The use of these embedded systems on transportation applications, has improved the efficiency of all kind of vehicles and reduced the number of accidents and fatalities.

Our contributions concern an ADAS designed for aircraft pilots, but they could be adapted to be used on other vehicles. It is the reason why we present in the next section, a short state of the art on Driver Assistance, before describing our objectives and contributions.

1.1 State of the art for Advanced Driver Assistance Systems on cars

Driver Assistance on cars has known a very fast development since twenty years: the European EUREKA Prometheus program in the nineties, various other European member states programs involving collaborative researches between labs and automotive companies, made different technological improvements, which are now integrated on sensors or devices used on many cars. The involved technologies are numerous, from motor control, communications (between vehicles and/or road infrastructures), sensing, data processing... Moreover, the joint use of GPS and of knowledge data base on the road maps, made available accurate enough localization methods, so that a driver could be warned when arriving to a dangerous turn or crossroad. Systems devoted to the driver or passenger monitoring inside the car cockpit (smart airbags or drowsiness detection) are not considered here.

Our contributions only concern an embedded system integrating sensors and computation units, devoted to obstacle detection and the estimation of the vehicle motions. So hereafter, only works on these applications will be considered, to cope with either sensing or integration issues.

Radar tools have been used for many years now, mostly for military and commercial use, and today we find more and more of them showing up in new vehicles in order to provide ADAS, especially for obstacle detection on highways. One of the main disadvantages of the radar system comes from undesired reflections which generate false alarms. Radar techniques also present some drawbacks for lateral-position measurements [1].

In the last two decades the automotive industry has done a significant amount of research in ADAS. The most well known ADAS is the Back-up aid system, already installed on millions of cars. Automatic parking systems have also been installed by carmakers of upmarket models like the Lexus LS-series vehicle. Another very useful and popular system is cruise control, often limited to the speed control when driving on highways.

The ACC (Adaptive Cruise Control) already integrated in the Sedan BMW, is based on a dual core computer platform. This system provides an ADAS based on information coming from a GPS, radar, and a set of pre-calibrated cameras. The ACC implements perspective transformation of the images in order to provide a Bird's eye view and a side view. The system also provides a tool called a driving tube, which corresponds to an augmented reality methods , in order to superpose a driving virtual lane in the image to indicate the estimated trajectory to the driver. This system includes even automatic maneuvers for controlling acceleration of the vehicle.

Night Vision systems using cameras with infrared illumination or infrared cameras are included on the highest-end vehicles like Mercedes S-class and BMW 7 series. The high cost of this kind of cameras has limited the use in only highest-end vehicles; it could be developed for high-cost professional vehicles like trucks.

Lane changing systems assist drivers when changing lanes. This system is included in the Audi Q7 luxury SUV. These systems warn the driver if a lane change occurs without a previous activation of the turn indicator.

Sign recognition systems are being studied by many companies. If the driver does not see an important sign such as a stop sign, the system will alert the driver.

Toyota includes a safety system, based in the IMAPCAR (Integrated Memory Array Processor for CAR) [2] developed by NEC, in the Lexus LS460. This processor has a special architecture including 128 Processing Elements, providing the processor with a peak performance of 12.8 MMAC/s or 100 GOP/s. Different ADAS algorithms are included: Lane departure

warning, Pedestrian protection, Collision avoidance, Blind spot detection, high beam assistant, etc.

Finally let us cite some embedded systems involving vision. First the Mobil Eye company in Israel, proposes a product for obstacle detection using vision: this device implements in real time, pedestrian or car detection, using a learning/classification approach based on region descriptors like HOG (Histogram of Oriented Gradients): it is another example of a very fast transfer from research results (the initial Dalal/Triggs works appeared in 2005) to a product ready to be integrated on the mass market. More locally LAAS has some cooperations with companies in Toulouse involved in transportation applications; the Continental unit has developed for many years embedded systems based on vision, while the SME Delta Technologies Sud Ouest provides smart cameras used to assist the driver of buses in urban environment.

1.2 State of the art for Driver Assistance on aircrafts

In the aircraft industry different ADAS exist and are now mandatory for the safety of the ground operations of these critical systems. A large percentage of aircrafts operation time is devoted to actions on the ground; in Europe it corresponds to 10 to 30 percent of the aircrafts activities.

Bad weather conditions presented in some places throughout the year represent a significant loss for an airline company and are a risk for the passengers.

Improving airport traffic is becoming a big necessity for airline companies. Most of the worst accidents in the airplane domain have been on the ground and are generally caused by environmental conditions or human failures.

Since 1993 a total of 12 accidents have been reported by [3]. In the last year, there were 3 accidents involving big airplanes, which occurred during navigation on taxiways or parking areas. All of these accidents involved contacts of the winglets of the airplanes with other objects; the main cause of these accidents was the absence of systems to help pilots determine the space occupied by his aircraft. Because of these accidents, the National Transportation Safety Board (NTSB) is evaluating where or not to impose the installation of anti-collision systems, based on vision or radar sensors, in order to help pilots.

Large airplanes (Boeing 747, 757, 767, and 777; the Airbus A380; and the McDonnell Douglas MD-10 and MD-11) need an ADAS tool for maneuvering throughout the airport during the day, but it is particularly required during the night. There are many blind spots in these big machines, especially because the pilot is very high with respect to the ground plane, so that the blind area in front of the nose is very large; the pilot has no visibility. Moreover, the wing tips are not easily visible by the pilot, so that it constitutes a risk during the taxiway navigation of the parking approach; in these aircrafts, the pilot needs to open the cockpit window to have a clear view of the wing tips.

Nowadays, most of the help provided to the pilot is done using signs between the pilot and human operators with a series of signals called Marshaling Signals. These signals include commands like move ahead, slow down, turn, and stop. During this procedure, the pilot follows the orders of the operator without the option to react to a possible operator failure.

Some special airport procedures have been established when there are hazard conditions. One of them corresponds to the “follow me” vehicle shown in the figure 1.1. For this procedure, a vehicle with large panels and illuminations drives in front of the airplane in order to indicate the correct path for the pilot. It is very useful in very large airports (Francfort, London, Paris in Europe), where a pilot can be lost during the navigation on taxiways, especially in bad weather conditions. But the use of these ”follow me” vehicles involves important delays in operations,

so delays also for passengers.

Figure 1.1: Follow me vehicle - Safety procedure used in many airports to guide the aircrafts. Taken from [4]

When doing a simple maneuver such as turning, different strategies must be followed because of the different dimensions and rotation axis of the airplanes; it is not sufficient to follow a marking on the ground. The pilot must execute an oversteering maneuver, making not visible markings and taxiways during the turn. Using an image with synthetic objects like taxiways boundaries inserted in the image, can make easier the execution of these maneuvers; it could reduce the human failure probability

Many of the existing ADAS used in aircraft today use localization systems based solely on GPS systems. Some systems have already been developed by companies like Thales. The OANS (Onboard Airport Navigation System) can be installed as an option on an Airbus A380; it dynamically displays the aircraft position over a high-resolution geo-referenced airport map. But the aircraft position is only computed from the GPS and IMU (Inertial Measurement Unit embedded on the aircraft) data; the confidence region is about 8 m. The OANS system makes use of an airport map to generate the synthetic images, but these databases themselves are not accurate enough and do not reference important visual landmarks like signs and beacons. So these systems are not accurate enough to guarantee that the wheels of a large aircraft remain on the taxiways and moreover, in some cases the aircraft is localized on an incorrect taxiway.

In [5] a real-time system combining infrared and visible cameras is presented. This system is intended to assist pilots during adverse weather conditions. Currently, a new European project called Alicia, which uses infrared and visible cameras combined with Radar and Lidar, is being developed.

Some last generation airplanes (a380, a340-500/600) include a system called ETACS (External and Taxi Aid Camera System) [6]. ETACS uses two cameras, which display the nose and the main landing gear in the cockpit. This system allows the pilot to detect possible collision of the airplane wing tips. The left side of the figure 1.2 shows the image displayed by this system.

The Ground maneuvering camera system [7] manages up to 6 cameras, this is a standard equipment for the Boeing 777-300, providing an automatic or manual viewing angle adjustment. The cameras are usually mounted behind the nose gear and in the horizontal stabilizers. On the right side of the figure 1.2 is shown the image displayed by this system.

Another system called TCAS (Traffic Collision Alerting System) [9] provides collision avoidance based on a communication link established automatically between the TCAS systems of the concerned aircrafts. This system warns the pilot if there is a collision risk, and indicates to the crew of each involved aircrafts actions that must be taken. To compute the maneuver, the TCAS system takes into account the dynamic models of the involved aircrafts. If there is a difference between the instructions done by the TCAS system and the control tower, the TCAS has always the priority. This system is designed to assist the pilot just for air operations;

Figure 1.2: Camera navigation systems - On the left, ETACS display appearance (taken from [8]). On the right, GMCS display appearance (taken from [7])

for the ground operations the presence of multiple TCAS units could degrade safety. For this reason, the system is usually placed in standby mode when the aircraft is on the ground.

Some of the existing systems are limited by a small field of view. In [5] is presented an ADAS that uses multispectral sources with just around 40 degrees of Field Of View. For the maneuvers done in the airports, this is still not enough.

Even if there are many systems that have been developed for ADAS, none of them satisfy the operational needs in terms of field of view, nor when there are low visibility conditions such as rain, snow, dust, fog, haze, or poor illumination. In order to help pilots in these situations, airports are equipped with visual aids like Ground Lighting, surface markings and Signs. Ground lighting provide taxiway edge lights, taxiway center lines, and runway edge lights. Surface markings provide stop bars, taxi indicators, edge lanes ...

In many of the current ADAS, DSPs or DSPs plus FPGAs have been used. A DSP only solution does not have enough compute horsepower to implement all of the processing needed for ADAS. An ASIC might be useful, but because the algorithms and techniques are evolving very quickly, an ASIC does not offer the option to easily upgrade the system.

ADAS must give information to the driver in real time and the decision must be taken immediately since only a few milliseconds can represent a big improvement for the safety of the users. A study realized in France in 2002 about about the required frequency of an obstacle avoidance system for highway navigation, concluded that such a system must process sensory data at 100Hz, taking into account delays required to confirm an alarm, and also, the reaction time of the driver once an alarm is generated.

An FPGA can execute the job of several DSPs to implement all of the algorithms needed for ADAS in real-time. An FPGA provides the developer, with a two-dimensional array of configurable resources that can implement a wide range of arithmetic and logic functions. A FPGA includes complex resources such as dedicated DSP or processor blocks, multipliers, dual port memories, lookup tables (LUTs), registers, tri-state buffers, multiplexers, digital clock managers and more.

In contrast, with a microprocessor or DSP processor, where performance is tied to the clock rate at which the processor can run, FPGA performance is tied to the amount of parallelism that can be implemented in the algorithms, making the FPGA a very strong signal processing

platform with the advantages of low power consumption, small size, easy installation, reliability, and flexibility.

In critical systems the software implementation represents more risk than a hardware implementation. In software there is a compiler that translates thousands of line code, resulting in a super complex system; a system too complex to be able to guarantee any sort of reliability. Software, for example, can experience electromagnetic noise resulting in the corruption of the program counter register and the system can completely crash.

In a hardware implementation, we can have better control of the available resources and of the implementation. For this reason we also justify the use of HDL language and FPGA design for these kinds of systems.

In this thesis, we need to process data for a human operator. For the development of an ADAS on aircraft, images will be displayed at 30 Hz in order to satisfy the visual comfort of the pilot. Real-time constraints push us to use high performance hardware like FPGAs.

1.3 Context of the thesis

This thesis was developed in the context of three projects: PICASSO (Plateforme d'Intgration de CameraS multiSenOrielles), COMMROB (Communication with and among Robots) and SART (Systme d' Aide au Roulage Tout temps).

The project PICASSO aimed at developing a real-time stereo vision algorithm to be executed either for transportation applications, but also for a surgery application using stereo endoscopic images. A FPGA-based implementation of a passive stereo vision algorithm was developed in the PhD of A.Naoulou [10], assuming that the stereo images were undistorted and perfectly aligned. Our contribution was the integration of the image distortion correction and the rectification algorithm, executed in real-time at the pixel clock. Finally all modules have been integrated on a multi-FPGAs platform, made with three Altera Cyclone 2 boards. Initially, the PICASSO project had to cope also with multispectral vision, fusing data acquired by a stereo camera pair and an infrared camera; but the infrared camera was a prototype developed at LAAS, and it has been found that the infrared images were not stable enough to be processed. In the project COMMROB, the objective was to implement a companion robot, able to assist a human in an indoor dynamic environment with the presence of other persons or robots. LAAS participated to a work package devoted to indoor navigation; the main LAAS contribution concerned an integrated system designed for obstacle detection from a belt of micro-cameras, mounted around a robot. This system presented in [11], was developed in the PhD of M.Ibarra Manzano [12]. It was implemented only one method for obstacle detection, using pixel classification from color and texture descriptors. Our contribution has been first to develop and integrate a second method, using the Inverse Perspective Mapping transform, and then the system was extended to fuse the classification results obtained in parallel from images acquired by 4 cameras, in a single robot-centered or environment-centered occupancy grid. All methods were implemented and evaluated on a Altera Stratix 3 development kit, mounted on a robot equipped with four cameras.

We have been mainly involved in the third project SART, an industrial project lead by the LATECOERE company in Toulouse. In this SART context, several functions required for a navigation system to help aircraft pilots have been developed. Different algorithms were first tested in software. Then, according to the estimation of resources and the needs of the algorithms, target FPGAs were chosen. Then, most of the algorithms were implemented on Xilinx Virtex6 and Virtex5 development kits. Finally, a custom board was developed to implement the final prototype. In this project, the constraint of VHDL use was imposed, because of the

high reliability of Hardware implementations using VHDL.

1.4 Thesis objectives

The objective of this thesis is the development of algorithms and architectures, integrated in an system embedded on vehicles, and devoted to obstacle detection and driver assistance for navigation under any weather conditions.

In this thesis, we will generate specifications and develop virtual components for: image transforms (rectification, correction, homography), maps construction, fusion, SLAM, contour detection, obstacle detection, and image enhancement. These virtual components will be developed and validated in real embedded systems, connected to real cameras.

Moreover, we want to evaluate the development of ADAS based on multispectral camera sources.

For Localization and Mapping of the robot, we have cooperated with other PhD students, which started from a C++ open source code called RT-SLAM, created at LAAS. Our objective consists in developing optimizations to the SLAM algorithm using hard coded accelerators taking advantage of the architecture of an FPGA.

Due to the certification constraints for equipments to be mounted on commercial aircrafts, we had (1) to use C instead of C++ , and (2) to use VHDL for the development of modules on FPGAs. This is the reason why methodologies based on high level synthesis, i.e. taking advantage of C to VHDL compilers, have not been evaluated during this thesis. Nevertheless, during the development of this thesis we used a High Level Tool based on Matlab for verfication of the different modules.

1.5 Contributions

In this thesis we will study the development of algorithms and architectures to be used on embedded Advanced Driver Assistance Systems. We will study how to migrate the different and existing algorithms to limited resource platforms. Different algorithms were adapted in order to be implemented into embedded architectures with strong real time constraints.

We are interested in the use of Co-design to share the processing between processors and hardware accelerators developed using FPGAs. We want to use the parallel horsepower available in the FPGAs.

We have been mainly involved in the SART project; so our main motivations are to perform obstacle detection and to develop algorithms for localization and mapping using embedded multispectral cameras, in order to provide visual aids to the pilot.

The figure 1.3 shows a block diagram of the different modules developped in this thesis, different target FPGAs from Xilinx and Altera were used in the development of the projects (PICASSO, COMMROB and SART) concerned in this work.

As a summary, our contribution aims at demonstrating the viability to use FPGAs to implement different vision algorithms to be integrated on ADAS, but also on autonomous robots; our results were used in three different projects: PICASSO (ADAS on vehicles), COMMROB (Robotics), and SART (ADAS for aircrafts).

In the research team RAP at LAAS-CNRS, the integration activity is becoming stronger; with this thesis we contribute to strengthen and make more visible this research activity.

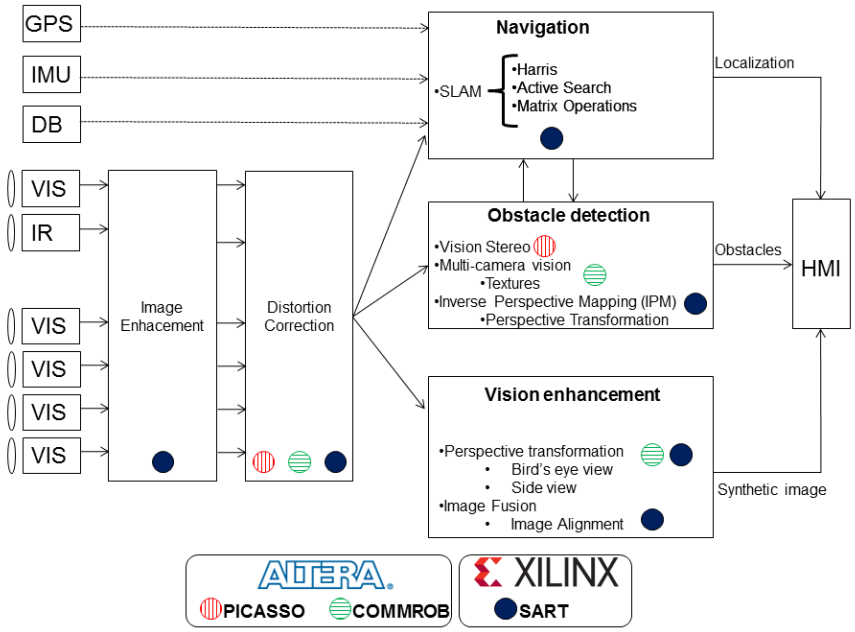

Figure 1.3: Block diagram of modules and projects - In this thesis we show different architectures for tasks involved in ADAS. Architectures for Navigation, Obstacle detection and Vision enhancement are described.

1.6 Publications

1. D.BOTERO, J.PIAT, M.DEVY, J.L.BOIZARD "An FPGA accelerator for multispectral vision-based EKF-SLAM" IEEE IROS'12 Workshop on Smart CAmeras for roBOtic applications. October 2012.

2. D.BOTERO GALEANO, J.PIAT, P.CHALIMBAUD, M.DEVY, J.L.BOIZARD "FPGA implementation of mono and stereo inverse perspective mapping for obstacle detection" Design and Architectures for Signal and Image Processing. October 2012.

3. D.BOTERO GALEANO, M.DEVY, J.L.BOIZARD, W.FILALI "Real-time architecture on FPGA for obstacle detection using inverse perspective mapping" International Conference on Electronics Circuits and Systems, December 2011.

4. D.BOTERO GALEANO, A.GONZALEZ, M.DEVY "Architecture embarque pour le SLAM monoculaire". Reconnaissance des Formes et Intelligence Artificielle. January 2012.

5. W.FILALI, D.BOTERO GALEANO, M.DEVY, J.L.BOIZARD "SOPC components for real time image processing: rectification, distortion correction and homography" IEEE Workshop on Electronics, Control, Measurement and Signals. Juin 2011.

6. M.DEVY, J.L.BOIZARD, D.BOTERO GALEANO, H.D.CARILLO LINDADO, M.IBARRA MANZANO, Z.IRKI, A.NAOULOU , P.LACROIX, P.FILLATREAU, J.Y.FOURNIOLS, C.PARRA "Stereovision algorithm to be executed at 100Hz on a FPGA-based architecture". Advances in Theory and Applications of Stereo Vision, N978-953-307-516-7, Janvier 2011, Chapter 17, pp.327-352

1.7 Document organization

This thesis is composed of four main chapters.

Chapter 2 presents the theoretical foundations for the different algorithms evaluated during this thesis. This chapter explains some image processing algorithms using vision; some algorithms have been extended to be used with infrared cameras. An introduction to the infrared technology and preprocessing algorithms to efficiently exploit its information are presented. An introduction to multispectral vision is given. Vision algorithms are used for obstacle detection and for SLAM. The obstacle detection is explained using two different approaches, one geometric approach based in projective geometry, and a second approach based in textures.

Chapter 3 presents different embedded systems and their advantages. After a description of multiple platforms, the FPGA architecture is described in detail. The methodology followed during this thesis to work using FPGA is presented as well as the new emergent technologies used today and how they work with the new FPGAs.

Chapter 4 presents the integration of the algorithms described in chapter 2 on FPGA-based architectures. This chapter describes the accelerators coded in VHDL, using co-design techniques.

Chapter 5 presents the results of the implemented algorithms and architectures. This chapter describes more of the projects (PICASSO, COMMROB and SART) involved during the development of this thesis.

Finally we present the conclusion and the perspectives.

2

Theoretical background on driving assistance

2.1 Introduction

Driving assistance is a very extensive field of research, widely studied specially during the last 10 years. In this thesis, we studied some of the algorithms used; we present in this chapter just those used during the development of this thesis. The algorithms presented here were chosen according to the needs imposed during the projects involved during this thesis.

In this chapter we will first present a brief introduction to image processing algorithms used in this thesis. A description of the Distortion Correction problem will be presented. Then, the Perspective Transformation is described as well as real applications now being used in different research fields. In the next section, we present the infrared, its principal properties, and some algorithms used to extract the information from an infrared High Dynamic Range (HDR) sensor. We present the state of the art of Multi-spectral vision, which corresponds to simultaneously use infrared and visible cameras. Then, we will present two main obstacle detection algorithms; the first of which is based in perspective effects (Inverse Perspective Mapping-IPM), while the second is based in an adaboost classifier that uses Textures properties in images. In the last section, we present the SLAM algorithm widely used in robotics for localization and mapping.

2.2 Pinhole Camera Model

An image corresponds to a representation of a three dimensional scene onto a two dimensional surface. Different camera models [13] are used to represent the mapping between a point in the space and the point where it will be projected in a 2d-image acquired with a camera. One of the most used models is the Pinhole.

The Pinhole model shown in figure 2.1 expresses the relationship between an image coordinate (u, v) and a world coordinate (X_w, Y_w) in function of the intrinsic (K) and extrinsic (R, t_c) camera parameters. The model is given by equation 2.1.

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = KR \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + Kt_c \tag{2.1}$$

The intrinsic parameters (K) depend oN the internal parameters of the camera and are represented with the matrix K given by the equation 2.2. The extrinsic parameters (R, t_c) depend of the orientation and the position of the camera respect to the world frame.

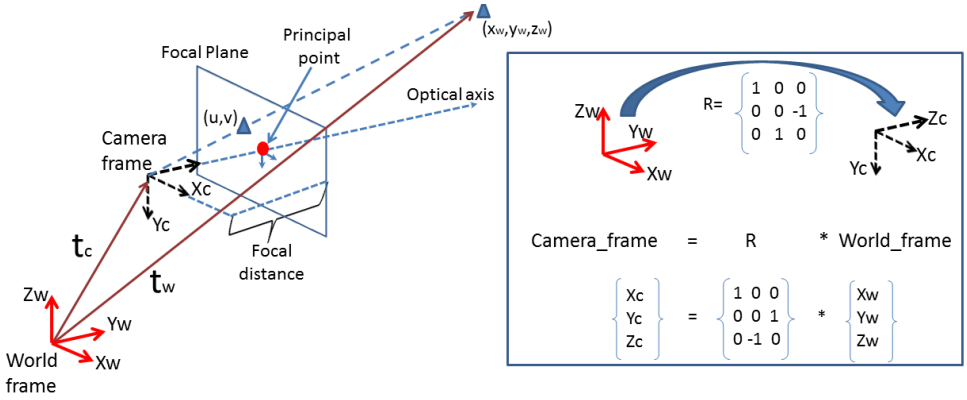

Figure 2.1: Pinhole Camera Model - In the figure, the camera frame is parallel to the world frame. For this reason, in this ideal example, the components of the rotation Matrix R are just 1, 0 and -1. Physically, it corresponds to have the camera parallel to the ground plane and the optical axis aligned with the Y-axis of the world frame.

$$K = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

where,
s = skew
$\alpha_u = k_u f$
$\alpha_v = k_v f$
k_u, k_v = pixel size in (pixel/m)
f = focal distance
(u_0, v_0) = principal point coordinates
Generally, in a CCD camera the skew factor is equal to zero, which means that the x-axis and y-axis are perpendicular.

2.3 Distortion Correction

One of the main problems of long field of view cameras in image processing is the Distortion. The Distortion is a phenomenon caused by the optics properties of the camera. Typically, there are two types of distortion : radial and tangential. Radial distortion is due to the lens properties while tangential distortion is due to the imperfect centering of the lens with respect to the sensor.

The distortion effect is a phenomenon that is more important in cameras with wide angle lenses. In some Pinhole cameras with small field of view the distortion is almost null, while in cameras with longer field of view it is very important.

Working with a distorted image increases the complexity of the algorithms. In an undistorted image, a straight line can appear like a curve. Distortion correction is needed in many image processing algorithms.

Different camera models exist to model the distortion, some applications can use a simple cuadratic model to undistort the images, however, a higher order model can be used to improve the distortion correction results. The next equation shows a sixth order model, this model is used in different Matlab toolboxes (California Tech). In this equation the normalized (pinhole) image projection $x_n = (x, y)$ is used to compute a new normalized coordinate $x_d = (x_d(1), x_d(2))$.

$$x_d = \begin{bmatrix} x_d(1) \\ x_d(2) \end{bmatrix} = (1 + k_c(1)r^2 + k_c(2)r^4 + k_c(5)r^6)x_n + dx$$

where,

k_c is a vector containning the distortion information, and dx corresponds to the tangential distortion given by the next equation:

$$dx = \begin{bmatrix} 2k_c(3)xy + k_c(4)(r^2 + 2x^2) \\ kc(3)(r^2 + 2y^2) + 2k_c(4)xy \end{bmatrix}$$

and,

$r^2 = x^2 + y^2$

When building an undistorted image, the distortion correction algorithm remaps the pixels of an image in new coordinates according to a distortion model. This remap can cause non-integer coordinates as shown in the figure 2.2.

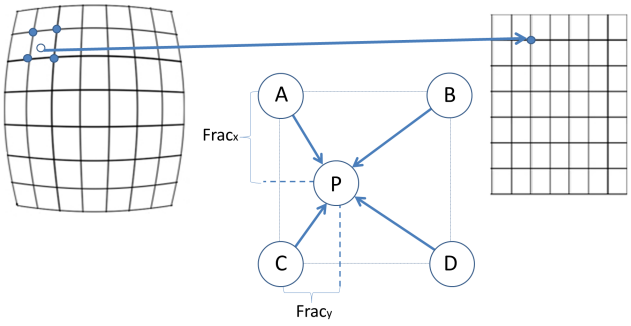

Figure 2.2: Bilinear interpolation - $Frac_x$ and $Frac_y$ are values between 0 and 1. They are used to assign the weights of the vicinity pixels (A, B, C and D) for the interpolation

The undistorted pixel can be replaced by the closest neighbor, however this can create artifacts in the reconstructed images that can affect the post-processing algorithms of the images. For example, in a corner detection algorithm, false corners can be detected.

Another approach to rebuild the unknown pixels is to implement a bilinear interpolation of the four pixels in the vicinity (A, B, C and D). When implementing a bilinear interpolation, weights are assigned to each one of the four neighbor pixels. For example, in the case that the unknown pixel is exactly in the middle of the four pixels ($Frac_x = 0.5$ and $Frac_y = 0.5$), the unknown pixel corresponds to the average of the four vicinity pixels.

The bilinear interpolation is done using the weighted sum given by the equation 2.3.

$$P = A.S + B.T + C.U + D.V \tag{2.3}$$

where,

$$S = (1 - Frac_x)(1 - Frac_y) \tag{2.4}$$

$$T = (1 - Frac_x)Frac_y \tag{2.5}$$

$$U = Frac_x(1 - Frac_y) \tag{2.6}$$

$$V = Frac_x Frac_y \tag{2.7}$$

$$P = A(1-Frac_x)(1-Frac_y)+B(1-Frac_x)Frac_y+CFrac_x(1-Frac_y)+DFrac_xFrac_y \tag{2.8}$$

Distortion correction is usually implemented detecting the corners of a pattern chessboard image. Then, using an optimization algorithm the distortion parameters are computed. Different software toolboxes and applications exist to implement distortion correction.

In the case of IR images, for calibration a simple chessboard cannot be used because the chesspatern will not be visible. Figure 2.3 shows a special chesspattern designed like a Printed Circuit Board (PCB) with an electric resistance in order to heat the cooper regions.

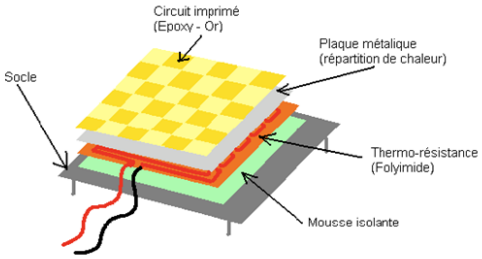

Figure 2.3: Infrared Cheesboard for calibration - This pattern has a small dimension

2.4 Perspective transformation

Homography is an important algorithm in computer vision widely used in many applications. When applying a homography to an image, the coordinates of the pixels of the image (x,y) are modified using an affine transformation. An homography results in the transformation of a plane. The rotation is particular case of a homography with H represented in the equation 2.9.

$$H_{rotation}(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

The homography is implemented using the equation 2.10

$$\lambda \begin{bmatrix} x_h \\ y_h \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.10}$$

A value λ done by equation 2.11 appears like a scale value for all the coordinates of the output image.

$$\lambda = xh_{31} + yh_{32} + h_{33} \tag{2.11}$$

From the equations 2.10 and 2.11, the values of x_h and y_h are done by the equations 2.12 and 2.13 respectively.

$$x_h = (xh_{11} + yh_{12} + h_{13})/(xh_{31} + yh_{32} + h_{33}) \tag{2.12}$$

$$y_h = (xh_{21} + yh_{22} + h_{23})/(xh_{31} + yh_{32} + h_{33}) \tag{2.13}$$

A homography can be used to build a Bird's eye view using a perspective transformation of an image, in this case a homography is used to build the Bird's eye view. The Bird's eye view corresponds to a virtual camera placed above the scene. See figure 2.4.

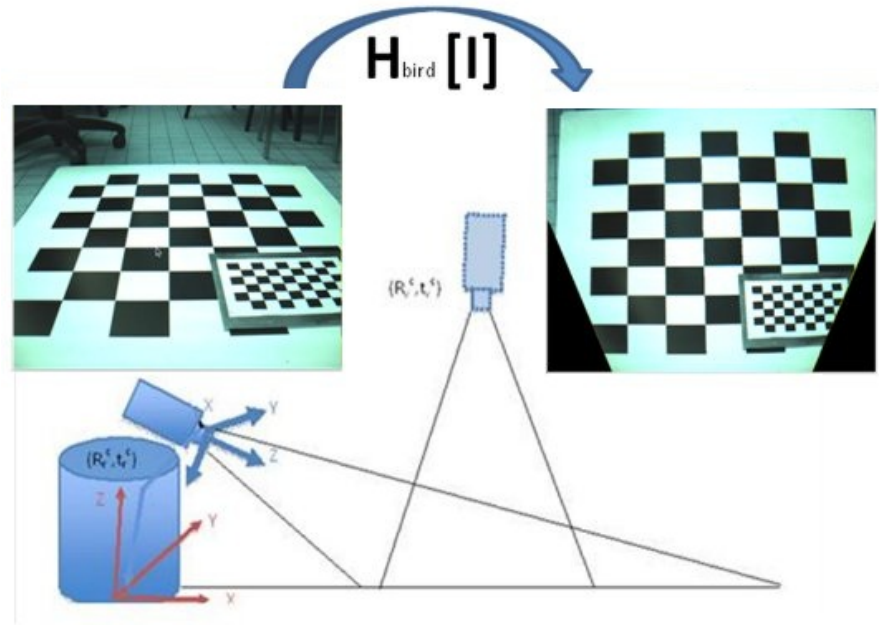

Figure 2.4: Bird's eye view transformation - Bird's eye view transformation.

In [14], homography is used to estimate the trajectory of a vehicle using a single camera placed on the roof of a car. In [15] and [16], it is used to generate X-ray panoramic images of bones using different views. In [17], detection and reconstruction of obstacles structures located over a ground plane is presented using homography.

For drive assistance, some vehicle fabricants have developed systems based in perspective transformations to rebuild a Bird's Eye view and a Side view of the car. The company Xylon has developed a multi-camera system which uses four fish-eye cameras and an FPGA to rebuild a synthetic image in real-time that corresponds to the different perspective views of the vehicle, including the Bird's eye view. See figure 2.5. A similar system is beginning to be commercialized by the company BMW.

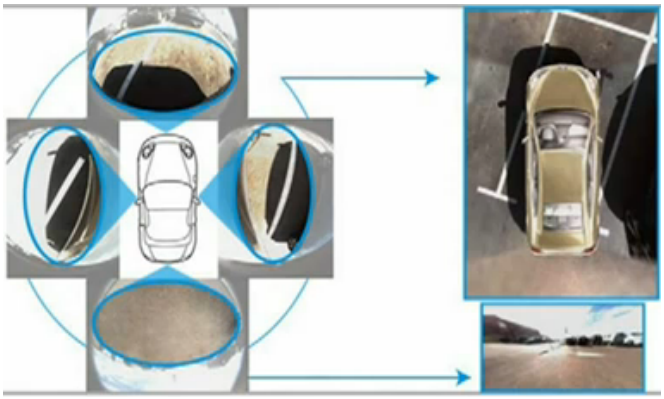

Figure 2.5: Xylon demo test vehicle - The system correct the lens distortions of the fish eye views, implements perspective transformations, and then displays a synthetic image. Taken from Xylon

The hardware acceleration of homography is useful for many applications included IPM. In [18] and [19], FPGA implementations of homography are presented; in both cases a SRAM memory is used.

2.5 Infrared

The human eye is sensible to wavelengths between 350 nm and 740 nm, this band of wavelengths is called visible spectrum. The infrared wavelengths are longer than that of visible light and includes wavelength that are between 740 nm and 14 um. Image 2.6 shows the electromagnetic spectrum (represented in function of the wavelength).

The infrared spectrum is subdivided into five gaps:
1. NIR (Near Infrared)
2. SWIR(Short-wavelength infrared)
3. MWIR(Mid-wavelength infrared)
4. LWIR(Long-wavelength infrared)
5. VLWIR(Very Long-wavelength infrared)

Using special cameras, wavelengths other than visible, can be observed by a human. Visible cameras capture the reflective light properties of the objects in the scene, while IR cameras are sensitive to the thermal emissivity properties of the objects.

The radiation measured by a IR camera comes from three sources:
1. Object's inherent temperature 2. Surroundings that was reflected onto the object's surface
3. Radiation that passed through the object from behind.

Thanks to the advances in infrared imagery, nowadays, we have High Dynamic Range (HDR) infrared cameras with a sensitivity of 16 bits. Usually the useful information in an IR image

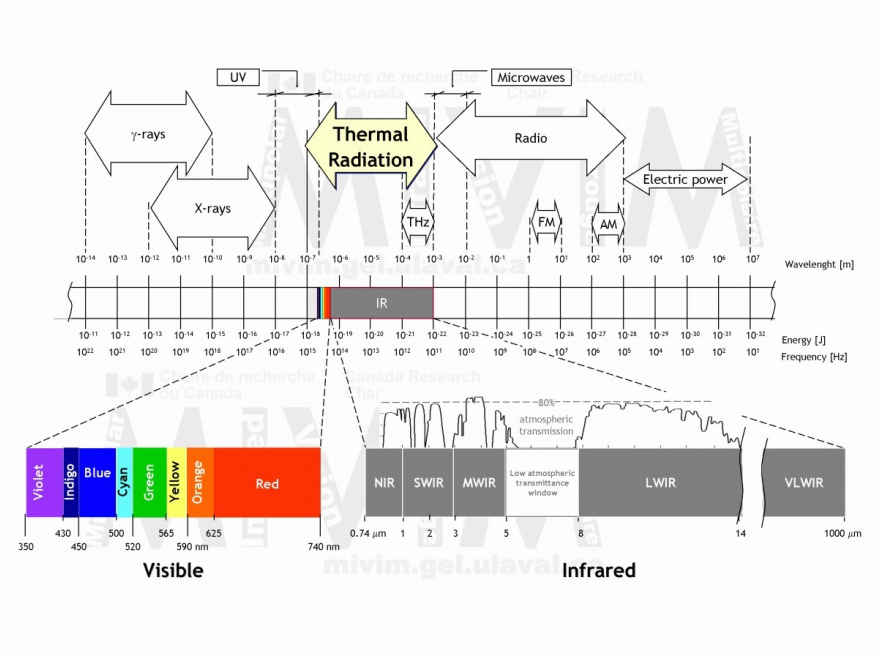

Figure 2.6: Electromagnetic Spectrum - Taken from http://mivim.gel.ulaval.ca

is concentrated in a little portion of this dynamic range; IR images have extremely low SNR (signal to noise ratio).

Different kind of IR sensor exists, with different spectral response according to their technology. See figure 2.7. IR cameras detect radiation in the electromagnetic spectrum with wavelengths from roughly 900 nm to 14,000 nm and produce images from that radiation. Common silicon detectors are sensitive to about 1050 nm, while InGaAs sensitivity starts around 950nm and ends between 1700 and 2600 nm, depending on the specific configuration.

The most common technology used in infrared cameras is the Microbolometer, which allows for the fabrication of uncooled cameras.

Infrared images are independent of the illumination conditions, giving a similar response during night and day. Thanks to their invariability respect to the illumination, IR cameras can be used to detect obstacles under low illumination conditions that would not be possible with a visible camera. This can be very useful for driving assistance. See figure 2.8.

2.6 Contrast Enhancement Techniques

Methods presented in the literature for contrast enhancement of images can be divided into two broad classes: contrast enhancement techniques and Dynamic Range compression (DRC) techniques. The main difficulty when implementing contrast enhancement is avoiding the noise amplification and degradation of the image quality.

Contrast enhancement techniques are widely used in image processing. These techniques may be generally divided into two categories: Unsharp-Masking-Based Methods (UMBMs) and Histogram Equalization-Based Methods (HEBMs).

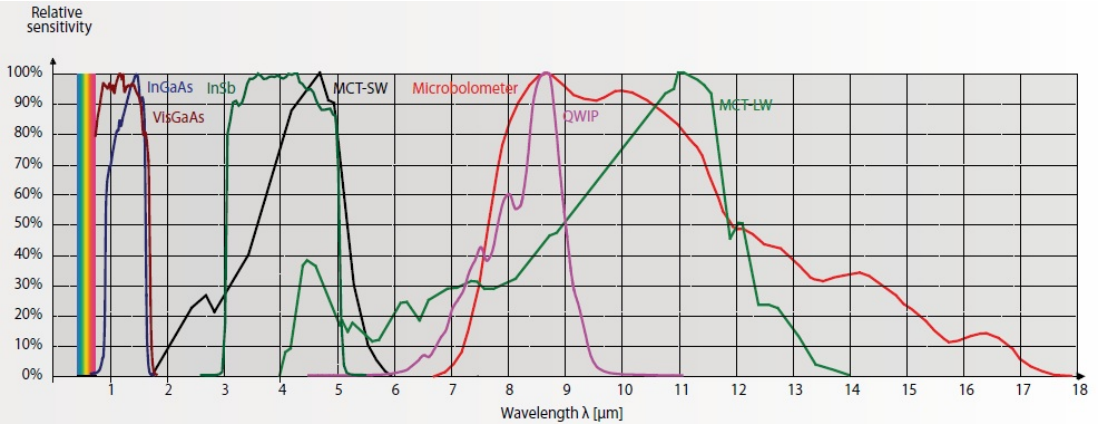

Figure 2.7: Relative response curves for different IR sensor materials - taken from [FLIR]

Figure 2.8: Infrared and visible view of a day and night scene - Infrared image was equalized using CLAHE

UMBMs [20] decompose the image into a low-pass and a high-pass component. A fraction of the high-pass image is added to the original image. The two main drawbacks of UMBMs are noise amplification in uniform areas, and generation of artifacts due to over enhancement of high-frequency details.

HEBMs are the most used techniques for contrast enhancement thanks to its simplicity and effectiveness. HEBMs are based in a redistribution of the gray scale levels of an image. These techniques are based on statistical characteristics of the image contained in the histogram of the image. From the histogram information, a transformation function is found to process each pixel of the image.

HEBMs are divided into two groups: global histogram equalization-based methods GHEBMs, and adaptive histogram equalization-based methods AHEBMs.

GHEBM uses a unique transformation to process all pixels of the image. In doing so, dark and bright regions of the image are treated equally and their contrast can be deteriorated. GHEBM gives good results when the distribution of the pixels is similar in the entire image.

Applying a GHEBM can increase the noise in the output image. It can also hide the important information in the input image because small areas, possibly containing important information, could have a negligible contribution to the computation of the transformation function.

Ideally, to have the best results in term of contrast, one histogram should be computed for the neighborhood of each pixel, but it is a very expensive procedure. This can be approximated using a Tesselation of the image, so the image is divided in NxM rectangular regions and for each of these regions a histogram is computed.

AHEBM transforms each pixel according to the histogram of the neighborhood region. For this, the image is divided into regions, and one histogram is computed for each of them. For the center pixel of each region, the histogram can be used to compute the best transformation; but for the others pixels, an interpolation between the transformations of the neighbor regions should be performed.

In AHEBM, contrast is enhanced locally, bright and dark regions are treated independently. This technique is more expensive in term of resources, because several histograms must be computed; each one corresponding to a different area of the image.

The problem of the typical AHEBM is that this algorithm can over amplify noise in homogeneous region of the image. To avoid this, in Contrast Limited Algorithm Histogram Equalization (CLAHE), a Clip Limit is used to limit the maximum contrast amplification.

2.6.1 Contrast Limited Adaptive Histogram Equalization (CLAHE)

In CLAHE, the image is divided in N sub-images. Then, for each one of the sub-images, a histogram must be computed. The histogram is clipped using a Clip Limit and all the bins above this clip limit are distributed. After the distribution of the excess, the Cumulative Distribution Function (CDF) is computed using equation 2.14. See figure 2.9. Finally, from the CDF a Look-Up Table (LUT) is computed using the equation 2.15.

$$cdf = \sum_{x_i<x} p(x_i) \tag{2.14}$$

Here $p(x_i)$ corresponds to the probability of each gray level scale, given by the histogram.

$$LUT(v) = round((cdf(v) - cdf_{min})/((MN) - cdf_{min})(L-1)) \tag{2.15}$$

To avoid the over amplification of noise generated by the typical AHEBM, the contrast amplification is limited with a CLIP LIMIT (Beta). Limiting the slope of the histogram mapping

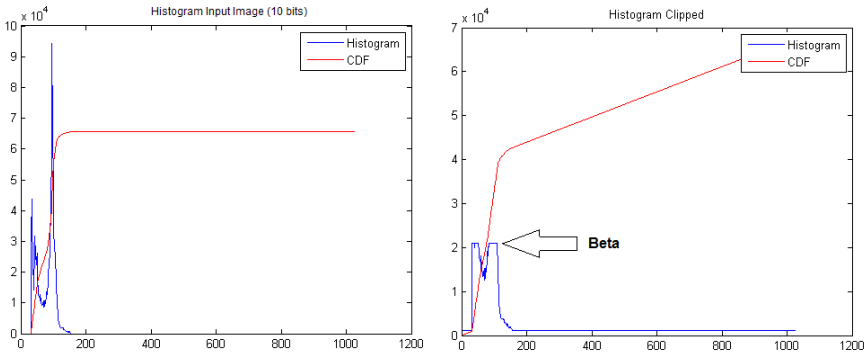

Figure 2.9: CLAHE histograms. - On the left, the histogram and the CDF. On the right, the histograms was Clipped, changing the CDF.

function is equivalent to clipping the height of the histogram. This limit is defined by the equation 2.16, and depends of the parameters α and S_{max}. Values of S_{max} and α are typically chosen empirically.

$$Beta = M/N(1 + \alpha/100(s_{max} - 1)) \tag{2.16}$$

where,
M = Width image
N = Height image
M N = Number of pixels
L = Gray-scale levels.
S_{max} represents the maximum slope
α = clip factor, it is a positive value between 0 and 100.

In the equation 2.16, when α is zero, the clip limit corresponds to M/N, which is equivalent to an uniform distribution. When α is equal to 100 (maximum allowed value), the clip limit is equal to $S_{max}(M/N)$, which corresponds to limit the maximum number of counts for each gray-scale value to Beta, and redistribute the extra counts between all the gray-scales with count less than Beta.

This algorithm originally needs recursion [21], which is not easily implemented in hardware. In [22] is shown that the redistribution of the excess can be approximated by a single step procedure. This single step procedure redistributes the initial excess only once among the bins that do not initially exceed the Clip limit. The bins that are pushed over the limit are clipped again and the secondary excess is discarded.

2.7 Multispectral vision

Infrared images can give a lot of information of a scene; however, in some situations the information of the infrared source is not enough. For driving assistance, color information is necessary to observe important information coming from traffic lights, signalization in the airports, etc.

Simultaneous use of infrared and visible images can be useful for drive assistance. Multispectral vision allows observing a scene under some extreme conditions, for example, the presence of smoke or fog or the absence of enough illumination.

A synthetic image can be created using the fusion of visible and infrared images. One of the most important difficulties in building a fusion image from infrared and visible cameras comes from the disparity of the images due to the physical separation (baseline) between the cameras. The disparity is quite important for objects near the cameras.

If no transformation is applied to correct the disparity in the fusion image, phantom effects may appear. The disparity depends on the Baseline, the focal distances, and the distance of the objects in the scene.

The disparity is given by the equation 2.17.

$$D = (B\alpha_u)/Z \tag{2.17}$$

where,

B = Baseline (distance between the optic center of the cameras).

α_u = focal distance

Z = Distance of the objects.

The disparity is not the only difficulty when creating multispectral images, with the available technologies is necessary to use two different cameras with different parameters (extrinsic and intrinsic). It creates misalignment and magnification problems between images.

Figure 2.10 shows infrared and visible images of the same scene of a taxiway in an airport. From the infrared image, the pilot can infer that he can continue because there are no obstacles; however, in the visible image, even if it is a dark image, apparently without a lot of useful information, there are some red lights signaling the pilot not to continue. Other important information from this image corresponds to the lights on the edge and the middle of the taxiway. According to the colors of these lights, the pilot can have an idea about the localization of the aircraft in the airport.

Figure 2.10: Infrared and visible image of the stop bar in an airport context. - Infrared image was equalized using CLAHE

The use of infrared and visible images simultaneously is a subject treated by many authors.

In [23], a system is presented to detect cars on roads using visible and NIR images. Their method is based in the extraction of a signature of the cars' "stop lights" using catadioptries. After implementing histogram operations to both images (IR and visible), the difference between images was analyzed. Their method showed the detection of cars at day and night. They verified their algorithm with static and dynamic scenes. Their algorithm presented timing limitations of up to 1.4 seconds.

In [24], infrared and visible images were used to detect pedestrians. They compared the histograms of pedestrian generated from infrared and visible images and showed that infrared

histograms have less variation than that of the visible ones. In their approach, the bright pixels in infrared images first gives an estimation of the pedestrian's positions. They tested their algorithms during summer and winter seasons.

In [25], a cooperative tracking using visible and infrared cameras is presented. They showed that using the visible and infrared images simultaneously gives robustness against temporary occlusion. Their system worked normally with the visible image and when the target was lost they switched to the infrared sequence.

In [26], a system based on four cameras (two color and two infrared) is used to implement pedestrian detection. They generated two independent disparity maps for IR and visible cameras, thus, improving the true-positive rate of pedestrian detection compared to a visible only source.

In [27], several methods to fuse color images and IR images are presented. In their method, intermediate images are fused by assigning different linear combinations to the RGB channels (R=IR-visible, G=Visible-IR and B=visible-IR) and the results were shown within a gray level image generated with a linear combination of them (OUT=0.03R+0.59G+0.11B). This fusion method does not lose information, but gives results that can distract the pilot. They also experimented with the HSV color space to implement the fusion (H=IR-Visible, S=Visible-IR and V=Visible-IR); this fusion method worked well for daylight images but it failed with the nighttime images. Experiments were also done with fusion methods based on feature extraction. A Laplacian filter was used to extract the edges of the visible image, then they added the edge image to the fused image. In all their approaches, they began with a projective transformation to warp the images (correct the rotation, magnification, translation and tilt).

In [28], a fusion method based in three steps is proposed; first they re-sized IR and visible images, then they localized the IR image in the visible one, and finally they treated both images and implemented the fusion in the red channel of the visible image. They chose the red channel because the red is usually the color to represent the thermal information.

2.8 Stereo Vision

The objective of the stereo vision is to recover the 3D information of a scene using two or more pre-calibrated cameras. In stereo vision two cameras separated by a known distance (Baseline) are used to build a map containing the depth information of different objects in a scene, this map is called Disparity Map.

The stereo vision is based on the triangulation principle shown on the left side of the figure 2.11. A point P observed from two different cameras will be perceived in different places in the focal plane of each camera. If camera parameters are known, the distance of the point P from the focal Plane can be computed.

The Disparity map is built with the correspondences between the Left and Right images using triangulation. To build the map, each pixel of one of the cameras must be matched with another pixel of the other camera. Some pixels will not be found because of the existence of occlusions and the different Fields of View of the cameras.

Stereovision algorithms typically work under the hypothesis that the cameras have been calibrated. The stereovision calibration must guarantee the epipolar restriction. Figure 2.11 ilustrates this restriction.

Thanks to the epipolar restriction, matching research should be only executed in the same row for both images. Due to this constraint, the research process can be reduced to a one dimension process and overall reducing the complexity of the problem.

In order to match the pixels, a signature is built for each pixel. The signatures of the pixels

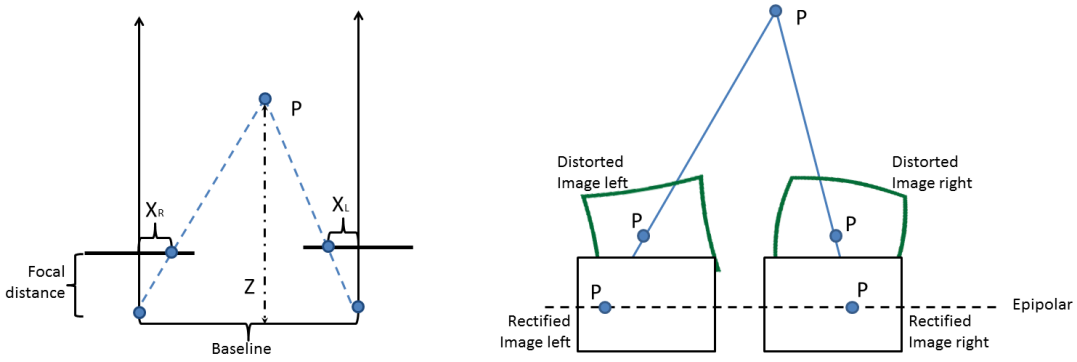

Figure 2.11: Stereo vision principles. - Left: Stereo vision triangulation principle. Right: Stereo vision epipolar restriction.

are matched between the images using a correlation. Different correlation scores have been used for this purpose: ZNCC, CC , SAD , Census, etc.

In [10], the disparity maps are built using the Census transformation. They showed that this correlation can give acceptable results and it is very adequate when used in an embedded system thanks to its low complexity. The Census transformation signature of a pixel is done by a bit string created with Boolean comparisons.

In [29], an FPGA implementation of stereo vision is implemented using the Census transformation. Using this correlation score, they were able to build disparity maps at 130 FPS with VGA resolution.

In the PICASSO project (described in the section 5.6), an especial base was conceived to align the stereo cameras using a mechanical system; this base is shown in the figure 2.12. However, given that the stereo algorithm is very sensible to misalignments, the results obtained were not very satisfactory. In this thesis, we implemented the architecture to perform image distortion correction at very high frequencies (up to 130 FPS).

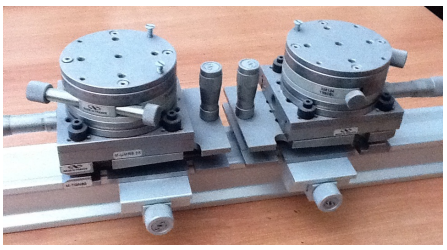

Figure 2.12: Configurable mechanic base for stereo vision -

2.9 Obstacle detection

Obstacle detection is a widely studied problem in the robotics and intelligent vehicles communities. LIDAR (Light Detection And Ranging) or RADAR (Radio Detection And Ranging) based methods are integrated in up-market cars but are generally limited by the nature of the

information they provide (depth only). Another option to detect obstacles is to use cameras as the only source of information from which a wide variety of features can be extracted.

Obstacle detection is a problem that has been studied for two or three decades, both in the robotics and the Intelligent Vehicles communities. Only visual-based methods are considered here; laser-based or radar-based methods are already integrated on upmarket cars, but generally, they are limited because sensory data are only acquired on a plane parallel to the ground, and are too poor to deal with interpretation.

2.9.1 Texture based Obstacle detection

Different approaches using Texture have been used to detect obstacles; In [11], color and texture characteristics are used in order to learn the ground appearance, and an Adaboost classifier to identify obstacle as non-ground areas in an image. Their approach was based on a classification using the LAB color space, they implemented an offline learning procedure using a database of the obstacles. Textures attributes used were mean, variance, correlation, contrast, homogeneity, cluster shade and cluster prominance. See figure 2.13.

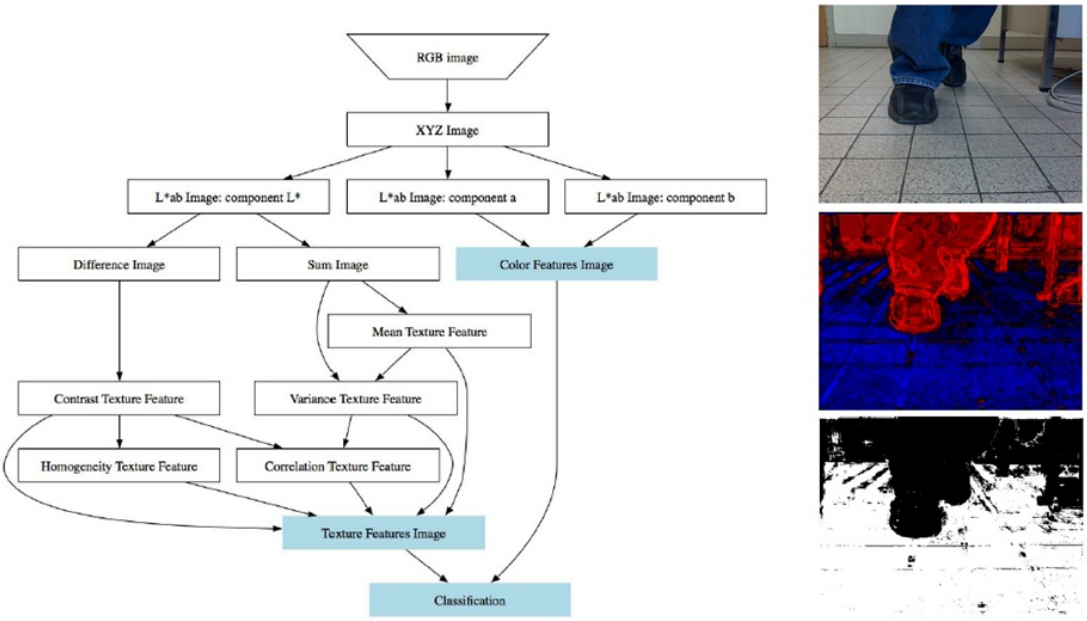

Figure 2.13: Texture object detection algorithm - Taken from [11]

The main disadvantage of this approach is the false alarms caused by the textures that have not been learned during the training process. In [30], an obstacle detection algorithms is presented. In their approach, textures attributes were learned online.

2.9.2 Inverse Perspective Mapping (IPM)

The IPM algorithm is a vision-based method that allows obstacle detection under the hypothesis of a flat ground. IPM is based on the perspective effect perceived from a scene, when observed from two different points of view. This method is based on a homographic transformation,

which allows the elimination of this perspective effect for the pixels in the ground plane. After eliminating the perspective effect for the ground pixels, subtraction between the camera views will reveal pixels that are not in the ground plane.

The IPM method was introduced in [31] and then exploited for obstacle detection [32] [33] [34] and for lane detection [35]. The IPM is a spatiotemporal algorithm that allows for the detection of obstacles from two images I_1 and I_2, either acquired by a moving camera [31], or by a stereoscopic sensor [32] knowing the relative positions between the two images. It uses a 3×3 homography matrix denoted H to transform one image, e.g. I_1, in order to remove the perspective effect with respect to I_2 for the pixels contained in the ground plane.

When applying the homography to I_1 denoted $H[I_1]$, the perspective of the ground points with respect to I_2 is removed, while the points outside of ground's plane lose their geometry. For example, a circle can become an ellipse. Therefore, the difference between $H[I_1]$ and I_2 will be almost null for the points contained in the ground plane.

2.9.2.1 Monocular IPM (MIPM)

In mono IPM, a single moving camera is used. Figure 2.14 represents two different views of a robot after a movement Mvt.

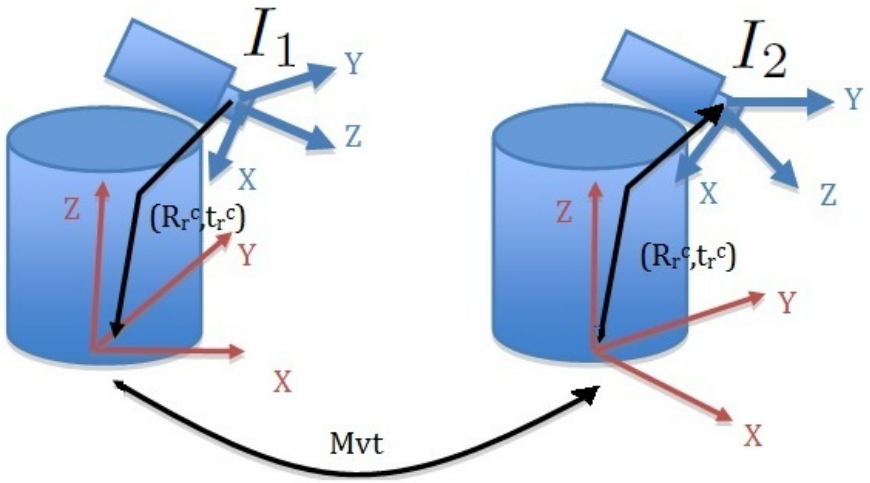

Figure 2.14: IPM coordinate notation -

The movement of the camera with respect to the world is expressed with the equation 2.18.

$$\begin{bmatrix} R_w^c & t_w^c \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_r^c & t_r^c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{mvt} & t_{mvt} \\ 0 & 1 \end{bmatrix} \tag{2.18}$$

The homography H between the two camera views is computed from the intrinsic (K) and extrinsic(R_r^c, T_r^c) camera parameters and from the known camera motion (Mvt).

IPM uses the camera Pinhole model shown in equation 2.1. Because IPM is intended to work for the ground points, the constraint (Z=0) is applied to equation 2.1, resulting the equation 2.19.

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = KR \begin{bmatrix} X_w \\ Y_w \\ 0 \end{bmatrix} + Kt \tag{2.19}$$

Applying algebraic properties, the term Kt is included in the H matrix, resulting equation 2.20.

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} = H \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{2.20}$$

Two image acquisitions from two different points of view are necessary to implement IPM algorithm, these acquisitions are expressed with equation 2.21 and 2.22 respectively.

$$\begin{bmatrix} s_1u_1 \\ s_1v_1 \\ s_1 \end{bmatrix} = K \begin{bmatrix} r^c_{r1} & r^c_{r2} & t^c_r \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} = H_1 \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{2.21}$$

$$\begin{bmatrix} s_2u_2 \\ s_2v_2 \\ s_2 \end{bmatrix} = K \begin{bmatrix} r^c_{w1} & r^c_{w2} & t^c_w \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} = H_2 \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{2.22}$$

If we compute (X_w,Y_w) from the equation 2.21 and replace in equation 2.22 we find equation 2.23.

$$\begin{bmatrix} s_2u_2 \\ s_2v_2 \\ s_2 \end{bmatrix} = K \begin{bmatrix} r^c_{w1} & r^c_{w2} & t^c_w \end{bmatrix} \begin{bmatrix} r^c_{r1} & r^c_{r2} & t^c_r \end{bmatrix}^{-1} K^{-1} \begin{bmatrix} s_1u_1 \\ s_1v_1 \\ s_1 \end{bmatrix} \tag{2.23}$$

Therefore the image coordinate of a ground point $(X_w, Y_w, 0)$, taken from two different points of view, can be computed using a homography transformation, like shown in the equation 2.24.

$$\lambda \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = T_{ipm} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} \tag{2.24}$$

Therefore T_{ipm} is given by the equation 2.25 and the term λ depends of the parameters h_{31},h_{32}, and h_{33}.

$$T_{ipm} = K \begin{bmatrix} r^c_{w1} & r^c_{w2} & t^c_w \end{bmatrix} \begin{bmatrix} r^c_{r1} & r^c_{r2} & t^c_r \end{bmatrix}^{-1} K^{-1} \tag{2.25}$$

The equation 2.25 corresponds to the Homography matrix that must be used to transform I_1, in order to align the ground pixels of I_1 and I_2. It means that after applying the homography to I_1, and implement a subtraction with I_2 the value of all the pixels in the ground plane will be almost null. And the pixels that are not part of the ground plane will be distorted, which creates an anomaly in the subtraction image, allowing the detection of the obstacles.

2.9.2.2 Stereo IPM (SIPM)

While MIPM is implemented with one camera that takes images from two different points of view, SIPM is based in a Stereo Pair of cameras, which their relative position is known.

SIPM is based on the perspective effect of the left image with respect to the right image. Because in SIPM the relative position between the two cameras (left and right) is fixed, the homography transformation to remove the perspective effect for the ground plane is constant.

For SIPM, from equation 2.26 setting the constraint $h_{33} = 1$, we have the equation 2.27, which is used to compute the parameters of the homography matrix.

$$\lambda \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{2.26}$$

$$\begin{bmatrix} . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ x_i & y_i & 1 & 0 & 0 & 0 & -u_i x_i & -u_i y_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -v_i x_i & -v_i y_i \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} . \\ . \\ . \\ u_i \\ v_i \\ . \\ . \\ . \end{bmatrix} \tag{2.27}$$

In SIPM for a closed-form solution of matrix H parameters, four points correspondences between images left and right are needed (each correspondence gives 2 equations). Nevertheless point correspondences are noisy, the use of more than four points can give a better solution, and in this case the equation system is over determined and must be solved using a classical least squares method [13].

2.10 Points of Interest

The detection of interest points is used in image processing for multiple applications: object recognition, SLAM, 3d reconstruction, etc. Multiple Point of interest detectors exist in the literature such as Harris [36], Shi and Tomasi [37], etc..

A point of interest is usually associated with a descriptor to implement a match. Multiple authors use the vicinity of the pixel like the descriptor of a point of interest; however, because the descriptor is usually used to implement multiple correlations, the descriptor must be optimized.

Another reason of the existence of different descriptors is the sensitivity of some descriptors to variations due to rotation, illumination changes, etc. Different feature descriptors exist in the literature such as SURF, SIFT, BRIEF, etc.

In the context of this thesis, the Harris point detector will be used because of the good trade-off between its simplicity and satisfactory results. The Harris point detector will be used to detect the landmarks needed to build the navigation map of the robot using the SLAM algorithm.

We evaluate the use of the BRIEF descriptor and different sized vicinity windows like feature descriptors.

2.10.1 Harris point detector

The Harris detector allows for the detection of edges and corners in the images. This point detector presents a good trade-off for a real time application: limited computation time, and good invariance to rotation, scale, illumination variation and image noise.

The Harris detector is based in the gray level partial spatial derivatives: Ix(,) and Iy(,) in x and y, respectively. The derivatives of image I are computed for each pixel using the kernels shown in the equations 2.28 and 2.29.

$$Kernel_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.28}$$

$$Kernel_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.29}$$

From the partial derivatives, a symmetric matrix M is built using equation 2.30. The matrix M is multiplied with a weight function w(x,y) to smooth the correlation. The weight function w(x,y) is a Gaussian mask gm. The Harris response R is determined by the eigenvalues of the matrix M and an empirical value k (usually between 0.04 and 0.06).

$$M = \begin{bmatrix} \sum(I_x^2) & \sum(I_x I_y) \\ \sum(I_x I_y) & \sum(I_y^2) \end{bmatrix} \tag{2.30}$$

According to the value of the response, the features in the images can be classified. See figure 2.15. If the magnitude of the Response is small, the point corresponds to a flat area. If the magnitude of the Response is big, it corresponds to:

Corner if $R > 0$

Edge if $R < 0$

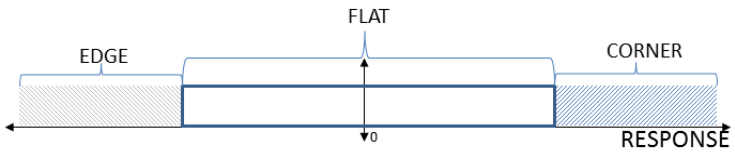

Figure 2.15: Harris response feature classification -

Harris response is computed using the equation 2.33.

$$R = det(M) - k(trace(M))^2 \tag{2.31}$$

where,

$$det(M) = \sum I_x^2 \sum I_y^2 - (\sum I_x I_y)^2 \tag{2.32}$$

$$trace(M) = \sum I_x^2 + \sum I_y^2 \tag{2.33}$$

For each point of the image, there is associated a response value. The interest points are chosen according to the final application based on the response. In [38] an algorithm is used to select the best Harris points using the sort algorithm presented in [39]. They use a filtering process to implement suppression of point localized near the chosen interest point, this way the interest points are distributed around the entire image, and avoiding to find multiple contiguous points.

In this thesis, our objective is to use the Harris detector to be used with the RT-SLAM, presented later in section 2.12.2. The RT-SLAM implements a Tessellation of the image, it means that the image is sub-divided in N sub-images, and the point with the best Response in each sub-image is used.

2.10.2 Binary Robust Independent Elementary Features (BRIEF)

The BRIEF descriptor is a very fast descriptor based on the intensity differences between random selected pixels in the vicinity of the interest point. See figure 2.16.

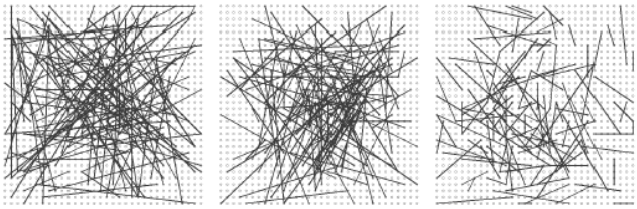

Figure 2.16: Brief comparison patterns - The image shows three different comparison patterns. This algorithm compares the magnitude of pair of pixels in a random way

In BRIEF, the descriptor of a interest point is usually encoded in a 128 or 256 bits chain. This chain is built computing simple comparisons between pairs of pixels in the neighborhood of the interest point, and then, according to the comparison, a bit 1 or 0 is assigned to the chain. This chain of bits can be used to implement a very simple research of the pixel using a simple Hamming distance.

In [40], the accuracy of BRIEF is compared against other well-known descriptors like SURF and SIFT; they showed similar recognition accuracy for the BRIEF descriptor.

2.11 Contour Detection

Contour detection can be used to detect the lines on the streets. Different contour detection algorithms have been studied as Sobel, Canny , Hough. We are just interested in the development of the Sobel operator because its good trade-off between simplicity and expected results.

2.11.1 Sobel

This operator uses the next two 3 by 3 kernels to approximate the gradients in X and Y.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.34}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.35}$$

Then, the Gradient is defined by the equation 2.36.

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.36}$$

Sobel detector can be used to detect the gradient's direction.

2.12 Simultaneous Localization and Mapping (SLAM)

One of the important aspects of robot navigation is the ability to locate itself and create a map of its surroundings. SLAM (Simultaneous Localization And Mapping) algorithms target such a goal by fusing multiple sensor inputs (depth scanner, inertial measurement unit, gps, vision...) to compute a position of the robot in its environment and to build a map.

The map contains positions of the robot and of landmarks in the 3D environment; a landmark is a 3-D point observed by a camera as an interest point.

SLAM allows a mobile robot to simultaneously locate itself and build a map of its surroundings. Vision-based SLAM tracks visual features extracted from camera images.

Vision based SLAM uses regions of interest extracted and tracked along images acquired from N embedded cameras while the robot is moving. The precision of the estimated robot position and of the built map depends on the number of tracked landmarks.

Vision based SLAM [41] [42] [43] allows a robot to operate in an unknown environment; building a stochastic map from information acquired by sensors such as inertial measurement units (IMU), global positioning (GPS) and cameras.

Different algorithms have been used to solve the SLAM problem. Typically, SLAM is solved using the Extended Kalman Filter (EKF); however, the disadvantage of this approach is the computer complexity due to the size of the big matrices used to represent all of the information. Optimization techniques have been used to implement SLAM using some optimizations (BUNDLE Adjustment). In [44] a detailed analysis presenting a comparison in terms of computational cost between bundle adjustment SLAM and EKF slam is presented, they explain why filtering techniques are more beneficial for small processing budgets.

Using visual information becomes compute-intensive as the number of tracked features directly impacts the SLAM precision and reliability. In this thesis, we propose to use an FPGA based accelerator as an image processing front-end for SLAM. The design flow started with high level tools; modules were created in HDL and compared against a software implementation. The resulting accelerator works in real-time with an infrared and a visible camera to provide visual information even in harsh conditions. The accelerator implements all of the necessary image processing and is designed to communicate with a general purpose host processor.

SLAM is an expensive technique in terms of resources, as the computation complexity is proportional to the number of tracked features. Vision-based SLAM exposes a lot of potential parallelism in the computer vision tasks (landmark detection and tracking). Today's architectures based on GPU's or multi-core processors, can take advantage of such parallelism to help speed-up the algorithm but do not satisfy constraints of embedded system: small power consumption, compacity, reliability... FPGA-based architectures offer the opportunity to design specialized hardware adapted to an algorithm in order to exploit all the available parallelism.

Embedded SLAM is a new interest domain in robotics because of the need to implement autonomous robots for critical missions like: Planetary exploration [45], Nuclear plant maintenance, and many others. Few authors have worked the SLAM algorithm using an embedded system with limited resources. In [46], an FPGA implementation using EKF is presented; the main development is centered in the parallelization of matrix operations. In [47], an FPGA implementation of SLAM using genetic algorithms and a Laser Range Finder is presented; its implementation is limited by the tendency to accumulate rotational errors. They achieved

an acceleration of about 14x compared to a CPU implementation using a Hardware-Software system on an FPGA Virtex 5.

When building a map of a place using the SLAM algorithm, detected interest points will be used to reconstruct a representation of the world. When using a camera, there are distortions that affect the projection and retro-projection of the points of the image to the real world. For SLAM, it is necessary to have an exact reconstruction of the scene; for this reason, some kind of image correction distortion algorithm is necessary. In almost all the SLAM applications just interest point are corrected, in the approach presented in this thesis, we correct all the image, because we need to work with an undistorted image for other algorithms like the IPM. See figure 2.17

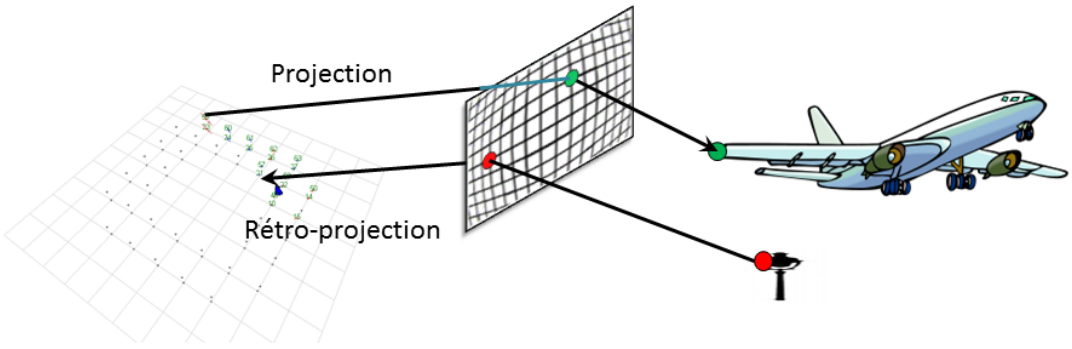

Figure 2.17: Calibration necessity for SLAM projection, illustrates the projection, retro-projection operations done during the SLAM algorithm. -

2.12.1 EKF SLAM

The typical Extended Kalman Filter (EKF) SLAM algorithm is composed of three basic steps; executed when a new image is acquired: Prediction, Correction and Initialization.

The Prediction step uses IMU, GPS and the robot model in order to estimate the robot motion since the last known position. The Prediction step also predicts the current robot position and where landmarks observed in previous images, and the current one, are located according to the map uncertainty.

The Correction step looks for true observations of these landmarks from an active search function, and computes a prediction error for each one, i.e. the discrepancy between the true and predicted positions of its observation in the current image. The prediction error (called innovation) is then used to update the map, i.e. to correct the robot and landmarks positions. The prediction error (called innovation) is then used to update the map, i.e. to correct the robot and landmarks positions.

In the EKF, we have a noisy measurement y done by equation 2.37.

$$y = h(x) + v \tag{2.37}$$

where, $h()$ is the observation function, x is the full state and v is the measurement noise.

The correction step is described with the next equations:

$$\bar{z} = y - h(\bar{x}) \tag{2.38}$$

$$Z = HPH^t + R \tag{2.39}$$

$$K = PH^t Z^{-1} \tag{2.40}$$

where, H is the Jacobian $(\delta h(X)/\delta x)$, R is the covariance matrix of the measurement noise and K is the Kalman gain.

The filter update is given by equations 2.41 and 2.42

$$x \leftarrow \bar{x} + K\bar{z} \tag{2.41}$$

$$P \leftarrow P - KZK^t \tag{2.42}$$

For the Correction step, the Active Search technique allows to avoid an exhaustive matching procedure in the entire image. This technique uses the prediction to search for a selected landmark in a region with a size proportional to the prediction uncertainty. In [48], advantages of Active Search are demonstrated using the information theory. In [49], it is shown that in real-time, with a high frequency image acquisition the active search method usually looks for matching in small regions.

Finally, for the Initialization step, points of interest are detected in selected areas of the current image, and corresponding 3D landmarks are initialized in the map. For the landmark initialization, the Harris detector can be used.

The vision-based SLAM precision and robustness can be improved by using sensors sensitive to different light spectrums. Working with a multispectral system allows for observing a scene in harsh conditions like smoke, fog, or in low illumination conditions.

2.12.2 RT-SLAM

In [50], RT-SLAM, an open-source implementation of inertial/visual SLAM algorithm and based on the EKF, works at 60Hz for images with VGA resolution. This SLAM implementation is designed to take advantage of the symmetry and sparsity of the involved matrices.

For the initialization process, the image is divided in sub-images and a Harris detector is used to detect new interest points. The point with the best Harris response in each sub-image is used.

In RT-SLAM, instead of processing the entire covariance matrix with a large size (more than 150x150 depending of the map size), 2 or 3 landmarks are processed in each algorithm iteration.

From the P matrix, a new submatrix containing the robot information and the landmark information is built. Using Inverse Depth Parametrization (IDP) representation, the robot information is contained in the top-right of the matrix. From the Landmark and the robot information, a new matrix of side 25x25 is used to implement the term HPH_t of equation 2.39. See figure 2.18.

For the correction step, RT-SLAM allows for the selection of different correlation algorithms: ZNCC, SAD, CC.

2.13 Conclusion

In this chapter we have presented many algorithms that could be used to make an ADAS, either for automotive or aircraft applications. Some previous works were done in the RAP team before

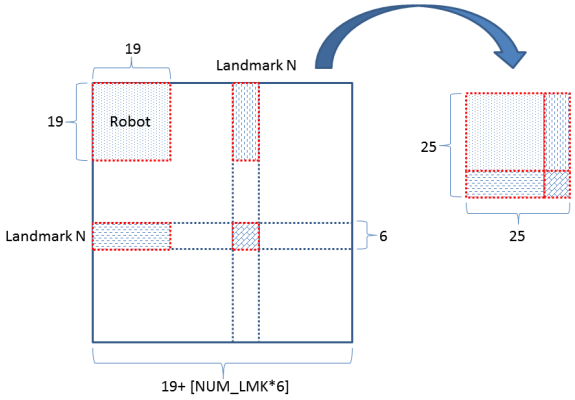

Figure 2.18: RT-SLAM decomposition of P before implementing HPH^t - Instead of process the entire matrix, in RT-SLAM each landmark is processed individually. This way only a sub-matrix of 25x25 must be processed. The sub-matrix is composed of a 19x19 matrix, which corresponds to the robot representation, plus the elements corresponding to the interest landmark.

my arrival to work in the SART project. I was involved in the stereo vision implementation, taking into account the rectification of images, involving image transformations. The same kind of operators was required also in order to build an occupancy grid from the results provided by an obstacle detection algorithms; so we began to work on these image transformations before to consider all functions required in our ADAS application.

The next chapter describes the platforms and methodologies availables to be used to deal with the design of an architecture dedicated to a set of algorithms.

3

Methodology and design of embedded systems based on FPGAs

3.1 Introduction

During the last decades, the evolution in embedded computers has grown thanks to the multiple advances in VLSI technologies. Before 1970, processors were composed of many integrated circuits, in 1971 the first processor into a single integrated circuit was developed by Intel. This processor called the Intel 4004 was a 4-bit processor with an operating frequency of around 100 KHz and composed of 2300 transistors using a manufacturing technology of 10 Micron. Nowadays we have 64-bit processors working at frequencies 30000 times bigger and composed of more than 1 billion of transistors with 22 nm manufacturing technology.

With our current technology we are arriving to a frequency operation limit due to limitations in the transistor fabrication process. At the nanometric scale used today, the transistor's gate is too thin, which creates an increase in the leakage current, increasing dramatically the static power. In the case of the FPGA fabrication, using specialized fabrication processes [51] [52], engineers have been able to create higher perfomance and lower power hungry transistors at the 28nm scale.

According to Moore's Law, the number of transistors is increasing, but the clock speeds are flattening. It explains why highest performance per watt and per chip area is achieved using parallel processing technologies. See figure 3.1.

The informatics computing has evolved from exclusive processing done by a single processor to real parallel processing. When using a single processor, some notion of parallel processing can be emulated with an Operating System (OS), which uses time multiplexation of resources between the different threads. Using a DSP, ASIC, FPGA, GPU or a multi-core system real parallel operations can be physically executed at the same time thanks to their special architectures.

In this chapter we will give a little insight to parallel processing units used in the design of embedded systems. Then, we will present the internal architecture of an FPGA and the basic tools used to manage complex systems using FPGAs. Finally, we will describe the difference

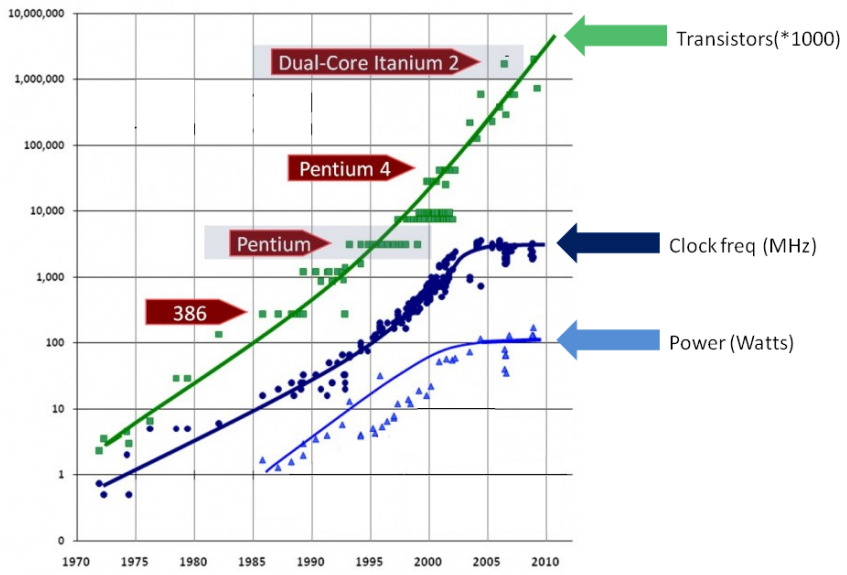

Figure 3.1: Moore's Law - The number of transistors and power consumption is constantly increasing, while the frequency is flattening. Taken from Kunle Olukotun and Herb Sutter.

between a Real Time Operating System (RTOS) and a General Purpose Operating Systems (GPOS).

3.2 Parallel Processing Platforms

For exhaustive parallel signal processing data, there are different available architectures; the most used today are: DSP, GPU, ASIC and FPGA. All of these architectures are different and each is used for a different purposes.

3.2.1 Digital Signal Processor (DSP)

A DSP is a processor system optimized to implement signal processing at very high speed. DSP's include a specialized architecture which allows parallel processing at the instruction level, this is called SIMD (Single Intruction Multiple Data). In the market there are floating point and fixed point DSP's.

From the programmer point of view, the parallel instructions are used with special assembler instructions included in the C program. Blackfin has 10-stage instruction pipeline. It has a hierarchical memory scheme to support memories running at the full speed of the processor. Up to 52 Kbytes of internal memory are available in the last Blackfin DSP's.

The DSP Blackfin 609 is a fixed point DSP based in a Dual-Core processor working up to 1GHz. The Blackfin arithmetic unit allows the execution of multiple operations in parallel: up to four 8-bit video ALUs or two multiplications and 2 accumulations of 32/40-bits.

3.2.2 Graphic Processing Unit (GPU)

Initially GPU's were designed as graphics accelerators, until engineers began to use them for parallel processing. A GPU consists of hundreds of small cores that can be used for graphics applications or high-performance computing. GPUs use programming models like CUDA and OpenCL, based on high-level languages like C, C++ or Fortran.

When working with GPUs, threading is handled automatically by the hardware thread manager. The programmer does not have direct control of the processors of the GPU; everything is done through Application Programming Interfaces (API).

In the current market we find three principal GPU providers: NVidia, Intel, and AMD. Some of the popular GPU from Nvidia are Geforce, ION, Quadro and Tesla. Tesla architecture is a very powerful GPU with the capacity to implement up to 4 teraflops in simple precision and 80 gigaflops in double precision. Quadro GPUs are intended to be used for professional graphics workstation. The GeForce is intended for traditional graphics market, Nvidia GeForce 8 architecture has 128 thread processors, each capable of managing up to 96 concurrent threads, for a maximum of 12,288 threads.

One of the main disadvantages of GPU's are related to cooling and energy. They are also large components, which are complicated to install in small places. GPU's are intended to be used like a co-processor, so it always works in parallel with a CPU, which limits their use in some embedded systems.

In [53], a time comparison between multiple image processing algorithms implemented in CPU and CPU+GPU is presented, they acheived a speedup of almost 100 times when using the GPU like a coprocessor. In [54], a comparison between FPGA, GPU and CPU perfomances are presented for different image processing algorithms (stereo vision and two-dimensional filters). In [55], tomography is accelerated 80 times respect the just CPU implementation using CUDA.

3.2.3 Application-Specific Integrated Circuit (ASIC)

An ASIC is a silicon device specialized to implement application specific tasks, in an ASIC the architecture is designed using a Hardware Description Language (HDL). In the ASIC the design is physically etched in the silicon using a fabrication mask. The main limitation of an ASIC is its price and the time to market, which limits its use to applications that requires very high performance or high volume production. ASIC are widely used in applications using RF signals such as radars, Television, etc.

The design using ASIC offers better performance, density and power consumption when compared to an FPGA. ASIC prototyping can be done using FPGAs, which allows taking advantage of FPGAs re-programmability. In [56] is described a design methodology to facilitate ASIC design based on FPGAs.

3.2.4 Field Programmable Gate Array (FPGA)

An FPGA is a general purpose silicon device composed of programmable gates, interconnection resources and other configurable modules that can be used to suit needs of a particular application.

The main element of the FPGA is the Logic Element (LE). Many LE are routed together with pre-fabricated interconnection resources and programmable switches. The LE of Altera and Xilinx are called Adaptive Logic Module (ALM) and Slice respectively. LE contains Look Up Tables, multiplexers, registers and other logic and connection resources. See figure 3.2.

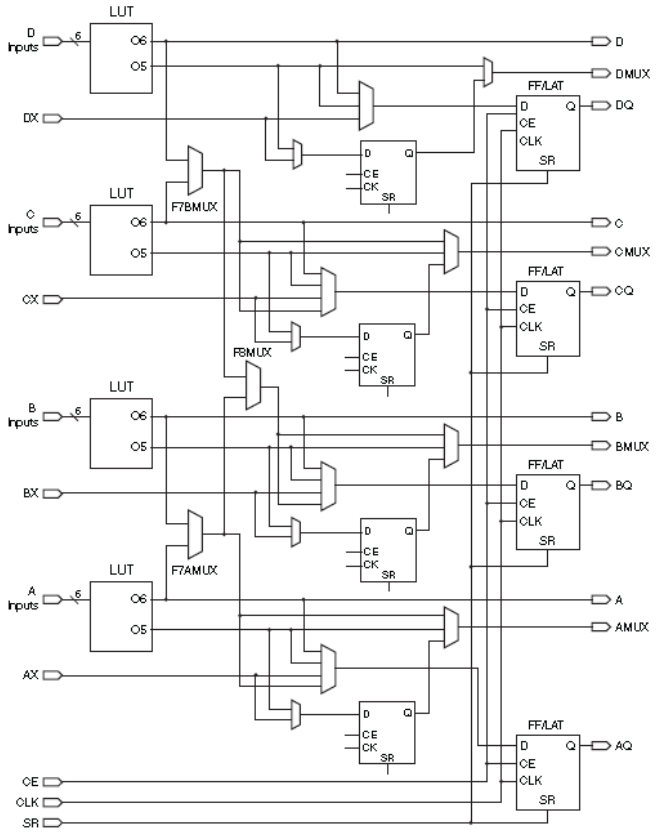

Figure 3.2: SLICE Virtex 6 simplified block diagram - Taken from [57]

Two Slices are grouped in Configurable Logic Blocks (CLBs) like shown in the figure 3.3. CLBs are connected to a Switch Matrix which allows connections to other CLBs.

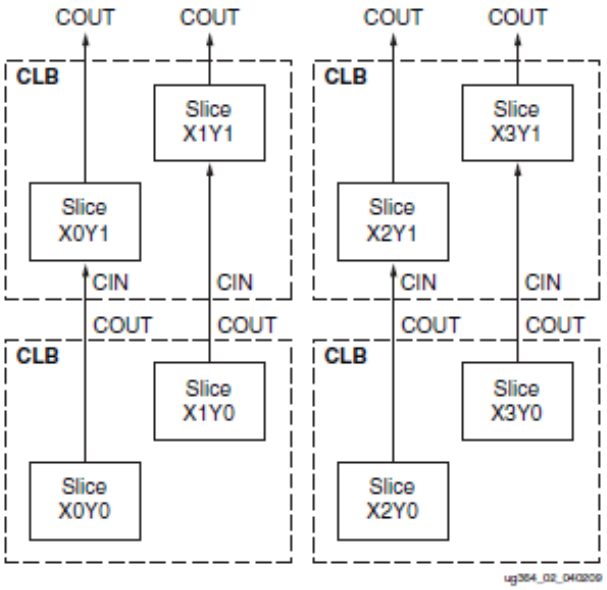

Figure 3.3: CLB block diagram - Taken from [57]

Virtex 6 FPGAs have a column based architecture, with all the different resources arranged in columns. See figure 3.4.

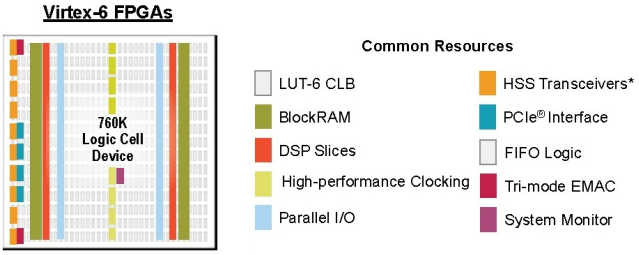

Figure 3.4: Virtex 6 column architecture. - Taken from [58]

In the FPGAs are silicon resources available to be used by the designer. The main resources available in the current FPGAs are hard Processors, RAM memory, Slices, DSP Slices, Multipliers, Gigabit transceivers, Triple-Speed Ethernet MAC, PCIexpress, Phase Locked Loop (PLL), etc.

In the FPGAs there are silicon multipliers, very useful in signal processing. In the case of Xilinx, in some FPGAs there are hundreds of small DSP blocks called DSP48. This module can be used to implement fixed point multiplication, additions, multiply accumulation (MACC), pattern detection, etc. The maximum performance of a DSP48 present in Virtex 6 FPGA is 600MHz obtained when using all the pipelines stages within the module. See figure 3.5

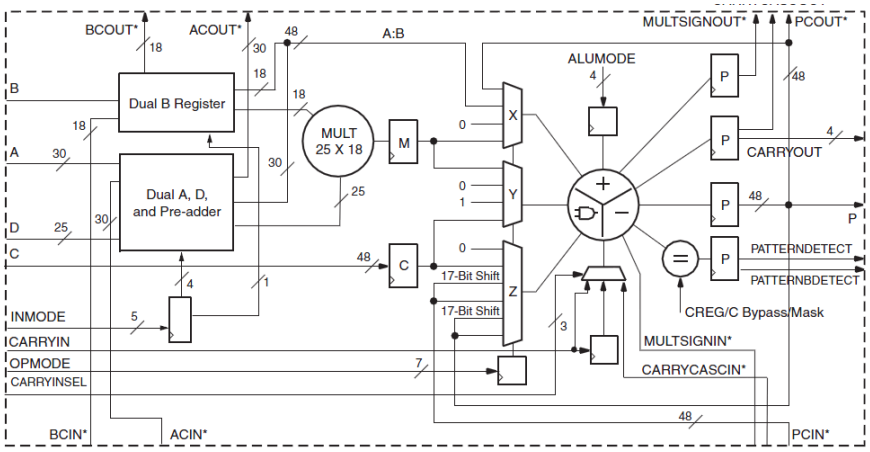

Figure 3.5: DSP48 block diagram - Taken from [59]

High communication speed protocols (PCIe, Aurora, Ethernet, etc.) are necessary in many applications, for this reason, FPGAs providers pre-fabricate in silicon into the FPGA gigabit transceivers called GTP for transmission and reception. See figure 3.6.

Thanks to the VLSI evolution, the miniaturization of transistors from 65 nm for Virtex 5 to 28 nm for Virtex 6, the capacity of FPGAs has been increased over six times. Today, one can find FPGAs with over 1 million LUTS, over 2 million registers, thousands of internal RAM and DSP blocks, many multi-Gigabit transceivers and many other resources. For this reason, in order to achieve the design goals, the methodologies used to develop and use FPGAs are in continuous evolution.

The table 3.7 shows some characteristics of different FPGAs of different vendors.

According to [61], FPGAs have a performance 100x higher than DSP in some highly parallelizable signal processing applications.

One of the main advantages of an FPGA is its flexibility to develop multiple tasks in parallel and in real time. With a General Purpose Processor (GPP), a program is executed in a sequential way, while in an FPGA, multiple processes can be executed in parallel.

Usually a GPP has a unique Arithmetic Logic Unit (ALU), which constraints the processor to execute only one arithmetic operation at a time. In a GPP the instructions execution are optimized to feed the ALU continuously with operands, but generally only a single result can be obtained on each clock event.

A GPP emulates multi-threading using processor resources in efficient way. For this, a scheduller controls resource multiplexation between the different threads, but only one thread is physically executed at a time.

An FPGA usually runs at lower speeds than GPP. Nowadays, processors can run up to more of 3 GHz, while the most powerful FPGA works at less than 1 GHz. What makes the difference when working with FPGAs is that many processes can be executed on each clock event. The number of process that can be treated depends of the algorithm and the available resources.

The maximum speed achieved to execute an operation on an FPGA depends on its nature and the way it was coded. For example, in a Virtex 6 FPGA many fixed point multiplications can be executed at 400 MHz using the embedded multipliers available in the FPGAs. In the case of V6lx240t there are 768 DSP48 [62]; that means, in theory it could be equivalent to use a processor of 460 GHz (768x600 MHz).

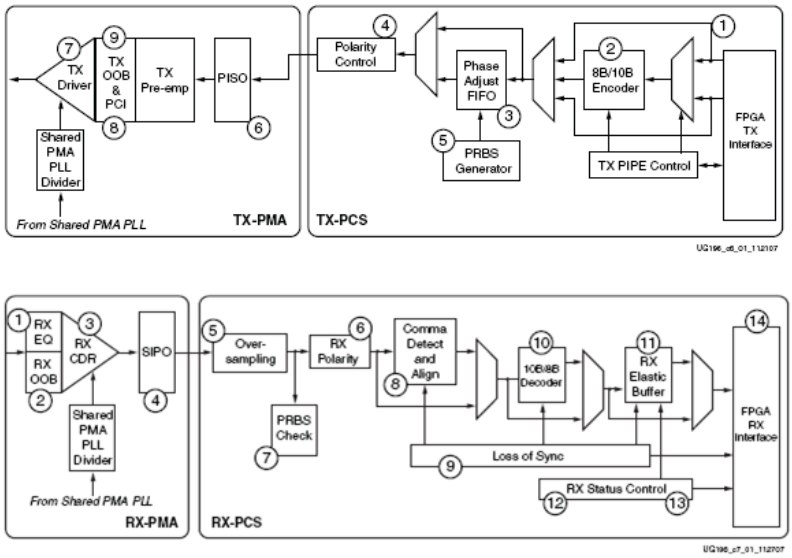

Figure 3.6: GTP block diagrams - Taken from [60]

Fabricant	ALTERA	XILINX	ACTEL
Famille	Stratix III	Virtex 5	FUSION+
Techno	SRAM	SRAM	FLASH
Processeur	NIOS II	PPC440 32bit Harvard architecture 7 stage pipeline 32k cache level 1 instruction et data	ARM Cortex M3
Max processeurs	NA	2	1
Type IP	Soft	Hard	Hard
Performance	300DMIPS	1100DMIPS	125DMIPS
Frequence	250Mhz	550MHz	100MHz
Rapport de perfo	1,2 DMPIS/MHz	2 DMIPS/MHz	1,25 DMIPS/MHz
Ressource bascule D	47k à 337k	20,4k à 122k	2300 à 38400
Resource RAM	1,8MBit à 18,4MBit	2,4MBit à 16,4MBit	27kBit à 270kBit
Resource FLASH	NA	NA	2MBit à 8MBit
ADC	NON	OUI	OUI
Nombre I/Os	296 à 1120 digital	360 à 960 digital	75 à 252 digital 20 à 40 analog
Utilisation ressources FPGA pour synthèse du processeur	3000LE (6%) 1500ALM (8%) plus petit STRATIX III 49500LE 19000ALM	0	0

Figure 3.7: FPGA devices comparison -

Thanks to the parallel architecture present in an FPGA, data processing can be drastically accelerated. The execution of the same algorithm in a just Software solution compared to a Hardware implementation take more clock cycles, because of the sequential nature of the Software approach. See figure 3.8.

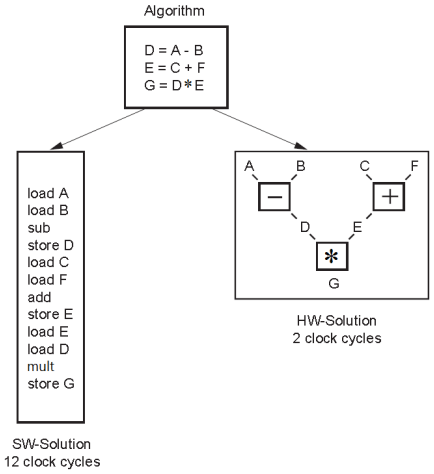

Figure 3.8: Comparison between hardware and software execution - Taken from [63]

There are 3 basic FPGA technologies: Static RAM (SRAM) Flash and Anti-fuse. According to the application, it is important to choose the suitable FPGA technology.

In the SRAM technology, transistors enable connections according to the information stored in an internal SRAM memory. The main advantage of this technology is that the components can be reconfigured, while disadvantage is the surface needed by the SRAM.

The Flash technology FPGAs are non-volatile, they have less power consumption on startup and offer more robustness against radiation with respect to SRAM technology. However, it does not have the same level of miniaturization of SRAM-technology, which reduces the density and increases the cost.

In the Anti-fuse technology, using a programming current a physical connection is permanently made between different points. This technology is less expensisve and gives best timing performance, however, the components cannot be reprogrammed.

3.2.5 Parallel Processing Platforms comparison

The different platforms presented in this section are widely used today world wide for different applications. The FPGAs present advantages and disadvantages respect to the others platforms in some aspects. See table 3.1.

The main disadvantage of the FPGAs are the design complexity that requires a big technical investisement, which implicates a longer time to market. With big FPGAs, just the compilation of the design can take hours, an then the validation of the design takes almost de 70% of the development time.

* In the case of the ASIC, the fabrication cost depends of the kind of production. If the component is used for mass production the cost is amortized, however, for unit production the ASIC design is very expensive.

Table 3.1: Parallel Processing Platforms comparison.

Platform	Size	Power	Flexibility	Reliability	Parallelism	Operation frequency	Design Complexity	Cost
FPGA	+	+	++	++	+++	+	–	–
GPU	–	–	+++	+	++	+++	+	+
ASIC	+++	+++	—	+++	+++	+++	—	(*1)
DSP	++	++	+++	+	+	++	++	++
GPP	++	++	+++	+	+	+++	+++	+++

3.3 Designing with FPGAs

When using an FPGA, the designer must take into account the architecture of the target, to efficiently solve a problem using in the best way the available hardware resources. It is important to do not forget that with FPGAs, even if the designer is writing code like in a C program, behind, the code is implementing physical connections between electronic circuits. These electronic circuits follow physical laws that constrains its use (propagation delays, fan out, power consumption, ...). When designing with FPGAs the designer must think in a parallel way to solve a problem to take advantage of the FPGA architecture, and not in a sequential way like usually is done when using a GPP.

The main design entries for FPGA development are the Hardware Description files and the Constraint Specifications file.

The Hardware Description files are generated by the user using a Hardware Description Language (HDL) or can be furnished by a third-party or the FPGA provider in the form of a Netlist(NGC). Tipically, a design contains multiple HDL files organized in a hierarchical way.

The constraint specifications are usually done using text files. The Constraint File (SDC for Altera and UCF for Xilinx) specifies important characteristics of some nets like operating frequencies, delays, timing offsets, multi-cycle paths, placing information, etc. These constraints will be used to drive the design implementation, this way the implementation tools will concentrate efforts in critical nets, and will not lose precious run time optimizing non-critical nets like the multi-cycle paths. A Multi-Cycle Path corresponds to a net in the system that can be executed in more than one clock event.

3.3.1 Hardware Description Language (HDL)

Hardware Description languages (HDL) can be used to describe the operation of digital or analogic electronic circuit. With the HDL, the functional description of a digital system is done using standardized text-based expressions. The most used HDL languages are VHDL, Verilog, and System Verilog.

The main characteristic of an HDL is that it allows the definition of concurrent statements, allowing the description of many processes executed in parallel. In a common used language like C, the program is executed sequentially.

For the development of this thesis we will basically use VHDL, which is a very rich language widely used around the world. The development of VHDL began in 1981 by the United States Department of Defence. Today, there are three main revisions of VHDL (1993, 2000 and 2002). VHDL-1993 is the most common used revision.

When using VHDL, there is the notion of a package. A package contains the definition of data types and operations between these types. There are standard packages necessary for

almost all design (STD LOGIC 1164, NUMERIC STD, STD LOGIC TEXTIO, STD LOGIC ARITH, ...). The user can also define its own packages.

An HDL includes some non-synthetizable statements used generally for simulation.

3.3.2 Setup and Hold

Due to the high density of the last generation FPGAs and the high frequency required by some algorithms, the propagation delays caused by the logic paths must be considerated.

A Flip Flop, present in LE, specifies a Setup and Hold time to guarantee a predictable behavior. The data arriving to a Flip Flop must become valid at least a specified time (setup) before the active clock edge, and it must stay valid at least another specified time (hold) after the active clock edge. See figure 3.9.

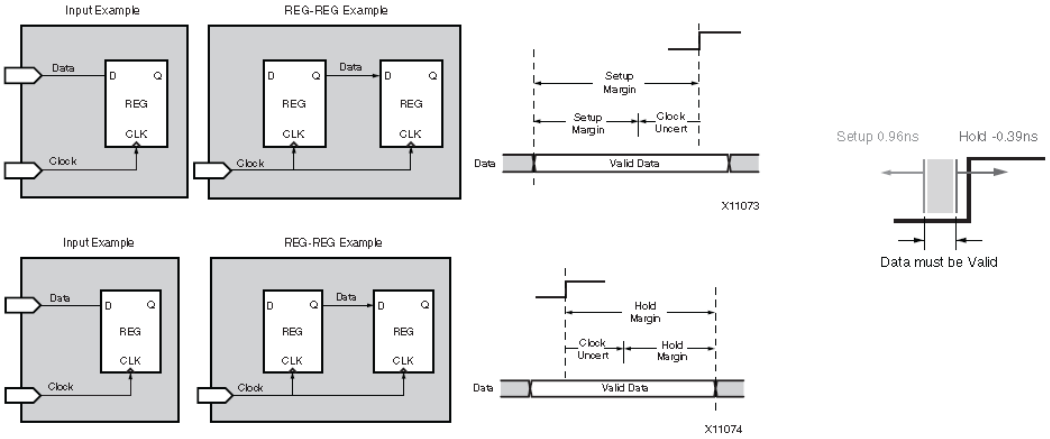

Figure 3.9: Setup and Hold - Taken from [64]

When there are Setup or Hold timing violations, a metastability condition can happen avoiding the correct functionality of the system. In FPGA design, satisfy the timing requirements is one of the most complicated part for the designer. To avoid metastability problems pipeline techniques are used to satisfy the timing requirements.

3.3.3 Pipelining

When using FPGAs, the designer must take advantage of the FPGA architecture to use the resources efficiently in order to solve a problem. An FPGA is a device that has many registers, usually when Pipeline is added to a design the main extra resource needed are registers.

The objective of pipelining is to increase the max operation frequency of a system, for this, a combinatory function is subdivided in steps and computed in multiple clock cycles. The operation frequency of an FPGA system is limited by the number of logic levels that a signal must cross. The logic elements present in an FPGA are not perfect, and they have associated impedances implying propagation delays. If a signal must be propagated through more logic elements, the delays will be more important, and the Setup and Hold times may be not respected.

Figure 3.10 shows two different ways to compute the same arithmetic operation. In the bottom implementation, register (represented with z^{-1}) were added between the adders. For

software engineers, it would be evident that the circuit on the top works faster than the circuit on the bottom. However, that is not correct when the operations have to be done at high frequency and the propagation delays are considerable, which corresponds in most of the cases to the FPGA design.

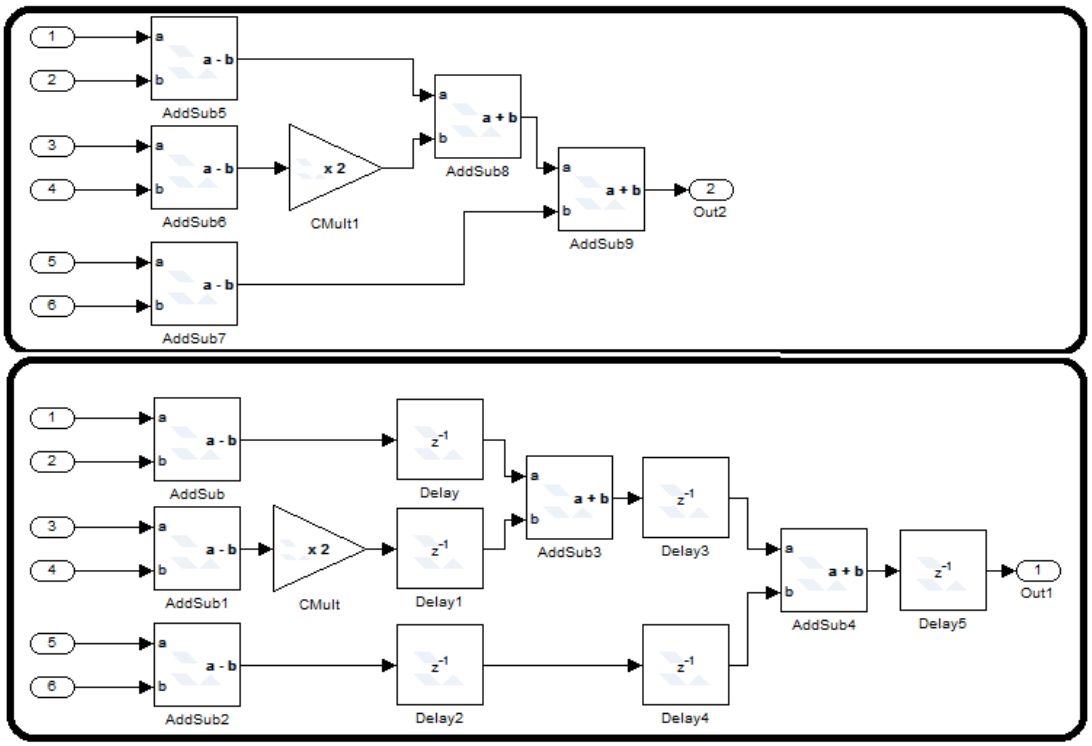

Figure 3.10: Pipelined vs. not pipelined implementation - The circuit on the bottom has an extra latency caused by the delay blocks; however this circuit can work at a higher frequency.

When adding pipeline, the designer must be careful with the data synchronization, to avoid a misalignment between the signals. In the solution using pipeline shown on the bottom of the figure 3.10, there are two z^{-1} blocks connected in cascade. Apparently, it does not affect the result, however, they have to be added for synchronization purposes.

Usually a system with pipeline stages has more latency, due to the registers added, however, after an initial latency on each clock cycle, a new result is obtained from the system. This latency is compensated with the higher operation frequency allowing better performance of the pipelined system.

3.3.4 Floating Point Operations in FPGA

Floating point operations are expensive when executed in FPGAs, however, in some algorithm they are mandatory. In an FPGA, a fixed point operation can be executed in just one clock event at high frequency and using low quantity of resources, some FPGAs even include silicon multipliers. In the case of floating point multiplications more resources are needed, because

Table 3.2: Throughput of floating point multiplications depends of the architecture

	Throughput
Only Software	2 MFlops
FPU+PLB	21 Mflops
FOU+APU	45 MFlops

there are not silicon Floating point multipliers available in the current FPGAs; so this kind of operations is implemented using the LE.

When using an embedded processor, a Floating Point Unit (FPU) can be coupled to the system to implement floating point operations. In the case of Virtex 5 FXT family, the PowerPC is coupled with the FPU using a special bus called the Fabric Co-processor Bus (FCB). The FCB is connected to the processor using the Auxiliary Processor Unit (APU) shown in the figure 3.11.

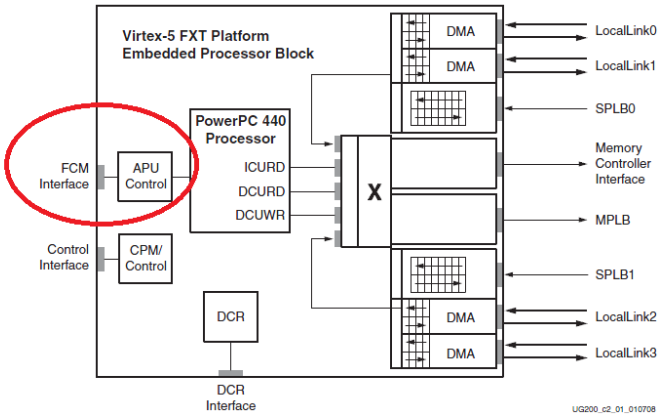

Figure 3.11: PowerPC interfaces - Taken from [65]

According to the architecture used to compute floating point operations, the performance can drastically change. In the case of Virtex 5 FXT, table 3.2 shows the achieved throughput for different architectures.

In a software only approach, just one floating point operation can be executed at a time. The execution of this operation takes many clock cyles to be completed if no pipelining is used.

When using an FPU, the operations are pipelined and a new operation is initialized on each FPU clock cycle. Inclusively, the PPC can continue to execute instructions while the FPU is computing other operations. With an FPU, more than one operation is being executed at the same time.

Floating point operations can also be implemented without any embedded processor. Different implementations of FPUs are distributed as open source codes. FPGA vendors also provide IPCORES in the form of a Netlist to synthesize floating point operations.

Figure 3.12 shows a graph comparing different implementations of floating point single precision additions using Virtex 5 FPGAs. Using this graph the designer can select the latency of the operations according to the frequency of operation and the resources needed. The implementation with the smallest latency has the worst timing performance. When the design

allows more latency, pipeline techniques are used to improve the timing performance. In the best case, a maximum performance of 410 MHz can be acheived.

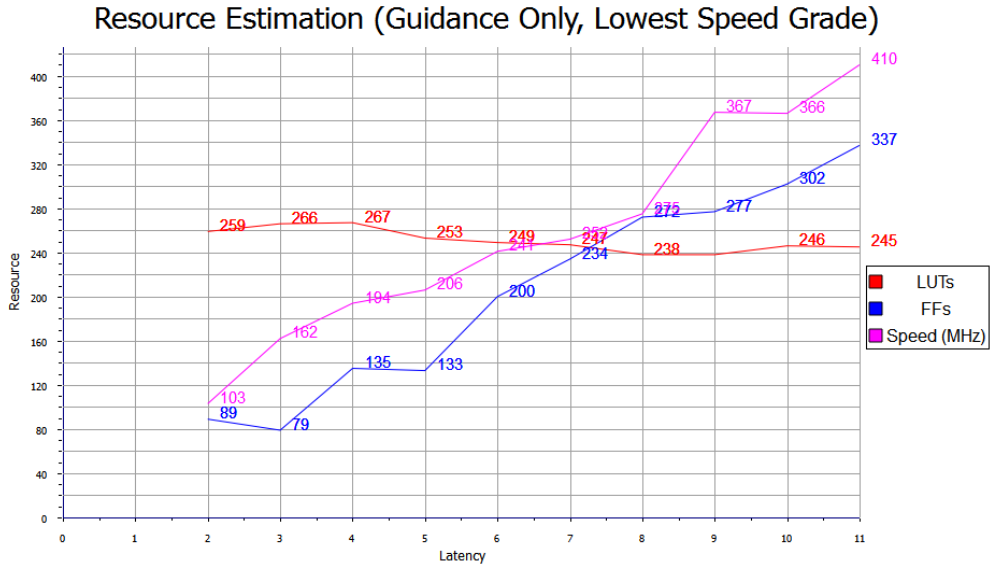

Figure 3.12: Floating point addition latency, operation frequency and resources - The graph represents latency, maximum operation frequency and resources for different configurations of the single precision floating point addition generated with the Xilinx IPCore tool.

3.3.5 FPGA Memory management

When designing with embedded systems, memory management can become a bottleneck if the correct decisions are not taken. There are many different kinds of memories with different characteristics. In this thesis we will work with four different memories: Internal FPGA RAM, SDRAM, SRAM, and Flash.

The main advantage of the internal memory of the FPGA is that many memories can be accessed in parallel, at very high speed and relatively easily from the user logic. Using the internal RAM user can build different storage architectures like FIFOs, true dual port ram, simple dual port ram, etc. Internal RAMs architecture depends of the FPGA vendor and family. In the case of Xilinx, the internal RAM is called BRAM and is organized in 4kbit blocks called M4k for the Virtex 5, and M18k blocks for Virtex 6. In the case of Altera, for example, Stratix FPGAs has different size memories into the same die (M512 and M4k).

The main limitation of the internal memories is the size. For image processing algorithms, it is very expensive to store the entire image in the internal RAM.

When a design needs extra memory, external memory can be used. The choice of the memory depends of the needed quantity, operation frequency, and type of accesses.

SDRAM is a high density and very high speed memory that is conceived to be accessed in sequential way using burst. The main disadvantage of the SDRAM memory is that if the accesses are not done sequentially, the performance is drastically affected. In current FPGA development boards we can find DDR2 and DDR3 SDRAM memories.

SRAM is built with a technology more expensive than SDRAM. Normally, the size of the SRAM available on the market is smaller than that of SDRAM. The main advantage of the SRAM memory is the possibility to be accessed in non-sequential ways without time penalization.

The Flash memory is a non-volatile memory normally used to store configuration data. It has the disadvantage that it is relatively slow, so for real-time operation, it is not desired to work directly from it. Normally, during startup of the system, the user has to copy the necessary data from the flash to the RAM memory, allowing real-time access to this information.

FPGA providers have different memory controller available to manage external memories. In this thesis, we worked with the Memory Interface Generator (MIG) and the Multi-Port Memory Controller (MPMC)[66] in the case of Xilinx[67], and with Avalon Memory Mapped controllers in the case of Altera.

3.3.5.1 Multi-Port Memory Controller (MPMC)

MPMC is a configurable memory controller that allows to connect up to 8 peripherals to an SDRAM memory. The MPMC supports different arbitration schemes and has independent read and write FIFOs for each of the eight ports.

The peripheral can be connected using different interfaces: PLBv46, VFBC, NPI, SDMA and XCL. The highest performance is obtained using the NPI (Native Port Interface), this interface defines a low level protocol to have direct access to the memory.

Another very useful interface is the VFBC (Video Frame Buffer Controller) interface. This interface works like a two-dimensional DMA and is very useful when working with images. VFBC is based on the NPI interface of the MPMC. The throughput achieved with the VFBC is smaller than with NPI. However, using VFBC facilitates the design task. The VFBC includes internal FIFOs to implement accesses to the memory. See figure 3.13.

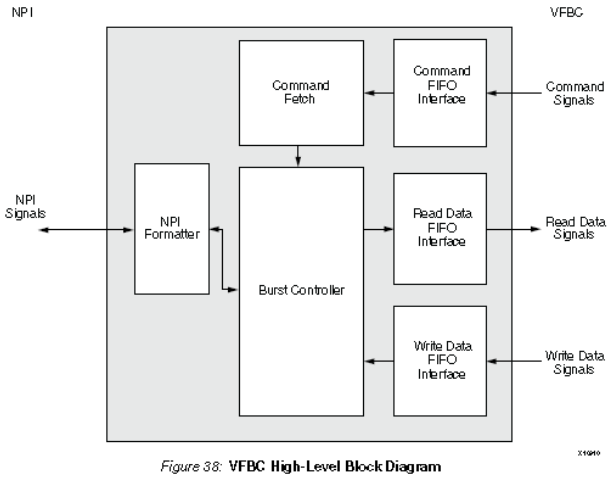

Figure 3.13: VFBC block diagram - Taken from [66].

When using the MPMC, the designer can create hardware accelerators that use the external memory using the different interfaces available in the MPMC. The accelerator must include the code to generate the correct signals specified in the MPMC specifications. See figure 3.14.

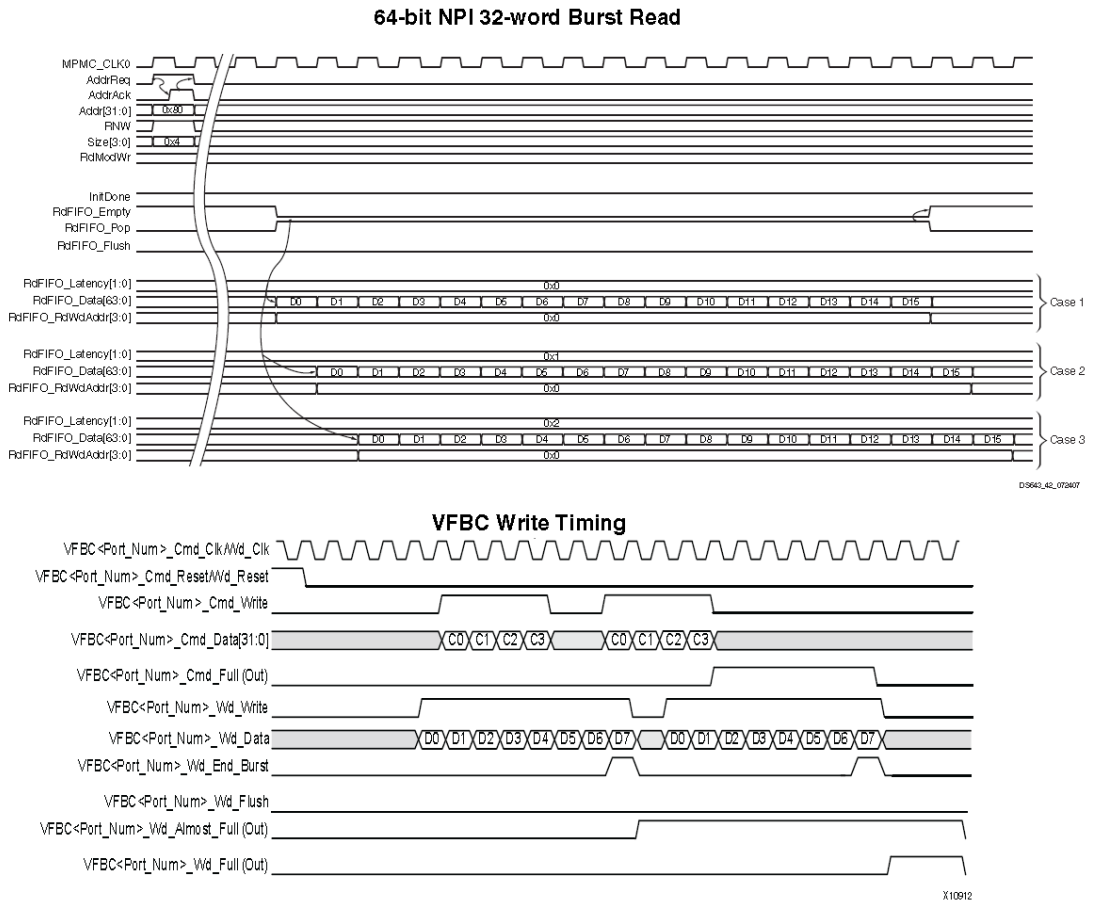

Figure 3.14: MPMC transaction timing diagrams - On the top, NPI transaction timing diagrams for a 32-word Burst Read. On the bottom, VFBC transaction timing diagrams for a VFBC write. Taken from [66].

The MPMC is intended to be employed using burst transfers. If single transfers are employed the throughput is highly degraded, because of the presence of an Initial Transaction Latency of at least 24 clock cycles. The document [68] presents a table describing the MPMC Latency and Throughput using different configurations.

3.3.5.2 Memory Interface Generator (MIG)

MIG [69] from Xilinx facilitates the generation of interface for different memories. MIG supports different SDRAM memories like DDR2, DDR3 as well as some SRAM memories like the QDR2. MIG is used in this thesis to generate the pin out and area constraints for a custom board developed in this thesis (described in section 5.9.1).

MIG also generates the timing and area constraint necessary to satisfy the high speed constraints of some memories. In the case of DDR and QDR memories, some buses must be driven using the same banks of the FPGA to avoid different propagation delays. For example, if a bit of an address of a QDR2 memory is delayed, it can result in an unexpected operation. MIG helps to guarantee this constraint just at the FPGA level.

MIG generates HDL controllers that can be integrated in a design; an example design is created by the tool in order to guide the user. This example design includes a simulation project, which can be used with a HDL memory model to test the operation of the system.

3.4 FPGA embedded Processors

Embedded processors are present in almost the 66% of the new embedded systems. In modern FPGA applications, it is common to find an embedded processor connected to the logic. In the last years the tendency to use processors simultaneously with the FPGAs has increased. This field of research is called Co-Design. When designing with FPGAs, using a processor can simplify a lot the work of the designer. For example, the design of state machines using an embedded processor is easier than creating them just using the FPGA logic.

Some non-critical tasks, like slow speed peripheral management can be designed easier using a processor. This way the FPGA designers can concentrate their efforts to accelerate critical tasks. Designing with a processor is easier than designing just using LE of the FPGA because of the considerations related to timing and resources when designing with hardware.

In 2002, Xilinx integrated a silicon PowerPC and today the two FPGA leaders (Xilinx and Altera) offer different architectures using embedded processors. In the 2012, the FPGA vendors began converging to a similar architecture based in ARM processors.

Nowadays, we can find two types of processors: soft processors and hard processors. Soft processors are entirely implemented using the logic primitives of the FPGA (Slices, BRAM, etc.), and is usually described using a Hardware Description Language. Hard processors are physical processor made in silicon and tightly connected to the rest of the logic resources of the FPGA.

Altera offers the Nios II soft-processor. Xilinx offers PicoBlaze and MicroBlaze soft-processors. Xilinx also provides hard core processors including the PowerPC440 and the PowerPC405.

In an FPGA design, the user can instantiate more than one processor in the same design, some Xilinx FPGAs even offer dual core PowerPC or dual core ARM processors, and if the designer needs more, he has the option to connect one or more extra soft processors to the system. In theory, the limit in number of soft-processors to be embedded in an FPGA depends of the size of the FPGA.

3.4.1 MicroBlaze and PicoBlaze

Xilinx has two soft processors : PicoBlaze and MicroBlaze.

PicoBlaze is a 8-bit processor optimized for Xilinx FPGAs with Reduced Instruction Set Computer (RISC) architecture. PicoBlaze is distributed as a VHDL module and has the advantage of being a royalty free processor.

The MicroBlaze has Harvard Memory architecture. See figure 3.15.

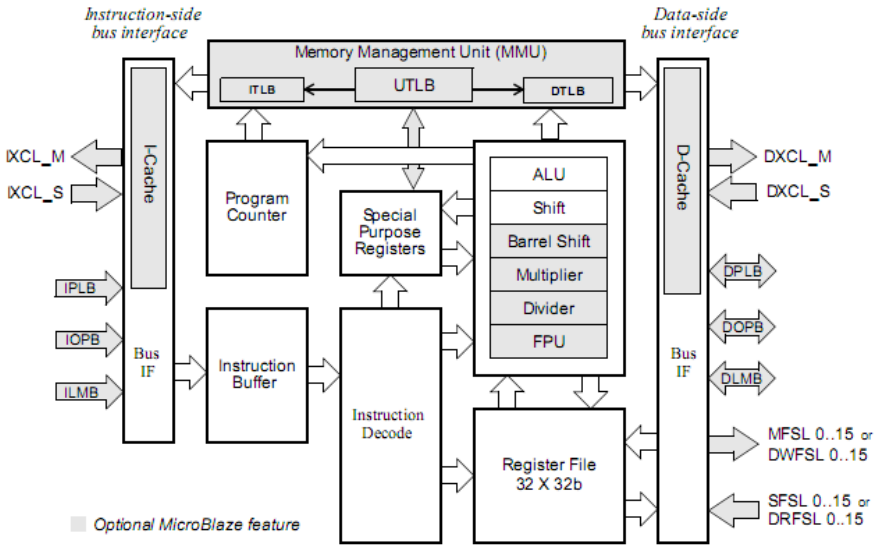

Figure 3.15: MicroBlaze block diagram - Taken from [70].

MicroBlaze has a specialized bus in order to connect co-processors called Fast Simplex Link (FSL). This link is a unidirectional point to point streaming interface based in FIFOs. MicroBlaze processor can be configured to have up to 16 FSL: 8 inputs and 8 outputs. The FSL is used from the point of view of the MicroBlaze programmer with C-macros.

3.4.2 PowerPC

The PowerPC (PPC) 405 and PPC 440 are hard processors available in the Xilinx FX family, they are physically built in silicon. These processors run up to 450MHz and 550 MHz respectively. PowerPC has a Harvard architecture with separate buses for instructions and data. See figure 3.16.

An important component of the PowerPC is the Crossbar. The Crossbar is a hard block and constitutes the main connection between the peripherals and the processor. It works like a central arbitration switch. The Crossbar provides three PLB connections.

A co-processor can be connected to the PowerPC using a bus called the Auxiliary Processing (APU), which allows for the hardware acceleration of software bottlenecks. One of the most common co-processors used today is the Floating Point Unit (FPU).

PowerPC includes hard Cache memories (instruction and data) built at one side of the processor.

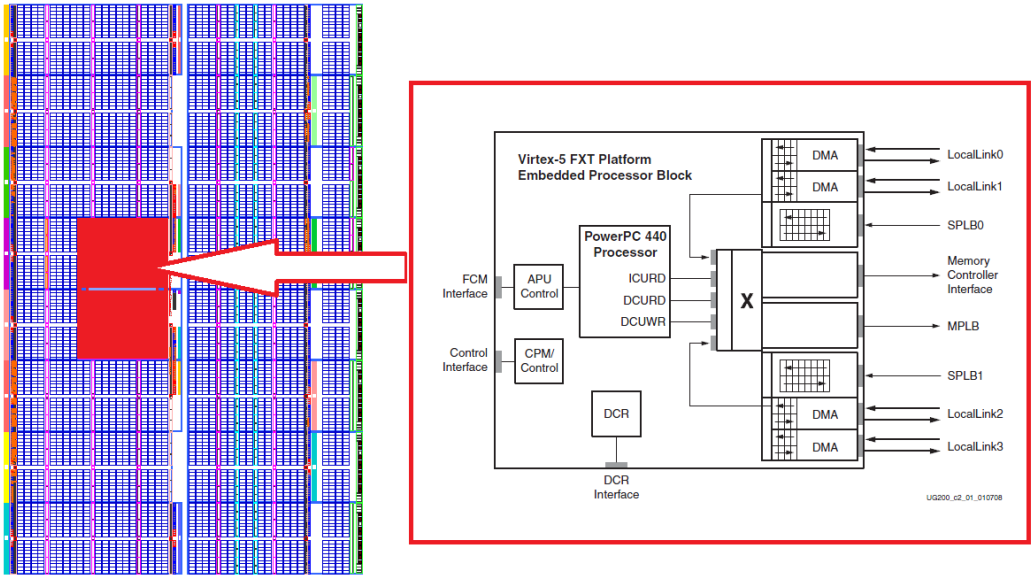

Figure 3.16: PowerPC in the die of a VFX70t - The Processor is in the middle of the die, tightly coupled with all the hardware resources. Taken from [65].

3.4.3 NIOS II

Nios II is a 32 bits soft processor from Altera with RISC architecture. See figure 3.17. In this processor, instruction and data buses are separated (Harvard Architecture). The processor is connected to the logic using the Avalon Switch Fabric Bus.

According to the needs of the application, NIOS II is available in three different configurations: (fast) : maximum performance (balance) : between performance and cost. (Smallest) : intended to be used in low-cost FPGAs.

3.4.4 Real Time Operating System (RTOS)

In some applications using processors, more than one thread must be executed. An operating systems allows the execution of multiple threads using a time multiplexation, allowing to share the resources between the threads.

An OS manages the resources of a processor, constituting the interface between the executed threads and the hardware resources. There are two types of operating systems: General Purpose Operating Systems (GPOS) and Real Time Operating System (RTOS). The main difference between the RTOS and the GPOS is the way how the OS schedules all the events.

A RTOS guarantees deterministic and short response for events. In the RTOS, the critical threads are executed in priority and only a highest priority thread can interrupt the execution of other threads. This priority scheme of thread executions allows meeting timing requirements of critical tasks.

A GPOS does not allow predictable response time, avoiding the execution on time of some critical tasks. When using a GPOS, user does not have precisely control of the software execution order. In the version 2.6 of Linux a kind of pre-emptive has been added, allowing tasks to

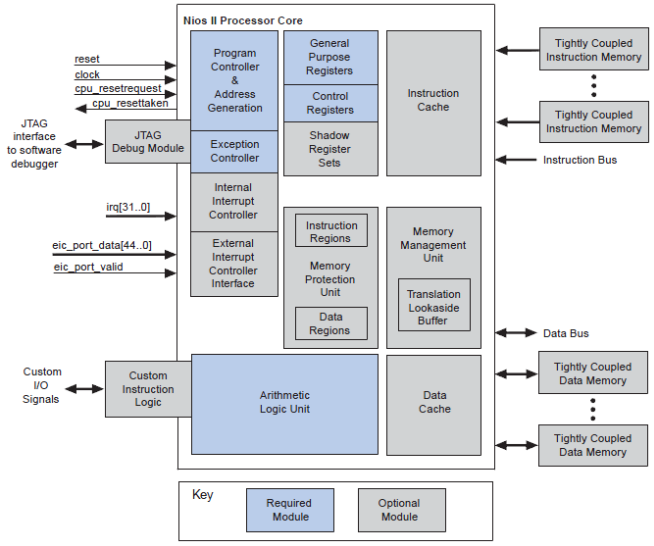

Figure 3.17: NIOS II block diagram - Taken from [71].

be interrupted according to their priority.

3.5 FPGA Bus architectures

A Bus is an infrastructure that supports transfer of data between masters and slaves devices. In a bus architecture, the components connected to the bus are memory mapped. To connect the processor to other hardware resources available in the system some standardized buses and interfaces exist. These interfaces specify the communication protocol between the processor and the logic.

Figure 3.18 shows a typical system including an Interconnection Bus. In the figure, we can see many masters and slaves. For example, Master 1 can be a Processor and Master 2 a DMA controller, and both have access to Slave 1 which can be a memory. In this scenario, someone must arbitrate the bus and give the DMA access to the bus when the processor is not using it. This means that some kind of signalization must be defined to establish the communication without conflicts. This signalization is usually defined like a protocol that defines the interface.

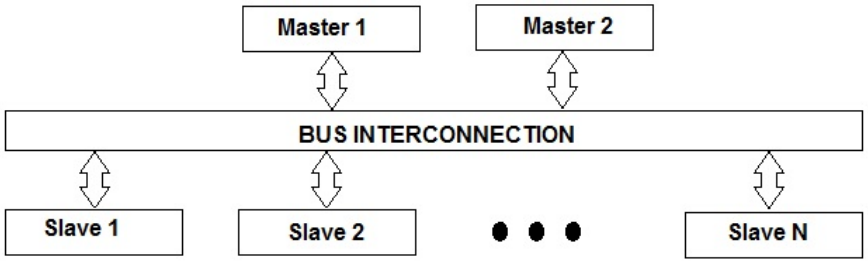

Figure 3.18: Interconnection Bus -

For Altera a central bus called Avalon Switch Fabric is used to connect the processor to the rest of the logic. Xilinx use a similar Bus called Processor Local Bus (PLB).

In the last FPGAs available since 2012, the main FPGA vendors are converging to use ARM processors with the same bus architecture AXI4. Thanks to this, the portability of designs between platforms of different vendors is becoming easier.

3.5.1 Avalon Switch Fabric

The Avalon Switch Fabric Bus [72] defined by Altera, allows the use of the Nios II 32 bits embedded processor. This bus allows just one master to access the bus at the same time.

In the Avalon specification are defined seven types of Avalon Interfaces: Avalon Memory Mapped, Avalon streaming, Avalon Conduit, Avalon Tri-State Conduit Interface, Avalon Interrupt interface, Avalon Clock Interface and Avalon Reset Interface.

When designing an Avalon peripheral, the designer must respect some protocols and signal name conventions. A peripheral can include more than one interface. The Avalon interface facilitates the migration and portability of IPcores between different projects.

Figure 3.19 shows an Altera Architecture using the Avalon Switch Fabric Bus to implement a homography.

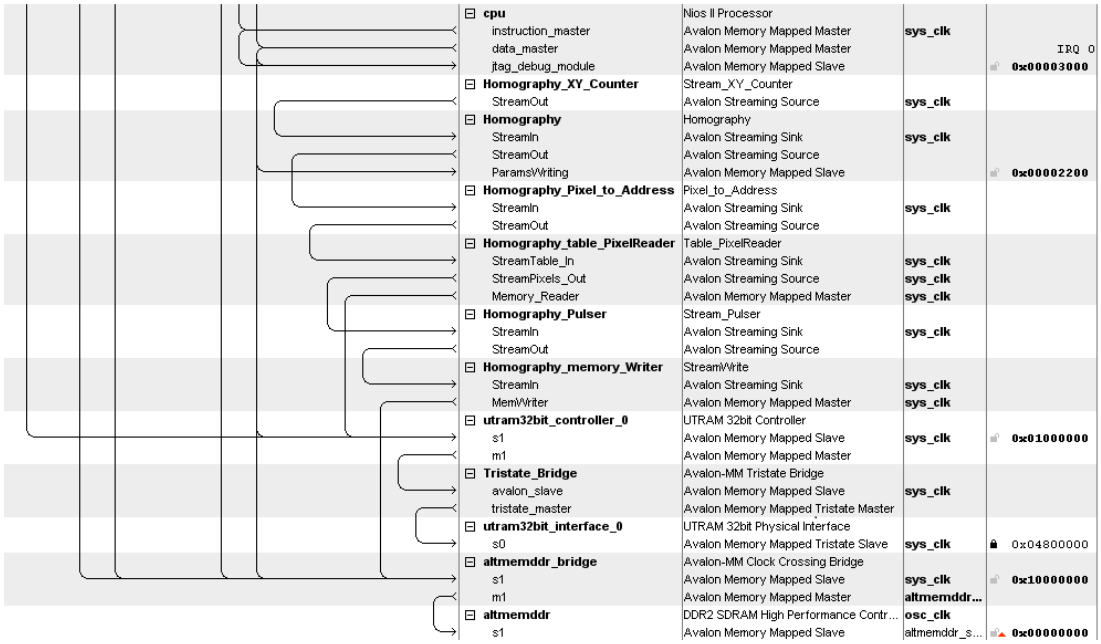

Figure 3.19: Avalon Switch Fabric - The diagram was generated with SOPC builder of Altera, this system implements a Homography of an image. The NIOS2 processor configures the Homography parameters through an Avalon Memory Mapped Interface. In this system, there are two memory components, a DDR2 memory and a SRAM memory, both of them are Avalon Memory Slaves.

3.5.2 Processor Local Bus (PLB)

PLB is a fully synchronous Bus designed by IBM. Xilinx PLB consists of a central bus arbiter and the necessary bus control and gating logic. PLB support up to 16 masters and any number of slave devices. When there is just one master any arbitration is needed. PLB defines three types of interfaces: PLB Master Interface, PLB Slave Interface and PLB Arbiter Interface.

Figure 3.20 shows a typical Xilinx Architecture using the PLB bus. In a system more than one PLB bus can exist; even in a mono-processor system.

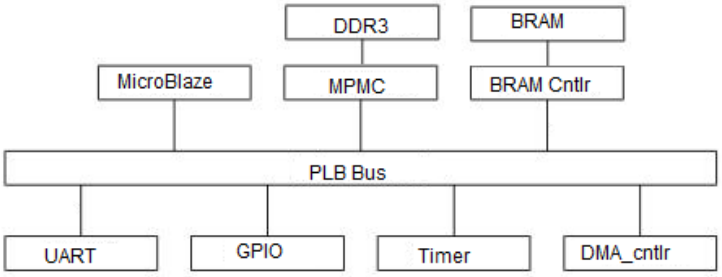

Figure 3.20: PLB Bus example - In this architecture, there are two masters: a MicroBlaze Processor and a DMA controller.

3.5.3 AMBA-AXI4

Advanced Microcontroller Bus Architecture (AMBA)- Advanced eXtensible Interface (AXI) defines a protocol for high performance bus interconnection.

This protocol uses a handshake mechanism based in two-way VALID-READY signals; this way both master and slaves can control data rate of the communication. AXI is a burst-based protocol, in which the first transaction an address is transferred. Then, the slave continues to compute the address of the other transfers in the burst.

3.6 FPGA Development Flow

There are two basic development flows:

- Description based in High Level Synthesis (HLS) tools.
- Description of the system using HDL languages

3.6.1 HLS Flow

In [73] a complete review of different HLS tools is presented, according with them, now we are in a transition point thanks to the progress of the last generation HLS tools, combined with the huge density of the last FPGAs and complexity of the new applications.

According with [74], a big design of 1-M gate elements require 300k lines of RTL code, resulting in an increase of 7x to 10x of design complexity compared with a HLS implementation. They present an analogy between the new HLS tools against the Hand coded RTL design flow

and the evolution of assembler language for software design to higher level languages like C and C++.

One big advantage of the HLS tools is that this flow facilitates the software/hardware co-design, because of the same programming language used to describe the hardware and to program the embedded processor. High level languages are more compact, faster to write and they offer better maintainability and readability than HDL, for this reason, in future designs due to the rapid increase of complexity, HLS tools will play a very important role.

Using HLS tools the behavior of the system is described using a high level language like C, C++, SystemC, etc., combined with compiler directives to specify concurrency, timing, pipelining, and other constraints.

Recently, Xilinx is using a new cockpit called Vivado [75] based in HLS. In their approach, first, a functional simulation of the system is done, where the designer creates a C test-bench that describes its behavior. The system behavior is verified using GNU compiler of Visual C++ simulator. Next, the RTL is automatically generated. Finally, a SystemC architecture level simulation is implemented. This way, the design implementation and verification is accelerated. Using Vivado, an algorithm that was simulated in 2 days can be simulated in 10 seconds.

3.6.2 Hand code RTL design Flow

In this thesis, we did not use HLS tools because the projects involved in the development of this work imposed the constraint of using VHDL for certification purposes.

Figure 3.21, shows the development flow followed in this thesis.

The first step on the development flow followed in this thesis is the identification and definition of the system requirements. From the system requirements, a software implementation is done to validate the algorithms and hypothesis proposed. The software implementation is done using a General Purpose Processor and using available libraries that could facilitate the software development. The software implementation is done using High Level Languages like SystemC, Matlab, C or C++.

After the validation of the algorithms in a Software only approach, the decision about use of an only software, only hardware or co-design solution must be taken. An initial decision about the necessity to use a processor and the kind of processor must be done. Critical tasks that using only a software approach does not satisfy the requirements can be accelerated or off-loaded using the hardware resources of the FPGA. Software design is easier than hardware design, which means that if a processor is available as much as possible of the design should be implemented in software. Using a processor allows for reducing the development time.

At this point the architecture is conceived, the different hardware and software tasks are identified. Block diagrams are generated to identify the different modules of the system. A flat architecture can be used to implement the design; however, a hierarchical approach can facilitate the designer task.

In a hierarchical design, a complex module is sub-divided into small modules, where each one is specialized to implement a part of the design. The main advantage of the hierarchical design is that it facilitates the readability and the debugging.

At this point, it is convenient to design temporal diagrams of the main signals, which will then be useful for the verification process. The interfaces between the different modules as well as the pipeline synchronization must be clearly defined. In this step of the design, the target FPGA must be selected in order to take advantage of the available resources and choose the size of the FPGA needed.

It is common to over dimensionate a little bit the size of the chosen FPGA to anticipate a future upgrade of the design and to facilitate the debugging and placement tasks. It will be

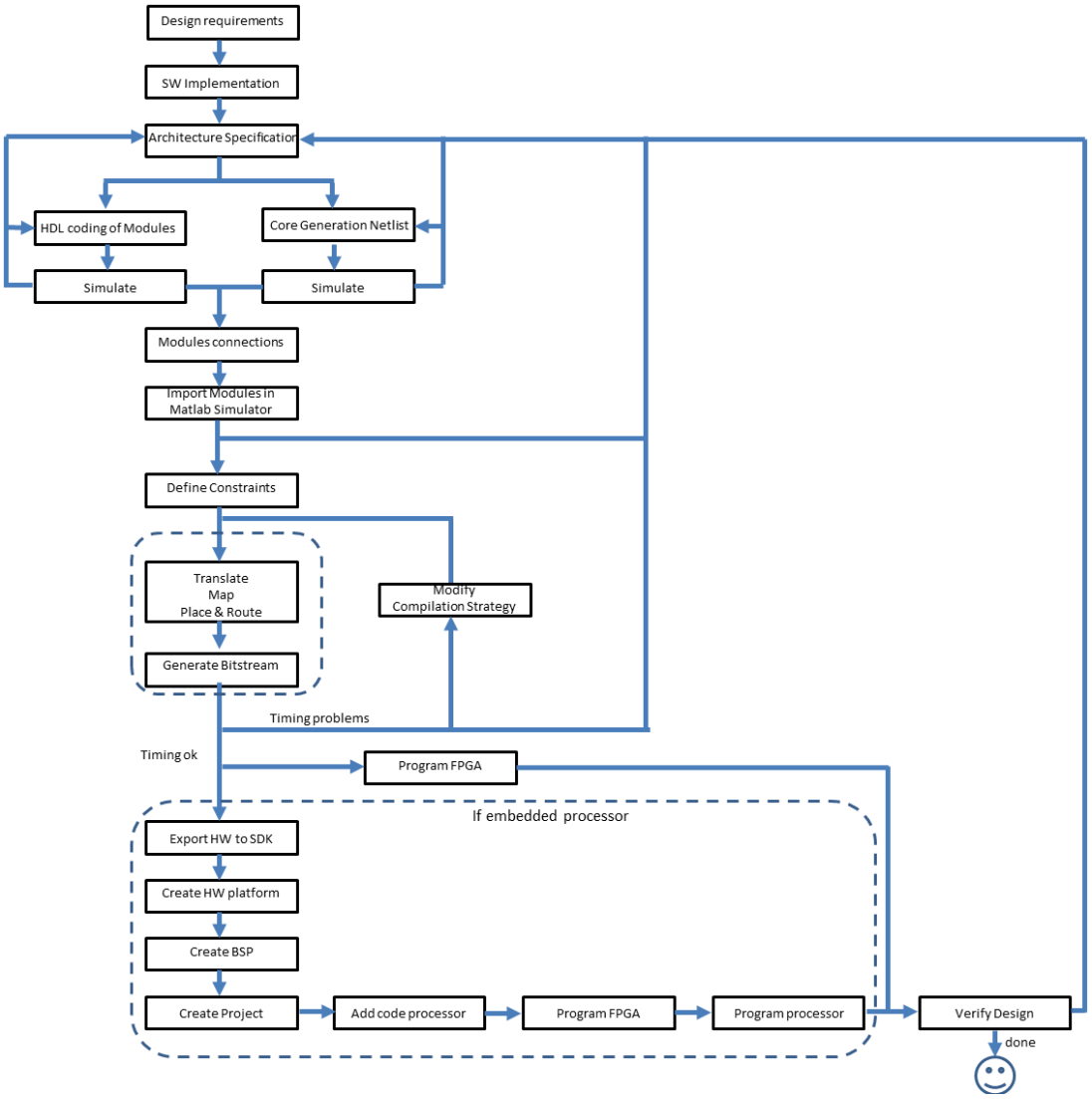

Figure 3.21: Development flow followed in this thesis -

never possible to use the 100 % of the available resources of the FPGA. For debugging purposes, will be convenient to keep enough free BRAM memories, because when using an FPGA scope analyzer [76] [77], the samples are stored in the FPGA internal memory.

Once the architecture is defined, the modules must be coded or generated if they correspond to IPCores. The modules are coded using a HDL language; keeping in mind that special attention must be taken to only use synthesizable statements. If special hardware resources (FIFOs, buffers, Gigabit transceivers,...) are used, they can be generated using an IPCore generator provided by the FPGA vendor. Some of the special resources of the FPGA can also be inferred from the HDL code if properly coded. Some hardware resources can just be instantiated directly from the FPGA wizard tool. If the modules are provided by a third-party, a Netlist or a HDL source code must be added to the design. In this step of the design, memory controller's modules are generated using MIG tool if necessary.

After coding and generating the architecture, the RTL viewer can be used to generate a schematic of the synthesized design. This schematic can be compared with the block diagrams implemented during the development of the specification of the system. Using the schematic, the designer can verify if the synthesizer inferred the correct logic according to the design requirements. Also, erroneously optimized nets, unwanted latches inferred from incomplete statements, and other bugs can be identified.

Then, all the modules must be checked using a low level simulation. In this step a functional simulation is implemented. This first simulation does not take into account the delays present in the different logic resources. Test-benches are generated using HDL code that generates the different stimulus to the blocks. For test-bench generation, is common to use non-synthesizable statements like WAIT, in the case of VHDL, to facilitate the test-bench generation.

If the simulation behavior does not agree with the conceived specification, the error must be identified. When a problem is found in the simulation, the designer must identify if the error comes from the Device Under Test (DUT) or from the test-bench. If the error comes from the DUT, the designer must verify if it is a HDL writing error or if it comes from the conceived architecture. If the error comes from the architecture, the designer should check architecture specification.

Once all the modules were tested independently, the designer proceeds to connect the different modules together. At this moment, it is convenient to implement a complete simulation of the design and correct possible bugs or modify parts of the architecture.

Then, the modules can be imported to System Generator to implement a higher level simulation with a better coverage. Thanks to the high level functionalities of Matlab (such as image reading/writing, graph plotting, random number generation, etc) the verification process is easier. Moreover System Generator allows to instantiate additional Xilinx IPCores. In System Generator, the HDL modules are imported as Black Boxes and can be compared with a Simulink model, which allows for estimating the operator precision loss inherent to data typing and dimensioning. Once again, if the system has bugs, the designer must identify if the bugs come from the conceived architecture, which would imply redefining some modules.

After simulating all modules in System Generator, the design can be connected with an EDK system. In the EDK system, the module can be connected to a PLBv4.6 or AXI system.

One very important design entry in FPGA constitutes the generation of constraints of the system. In this step, the designer must add the necessary constraints (clock periods, pin out, pin standards, offsets, multi-cycle paths...). Special attention must be made to not over constrain the design which could complicate the implementation task.

The implementation task, is composed of three steps: Translate, Map and "Place Y Route". Understanding these phases allows the designer to efficiently optimize the design. Each one of the phases gives important information that can be used to do the closure of timing require-

ments.

The first Phase of the implementation process is the Translate step. Here, the generated Netlists from the synthesis are merged into a single Netlist. The second phase corresponds to the Map of the design. In this phase, the logical symbols from the Netlist are mapped to physical components (slices, DSP48, etc). In the last phase, Place and Route, components are placed onto the chip.

After implemented the design, the timing reports must be analyzed to verify if the placed architecture satisfies the timing constraints of the system; if constraints are not satisfied, a timing closure strategy must be followed.

In the case of Xilinx, different techniques are proposed to achieve the timing closure [64]. See figure 3.22. The timing response in an FPGA is impacted by many factors (architecture of the system, coding style, compilation strategy,...). Satisfying the timing requirements can be the most difficult part of the FPGA development flow. After each phase of the FPGA implementation flow, a timing report is generated. These reports give information about the critical nets that do not satisfy the timing constraints. The nets that do not satisfy the constraints will have a negative slack. The slack of a net depends on the clock requirement, the data path (depends of the levels of logic), the clock path skew and the uncertainty (mainly depends of the clock jitter).

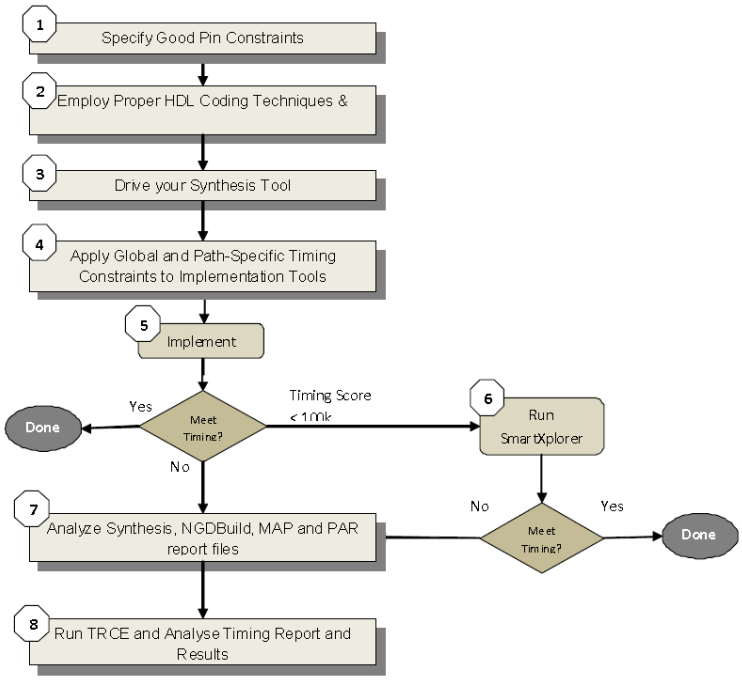

Figure 3.22: Timing closure flowchart - Taken from [78]

Using a software called PlanAhead of Xilinx, these Nets can be physically identified in the FPGA. This way, area constraints or timing constraints can be included or special optimization can be done to a specific net.

If the timing closure was possible, the design can be used either to program the FPGA or to begin developing the Software for the embedded processor, if used. For FPGA programming,

a Bitstream containing the programming information is generated. When using an embedded processor the design is imported to the SDK. In the SDK, a file containing the memory map of all the peripherals present in the bus is generated.

To begin with the procesor software development, the designer must identify if an RTOS (Xilkernel, MicroC/OS II, Embedded Linux, RTK,...) is going to be used or if special libraries need to be used to implement some tasks (for example, when using a TEMAC, the Nichestack [79] or the Lightweight (lwIP) [80] stacks provided by third parties can be included). Next, a Board Support Package (BSP) containing all the drivers and hardware configuration is created. Test programs can be created automatically by the SDK facilitating the software development.

Then, the software for the processor is generated and debugged. If Bugs are present, the problem can come again from the conceived architecture, if it is the case the architecture must be rechecked. After corrected all the bugs, a file containing the program information is generated (Executable Loader File-ELF).

The designer can include profiling tools, which help to identify parts of the code that must be optimized. Optimization can be done designing hardware accelerators to offload parts of the algorithm from the processor to the FPGA. To implement the profiling, typically a timer must be added to the system. The timer is used to count how long time the processor is executing each one of the functions. Profiling is an invasive technique but allows to have a very precise idea of the software bottlenecks.

The FPGA hardware programming information (Bitstream) and the Processor software (ELF) can be merged in a single file that can be used to program the FPGA from a compact flash for a standalone operation. This file contains all the programming information necessary to configure the FPGA, this file is loaded from a non-volatile memory to perform the configuration of the FPGA.

One very powerful tool used during the verification of the FPGA designs is the scope viewer, which enables the designer to view the internal signals of the system. The main advantage of this tool is that the real system is tested as if the designer would dispose of a physical digital analyzer. Many options to trigger events can be used to facilitate the debug.

The FPGA scopes store samples in internal FPGA memories. For debugging purposes, it is convenient to dispose of enough unused internal ram memory to visualize multiple signals if necessary. Once the design is validated, the final implementation can be migrated to smaller device because the resources needed for the scope wouldn't be necessary anymore.

3.6.3 System simulation

The simulation constitutes a very important stage in system design. For FPGAs, the simulation is usually done with a third-party software called ModelSim from Menthor Graphics.

Xilinx and Altera integrate a simulator in their development environment. Use the FPGA provider simulator instead than ModelSim facilitates the designer tasks, because all the libraries are already linked with the FPGA components.

For simulation, a test-bench is normally coded in HDL to generate the stimulus of the Device Under Test (DUT). One of the more complicated parts in verification is having good coverage. Coverage is a measure related to the percentage of the different possible scenarios that have been tested, resulting in a measure of the verification quality.

Simulation using models is a very useful tool of system design. Memory providers offer behavioral models of their components; usually, they are provided like a HDL module or a Netlist. It can be useful for a premature phase of conception of algorithms for a custom board. For example, in the context of this thesis a board containing QDR2 memories was conceived.

Using the memory model, the algorithms using the QDR2 memory were tested before the board fabrication.

When designing a custom PLB peripheral, a Bus Functional Model (BFM) simulation can be useful. This model allows for the simulation of the PLB bus, which is very useful to early detect errors. The main advantage of this simulation is that is faster than post-synthesis or timing simulation. The BFM is a tool created by Xilinx to help designer's tasks.

Another kind of simulation which includes the Hardware in the Loop is also available. The Co-simulation is done using the true hardware to simulate the algorithms. When doing Co-simulation, the runtime is drastically reduced allowing the implementation of more exhaustive test-benches.

For simulation, the VHDL package STD LOGIC TEXTIO can be very useful. This package allows reading and writing files. This way the user can read an input file to feed the design with stimulus contained in a file. To verify the correct operation of the design, the outputs can be written in another file, which can be verified using a script.

3.7 Conclusion

In this chapter we have presented different processing platforms used in embedded systems. We presented a description of the internal architecture of an FPGA and the different development flows used to design with this complex components. We presented different tools and techniques that will be used to develop the different algorithms used for the design of ADAS.

In the next chapter, we present a description of the architectures developped in this thesis for ADAS applications.

4

Hardware Integration

4.1 Introduction

In this chapter, we will describe the different architectures designed during this thesis. These architectures will be used in FPGA's of different vendors (Altera and Xilinx) in three different projects that will be described in the next chapter. Figure 1.3 presented in the introduction of this thesis, shows a block diagram of the different architectures described in this chapter.

The first section of this chapter describes the implementation of the image pre-processing algorithms: Distortion Correction and the Histogram Equalization. Second section presents the Perspective transformation. Different architectures to implement homographies are evaluated to optimize the performance. Third section describes the implementation of the fusion of multispectral sources. Fourth section presents the implementation of obstacle detection algorithms using two different algorithms: IPM and Textures. Fifth section presents the integration of the distortion correction algorithm to implement Stereo Vision. The last section describes some accelerators implemented to improve the execution of the Embedded SLAM: Point detector, Floating point matrix algorithm optimizations and Active Search using different techniques.

4.2 Image preprocessing

When working with images some preprocessing algorithms must be performed to improve the results. For the development of the components of this thesis we developed distortion correction and histogram equalization.

Distortion correction is an algorithm mandatory for almost all the image processing algorithms using cameras with long Field Of View (FOV), which have a lot of distortion. In the case of this thesis, undistorted images are necessary for stereo vision, multispectral vision, SLAM and the obstacle detection using IPM.

Histogram Equalization is applied to infrared images in order to reduce high dynamic range of the infrared camera used, without losing important information and improving the signal to noise ratio without over amplifying the noise. Reducing the dynamic range allows us to work with 8-bits instead of 16-bits images saving hardware resources in the implementation of the different algorithms.

4.2.1 Distortion Correction implementation

The goal of this module is to execute the distortion correction on high resolution images at 100 FPS. As shown in the figure 4.1, the pixel corresponding to the image coordinate (X_d, Y_d) is replaced by the interpolation of the four vicinity pixels around the coordinate (X_u, Y_u). An interpolation is necessary because the coordinates xu,yu may not be an integer coordinate.

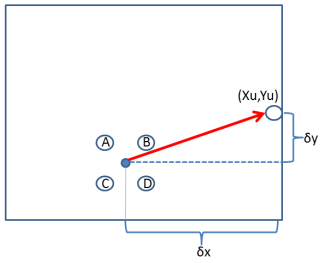

Figure 4.1: Distortion correction pixel interpolation - Pixel in coordinates X_u,Y_u) is computed using the interpolation of four vecinity pixels (A, B, C and D) indexed using the information encoded in δ_x and δ_y.

The interpolations are computed using the equation 2.8. We decided to store the fractional parts ($Frac_x$ and $Frac_y$) of the coefficients, instead of the interpolation weights (S, T, U and V) to save memory resources and to have better precision. Based on the fractional parts, the interpolation weights are computed on the fly using equations 2.4, 2.5, 2.6 and 2.7.

To implement distortion correction, calibration images are acquired with the target camera and processed offline with a Matlab toolbox, developed by the California Institute of Technology. This toolbox generates the intrinsic and extrinsic parameters to correct the distortion, and align cameras if stereo calibration is required. From the camera calibration parameters, a table of distortion coefficients is generated. The table encodes the differences (δ_x and δ_y) between the distorted and undistorted coordinates for each pixel of the image.

The values δ_x and δ_y can be non-integer numbers. To represent these values, we use a fixed point representation to save resources. To represent fractional and integer parts of δ_x and δ_y 32-bits were used as shown in the figure 4.2. This memory width size is enough to encode the distortion coefficients with the camera used, and correspond to a standard memory width.

Sign_x	Int_x	Frac_x	Sign_y	Int_y	Frac_y
31	30:23	22:16	15	14:7	6:0

Figure 4.2: Coefficient codification - The coefficient for each pixel encodes the integer displacements int_x and int_y to remap the pixel, and the fractional parts $frac_x$ and $frac_y$ to implement an interpolation

Figure 4.3 shows the architecture used to implement the distortion correction. The distortion correction is implemented using pixels indexed with the information of the table of coefficients, thus a buffer containing some lines of the images is necessary when receiving the video stream. The buffer image memory must support random accesses at high frequencies.

Figure 4.4 represents the distortion correction procedure implemented in this module. A predefined number of lines N around the pixel to be undistorted are temporarily stored in a memory. The memory is managed like a Circular Buffer to minimize memory size requirements.

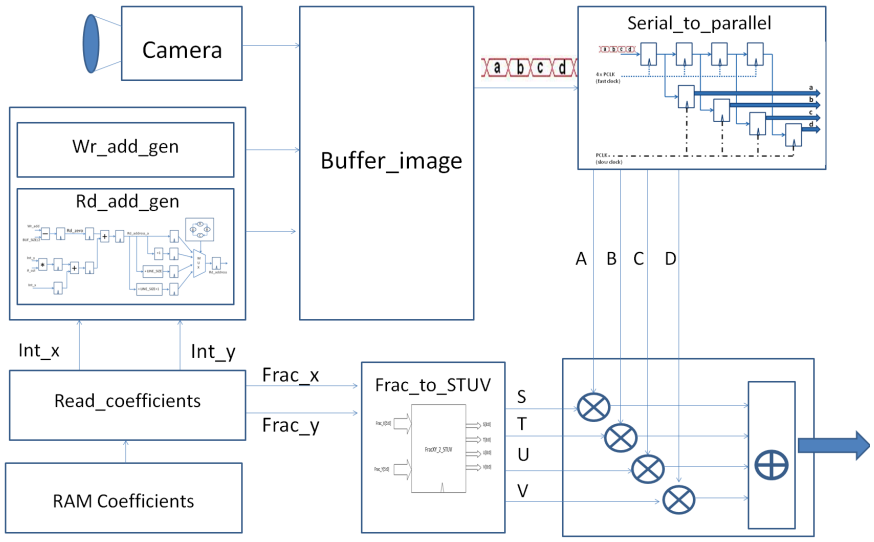

Figure 4.3: Distortion Correction architecture - This module uses two memories, one is used to store some lines of the image stream (buffer image) and the other one is used to read the coefficients (RAM COEF).

The size of the Circular Buffer imposes a constraint over the max distortion that can be corrected by the module. With a buffer size of N lines, the max distortion supported is $\pm N/2$ lines on the Y-axis, while there is no limit on the X-axis.

To undistort each pixel, a group of four pixels is used to interpolate each output pixel. The module "Rd_add_gen" generates the addresses necessary to read the four vicinity pixels needed to interpolate each undistorted pixel. The addresses are generated using the integer part of the coefficients (Int_x and Int_y). Basically, this module computes the signal "read_zero", which corresponds to the address of the pixel to be read from the circular buffer if there was not distortion ($\delta_x = 0$ and $\delta_y = 0$). Then, the read addresses are generated using the integer parts of the coefficient (Int_x, Int_y), the write address (Wr_add) in the buffer image, the line size and the buffer size. See figure 4.5.

To improve the timing performance of the "Rd_add_gen" module, it was fully pipelined as shown in figure 4.6.

The four read pixels (A, B, C and D) are multiplied with S,T,U and V respectively, the products then are added and truncated to determine the value of the unknown pixel. In this module all data is managed using fixed point data representation.

The vicinity pixels (A, B, C and D) are read at 4 x pclk frequency, then they are interpolated at pclk frequency. Simultaneously, the undistorted pixels are stored in the buffer image. The module shown in figure 4.7 manages the clock crossing. No special circuit is needed to do a cross clock domain because both the fast and the slow clocks were generated with the same PLL (they are in phase). Care must be taken with the synchronization of the pipeline.

The Distortion Correction table is stored in a non-volatile memory and loaded to a SDRAM memory at boot time. This allows us to simultaneously achieve real-time and standalone operation. The non-volatile memory is a relative slow memory, however it is necessary to avoid transferring the coefficients from a host at boot time.

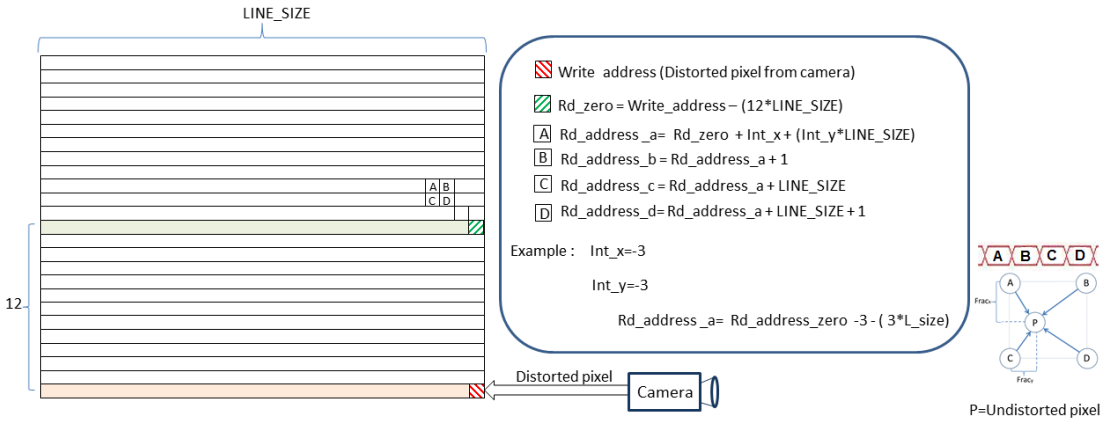

Figure 4.4: Internal Memory Management - When a new pixel arrives from the camera, it is stored in the memory. At the same time, an undistorted pixel will be generated using the interpolation of the vicinity pixels (A, B, C and D). This figure represents an illustrative example using a small buffer size of 24 lines. The final implementation has a buffer size of 256 lines to support a max distortion of ±128 lines.

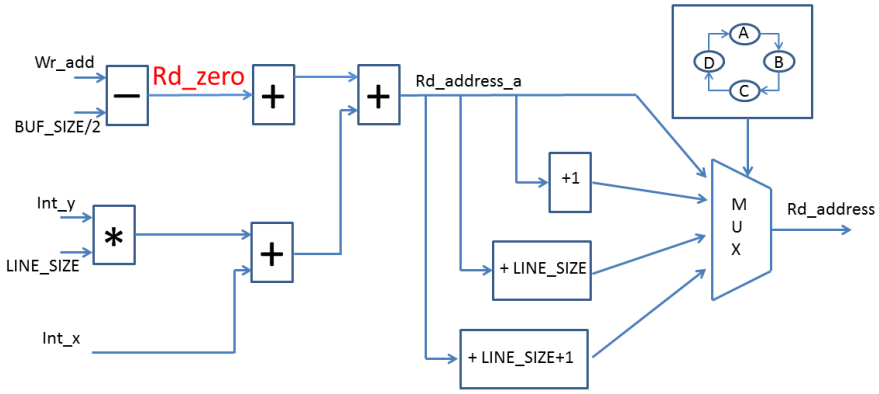

Figure 4.5: Read Address generator - The read addresses are generated using the write address (Wr_{add}), the integer parts of the coefficient (Int_x and Int_y), the line size and the buffer size.

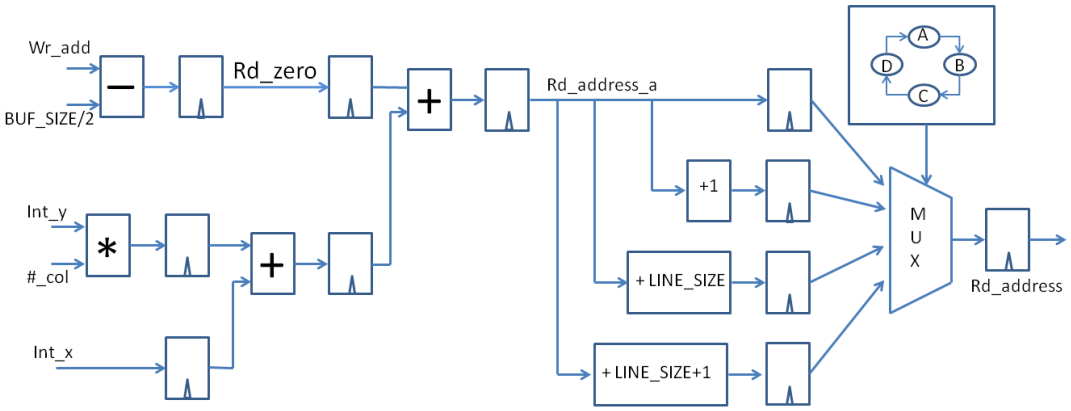

Figure 4.6: Read Address generator fully pipelined - This module implements the same function than its version without pipeline, but because the critical path is smaller, it can work at higher frequencies.

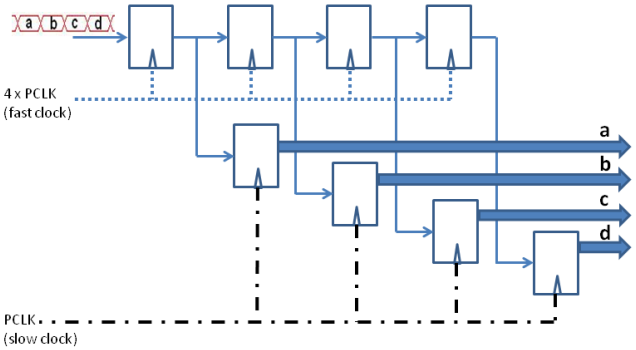

Figure 4.7: Serial to parallel Distortion Correction module - Vicinity pixels are read at 4xpclk, then they are processed in parallel at the pclk frequency.

4.2.2 Histogram Equalization implementation

When working with HDR infrared images, dynamic range reduction is necessary, first, to save on memory and logic resources, and second, to display the images on a conventional DVI screen.

Histogram equalization (HE) tries to enhance global contrast of the images by distributing the intensities over the image histogram. In the case of our HE, the dynamic range of the images is reduced from 16-bits to 8-bits.

In the architecture presented here, the HE is performed in two steps. In the first step (Pre-histogram), using a priority encoder we extract 10-bits from the 16-bits. In the second step, the CLAHE algorithm is used to reduce the dynamic to 8-bits and at the same time enhance the contrast of the images.

4.2.2.1 Pre-histogram

This module receives a gray level image with 16-bits per pixel and outputs a gray level image with 10-bits per pixel. The 16-bits wide input is reduced by isolating 10-bits starting from the position of the maximum non-zero MSB (Most Significant Bit).

Working with 10-bits dynamic range for the HE, allows to reduce the histogram number of bins to 1024 thus saving on logic and memory. Performing the HE on 16-bits would yield better visible results but does not prove significant improvements for the requirements of this work.

4.2.2.2 CLAHE

This module receives a gray level image with 10-bits per pixel and outputs a gray level equalized image with 8-bits per pixel. Thus the CLAHE module reduces the dynamic range and improves the global contrast of the image.

In our CLAHE implementation, after simulation in Matlab we found a good performance using sub-images of size 256×256. With this block size and with an image resolution of 1024×768, the image is subdivided in twelve sub-images.

In a first step of the CLAHE implementation, each sub-image is processed independently and then a bilinear interpolation is implemented to ensure a smooth image with no block contouring artifacts.

To implement CLAHE, the histogram of each sub-image must be computed. The histogram of each sub-image is computed using an internal BRAM with a feedback using an adder as shown in the figure 4.8.

The module shown in figure 4.8 is instantiated once for each sub-image, this module is enabled by a sequencer according to the pixel coordinates and the image data valid signal. The histograms are build progressively, and at the end of the image all of them will be already computed. See figure 4.9.

In our approach, all the computed histograms for each sub-image are processed simultaneously during the vertical synchronization of the image. Each one of the computed histograms are swept multiple times to build progressively the LUTs. The LUTs transformations are stored in another set of BRAM memories. These LUTs will be used to process the pixels of the next frame. As shown in figure 4.10, the LUT is processed in five steps following the next sequence:

1. The excess of the histogram above the Clip Limit is computed (the CLIP Limit was previously set).

2. The histograms are clipped and the excess is distributed.

3. The CDF is computed from the clipped histogram using the equation 2.14.

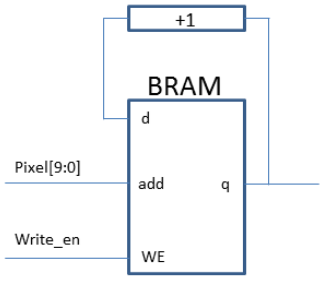

Figure 4.8: Histogram tile block diagram - Each histogram is computed using a BRAM memory and a counter. The input pixel is used to address the memory and increment the current count stored. Before beginning to implement the histogram, the memory is written with zeros to initialize the count.

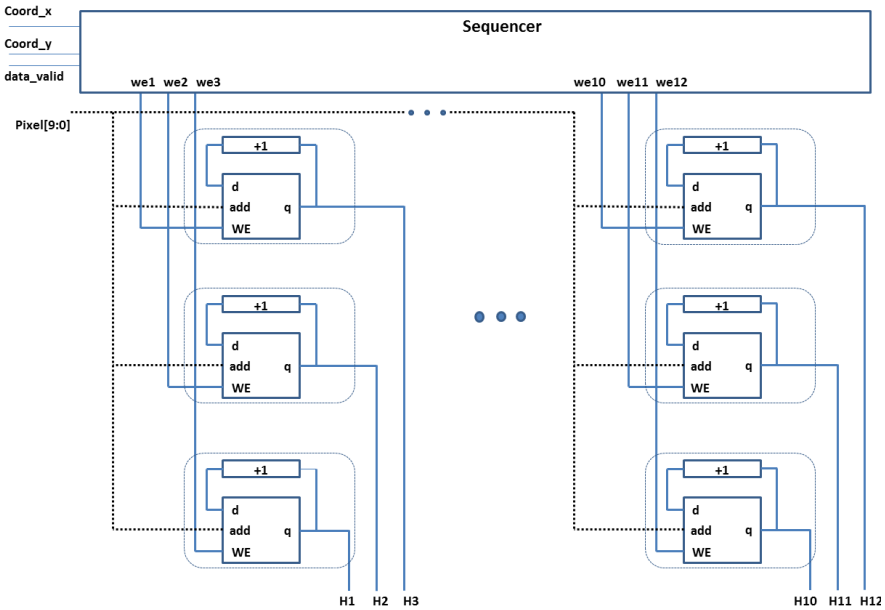

Figure 4.9: CLAHE histogram array architecture - When a new pixel arrives, just one of the histogram sub-modules is enabled, and the corresponding bin count will be incremented according to the pixel value. The histogram sub-module is enabled according to the coordinate of the arriving pixel.

4. The LUT is computed using equation 2.15. To reduce the resource utilization we fixed cdf_{min} to zero, this way, the division needed to compute the LUT is replaced with a shift operation.

5. The memories used to build the histogram are erased leaving the hardware ready to process the next frame.

In our architecture, a soft-processor is used to set the Clip Limit value.

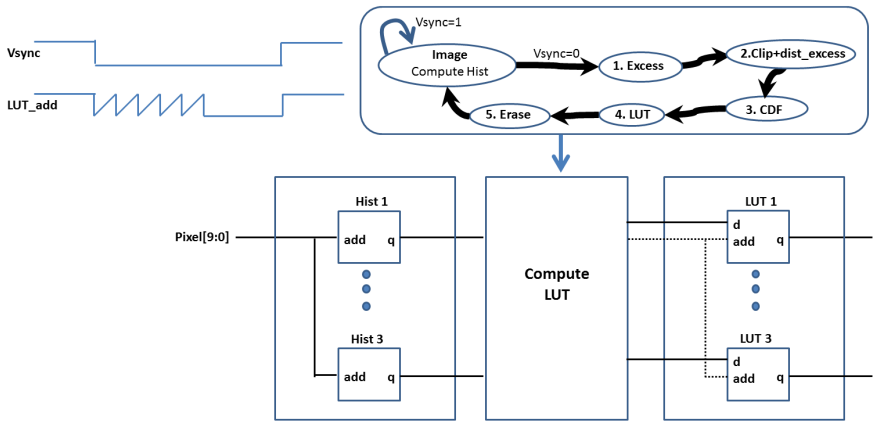

Figure 4.10: CLAHE histogram processing architecture - A two set of BRAMS are used in the CLAHE implementation: The first set is used to compute the histograms for each sub-image, and second set is used to store the LUTs for each sub-image.

Because in CLAHE each sub-image is transformed using a different LUT, a bilinear interpolation is neccesary to smoothen the images. The input pixel is transformed simultaneously using all the computed LUTs. Then, the block "Select Neighbor Region Pixels" selects four from the transformed pixels. A weight is assigned to each one of the four selected transformed pixels to perform a weighted sum. See figure 4.11.

In order to select the four quadrants vicinity pixels used for the bilinear interpolation, each sub-image is divided in four quadrants, and depending of the quadrant of the pixel to transform, the four neighbor LUTs are chosen as shown in the figure 4.12. For the pixels corresponding to the quadrants in the border of the image, a special treatment must be done.

In the approach used in this thesis, using the information from the frame N we compute the transformation to be applied to the frame $N + 1$. This approach is valid and does not represent a significant loss of quality under the hypothesis that the image properties between two consecutive frames are very similar when working at 30 FPS.

4.3 Perspective transformation implementation

This module is intended to implement the perspective transformation on images in real-time at 30 FPS with a resolution of 1024 x768. The architectures presented in this section can be used to accelerate any application that uses the homographic transformation.

Figure 4.13 shows two approaches that can be used to implement a homography.

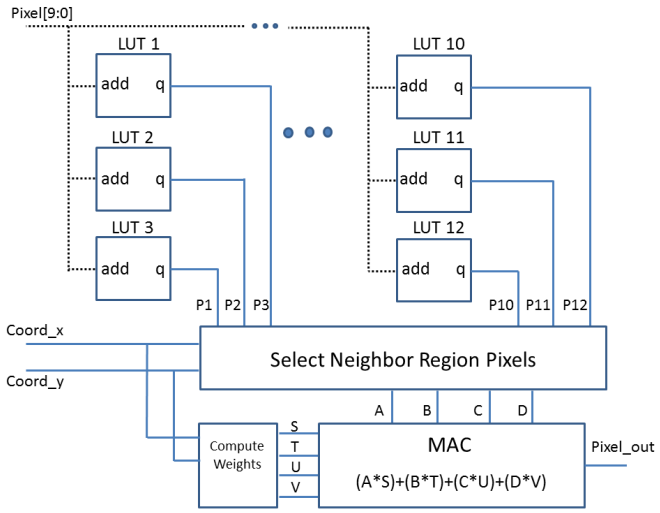

Figure 4.11: CLAHE LUT interpolation - The diagram represents the 12 LUT's used to transform the input pixel. Each new pixel is transformed simultaneously using the 12 LUT's. From the 12 transformed pixels, a module selects the necessary pixels to implement an interpolation. The selection and the weight of this pixels depends of the coordinates $(Coord_x, Coord_y)$.

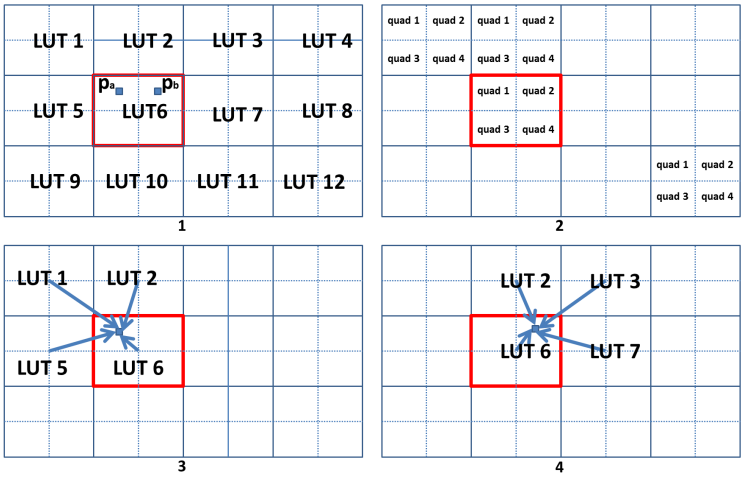

Figure 4.12: CLAHE quadrant interpolation - According to the quadrant different transformed pixels are choosen. In (1) are shown the pixels p_a and p_b corresponding to the sub-image 6. In (2) are shown the quadrant division of each sub-image. In (3), the pixel p_a corresponding to the quadrant 1 of the sub-image 6 is interpolated using the information from the LUTs 1,2, 5 and 6. In (4), the pixel p_b corresponding to the quadrant 2 of the same sub-image is interpolated using the information from the LUTs 2, 3, 6 and 7. The size of the rows shown in (3) and (4) represent the weights of the interpolation.

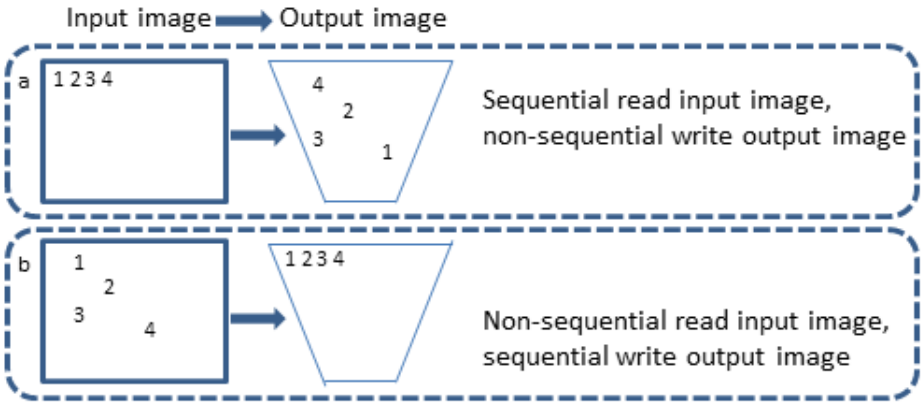

Figure 4.13: Homography accesses sequential and random approaches. -

In the first approach (a), sequential reads of the input image and non-sequential write of the output image are performed. The disadvantage of this approach is that all the pixels of the output image may not be updated because according to the homography matrix, some output coordinates may not exist. It implies the necessity of an extra post-processing algorithm.

In the second approach (b), non-sequential reads of the input image and sequential writes on the output image are performed. Using the homography matrix, each output pixel is mapped to an input pixel. The output coordinates can correspond to non-integer coordinates, thus an interpolation could be implemented to improve results. Output coordinates can also be outside the image resolution (these pixels are usually replaced by a null value).

Architectures presented in this thesis perform (b) non-sequential reads of the input image and sequential writes of the output image.

The perspective transformation of an image is done using the equations 2.12 and 2.13 to compute the transformed coordinates X_h, Y_h for each pixel. In our implementation, a soft processor is used to compute the homography matrix, and the transformed coordinates are computed using the FPGA hardware resources. See figure 4.14.

In this architecture, to be used for each frame.

Using the homography matrix, each output pixel (X_h, Y_h) is mapped to an input pixel (X_{in}, Y_{in}). The output coordinates can correspond to non-integer coordinates, thus an interpolation could be implemented to improve results, however, in the implementation presented here, the transformed coordinates are just truncated. It means that the output pixel will be always replaced by the pixel corresponding to the top-left side of the coordinate. Output coordinates can also be outside the image resolution (these pixels are here replaced by a null value).

The transformed coordinates are completely computed using LE of the FPGA. A total of 8 multiplications, 6 additions and 2 divisions are done in parallel, these operations are computed in a pipelined fashion with a latency of 38 clock cycles.

The most critical arithmetic operation of the homography is the division. To implement a homography, two divisions are needed to process each pixel of the image. For its implementation the High-radix divider [81] IPCore of Xilinx was used. The divider module generated with Core Generator of Xilinx has a latency proportional to the operand sizes, in our case the latency was 36 clock events.

Using the homography arithmetic unit presented in the figure 4.14, we propose two different architectures to implement the homography. The main difference between the two architectures

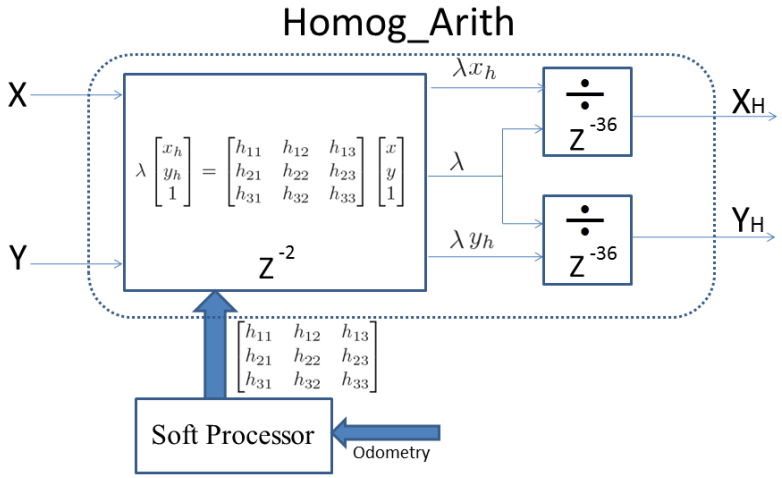

Figure 4.14: Homography Arithmetic Unit block diagram - In our approach a Microblaze processor computes the homography matrix using the odometry, and sends it through a FSL (Fast Simplex Link)

presented is the memory management. In the architecture presented in figure 4.15, a SDRAM memory is used to implement the entire algorithm, while in the architecture presented in figure 4.16 a Cache memory is added to speed-up the algorithm.

In the next subsections, we will describe the operation mode of each of these architectures.

4.3.1 Homography without Cache memory

Figure 4.15 shows the architecture of the homography without Cache memory. In this approach, the input image (already stored in the SDRAM memory) is read in a random way pixel by pixel according to the homography matrix.

This module access the memory using the protocols shown in the figure 3.14. Pixels are read using the NPI interface of the MPMC. Output pixels are written in burst mode using the VFBC interface of the MPMC.

This module uses a counter (XY_COUNTER), which generates the non-transformed coordinates (X_{in}, Y_{in}) for each pixel consecutively. These coordinates are transformed by the arithmetic unit (HOMOG_ARITH) presented in the figure 4.14.

Then, the transformed coordinates (X_h, Y_h) are sent to the module XY_TO_ADD, which generates the physical address using the base address and the resolution of the image.

Finally, the module WR_RD_HOMOG retrieves from memory the pixel corresponding to the coordinates (X_h, Y_h), which will be written in the output image.

This architecture reveals a memory access bottleneck when using SDRAM memory. The read operation being performed for single pixels, suffers a high initial transaction latency penalty (30 clock cycles), making it unable to handle the source camera frame-rate (30 FPS). Because the read addresses may not be sequential burst accesses cannot be done.. One solution to tackle this problem is to use SRAM memory but it would increase the system cost. Another solution is to implement a Cache management strategy to tackle memory accesses issues.

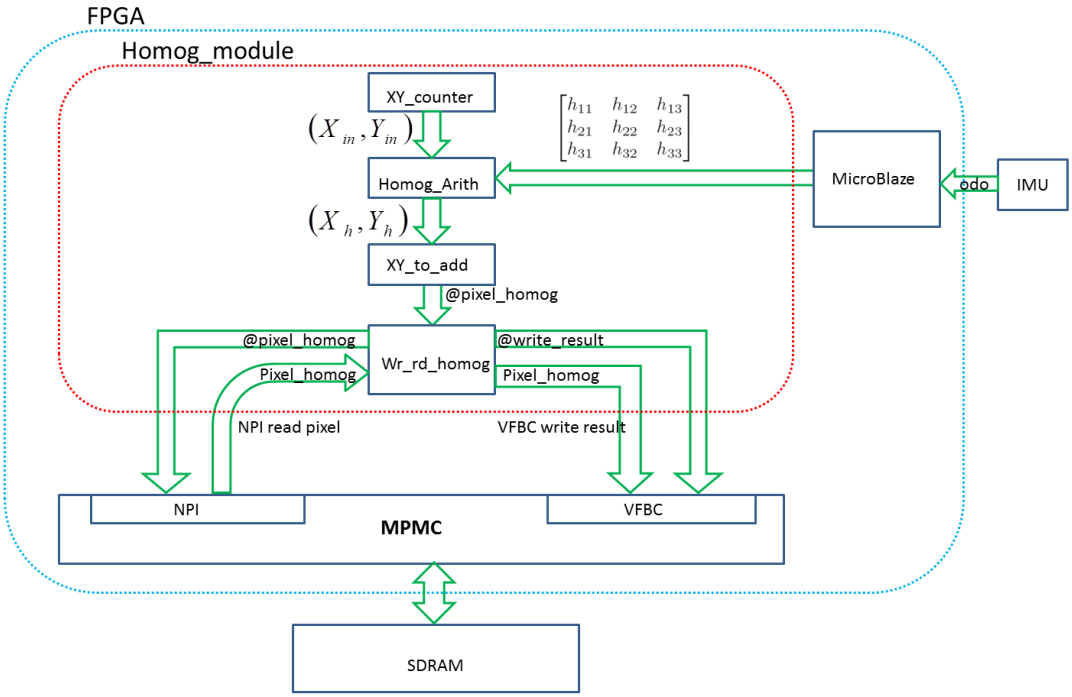

Figure 4.15: Homography module without Cache memory - This module implements the homography of an image. The main disadvantage of this module is that the accesses to the memory SDRAM are done in byte mode, with random access, which implies a time penalization.

4.3.2 Homography with Cache memory

In order to run the algorithm in real-time at no additional cost, we chose to implement a Cache memory designed to speed-up the homography and make use of the burst access of the SDRAM memory.

Figure 4.16 shows the architecture implementation of the homography using a Cache memory. Here the input image is stored in memory and it is sequentially loaded in the Cache memory by the module DATA_VFBC_CNTLR. The Cache memory is managed like a circular buffer.

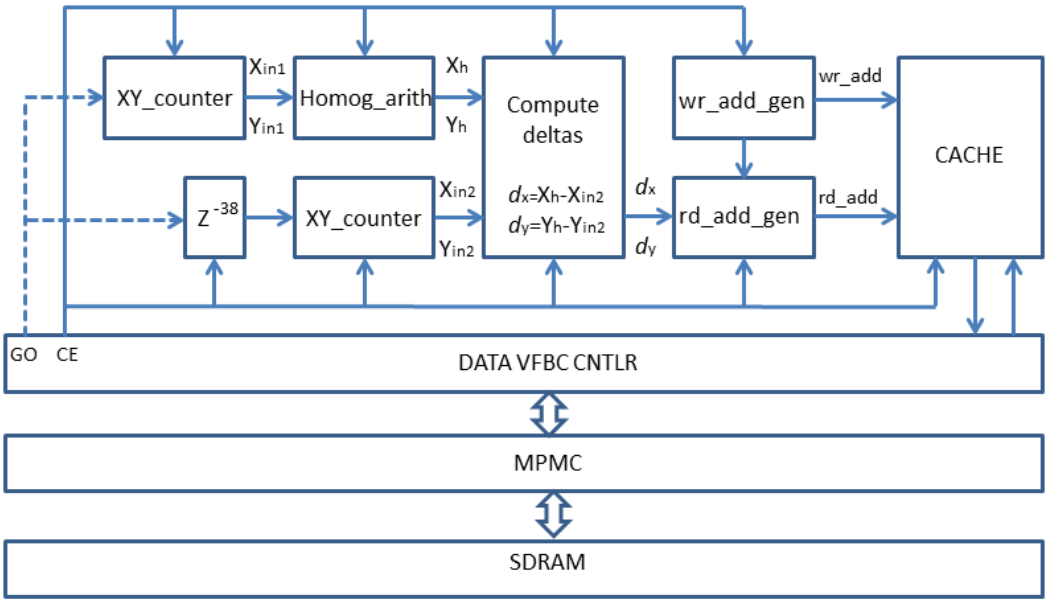

Figure 4.16: Homography module with Cache memory - Block diagram of the homography module using a Cache memory. In this architecture the Cache memory is used like a circular buffer.

The non-transformed coordinates (X_{in1}, Y_{in1}) of each pixel are generated by the module XY_COUNTER. These coordinates are processed using the homography matrix by the arithmetic unit module presented in the figure 4.14. The non-transformed coordinates (X_{in2}, Y_{in2}) are generated by a second XY_COUNTER, which was synchronized with the pipeline taking into account the latency of the homography arithmetic unit.

The module COMPUTE_DELTAS generates the signals (δ_x and δ_y), which correspond to the differences between the transformed coordinates (X_h, Y_h) and the non-transformed coordinates (X_{in2}, Y_{in2}). The signals (δ_x and δ_y) and the wr_{add} are used to generate the read address of the Cache memory.

The module DATA_VFBC_CNTLR accesses sequentially the SDRAM memory through the MPMC using a single full duplex VFBC interface. The signal CE (Chip Enable) is asserted according to the state of the VFBC FIFO's.

The Cache memory is implemented using internal BRAMs memories, which store N lines of the image, thus limiting the homography to a max δ_y of $Y_h - Y_{in} = \pm N/2$.

4.4 Multispectral Vision implementation

To implement the fusion between two images, acquired with two different cameras, with different characteristics and from different point of views, first of all, we must correct the distortion and align the images. For this, we use a table to encode simultaneously the distortion correction and the fusion.

Because the images are taken from different points of view, a perfect alignement between the two camera views is not possible, however, a perspective transformation can be applied to align two planes from the two images. In our implementation, we decided to align the ground planes, because in driving assistance, most of the important information is on this plane, especially for an aircraft environment.

Figure 4.17 shows the block diagram of the IR-VIS Fusion module.

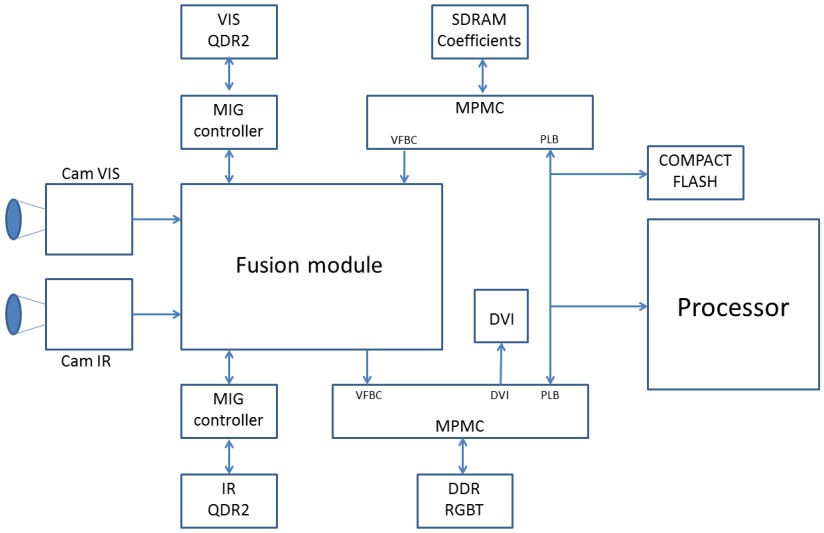

Figure 4.17: Fusion IR and VIS using QDR2 memory -

To implement the fusion of IR and VIS images, using images with high resolution and important distortions at 30 FPS, we propose to use two QDR2 memories and two DDR2 memories. The QDR2 memory has the advantage that random accesses can be done without no timing penalization. This memory allows four access on each clock event (two writes and two reads) like shown in the figure 4.18.

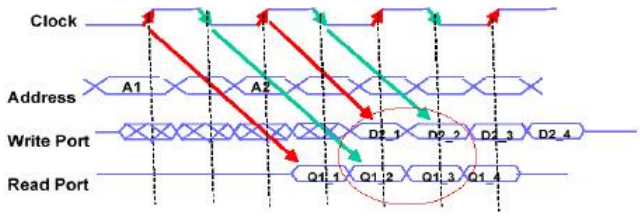

Figure 4.18: QDR2 timing diagrams - Taken from QDR Consortium

In each one of the QDR2 memories, a circular buffer contains some lines of each image (VIS

and IR). To build each one of the pixels of the output image, 4 accesses to the QDR2 memories are done (to read the group of 4 pixels needed to implement a bilinear interpolation).

In this architecture there are two tables of coefficients, one table for each camera. The table encodes the transformation to be done to each one of the pixels of the image. The transformation contains simultaneously the correction distortion, homography and scaling information. These tables were also computed offline and stored in a Compact Flash. The data is transferred from the Compact Flash to the DDR2 memory, from where it is normally read.

The output image is written through a VFBC interface to another DDR memory (called in the figure DDR RGBT). Through unused ports of the MPMC the output image can be visualized, or processed.

The perspective transformation to be applied to align the ground planes of the Visible and infrared images corresponds to a fixed homography that is computed using four or more correspondence points between the IR and the VIS images for the ground plane. Then, using the equation 2.27, the system is solved using the LMS algorithm.

This homography is valid under the hypothesis that the extrinsic parameters of the camera will be constant. However, this is not completely valid because of the damping effects of the vehicle caused by the accelerations.

To align the images using the Homography, either of the methods described in sections 4.3 can be used. After applying the perspective transformation using a Homography, the pixels that aren't in the ground pixels will not be aligned and will create ghosts in the fusion image.

Aligned images can be mixed to generate a new image containing the information of both infrared and Visible. This synthetic image can be presented to the pilot and can be generated using combinations in different color spaces.

For the HSV fusion, the channel H is mixed with the infrared. For the YcbCr, the channel Y is mixed with the infrared. For the RGB fusion, pixels with a high gray scale level of Red, Green or Blue are incrusted in the infrared image. A threshold determined using the standard deviation and the mean of the gray level is used to detect the pixel that should be inlayed.

4.5 Obstacle detection implementation

For obstacle detection, two different approaches were implemented, The first approach IPM, uses a perspective transformation between two images, while the second approach uses textures attributes combined with an adaboost classifier.

4.5.1 IPM

The IPM or SIPM implementation requires the computing of three distinct operations on the image stream. For each pixel of the image, a distortion correction is applied followed by a homographic transformation. Then, a subtraction is performed to reveal the image of obstacles. The homographic transformation is the most costly part of the algorithm and requires optimization for real-time operation when using a high-latency/high-throughput RAM (DDR-SDRAM).

4.5.1.1 Monocular IPM

Monocular IPM works by computing the difference between one image I_1 and the homography of a second image $H[I_2]$. The homography matrix is computed using the relative movement between the two images and the extrinsic and intrinsic camera parameters using the equation 2.25. The relative position comes directly from the odometry or can be estimated using the

SLAM algorithm. The ideal would be to use the SLAM output to avoid positioning errors that are reflected in false detections.

To perform this algorithm, an image must be stored in order to perform the IPM using the next acquired image. One drawback of this method is its memory requirement, as it needs to store at least one image. We evaluated different options to be able to implement the IPM in real time using a SDRAM memory.

IPM algorithm needs a memory to store the reference image I_1 that will be subtracted with $H[I_2]$. The reference image must be stored in an external memory (SRAM or SDRAM). Because the SRAM is more expensive than SDRAM, the development of the algorithm using a SDRAM memory is very attractive. To implement the design using a SDRAM, considerations like burst and sequential accesses have to be taken into account in the architecture design, otherwise time penalization will occurs. To use the SDRAM, we included a Cache memory allowing real-time operation.

Figure 4.19 shows two different approaches using an internal BRAM as Cache memory. In the first approach (a), IPM algorithm is implemented using the equation 4.1. In the second approach (b), IPM algorithm is implemented using the equation 4.2. Both approaches are similar but approach (a), yields better results thanks to a lower latency (see Figure 4.20).

$$abs(H_{12}[I_n] - I_{n-1}) \tag{4.1}$$

$$abs(H_{21}[I_{n-1}] - I_n) \tag{4.2}$$

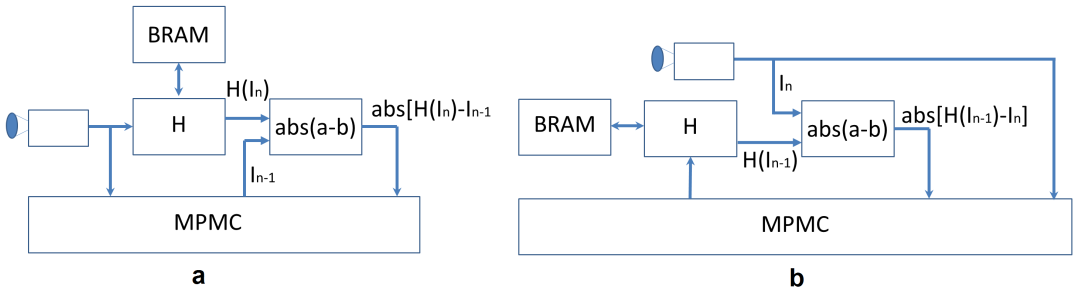

Figure 4.19: IPM block diagrams - On the left, IPM implementation using equation 4.1. On the right, IPM implementation using equation 4.2

4.5.1.2 Stereo IPM

To implement the SIPM two different approaches are possible:

1. In the first approach, homography is applied to just one of the cameras. The left image is transformed with a homography like shown in the left side of figure 4.21, resulting with a virtual left camera placed in the same location of the right camera. For pixels in the ground plane, the image from the virtual left camera corresponds to the pixels of the right camera.

2. In the second approach, homography is applied to both cameras. The left and right images are transformed to a bird's-eye view like shown in the figure 4.22. There will be two new

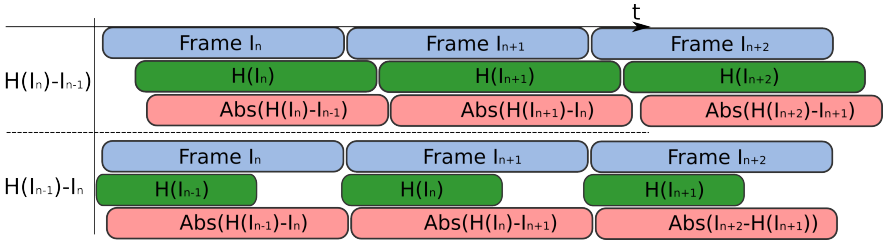

Figure 4.20: Gantt chart to compare latency of both MIPM methods - $H[I_{n-1}]$ can be built at the start of frame of I_n. To build $H[I_n]$ some lines of I_n must be already in memory, which implies an extra latency

virtual cameras placed in the spot, with the focal plane parallel to the ground plane. The advantage of this approach is that in the output image, corresponding to the bird's eye view, the results could be exploited directly to extract obstacles distances.

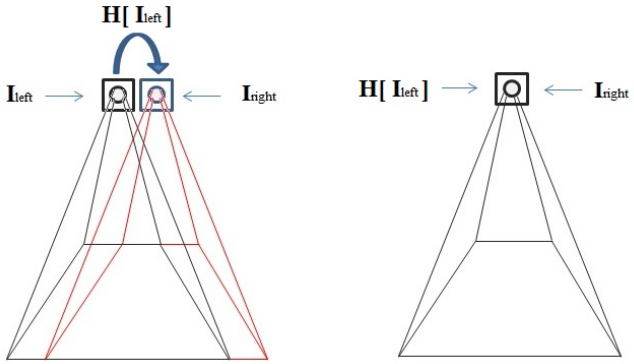

Figure 4.21: Stereo IPM representation - $H[I_{left}] - I_{right}$, here I_{left} is transformed with a Homography to remove the perspective of the ground points with respect to I_{right}.

The homography needed for SIPM can be implemented using the module described in section 4.3, with the difference that in SIPM a fixed homography is used, thus the soft processor does not need to update periodically the homography matrix. For SIPM the homography transformation can be known in advance and hard coded at compile time.

Another approach to implement SIPM is the use of a table to encode the homography transformation. The advantage of using a table to encode the homography transformation is that the distortion correction can be simultaneously encoded without no extra cost in terms of FPGA resources.

4.5.2 Texture Classification

In this thesis we present an architecture implementing the algorithm described in [11] replicated to four cameras. The obstacle detection is implemented in parallel for each of the cameras, and a output image is generated containing the classification results of the camera belt. The camera

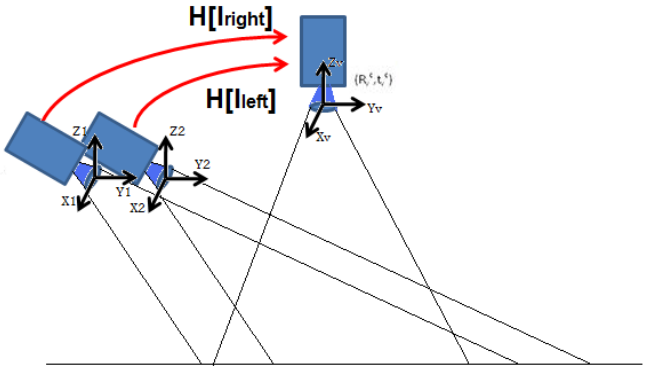

Figure 4.22: Stereo IPM Bird's eye view approach - Both cameras I_{left} and I_{right} are transformed with a homography to have a bird's eye view from the same point of view.

belt constitutes a multi-camera system used to create an occupancy grid of the obstacles around the vehicle.

The acquisition and processing pipelines of N images are implemented in parallel, using the algorithm presented in the section 2.9.1. Each image flow is processed at pclk frequency, then the module MEM_WRITER writes the processed images to the SRAM using a Dual Clock FIFO. See figure 4.23.

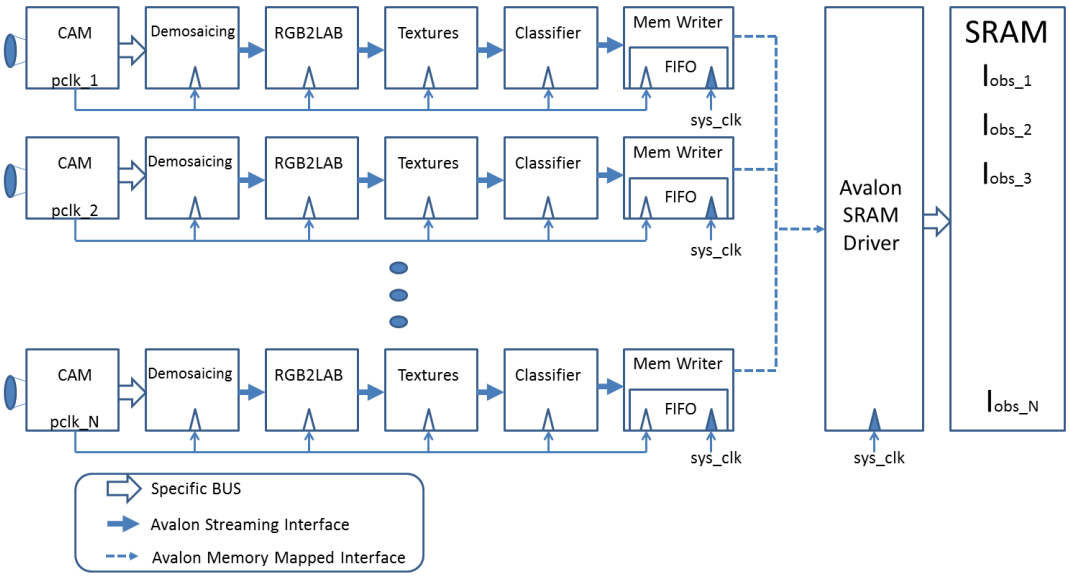

Figure 4.23: Obstacle detection pipelines - ..

The MEM_WRITER module access the memory using the Avalon memory mapped interface. When there are enough data in the FIFO, the MEM_WRITER module ask the Avalon arbiter for the control of the BUS. If nobody is using the BUS, the arbiter gives the access to the concerned MEM_WRITER, otherwise, after the bus is free the transference begins. See

figure 4.24.

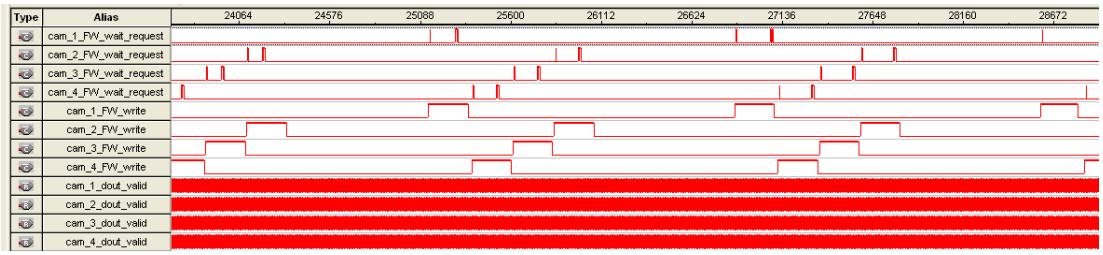

Figure 4.24: Multiple camera accesses to Avalon Bus - The cameras access the bus according to the status of the FIFO.

The FIFOs of the MEM_WRITER modules are handled using a kind of Schmitt trigger behavior managed with its Almost Full and Almost Empty configuration.

From the processed images already stored in memory, an occupancy map was generated in real time using different frame references. See figure 4.25. This architecture uses a special module called Pixel_Fetcher, which receives the addresses of the pixels to be read, and writes the output pixels consecutively in another destination memory space. In order to create the Occupancy grid, two external memories were used in this system.

In order to build the Occupancy grid, the following steps were done:

1. In a SRAM, the processed input images are stored.
2. The module Pixel_Fetcher_A reads pixels from the processed images already stored in the SRAM memory. The pixels are read according to the information of a table stored in the SDRAM memory. This table encodes a perspective transformation and a fusion of the four obstacle images. The result of this process is an Occupancy grid, which is written in the SRAM memory. This Occupancy grid is written in the SRAM memory because later it may be accessed randomly, in order to implement change the frame of reference.
3. The soft processor NIOS2 reads the odometry through the Avalon Memory interface comming from the robot.
4. NIOS2 processor configure the module Dynamic_Rotation_Translation.
5. The module Dynamic_Rotation_Translation creates the address (@) necessary to read the Occupancy grid in order to create a transformed Occupancy grid centered in different frame of references.
6. The transformed Occupancy grid is stored in the SDRAM.
7. The transformed Occupancy grid is read by the DMA in order to be sent using UDP Ethernet communication to the Host processor.

In the implemented architecture, an FPGA implements the acquisition and the image processing of the four cameras in parallel. At the same time the odometry information is received from the Desktop PC through an Ethernet link. According to this information the obstacles images are aligned to build an occupancy grid. This occupancy grid can be built using the robot or the world frame of reference. The occupancy grid is transmitted to a Desktop PC, which displays the images on the screen. The Desktop PC reads the odometry from the Scout

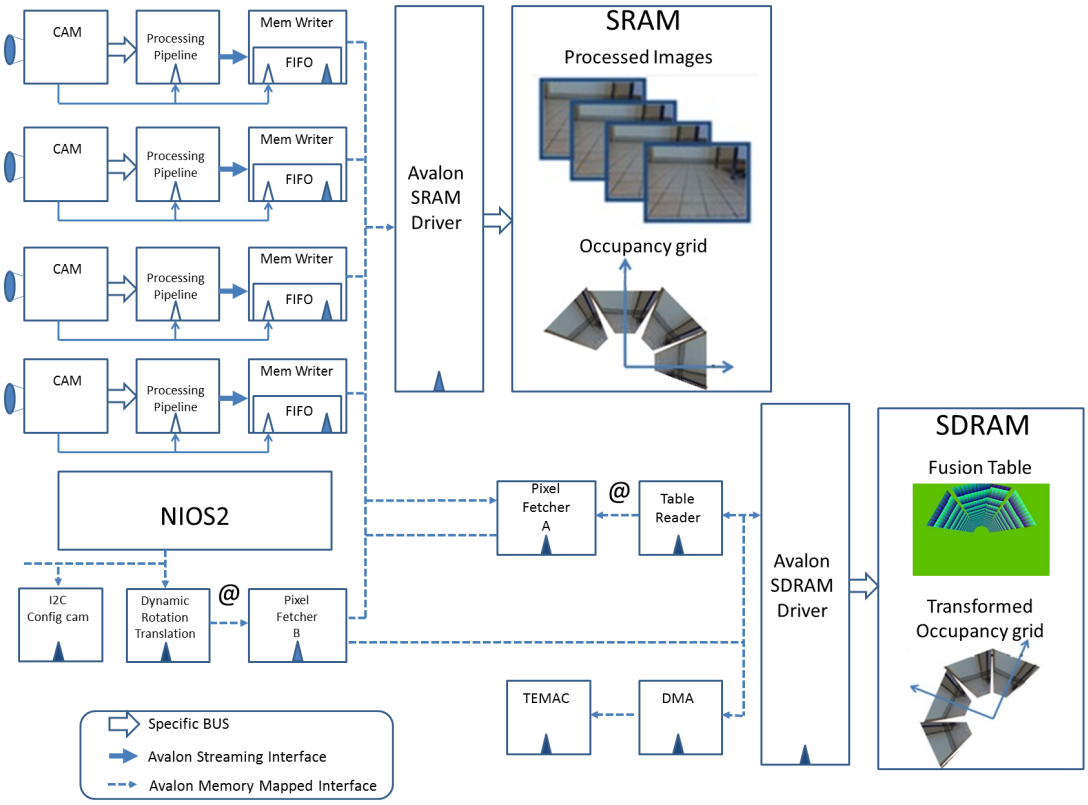

Figure 4.25: Occupancy grid with different frame references - ..

[82] robot, using posters coming from a script that accesses the robot encoders and generates the coordinate (x,y) and the rotation angle of the robot. See figure 4.26.

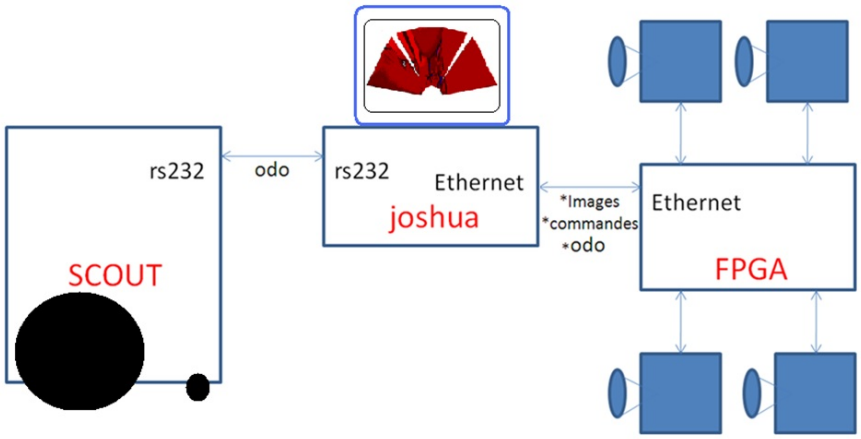

Figure 4.26: Commrob block diagram -

4.6 Stereo vision implementation

In [10], stereo vision was implemented in an FPGA without the distortion correction algorithm, in their implementation the Census correlation is used because of its low complexity properties. However, they worked with distorted images, and the alignment of the cameras was done using the system shown in the figure 2.12; in this thesis we present an improvement of that stereo vision architecture using real-time distortion correction.

The implementation of the distortion correction uses the architecture described in section 4.2.1. This architecture allows us to simultaneously correct the distortion of the cameras and satisfy the epipolar constraint described in section 2.8.

Figure 4.27 presents a block diagram of the stereo vision system, composed of three Boards connected together. Each board has a low-cost FPGA Altera Cyclone 2 (EPC20F484C8), a SRAM memory (CY7C13800), and a non-volatile memory type FLASH (AM29LV320M).

For the implementation of the Stereo vision system presented here, no soft processor was used because the target FPGAs were small devices (Cyclone 2). Consequently, in order to access the SRAM and the FLASH memory, VHDL drivers were used, which complicated the development of the different parts of the algorithm. Specially the FLASH memory driver presented a lot of difficulty. These drivers were developped by Delta Technologies Sudouest, but they were not enough documented.

In our system we have two JAI cameras model CV-A33 [83], with camera link interfaces working at 100 FPS with a pixel clock of 40MHz. The disparity map is sent also through a Camera Link Interface to a Desktop PC that is used to display the output images. All the synchronism output signals are generated synchronized with the undistorted generated images, for this, a finite state machine was used. The finite state machine takes into account the latency of the distortion correction module. This latency corresponds to the time necessary to store a quantity of lines necessary to implement the remapping of the pixels.

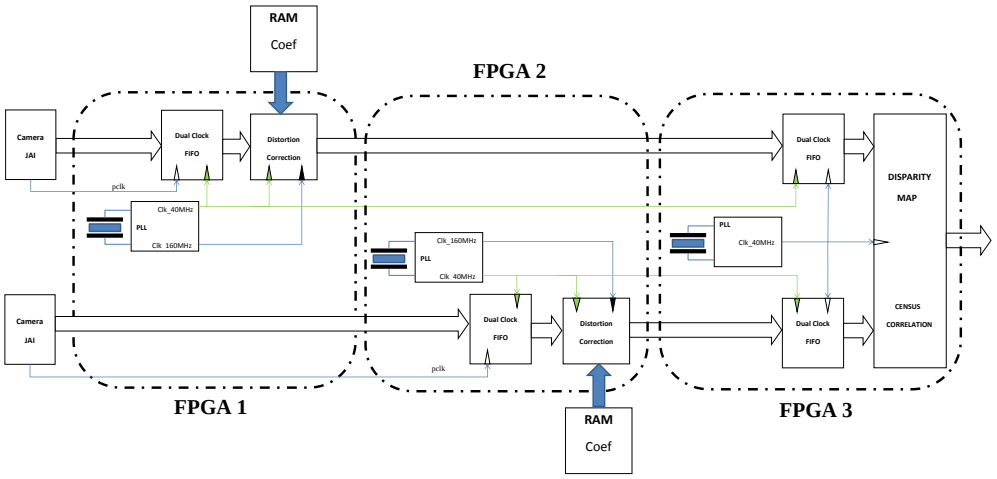

Figure 4.27: Picasso Hardware - Picasso hardware is composed of three Cyclone 2 FPGAs connected together through GPIOs, and sharing the same clock.

In our system one difficulty comes from the fact that each camera has its own clock completely asynchronous with the system clock. In order to avoid metastability problems, a dual clock FIFO is used to implement the cross clock domain. The images coming from two different clock domains are processed in a new clock domain managed with internal PLLs of the FPGAs.

The correction distortion is implemented using coefficients computed offline using a stereo vision Toolbox. Following the same methodology of the section 4.2.1, a table is generated to implement the replacements. The table is uploaded to the stereo system through a USB2.0. A board with a FX2 [84] allows to implement USB2.0 communication. Using the FX2, from the point of view of the FPGA, the USB2.0 protocol is managed like a FIFO. The FX2 is a complex device including a high-speed 8051-microcontroller. This device is programmed in C according to the requirements of the application, it supports Isochronous and Bulk transfers. In our case we used Bulk transfers because we needed to guarantee the integrity of the data. In our system we configured two endpoints, one to upload the coefficients to the FPGA system, and another to verify the correct transference of the data.

Figure 4.28 shows the architecture of the system used to upload the coefficients to the stereo system, in normal operation, after power-up the system, the coefficients are loaded by the sequencer module from the Flash memory to the SRAM memory.

In order to store the coefficients in the FLASH memory we followed the next steps:

1. FX2 is recognized by the Host.

2. Coefficients are sent from the Host to the FX2.

3. FPGA reads the FX2 FIFO and writes the data in a SRAM memory.

4. Data is transferred from the SRAM memory to the FLASH memory.

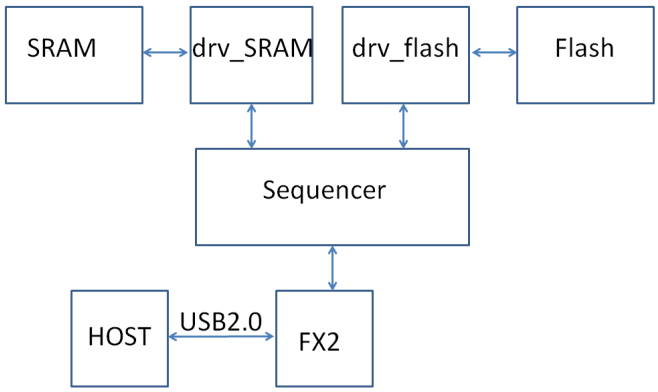

Figure 4.28: Distortion coefficients upload mechanism - When setting up the system, the distortion coefficients are transferred from a host processor through a USB2.0 link.

4.7 Embedded SLAM

In this section, we present the architecture of an FPGA accelerator for a co-design implementation of a Simultaneous Localization and Mapping algorithm (SLAM) from multispectral vision. The developed SLAM architecture is designed to accelerate the RT-SLAM architecture described in section 2.12.2.

The accelerator implements all the image processing tasks of the SLAM algorithm using multi-spectral sources. This accelerator is based on a FPGA that exploits the computing pipeline inherent to the image processing tasks. Thus, we can speed-up image pre-processing (histogram equalization, distortion correction), landmark detection and landmark tracking to reduce the SLAM complexity. Such accelerator can then be used with a companion core that performs the remaining SLAM tasks with an improved efficiency.

Figure 4.29 shows the global architecture of the accelerator; the system is driven by the pixel clock; images are processed on the fly.

At boot time, a soft processor is in charge of loading the camera calibration data from an external storage into SDRAM memory to configure the distortion correction modules.

For the infrared image, the pixel pipeline starts with the CLAHE module that performs the histogram equalization on the infrared images. This module aims at extracting the usable information (8 bits) from the IR camera dynamic range (16 bits). Then, distortion correction is implemented in parallel for infrared and visible images, based on calibration data stored in memory.

Several parallel flows are then activated on each undistorted image in order to extract features.

1. For every landmark selected in the current map, the active search module performs a correlation in order to look for its true observation in the image region corresponding to its predicted observation.

2. The Harris module extracts the N best Harris points to be used in order to initialize new landmarks in the next SLAM iteration.

3. The undistorted images are also stored in the SDRAM memory via the MPMC to later be displayed through the DVI link for debugging purposes, or for an HMI interface. Other processing could be applied in parallel, but only operations related to SLAM are considered here.

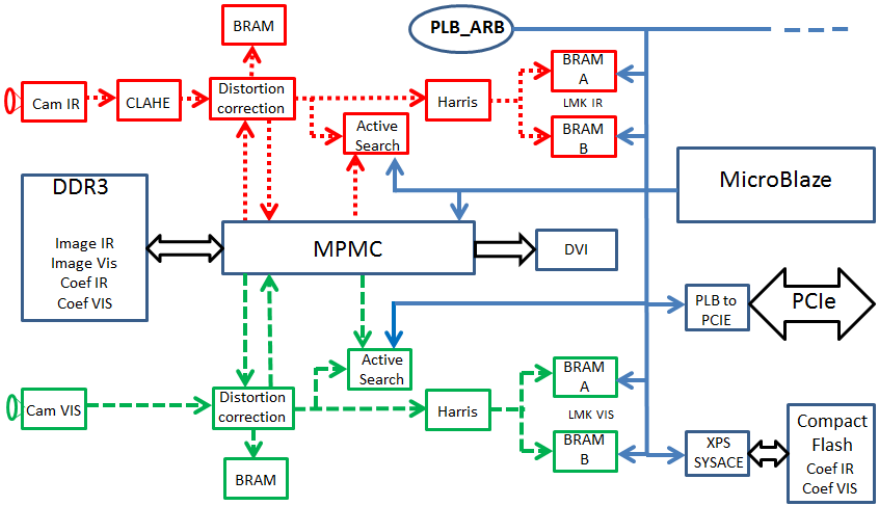

Figure 4.29: SLAM accelerator block diagram - The red dotted line represents the data flow of the infrared images, the green dotted line represents the data flow of the visible images and the blue one represents the PLB Bus

Thanks to the available parallelism, the different pipelines free the CPU from image processing tasks. An only software implementation is limited by its arithmetic unit and its sequential architecture scheme. The pixel level pipeline architecture for each camera front-end (Camera → Clahe → Distortion correction → { Active search - Harris } → Memory). See figure 4.30.

This accelerator allows a low latency computation of the SLAM inputs with a small memory footprint that permits most data used by the pipeline to be stored in local BRAM. Using local BRAM also helps to avoid the bottleneck issues in memory access.

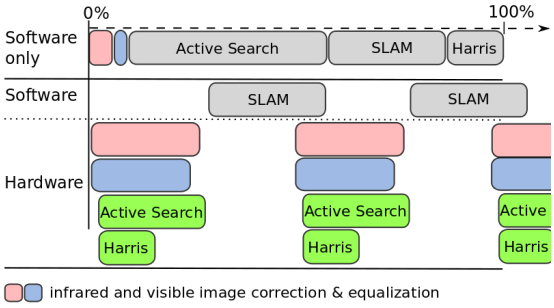

Figure 4.30: Gant diagram representing SLAM speed-up given by the co-design -

At each iteration, the PCIe link allows communications between the SLAM accelerator and the host processor executing the SLAM algorithm. It can either be an embedded processor

like a PowerPC (connected through the PLB bus), a conventional PC (in a machine with a PCIe slot connected through the PLB to PCIe bridge) or a multiprocessor system. It needs the Harris points and the Active search results in order to update the map, while the SLAM accelerator requires positions and sizes of the predicted regions in which landmarks have to be searched.

Figure 4.31 shows the data flow between the SLAM accelerator and the host processor, using three oriented edges:
1. The Landmark list is sent to the host processor. It gives the coordinates (x,y) of Harris points detected in the current frame; such a point is an observation used to initialize a new landmark.
2. Requests are sent to the Active Search module. The host processor requires the accelerator to perform a search of each selected landmark in its predicted region characterized by its position and a vertical and horizontal uncertainty. A window deformation configuration can be also sent to the system in order to implement an homography to transform the landmark signature according to the motion made by the camera since its initialization.
3. Results of the Active Search module are sent to the host processor; it gives coordinates of the true observations (i_{obs}, j_{obs}) of selected landmarks (best correlation score for the landmark in its predicted region).

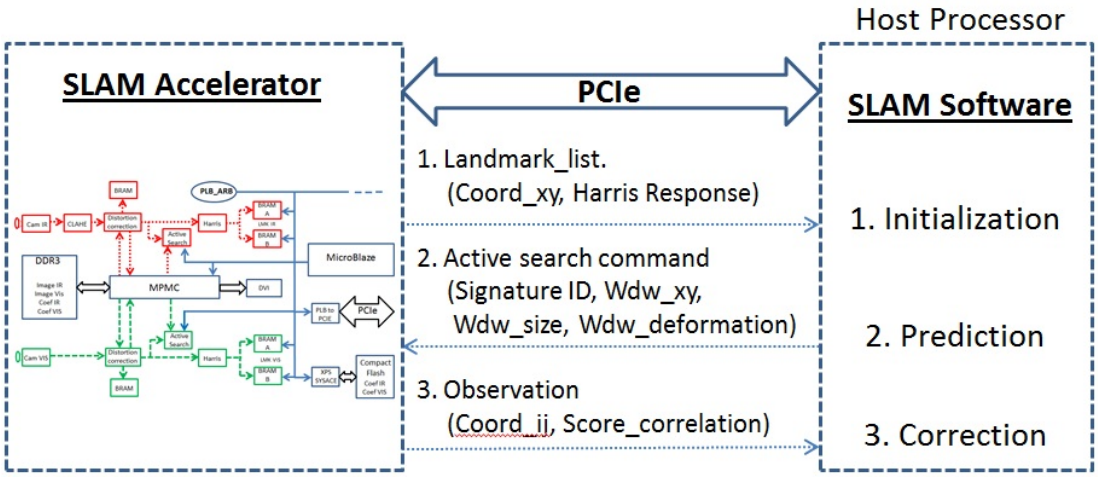

Figure 4.31: Communication through PCIe between the host processor and the SLAM accelerator -

4.7.1 System prototyping

The final purpose of this accelerator is to be integrated with a Virtex 5 FPGA with an embedded PowerPC core and a hardware accelerator (floating point matrix product) to run the EKF-SLAM algorithm. Evaluating the software only implementation of the vision based EKF-SLAM revealed that up to 53% of the processor time was dedicated to run the image processing tasks (50% for active search).

A first prototype consisting of a host PC running vision related tasks connected to a Virtex 5 development kit (ML507) through PCIe allowed us to evaluate the PowerPC implementation of the EKF-SLAM and showed performance of up to 5FPS. The main performance limitation came

from data serialization (conventional PC and PowerPC does not have the same endianness) and PCIex communication speed/latency (limited to 28MB/s with high latency in our design). This 5FPS speed can appear poor compared to real-time implementations, but allowed to evaluate the SLAM on an existing image set and to profile the code running on the PowerPC. The PowerPC profiling revealed the need for a floating point matrix product accelerator to achieve real-time performance for the required number of landmarks.

A second prototype consisting of a host PC connected to a Virtex 6 development kit through PCIex allowed us to evaluate the behavior of the presented architecture on an existing image-set with the same speed-limitation stated above (PCIex bandwidth/latency).

A last prototype (Figure 4.32) consisting of a host PC connected to the Virtex 5 and Virtex 6 development kits through PCIex will allow us to evaluate the whole system on an image-set with SLAM results being displayed and logged on the PC. This prototype is not finalized at the time of writing this thesis, but should be working very soon.

These prototypes were not intended to provide real-time performances but allowed us to perform Hardware In the Loop (HIL) simulation of the final system. Moreover, it allowed us to profile the architecture, identify bottlenecks, and make decisions on SLAM related aspects (landmarks size and descriptor, number of landmarks to initialize, number of landmarks to track ...). See figure 4.32.

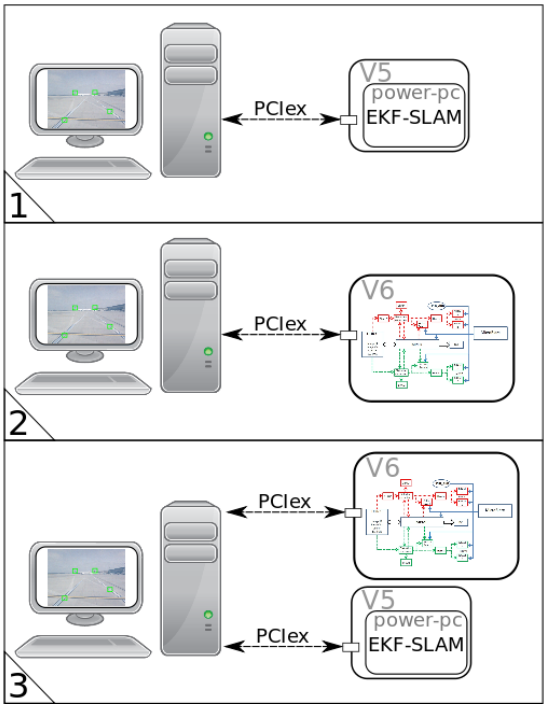

Figure 4.32: SLAM Prototyping - Top : prototyping with Virtex 5 + PC, **Middle** : Prototyping with Virtex 6 + PC, **Bottom**: Prototyping with Virtex 5 + Virtex 6 + PC

4.7.2 Matrix operations accelerators implementation

Using profiling tools, the bottlenecks of the SLAM algorithm were identified. When executing the SLAM algorithm, most of the time the processor was executing pixel operations on images and floating point operations in matrix.

Thanks to the architecture available in an FPGA, we developed a core tightlhy coupled to the system to accelerate the implementation of the matrix multiplication. The developed multiplier (AxB=C) is a generic co-processor which can be used to implement multiplication of matrices with sizes A=[2xN] and B=[NxN]. Figure 4.33 shows the block diagram of the developed matrix multiplier core.

The accelerator is coupled to the processor by the PLB bus. The matrix are first loaded in internal BRAMs (bram_param_a and bram_param_b), from where the accelerator will recuperate the data. The output result matrix is written to another BRAM(bram_param_c). See figure 4.34

Matrix multiplication accelerator was implemented using a Systolic array. The array combined with FIFO's allows the implementation of a floating point matrix multiplication without stalls. In the implemented architecture, a systolic array is fed continuously at a frequency of 300 MHz. This architecture allows us to implement a pipeline executing in parallel 3 multiplications, 3 additions and 1 accumulation on each clock cycle. See figure 4.35.

In our approach, during the time interval t=[1:N], the first 3 elements of each row of matrix A ($[A_{11}, A_{12}, A_{13}]$ and $[A_{21}, A_{22}, A_{23}]$) will be inyected to the systolic array, in order to implement the necessary operation with all the elements in first 3 rows of the matrix B. On each clock event, a set of three new elements of the Matrix B are inyected to the systolic array. Then, during the interval t=[N+1:2N] a second set of six elements of matrix A will be used, and so on. See figure 4.36

Data that will be injected to the systolic array are prepared by hardware modules, which have direct access to the BRAM (where the operands should be already stored). The elements of the matrixes A and B are completely stored in FIFO's that will be popped to feed continuosly the systolic array. See figure 4.37

Elements of the matrix A are managed with two FIFO's, which feed two sets of delay lines (delay_line_a and delay_line_b). Then the delay line modules are used like ping-pong buffers, so when operating the first 6 elements present in the delay line a, the delay line b is being prepared with the next 6 elements of the matrix A.

For the matrix B, 3 FIFO's are used. In each FIFO a different row of the Matrix B is stored. On each clock event, 3 new elements of the matrix B are read (one element from each FIFO) and sent to the systolic array.

The BRAMs are implemented as dual port memories, thus the contents of the memory can be accessed by a PLB master and by the co-processor. The PLB master can be a processor or a DMA controller.

The result matrix C with size [2xN] is computed progressively because of ressources limitation. For this, the temporary result matrix is stored in internal registers (C_{tmp}), and the multiplications are subdivided in small multiplications. Two floating point adders are used to accumulate the temporary results coming from the systolic arrays. The input of these adders are the temporary result and the output of the systolic array.

Thanks to the developed architecture, the matrix multiplications are executed at a higher frequency than the PLB frequency. Normally the processor will configure 2 DMA engines to load the operand matrix A and B, and the co-processor will execute the operation. The co-processor will interrupt the processor when the operation is done. This allows the processor to execute other tasks while the operation is implemented using the hardware resources of the FPGA.

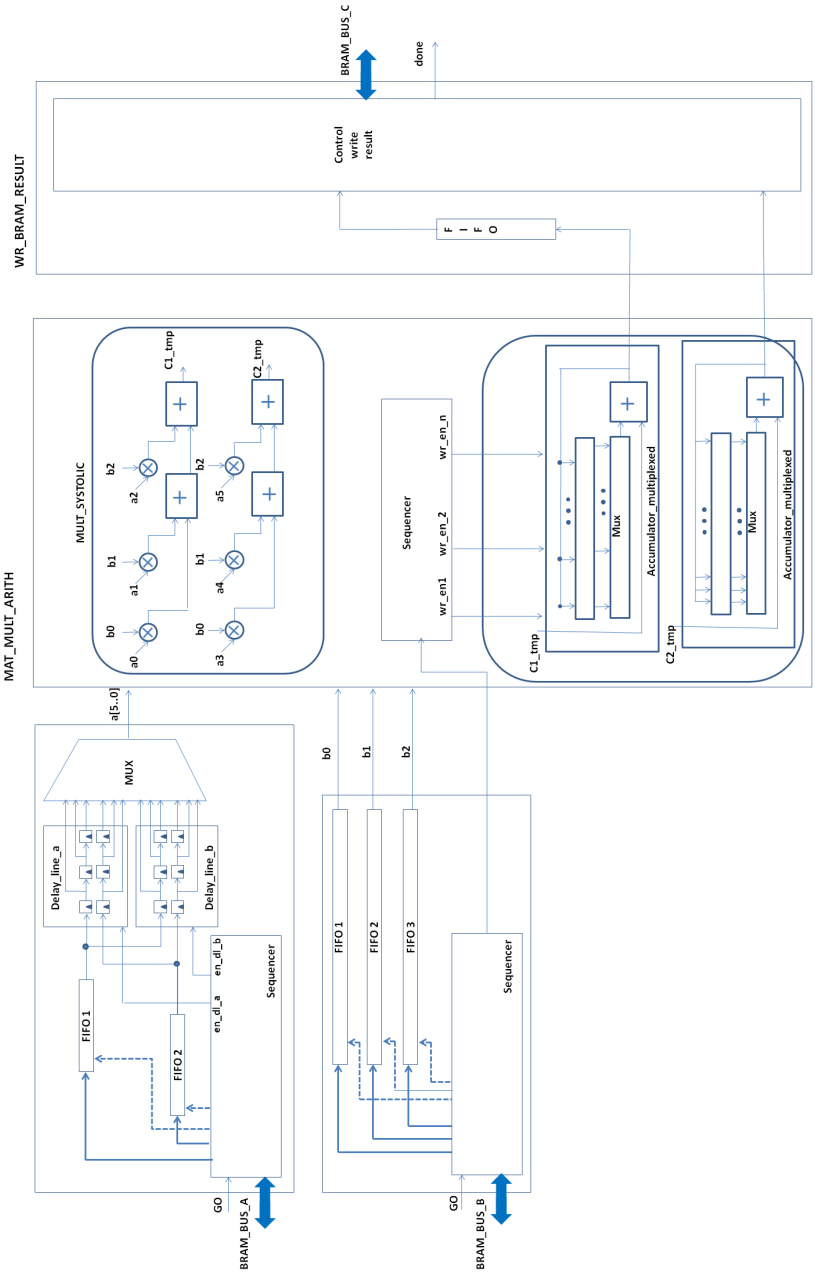

Figure 4.33: Block diagram matrix multiplication module - Floating Point matrix multiplication module

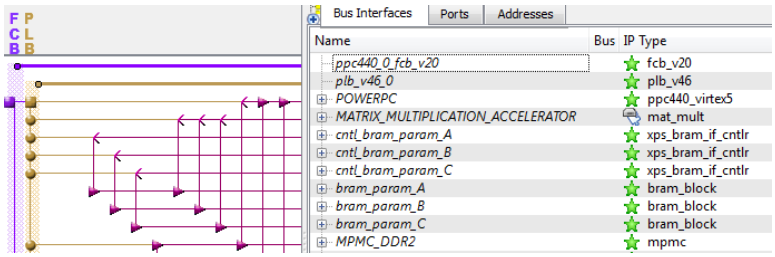

Figure 4.34: Multiplication Accelerator PLB connections -

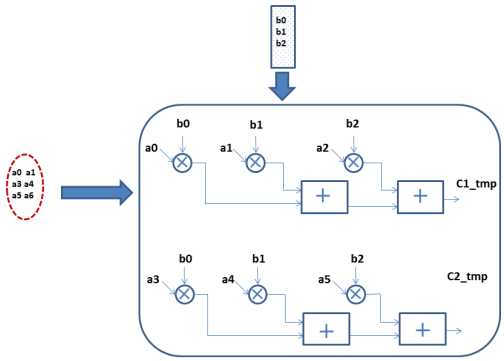

Figure 4.35: Multiplication Systolic Array - A systolic array composed of floating point additioners and multiplicators.

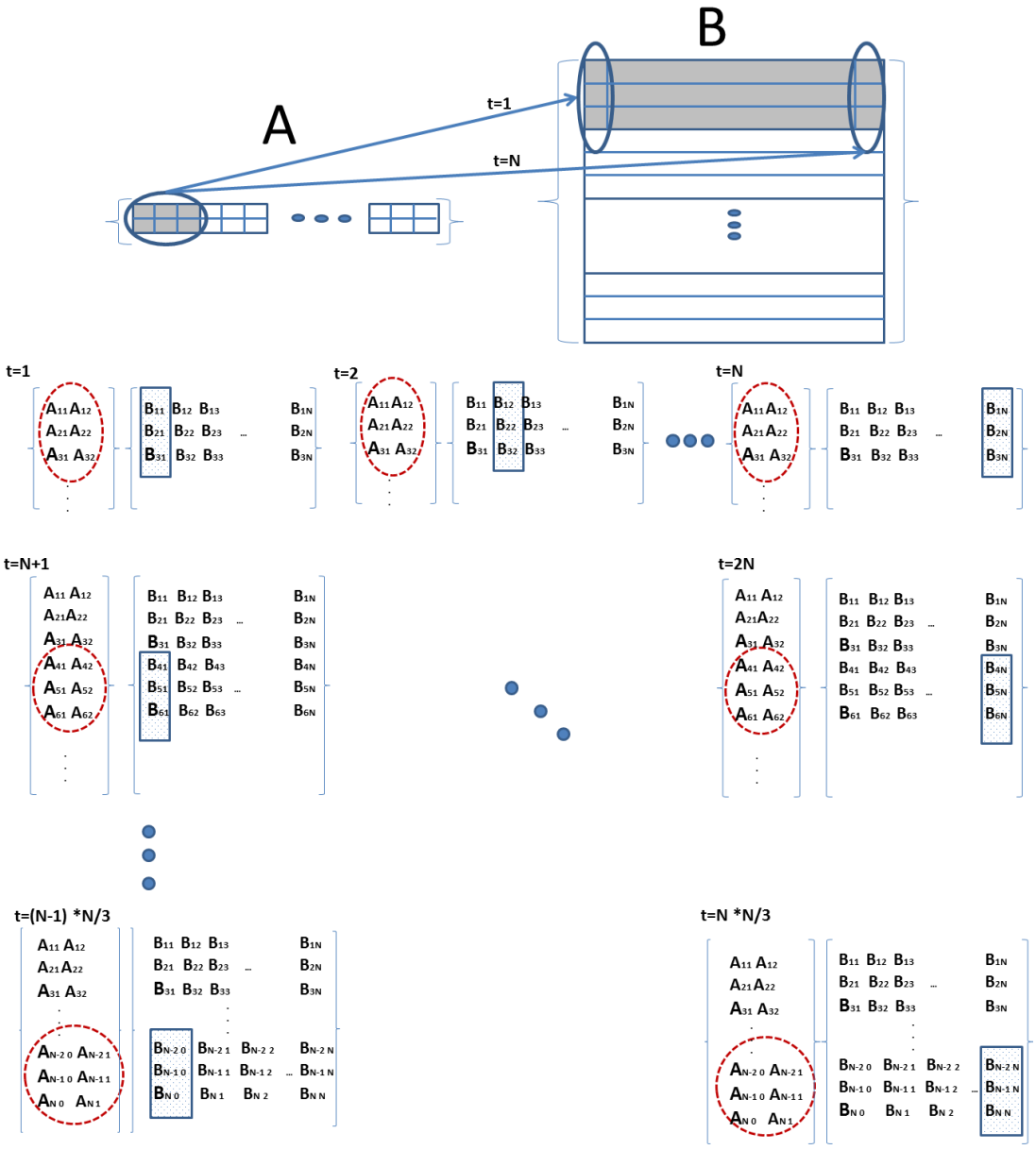

Figure 4.36: Multiplication descomposition of the matrix to feed the Systolic Array -

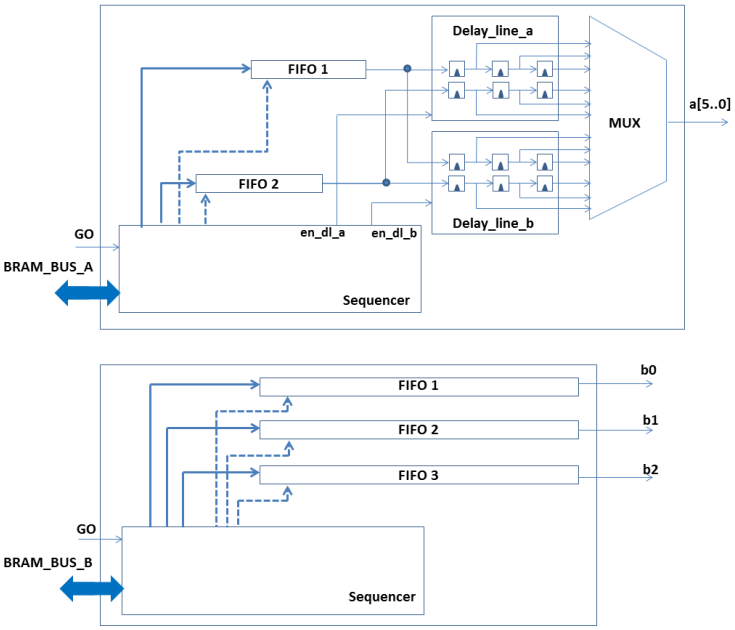

Figure 4.37: Matrix multiplication accelerator FIFO's interface - Modules that prepare the data that will be sent to the systolic array. FIFO's are filled with different rows of the matrix operands. On the top, the module which uploads the matrix A. This module uses two FIFO's, one has the first row of the matrix A and the other one has the second row of the matrix A. On the bottom, the module which uploads the matrix B. This module uses three FIFO's, in the first FIFO will be stored rows number (1,4,7,...), in the second FIFO rows number (2,5,8,...) and in the third one rows number (3,6,9,...) of matrix B. These FIFO's will be popped on each clock cycle allowing to feed continuously the systolic array

The accelerator was completely simulated using Modelsim and System Generator before being connected to the processor. Single precision was used to implement all the operations because we found a good trade-off between the resource utilization and the acquired results.

4.7.3 Harris point detector implementation

The architecture to compute the Harris point detector is divided in three modules: Harris memory unit, Harris Arithmetic Unit and Harris Sort Unit. See figure 4.38.

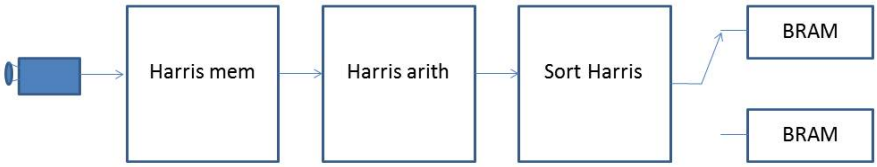

Figure 4.38: Harris detection block diagram - The system is composed of three main blocks: memory unit, arithmetic unit and Sort unit. The Sort Unit detect the best computed score in each subimage. Result is writen in a pair of BRAM's used like ping pong buffers.

The Harris memory unit module uses internal BRAM memories to store five lines that will be swept to manage a window of size five by five. See figure 4.39.

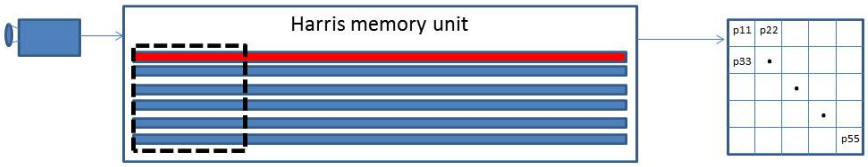

Figure 4.39: Harris memory unit. - In five BRAMS will be stored the last five lines of the images. This BRAMS are swept to generate a window of size 5x5, to feed continuously the Harris arithmetic unit.

This five by five window is sent to the Harris arithmetic unit, which has a pipeline generating a latency of 13 clock events. This module first implements all the gradients on X and Y in parallel using the convolution kernels shown in the equations 2.28 and 2.29. The computed gradients are then multiplied by a Gauss mask in order to smooth the detector. In the last module the terms of the matrix M are computed using the equation 2.30.

To implement the Gauss mask, some approximations are used to avoid the need of floating point multiplications. The gauss mask is implemented with multiplications and division by power of two allowing to save resources, as shown in equation 4.3.

$$\begin{bmatrix} 0.0751 & 0.1238 & 0.0751 \\ 0.1238 & 0.2042 & 0.1238 \\ 0.0751 & 0.1238 & 0.0751 \end{bmatrix} = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 26 & 10 \\ 10 & 16 & 10 \end{bmatrix} /128 \tag{4.3}$$

From the components of M, the trace and determinant are used to compute the response using the equation 2.33. The factor K is also approximated with a multiplication and a shift.

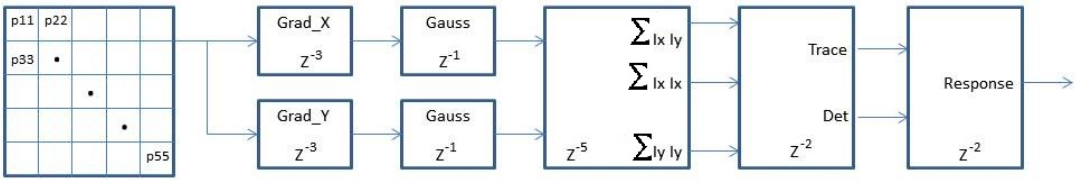

Figure 4.40: Harris Arithmetic Unit - This module implements all the arithmetic operation necessary to compute the Harris response given by the equation 2.33. For each pixel, using the window around the interest pixel, a response will be computed on the fly.

The module Harris Sort implements a Tessellation of the image. The tessellation is implemented on 128×128 windows in the image. Using a power of two size sub-image allows us to reduce resource utilization. The best Harris point is detected in each sub-image. For each camera, we detect 48 Harris points that will be written in a BRAM memory connected to the PLB Bus.

Because we are using power of two size windows for tessellation, in order to detect the first pixel of each sub-image, we need just to analyze the N Less Significant Bits of the coordinates (x,y) of the pixel. With 128x128 size windows, the pixels corresponding to the first data of each sub-image have the bits [6 downto 0] equal to zero.

The Most Significant Bits of the coordinates (x,y) are used to generate the sub-image index, which is used to access the memory where the Harris point information is stored. With 128x128 size windows, and an image resolution of 1024x768, the sub-image_index is computed using the next equation:

$sub\_image\_index <= (coordinate\_y[9downto7]\&b"000") + coordinate\_x[10downto7]$

In our implementation, the Harris point is initialized with the coordinates and the response of the first pixel of each sub-image, then, if a point with a best response is found in each sub-image, it is updated with the information of the new point. See algorithm 1.

Algorithm 1: Harris sort algorithm

```
begin
    // Initialize Harris with values first pixel sub-image
    if first_pixel_subimg = 1 then
        update_harris_point(sub_image_index, response, coordinates)
    else
        // A better response was found for the sub-image N
        if max_response_sub_image < current_response then
            update_harris_point(sub_image_index, response, coordinates)
```

The Harris point detector is coupled to a host processor using the PLB bus. The host processor can access the BRAMS to get the scores and the coordinates (x,y) of all Harris points. A ping-pong buffer scheme (Harris lmk a and Harris lmk B) is used to allow the host processor to access Harris results of image N when computing Harris points of image $N + 1$. See figure 4.41.

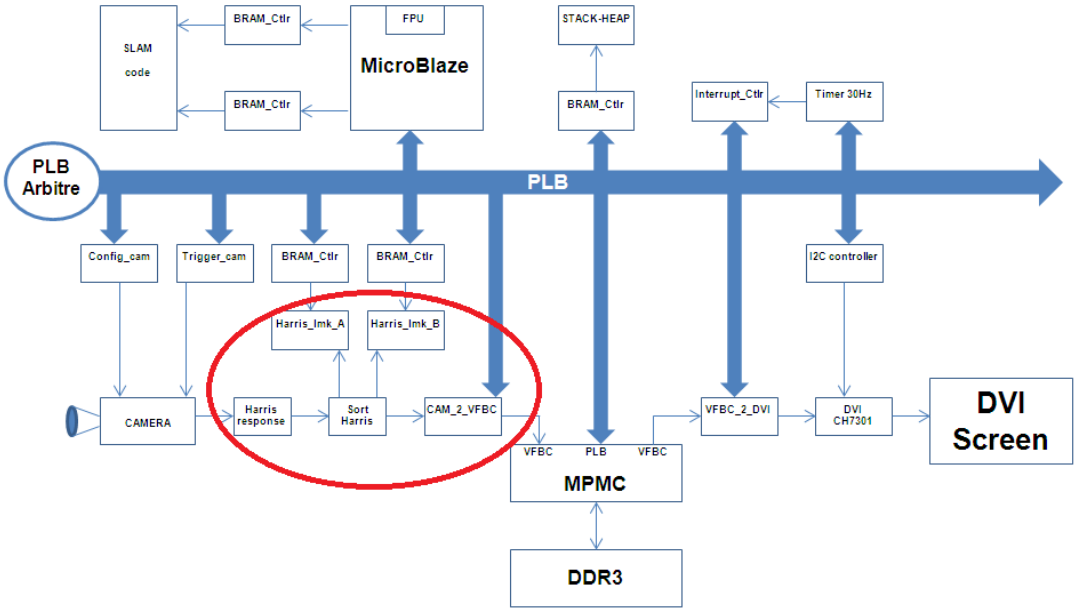

Figure 4.41: Harris unit integrated to the PLB system - In the ellipse on the bottom left, we highlight the Harris pipeline. Images coming from the camera are processed on the fly. The module sort Harris, organize the response to choose the best response in each sub-image. A ping pong schema is used to send the Harris information to the Host processor.

Harris point detection is performed on fixed point data representation to save on logic.

4.7.4 Active search implementation

This module implements on the fly a correlation between the known signature of a landmark (a block around the first observation of the landmark) and an image flow. The Active Search is done in an uncertainty ellipse that is approximated with a rectangle. See figure 4.42.

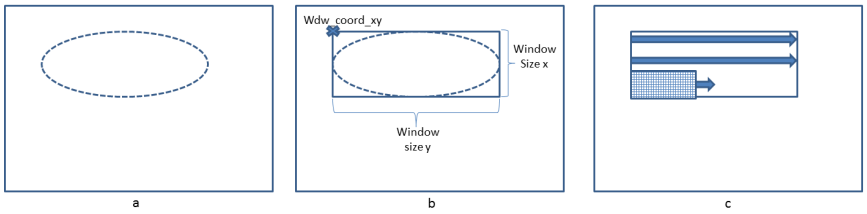

Figure 4.42: Active Search - On the left (a), the uncertainty ellipse. On the middle (b), the ellipse is approximated with a rectangle. On the right (c), the rectangle is swept to implement the correlations with the signature of the interest Landmark.

The Active Search module is divided in four main modules : Image Flow Window Management, Window Signature Management, Correlator and the Sort correlation module. Three different correlation scores were coded to compare the results: Sum Absolute of Differences (SAD), Cros-Correlation (CC) and Binary Robust Independent Elementary Features (BRIEF). See figure 4.43.

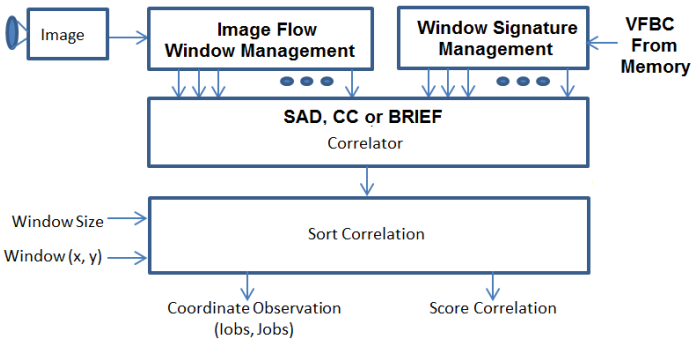

Figure 4.43: Block diagram of the Active Search module. - The output of this module are the coordinates (Iobs, Jobs) and the score of correlation

The Active search module receives from the processor the parameters of a predicted zone where the signature of a landmark should be searched. The parameters received from the processor are the Window research size (that corresponds to the ellipse of uncertainty approximated by a rectangular region) and the Window research coordinate. This module search the coordinate (I_{obs},J_{obs}) of the best candidate pixel according to the correlation of the signatures. A score of correlation will be asigned to the best correlated pixel, then the processor implementing the SLAM algorithm according to this score will decide if it is a good candidate.

We chose to use a window size of (16×8), the dimension of this signature is larger than the one proposed by [49], who found that a 11×11 window is enough to track a landmark. Using a non-square window allows us to save memory resources. Indeed a signature with more lines would require to store more lines of the image for the correlation, thus increasing the memory need. The use of a non-square window allows us to accurately track the landmark without additional cost in terms of memory. If we would use a window size of 16×16, we would need 8 additional memories.

The landmark signature is read from the SDRAM memory through a VFBC interface and stored in a set of registers. This way, all the elements of the signature are available in parallel for the Active Search module in a set of registers. Then, when a new pixel arrives from the camera a new correlation can be done.

In the active search module we should look for a pixel in a specified zone using the correlation score. In our implementation, the hardware is always computing the correlations but scores will be taken into account only in the chosen zone by the processor. Once we have swept all the chosen zone, a new active search can begin, it allows the reutilization of the same hardware.

In the proposed architecture, the host processor can configure the active search of multiple landmarks per frame, with the only requirement that the search region lines do not overlap. This constraint can easily be solved by re-instantiating the active search module to match the max number of tracked landmarks, at the cost of more logic and memory.

For SAD, on each clock event we implement 16×8 subtractions and the accumulation of this subtraction is implemented in a pipeline.

For the CC, 16×8 multiplications are implemented in parallel, and the results are accumulated using adders connected in a pipeline fashion way.

For BRIEF, 128 pseudo-random comparisons are implemented in parallel between couples of pixels of the descriptor and the target patch. The 128 comparator module is generated with HDL using a random pattern generated offline using Matlab. Using Matlab, a text file is generated with the HDL syntax necessary to be included in the code. Figure 4.44 shows a block diagram of the BRIEF correlator.

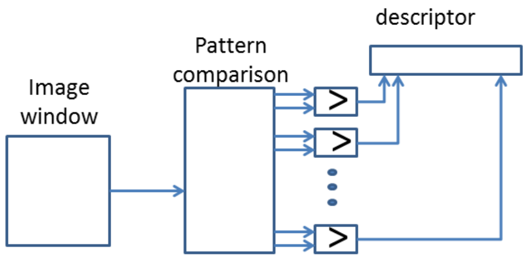

Figure 4.44: Block diagram of the BRIEF module. - 128 comparison are implemented in parallel between different pixels of the image window. According to the comparison a descriptor is created.

4.8 Contour detection implementation

This module is coupled to the Harris point detector and reuses the resources of the Harris module (uses the same window managed for Harris). The gradients are computed from the

Harris memory unit contents.

To implement contour detection, Sobel detector is implemented using the approximation shown in the equation 4.4.

$$\sqrt{G_1^2 + G_2^2} \approx \mid G_1 + G_2 \mid \tag{4.4}$$

The kernel convolution is implemented using the module shown in figure 4.45 connected in two different ways to the Harris Memory Unit. According to the input connections of this module, the Gradient on X or Y can be computed. One instance of this module is necessary to compute each one of the gradients.

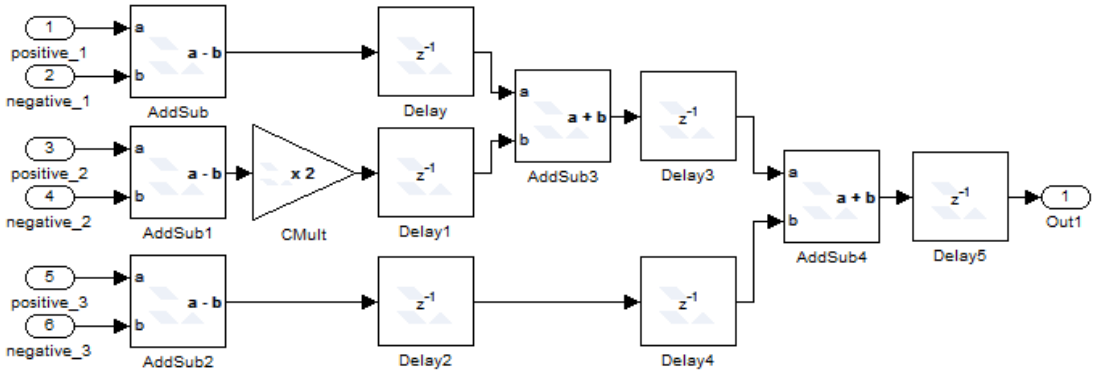

Figure 4.45: Sobel basic element architecture -

The module is implemented using System Generator. The architecture uses a pipeline that generates a latency of three clocks. Two instances of this module are instantiated in the design to compute on the fly the Gradients on X and Y simultaneously.

To store the Contour information, no extra memory resource is needed. For the color (RGB) image we have 24 bits, and with extra 8 bits used to encode the contours, we have a total of 32 bits, equivalent to the memory width used in the system.

4.9 Conclusion

In this chapter, we described the different architectures implemented in this thesis. All of the architectures presented here were developed to be used in some research project involved during the develpment of this thesis.

In the next chapter, we present results, projects and platforms used in the development of this thesis.

5

Results and Projects

In this chapter, first, we present the results obtained for the different algorithms: Distortion correction, CLAHE, Multispectral Vision, Obstacle detection, Multi-camera system, Stereo-vision, Harris point detector, and Contours detection.. the complete SLAM algorithm has not been integrated on FPGAs, because this algorithm has numerical computations that cannot be easily implemented on FPGAs. So it is required to distribute the complete algorithm on a classical processor and on FPGA modules; this work is on the way.

Then, we present tables describing the resource utilization for each one of the generated modules. Here, the timing characteristics of the modules are commented.

Next, we present the development done for the SART project. We participate to the design of a custom board developed to implement the SART algorithms; this board characteristics are described; its test and evaluation on some proposed architectures is on the way.

In the last part of this chapter, we present the platforms used to test all of developed algorithms and architectures.

5.1 Distortion correction results

Figure 5.1 shows the input and output images of the distortion correction module. In the distorted image (on the left) the white lines are not aligned with the chessboard, revealing the distortion of the camera lenses. Then, after applying the distortion correction algorithm, the output image is a transformed version of the input image without distortion. This algorithm was implemented at 100 Hz with a VGA resolution.

Figure 5.2 shows a comparison between the output of the Distortion correction executed in Matlab and executed on FPGAs.

For the Distortion Correction, a bilinear interpolation was done to improve the quality of the output image. If this interpolation is not executed, there are artifacts especially in the images contours. These artifacts can affect the results of the post-processing algorithms. See figure 5.3.

Our architecture can be used to implement almost any transformation of the image; figure 5.4 shows a sinusoidal transformation applied to acquired images. This kind of transformation has been implemented only in order to test the distortion correction module: only the tables are adapted with respect to the transform to be applied on images; the architecture, starting from a pixel in the transformed images to the corresponding region in the original ones, is the same.

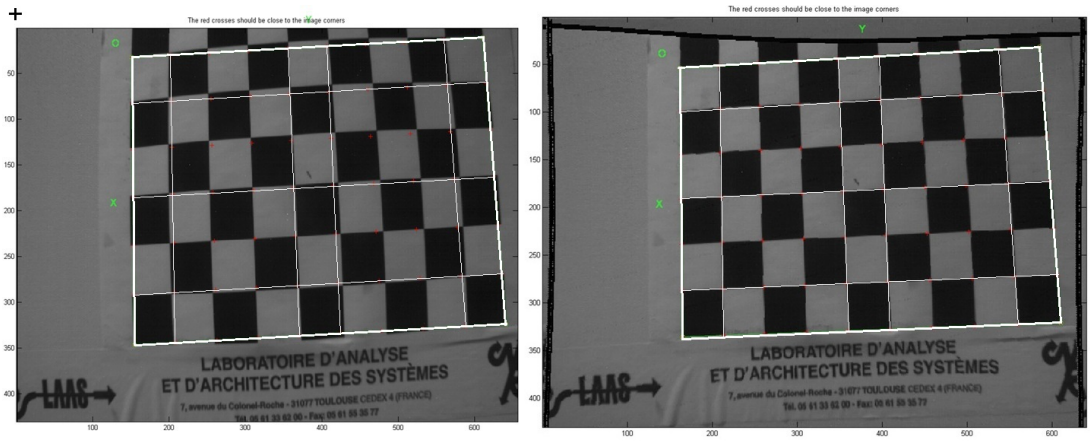

Figure 5.1: Distortion correction result - On the left, the input image. On the right, the undistorted image on the FPGA.

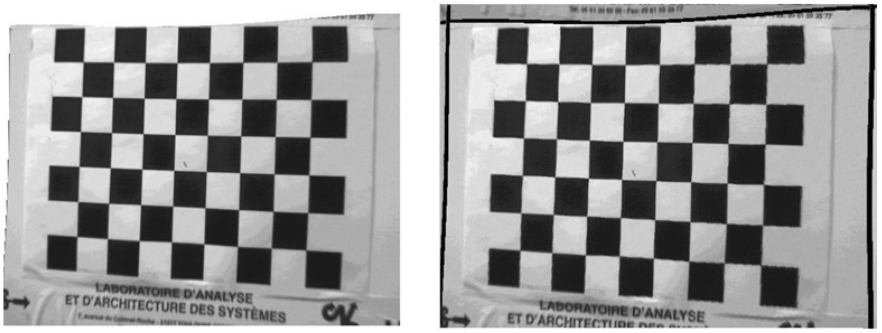

Figure 5.2: Matlab comparison distortion images - On the left, the undistorted image using Matlab. On the right, it is undistorted using the FPGA

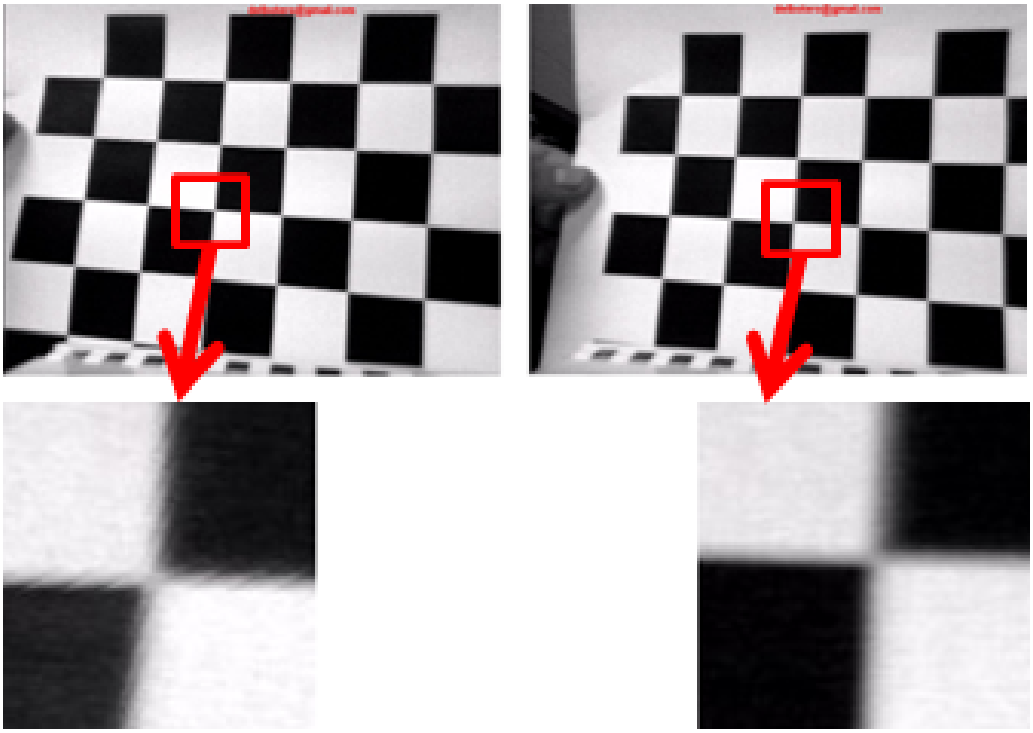

Figure 5.3: Comparison Distortion correction with and without Interpolation - Left, undistorted image without interpolation. Right, undistorted image with interpolation.

Figure 5.4: Sinusoidal transformation - The input image is transformed using a table encoding a sinusoidal transformation. This way, we tested the correct operation of the module.

5.2 CLAHE results

Figure 5.5 shows the histogram and the CDF before and after the image truncation to 10-bits using the priority decoder described in section 4.2.2.1. Like seen in the figure, the CDF properties are not affected significantly after reducing the dynamic of the image.

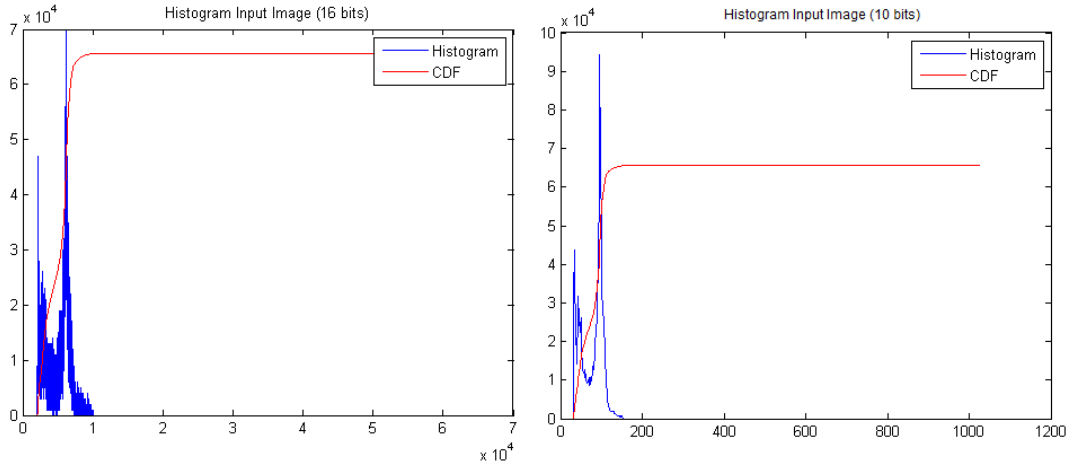

Figure 5.5: Histogram and CDF after and before truncate bits - On the left, $2^{16} bins$ are used to compute the histogram. In the middle, the histogram is built using 2^{10} bins.

Figure 5.6 shows the System Generator simulation of the different steps of the CLAHE algorithm. As explained in section 4.2.2.2, the transformation LUTs are built in five steps.

Figure 5.7 shows the input (a), intermediary (b), and output (c) images of the CLAHE algorithm.

The Clip limit is changed using a soft processor. A different contrast enhancement is implemented with different values of Clip limit. In our approach Distortion correction is implemented after CLAHE. See figure 5.8

5.3 Perspective transformation results

We showed the implementation of homographies for IPM using a SDRAM memory. To achieve real time operation (30 FPS with a VGA resolution) a cache memory was added to the system. Architecture presented in subsection 4.3.2 using a cache memory achieves an improvement of at least 30x in terms of maximum frequency with respect to the architecture presented in subsection 4.3.1.

The homography transformation module was initially tested using System Generator [85] of Xilinx. The VHDL module was imported like a black box and simulated to estimate the loss inherent to data quantization. Figure 5.9 shows the black box corresponding to the VHDL model, which is compared with a Simulink model. For this, both the Simulink and the VHDL model are fed simultaneously with the same stimulus, then the output from the VHDL module is subtracted with the output of the Simulink model. See figure 5.10

The Simulink model includes unitary delays to synchronize the Simulink model with the pipeline of the VHDL model, this way the output from both models can be directly compared

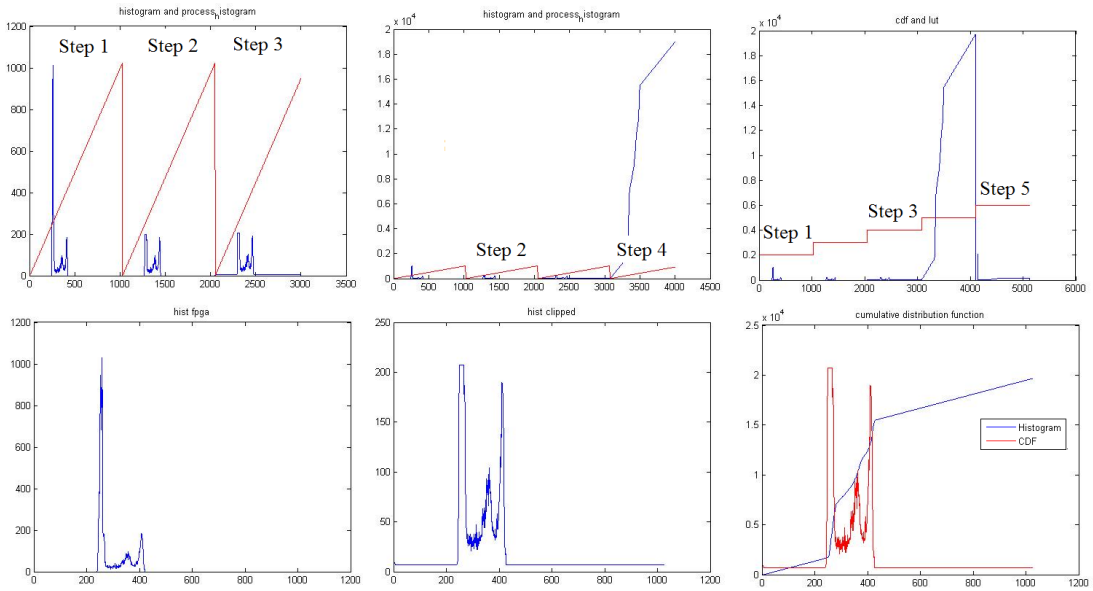

Figure 5.6: CLAHE System generator simulation - On the top, the images illustrates the five steps done to compute the LUT. On the bottom, the left image shows the original histogram, the middle image shows the clipped histogram, and the right image shows the histogram and the CDF.

Figure 5.7: CLAHE intermediary images - On the left, the input image. In the middle, the intermediary image, here each sub-image is processed independently. On the right, the output image after implementing interpolations.

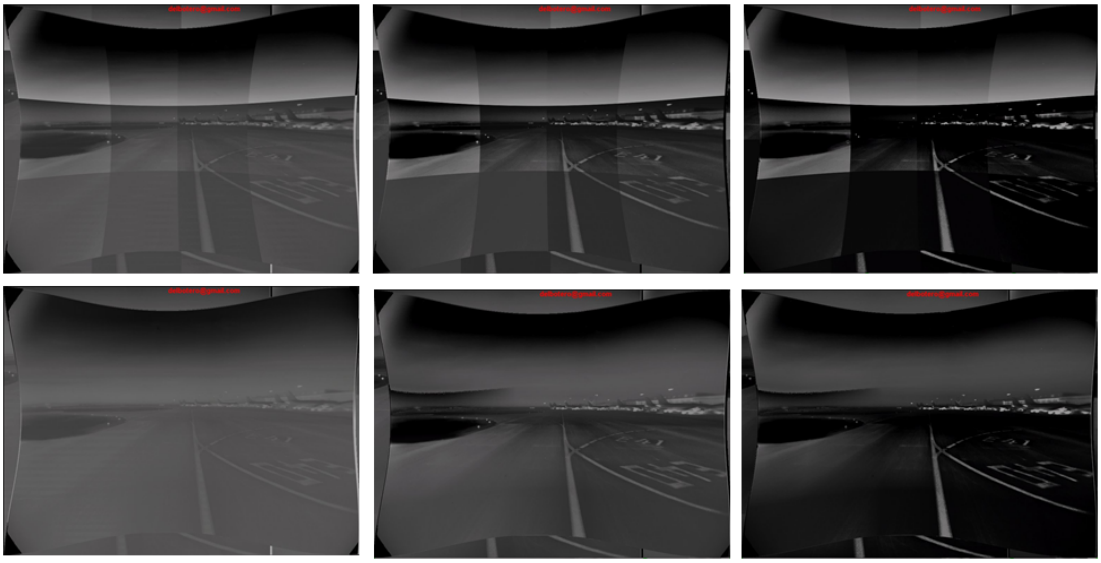

Figure 5.8: CLAHE output images with different Clip Limits - On the top, the images before interpolations. On the bottom, the images after the interpolation. From left to right, the Clip Limit is increased.

and plotted.

When implementing the homography, non-integer coordinates may be obtained. In our implementation, we used the integer part of the coordinate to replace the pixel and the fractional part was discarded. The ideal procedure would be to implement an interpolation between the four neighbor pixels. Because no interpolation was used in our implementation, in the discontinuities of the transformed images can appear artifacts. See figure 5.11.

Pipelining was necessary to implement the homography. For this module, all the operations are done at 100 MHz. In order to illustrate the necessity of the pipeline, figure 5.12 shows a homography computed using only combinatory logic.

5.4 Multispectral vision results

Multispectral fusion was simulated using the architecture described in the section 4.4. The MIG was used to generate the drivers of the QDR2 memories. An HDL model provided by the QDR2 memory constructor was used. In the simulation, small resolution images were used because the simulation takes a long time to be executed.

Figure 5.13 shows the resulting images after implementing a fusion between the Visible and Infrared images using Matlab. To implement the fusion, the images were first aligned using a homography, and then different fusions were implemented using different color spaces: HSV, Y'CbCr, and RGB. The implemented homography allows to align the ground planes of both images, we have chosen this plane, because according to the airport context, it is the plane where there are more usefull information for the pilot.

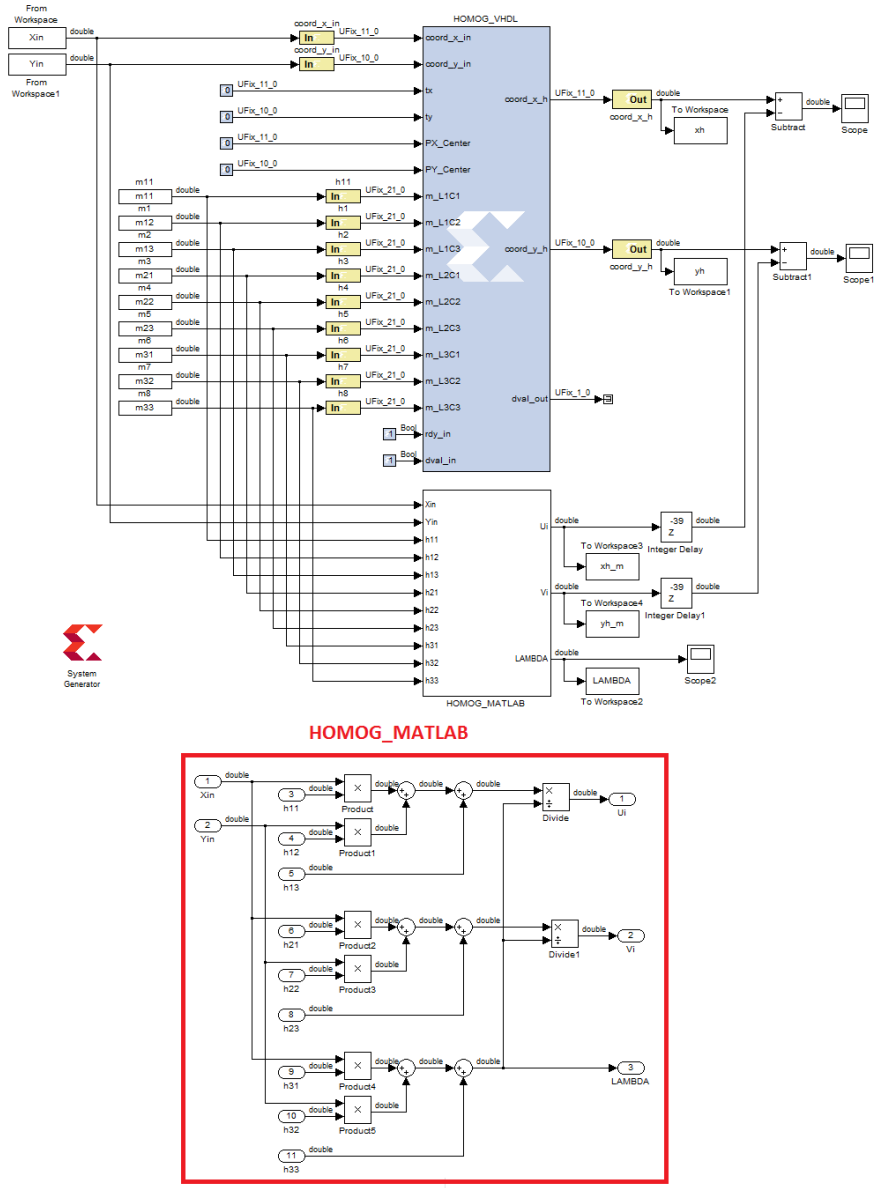

Figure 5.9: Simulink homography verification diagram - Verification of the arithmetic unit of the Homography module using System generator of Xilinx. Output from a black box with the VHDL is compared with a Simulink model

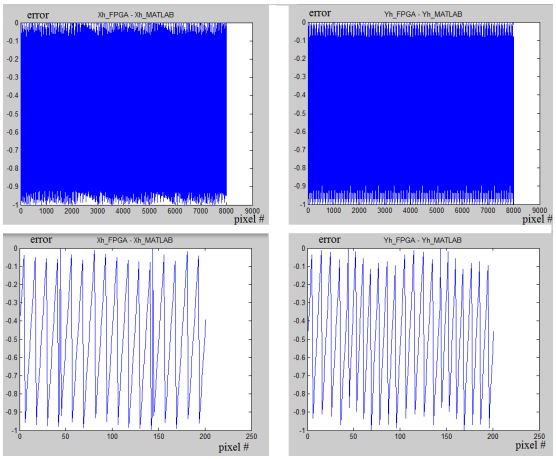

Figure 5.10: Graphic of quantization error of the homography module - The difference between the VHDL and the Simulink model response corresponds to the fractional part discarded in the FPGA implementation. For this test the parameters of the homography matrix were constant.

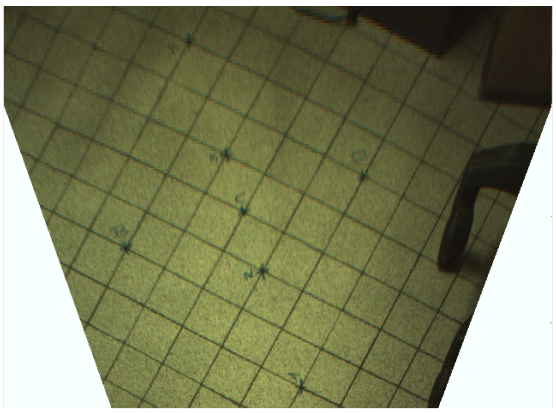

Figure 5.11: Homography effects of interpolation absence. - In the top part of the figure, the lines are not correctly reconstructed; using an interpolation a better result may be obtained.

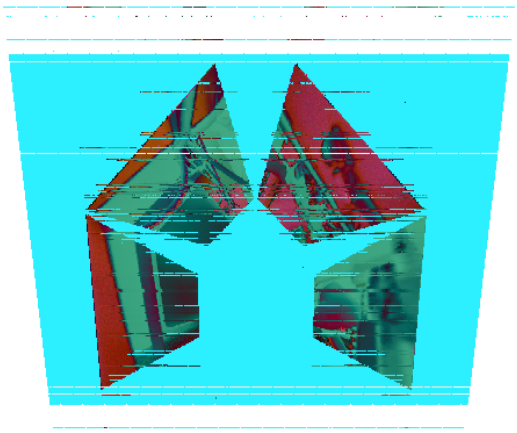

Figure 5.12: Homography necessity of pipelining. - This image corresponds to the occupancy grid generated in the COMMROB project, here we implemented all the arithmetic operations corresponding to the homography in one clock cycle (without pipeline). Because of the setup and hold violations due to the long paths, the output image is noisy.

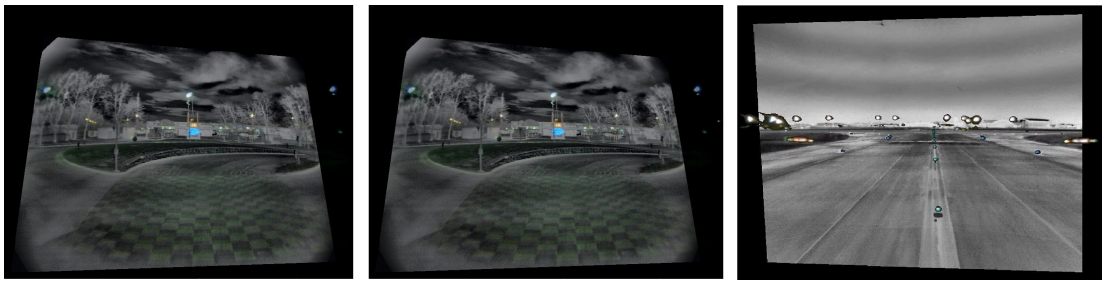

Figure 5.13: Fusion IR and VIS in different color spaces - On the left, Fusion using RGB color space. In the middle, Fusion using HSV color space. On the right, Fusion using YCbCr color space

5.5 Obstacle detection

Two different algorithms were implemented and tested for obstacle detection.

The first algorithm used the IPM technique, both the SIPM and MIPM algorithms were evaluated.

The second algorithm used the modules developed by M.Ibarra Manzano [86][12]. Our contribution consists in integrating all those modules in a multi-cameras system.

5.5.1 IPM results

Figure 5.14 shows the output of the MIPM algorithm. For this test, the odometry was read from the scout robot.

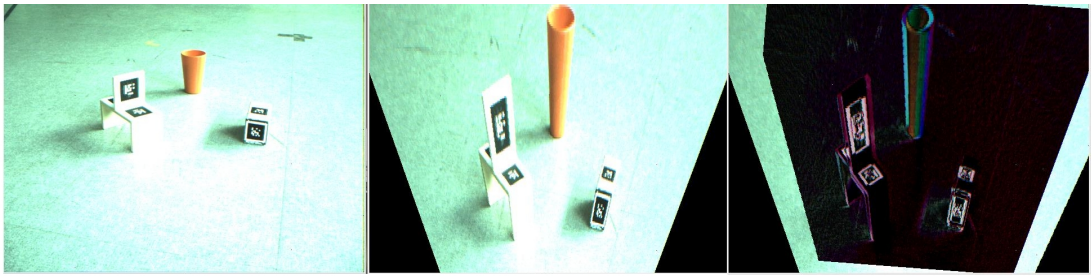

Figure 5.14: IPM output image - On the left, the image taken in t=N. In the middle, the homography of the image taken in t=N+dn. On the right, the subtraction of In-H[In+dn]

To compute the Homography matrix needed for SIPM, the inner corners of the chessboard undistorted images acquired with the stereo pair are detected. See figure 5.15.

Using the correspondences between the corners found in the chessboard images for the left and right camera, the equation 2.27 is implemented. The solution of the system of equations is implemented offline at the calibration time. Then, with the H matrix, a table of addresses is transformed to encode the results. See figure 5.16.

The SIPM was tested using a custom board with cyclone 2 FPGAs and a SRAM memory; we arrived to implement SIPM at 100 FPS with VGA image resolution. In our custom board, the tables were transferred using USB2.0 and stored in non-volatile memories. In the boot sequence, the distortion-homography tables were transferred from the non-volatile memory to the RAM memory.

Figure 5.18 shows the output of the SIPM algorithm. In the figures, all of the objects that are not in the ground plane are transformed like if they were in this plane, so their geometry is distorted. This change of geometry generates an anomaly in the subtraction image. In the left camera images, only a distortion correction is applied; while a distortion correction and a homography are applied to the right camera images.

IPM is sensitive to the reflections of lights. In an outdoor application, several reflections can appear especially when there are raining conditions. These reflections represent a big challenge that should be processed with extra processing. See figure 5.18.

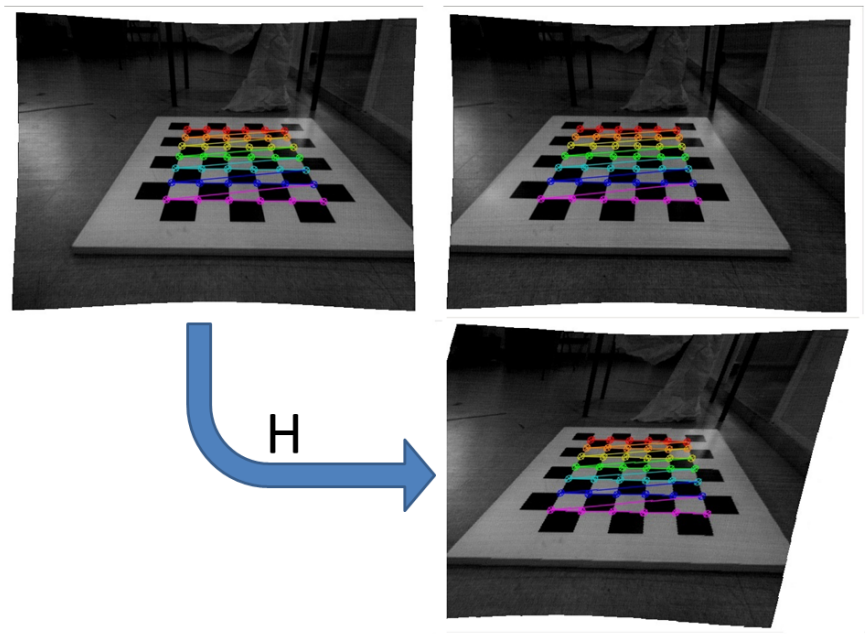

Figure 5.15: SIPM calibration procedure - Top-left: Chessboard left image undistorted. Top-right: Chessboard right image undistorted. Bottom-right: Homography of chessboard left image undistorted.

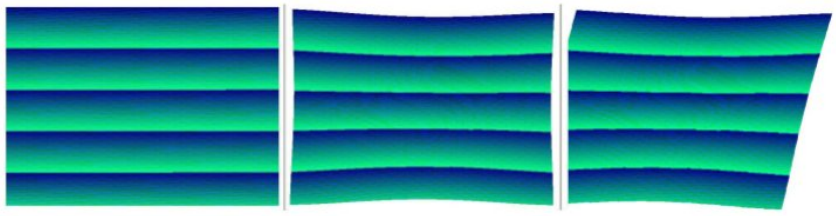

Figure 5.16: SIPM tables encode homography and distortion correction simultaneously. - On the left, the input table which represents a null transformation. In the middle, the table which encodes correction distortion. On the right, the table which encodes distortion correction and homography simultaneously.

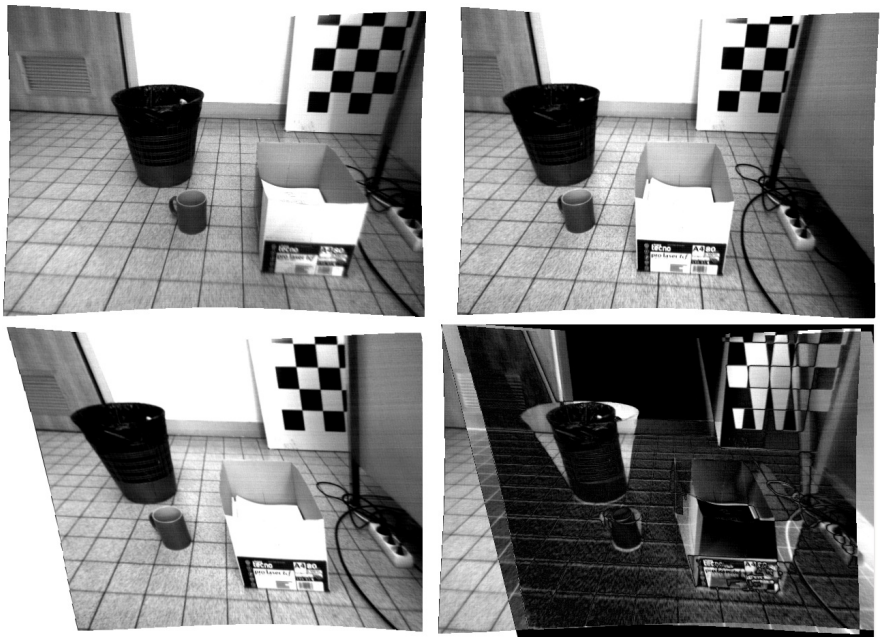

Figure 5.17: SIPM results - Top-left: Left image undistorted. Top-right: Right Image undistorted. Bottom-left: Homography of right image undistorted. Bottom right: $I_{right} - H[I_{left}]$

Figure 5.18: Shadow effects in SIPM - Top-left: Left image undistorted. Top-right: Homography of right image undistorted. Bottom-left: Abs($I_{right} - H[I_{left}]$). Bottom right: Output SIPM, threshold of Abs($I_{right} - H[I_{left}]$)

5.5.2 Textures Obstacle detection - Camera belt

The detection of obstacles based on texture was done using the algorithm presented in [86]; we integrated the corresponding modules on a bigger FPGA in order to support a multi-cameras system. We implemented the fusion algorithm which generates an occupation grid from the four camera sources.

The fusion was done using a table which encodes the different transformation to be done to each image. The table encodes the origin address of the pixel to be used to build each one of the pixels. Because we needed random access to build the occupation grid, the processed images were stored in a SRAM, which allowed us to implement the fusion in real time.

Figure 5.19 shows the disposition of the cameras around the robot. Here four cameras TRd5m [87] of Terasic were connected through a parallel connection to a high speed port of the Stratix 3 development kit [88]. The parallel connection limits the size of the cable used to connect the cameras to the FPGA.

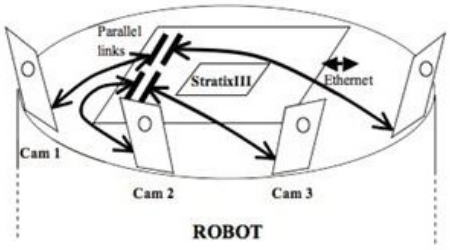

Figure 5.19: Camera-belt disposition - On the left, the Terasic cameras. In the middle, the Startix development kit. On the right, the disposition of the camera belt.

Figure 5.20 shows the Fusion homography module. This module implements the image processing of four Bayer images and stores the processed images in a SRAM. Then, a module reads a table from a DDR2 memory containing the addresses of the pixels that should be used to create the output image. Using the addresses, the fusion image is built and stored in the DDR2 memory. All these processes are done at 30 FPS, however, we displayed the output image at only 2 FPS in a PC using UDP/IP transfers.

The odometry is used to build the occupation grid with different frame references. In a future improvement of the system, the odometry can be used to build the occupation grid of a predefined exploration area progressively by the implementation of the fusion between the local stored occupation grid and the current occupation grid.

Figure 5.21 shows the output after combining the four camera processing. On the bottom, the occupation grid was rotated and translated using the odometry read from the robot. For this, we used the module described in the section 4.3 to implement the geometric transformation.

This geometric transformation corresponds to a rotation and translation because we assumed that the ground is flat.

Figure 5.22 shows a micro-camera developed at LAAS, which would replace the Terasic cameras used on the camera-belt. This camera has a serial LVDS interface to communicate to the FPGA, instead of a parallel interface. This way, the camera can be placed at longer distances from the FPGA.

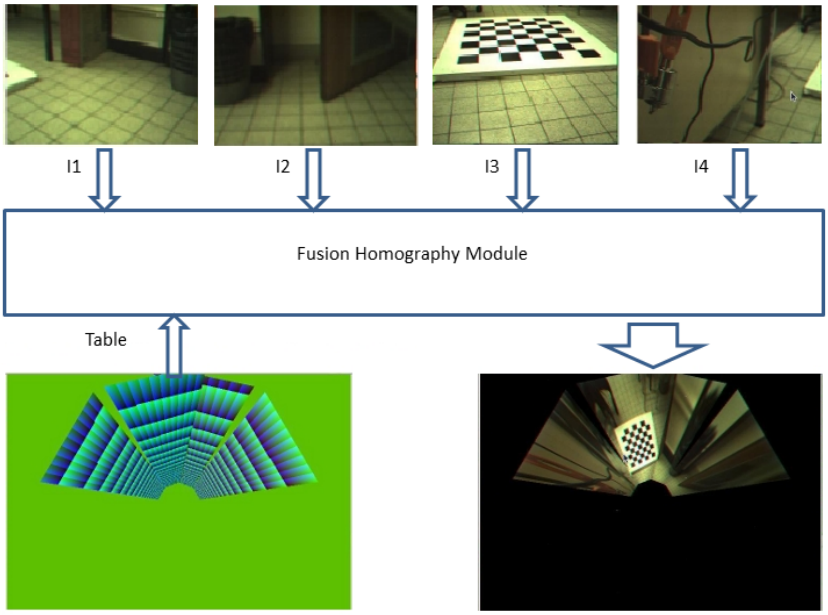

Figure 5.20: COMMROB Multicamera fusion image - Information from four cameras are acquired and a fusion image is built using a table. The module Fusion Homography module generates the Bird's eye view at 30 fps.

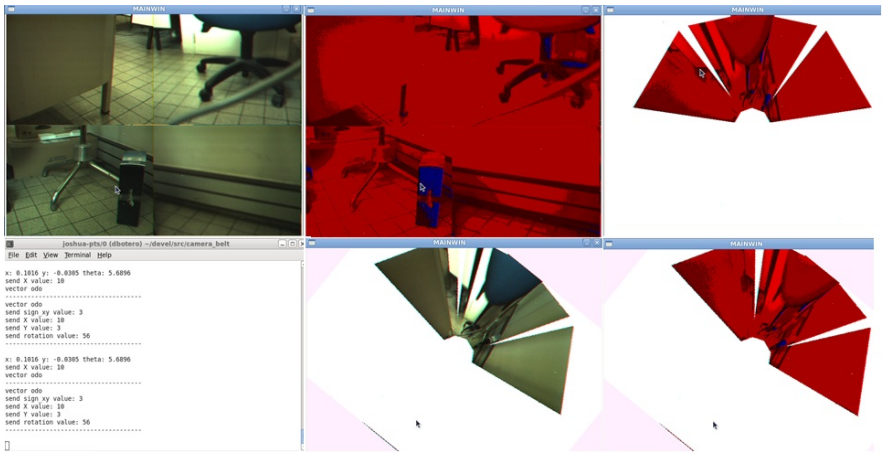

Figure 5.21: Occupancy grid - Using the four cameras, the algorithm developed by [86] was applied to each one of the four cameras. Then using a table, an occupancy grid was generated with the information of the obstacles. The occupancy grid was rotated and translated according to the information acquired from the odometry of the robot.

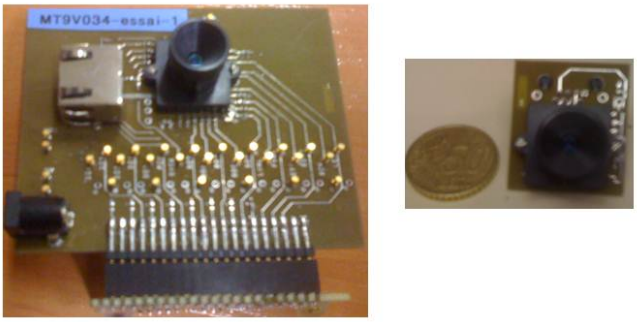

Figure 5.22: Micro camera developed at LAAS for the camera Belt - For a future upgrade, we would like to replace the Terasic cameras with smaller cameras with a LVDS connection allowing for longer separation distances between the cameras

The integration of the system was done using SOPC builder. All of the components communicated together using Avalon interfaces. See figure 5.23.

5.6 Stereo-vision results

The Distortion correction module was also used to build disparity maps using stereo-vision algoithms. For this, a platform developed by Delta Technologies was used. The platform is based on many boards connected through a standard bus. Each board has a Cyclone2 FPGA, a SRAM memory and a Flash memory. An extra board with USB2.0 capability is also included in the system and is used to write the tables of distortion correction adn alignment.

In the first board, the distortion correction of the left camera was implemented. In the second board, the correction distortion of the other camera was implemented. In the third board, the two undistorted and aligned images were processed to generate the disparity map using a Census correlation.

The distortion correction algorithm was implemented at 100 FPS using the precomputed tables stored in the non-volatile memory. This algorithm guarantees no distortion and the epipolar restriction which facilitates the implementation of the disparity map.

Figure 5.24 shows the output of the stereo algorithm after applying the distortion correction to the cameras.

For the implementation of this algorithm, no embedded processor was used. The target FPGAs were small devices, thus increasing the complexity design efforts. All the memory and communication management was done usign VHDL hand coded modules.

5.7 Harris point detector results

Harris point detection is performed on fixed point data representation to save on logic. A System Generator simulation with comparison to floating point values computed by Matlab, showed the approximation error to be small. Figure 5.25 shows the simulated System Generator model, and the figure 5.26 shows the quantization error.

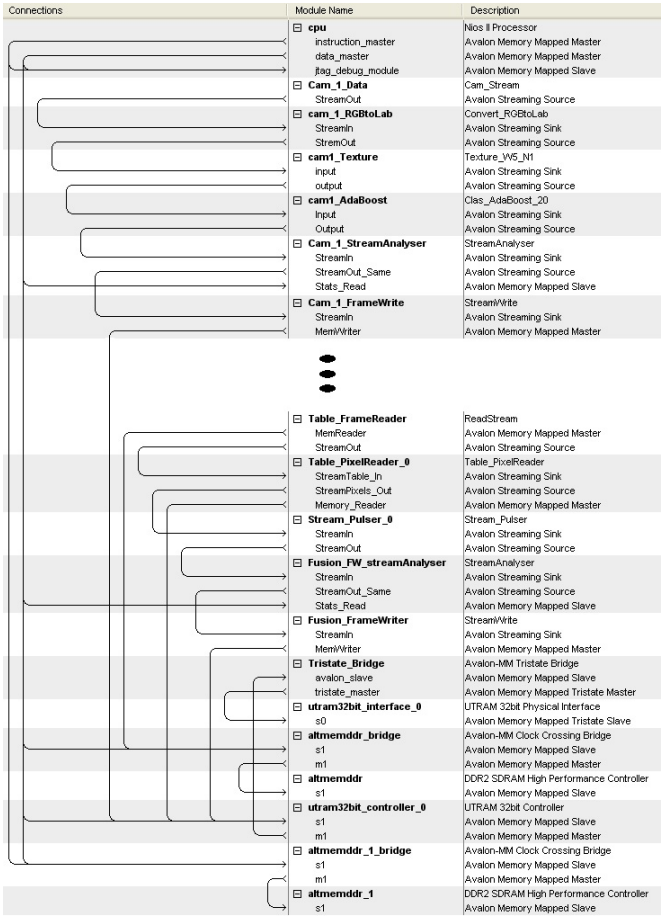

Figure 5.23: Cammera Belt architecture in SOPC Builder - For simplicity of the diagram, just one camera is represented and the Ethernet components are not shown.

Figure 5.24: Disparity map after distortion correction - On the top, the input image. On the bottom-left, the disparity map generated at 100 fps. On the bottom-right, the disparity map after a post-processing to eliminate noise.

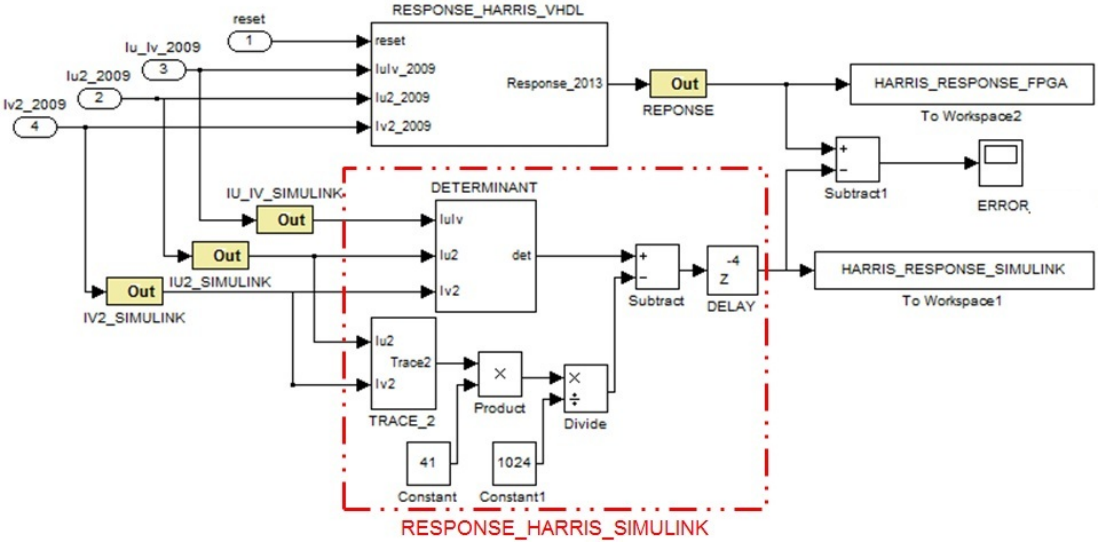

Figure 5.25: Harris module in system generator and the Simulink model - On the top, a black box containing the VHDL module of the Harris Arithmetic unit. On the bottom, the Simulink model; the Simulink model includes delay blocks to take into account the latency generated by the pipeline of the VHDL model. Both models are stimulated with the same input. Then, the output responses from the two models are compared to verify the correct operation of the module.

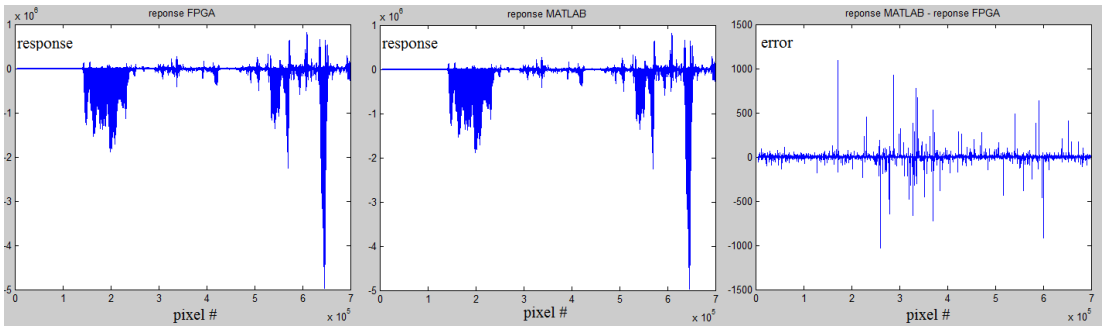

Figure 5.26: Response quantization error -

Using Matlab, the detected Harris points are displayed. This way we evaluated the correct functionality of this module. See figure 5.27.

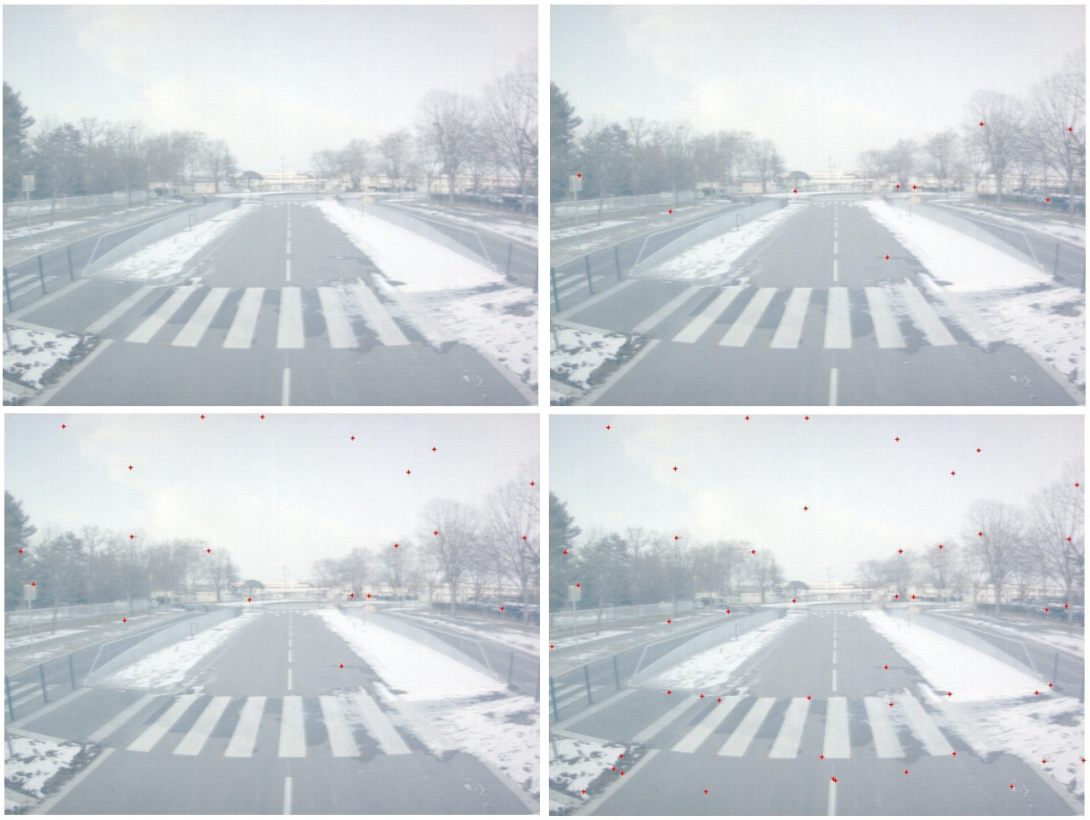

Figure 5.27: Harris point detector output in System Generator - A threshold is used to display only the points with the best response. When the threshold is reduced, more points are detected.

To validate the algorithm, a soft processor was used to superpose markers to the output image to the detected points. The soft processor writes square markers over the output buffer image which is displayed using a DVI interface. Because the processor is not able to design all the markers at 30Hz, they are continuously erased by the new arriving images, however, using this method we are able to display the Harris points for demonstrating objectives. See figure 5.28.

The inconvenient of this display method is that the points will seem to be blinking because they are constantly erased by the images arriving from the camera, however, this does not affect our final application, because the final objective of our Harris detector module is to feed the SLAM algorithm, thus the points will be read by a Host processor at 30Hz.

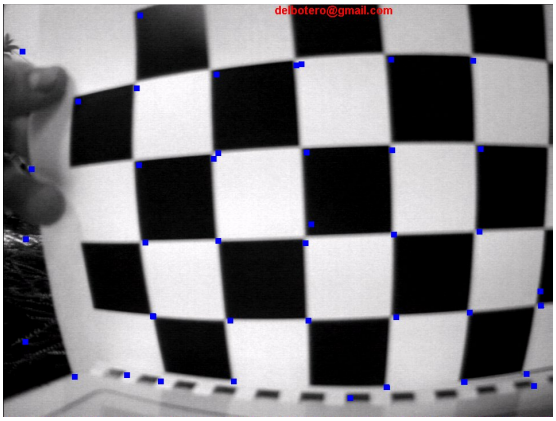

Figure 5.28: Harris point detector output - The corners are highlighted with a square.

5.8 Contour detection results

Gradients information is computed on the fly and stored in memory. These results will be used in the SART project by the Host processor to implement a matching with a database containing a map of the airport represented with contours. The data base is furnished by THALES, the matching algorithm is being studied in another thesis developed at LAAS, also in the context of the SART project.

Contours were detected using the Sobel detector, the algorithm is applied to the acquired and processed images, and is displayed in the DVI screen for validation purposes. In the figure 5.29, the gradients of the visible and the infrared undistorted images are displayed simultaneously.

Figure 5.29: Superposition input images and gradients. - Gradient is displayed simultaneously with the acquired image

In the figure 5.30, the gradients are segmented using a threshold and superposed to the input image.

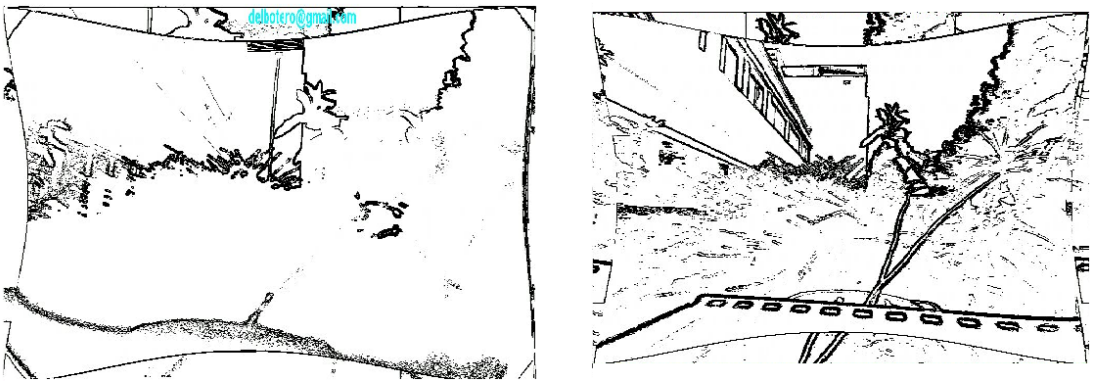

Figure 5.30: Gradient applied to undistorted images. - A correction distortion and CLAHE algorithm was implemented to the images before implement the CLAHE.

Table 5.1: Resources required by our accelerator.

	Slice Registers	Slice Luts	BRAM	DSP48
CLAHE	1280	4308	24	60
Distortion Correction	404	425	32	4
Active Search	603	557	9	0
Harris	3045	2269	6	44
Whole System	26375	22858	240	125
Available in device	301440	150720	416	768

5.9 Timing and Resource utilization

Table 5.1 shows the resources required for each one of the accelerator modules. The results for Distortion correction, Active Search, and Harris detection are shown for each camera. The two last lines of the table show the total required resources and those available ones on the target FPGA (Virtex-6 VLX240T) respectively. The system was compiled with the balanced strategy of compilation proposed by Xilinx. Many of the extra resources were used for the implementation of the Soft Processor (Microblaze), the MPMC, and the PLB infrastructure.

Table 5.2 shows the resources used to implement the IPM algorithm. The whole system includes the homography with cache module, the subtraction module, the Microblaze processor with extended FPU, the MPMC, and the PLB infrastructure. The target FPGA is a Virtex 6 vlx240t.

For MIPM, the equation 2.25 was implemented using the soft-processor (Microblaze). To implement this equation six trigonometric functions (three sinus and three cosines), five multiplications of 3x3 matrixes, one multiplication of 4x4 matrix, and one matrix inversion should be implemented. The matrix inversion was implemented using the method of Gauss-Jordan. Table 5.3 shows the execution time for each of these operations.

If we set the pitch and roll to zero (because of the flat ground), the operations needed to compute equation 2.25 are: two trigonometric functions (one sine and one cosine), two multiplication of 3x3 matrix and one multiplication of 4x4 matrix. In this case no matrix inversion is needed. Table 5.4 shows the execution time for each one of these operations.

Table 5.2: Resources required for IPM.

	Slice Reg	Slice Luts	RAMB36E1	DSP48
Homography without Cache	5684	2983	0	9
Homography with Cache	5907	3469	88	9
Whole System	21357	16260	172	14
Available in V6lx240	301440	150720	416	768

Table 5.3: Execution time required to compute equation 2.25 in Microblaze, taking into account the 3 euler angles and the x-y-z coordinates. Matrix inversion was implemented using Gauss-Jordan Method

Operation	Time
Trigonometric functions	1.4 ms
Matrix Inversion	170 us
Matrix Multiplications	70 us
Total time	1.7 ms

5.9.1 SART

An airport is a very complex place in term of traffic management. When there are bad visibility conditions some procedures have been adopted to guarantee the safety of the people, however, these procedures depends of the expertise and capacity of the pilots.

The objective of this project is to develop a system to help the pilot of an aircraft to drive through an airport when the visibility conditions are poor. The system should implement obstacle detection and should help the pilot to place himself in the airport with little error. Current systems work only with GPS for localization, and can have errors of about 10 meters. In SART, with the fusion of multiple sensors, this error must be reduced to some centimeters. The SLAM algorithm is used to create a map of the airport and simultaneously localize the aircraft in this map.

We wanted to develop a black box that will be installed in the aircraft to provide driving assistance to the pilot. The system shows the pilot a synthetic image with graphic information about some important features like the detected obstacles, localization of the aircraft, signalization, taxiway position, and more. The synthetic data is generated from the embedded algorithms and from a database containing all the information about the airport. See figure 5.31.

This synthetic image must provide enough information to the pilot to anticipate his or her actions while moving through the airport. Even under poor visibility conditions, using the synthetic data the pilot should be able to navigate in the airport under any weather condition.

SART is a project developed in collaboration with many companies. In this project, two main parts have been developed: The Fusion Logic (LDF) and the Human-Machine Interface (HMI). Figure 5.32 shows a simplified block diagram of the different algorithms executed for SART.

Between the two modules (LDF and HMI), a bidirectional communication channel must be established. From the HMI, some configuration commands will be sent to the LDF; from the LDF to the HMI, the images and results of the processing must be sent at 30 Hz. A protocol

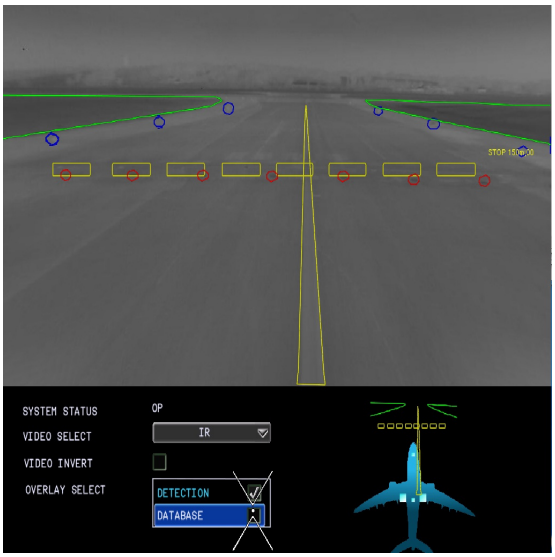

Figure 5.31: SART HMI displayed image - Using the estimated position predicted with the SLAM algorithm and the airport database, a synthetic image will be created with the superposition of the acquired-processed image and the database information.

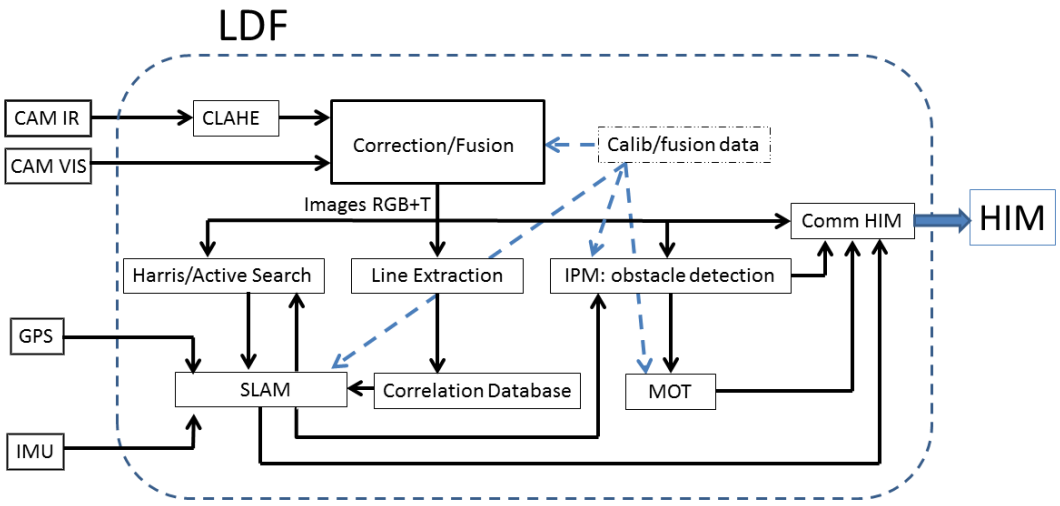

Figure 5.32: Simplified block diagram SART algorithms -

Table 5.4: Execution time required to compute equation 2.25 in Microblaze, just using the yaw angle and x-y-z coordinates

Operation	Time
Trigonometric functions	355 us
Matrix Multiplications	56 us
Total time	412 us

to represent the information was defined between the project partners. Figure 5.33 shows the interface between the LDF and the HMI and the image sent from the LDF to the HMI.

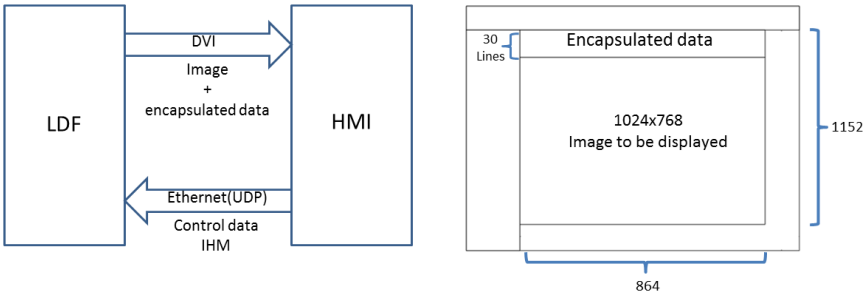

Figure 5.33: Interface HMI and LDF - .

The main constraint of the channel is that the processing results must be synchronized with the images. For this, we chose to use a DVI link in order to send images synchronized with the results. Processing result data was added in the first 30 lines of the image, and then we send the result image that will be displayed to the operator. We decided to use a standard VESA resolution image. With an image to be displayed of resolution 1024x768, and 30 extra lines for the result data, we would need a resolution of 1024x798, which is not standard, so we choose the next standard resolution (1152x864).

The result data sent contains information respect to the obstacles detected in the scene, the SLAM results, as well as other information needed by the HMI in order to rebuild a synthetic image. The synthetic image represents the information to the pilot.

Since in the global SART architecture, we also had an Ethernet channel to send data from a maintenance console to the HMI, we also used this channel to send the commands from the HMI to the LDF. For the implementation of Ethernet, we used the open source lwIP [80] stack. This stack allows for the implementation of UDP/IP and TCP/IP communications in embedded devices with limited resources.

Using the lwIP stack, a MFS image can be used to store the image of an html webpage in the compact Flash. MFS supports ActiveX controls to access the hardware from the Ethernet console. This is very useful for designing a Console of maintenance for aircraft systems. Nowadays, the tendency of the equipment installed in the aircraft is to verify components through a network. It facilitates the maintenance of the numerous embedded system installed in these complex systems.

5.9.1.1 SART Cameras

After a study of the camera response under different weather conditions (fog, humidity, temperature...) of the different signalization present in an airport, the infrared band III (8-12 μm) was chosen. This study was implemented by CEAT (Centre d'Essais Aronautiques de Toulouse), a lab from LCPC (Laboratoire Central des Ponts et Chausses, now integrated in the IFSTTAR research organism) and FLIR.

Figure 5.34 shows the camera used in the SART project. The system is composed by two cameras in two different bands.

Figure 5.34: SART cameras - ..

An infrared camera made by FLIR ATS company with an uncooled infrared LW detector with technology micro bolometer and a Focal distance of 12 mm was used. The sensor works in the Band III (LWIR) between 8 and 14 μm. A big Field Of view is needed in the application. The camera includes some processing needed to improve the image quality. The NUC is implemented periodically or can be controlled through a serial interface. This infrared camera has been designed especially for this project; by now, it is not available on the market.

The visible camera used was fabricated by NIT. This camera uses CMOS technology with a spectrum response between 380 and 950 nm. The fabricant specifies a distortion of 28 percent.

Both the Infrared and the Visible cameras have long FOV. Our cameras have a horizontal and vertical FOV of about 72 and 57 degrees respectively. Because of the Long FOV there is an important distortion in the camera.

Both cameras works at 30 Hz, two modes are provided, with external trigger and without external trigger. When there is not external trigger, both cameras can be synchronized, in this scheme the infrared camera will synchronize the visible camera.

Each camera sends a time stamp in the last line of the image. This time-stamp is used to diagnose the image loss.

Figures 5.35 and 5.36 show the infrared and visible images before and after correcting the distortion. Contrast of the infrared image was enhanced using CLAHE.

5.9.1.2 SART Architecture

Figure 5.37 shows the block diagram of the internal architecture of the FPGAs conceived for SART. On the left, we have the Virtex 6 domain where all of the pixel level operations are executed. Here, a soft processor is used to execute some non-critical tasks like the configurations of the hardware, and to compute the Homography matrix to perform the IPM algorithm. On the Virtex 6 domain we have the SLAM accelerator described on the section 4.7 and the

Figure 5.35: Visible Image SART Camera - On the left, the original distorted image. On the right, the image after distortion correction

Figure 5.36: Infrared Image SART Camera - On the left, the original distorted image. On the right, the image after distortion correction. In all images contrast was enhanced using CLAHE

homography module described in the section 4.3.2. On the right, the Virtex 5 domain has a PowerPC processor working at 400 MHz that is used to implement the SLAM algorithm. Both FPGAs communicate together using a PLB to PCIe bridge.

5.9.1.3 Parallel Processing

Figure 5.38 shows a temporal diagram showing the execution of the different tasks. In order to satisfy the real-time constraint of 30fps, many processing algorithms are executed in parallel. The image represents the process of two frames coming from the camera. A latency of 66 ms exists to transfer the results from the LDF to the HIM.

5.9.1.4 Memory accesses

Figure 5.39 shows a temporal diagram showing the memory accesses to the different memories used for SART. In this figure are represented the multiple memories accesses done in parallel.

Initially the infrared and visible images are stored in a QDR2 memory. While the infrared image is arriving, the histogram needed for CLAHE is computed and the histogram are built using BRAM memories.

After enough lines of the infrared and the visible images are stored in the QDR2 memories, the distortion correction and fusion module begins. For this process, four memories are involved: two QDR2 memories containing the original visible and infrared images are read, one DDR2

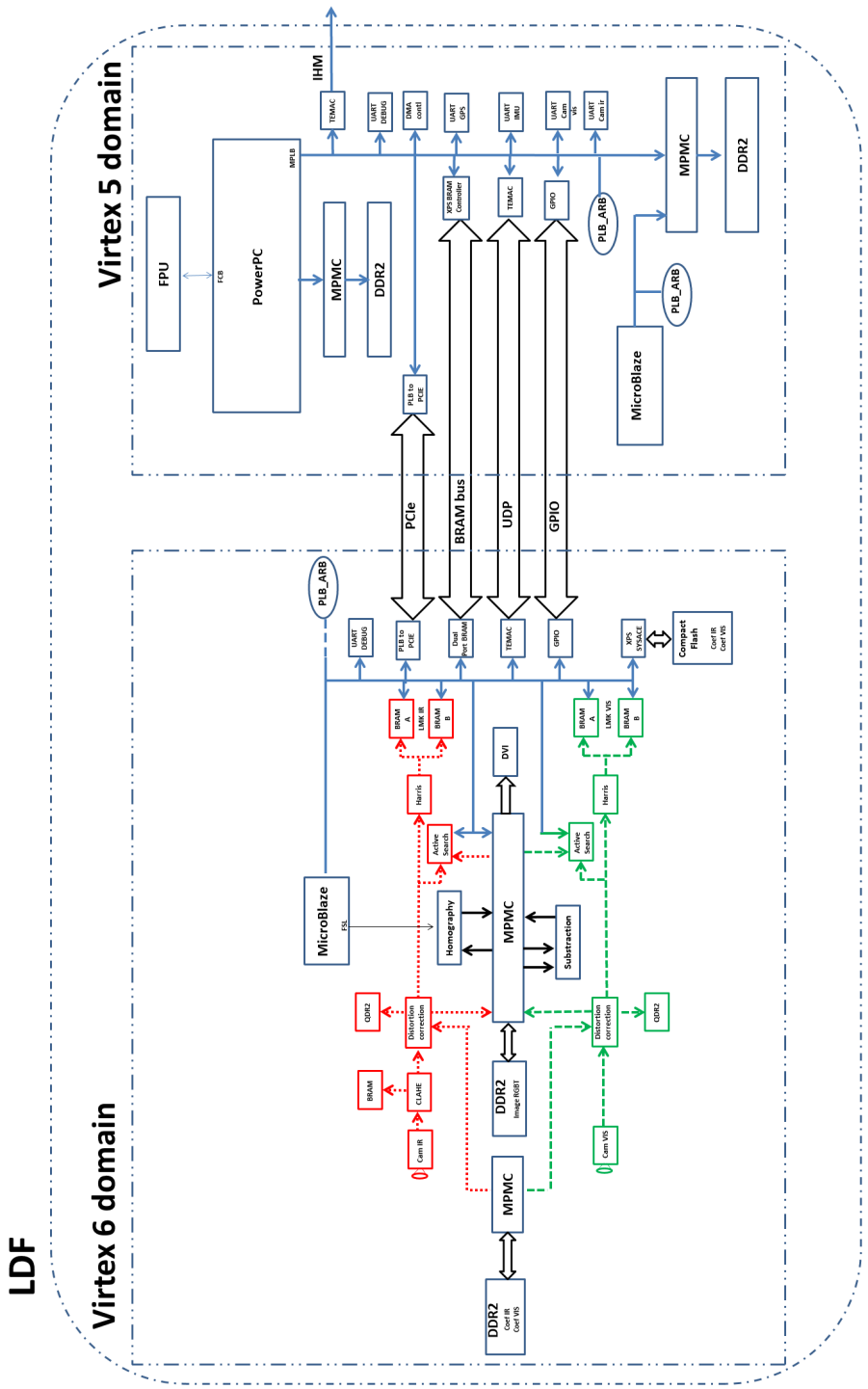

Figure 5.37: Block diagram of the SART System - The system is based on two FPGAs connected through the Gigabit transceivers. On the right, the Virtex 6 domain. On the left, the Virtex 5 domain.

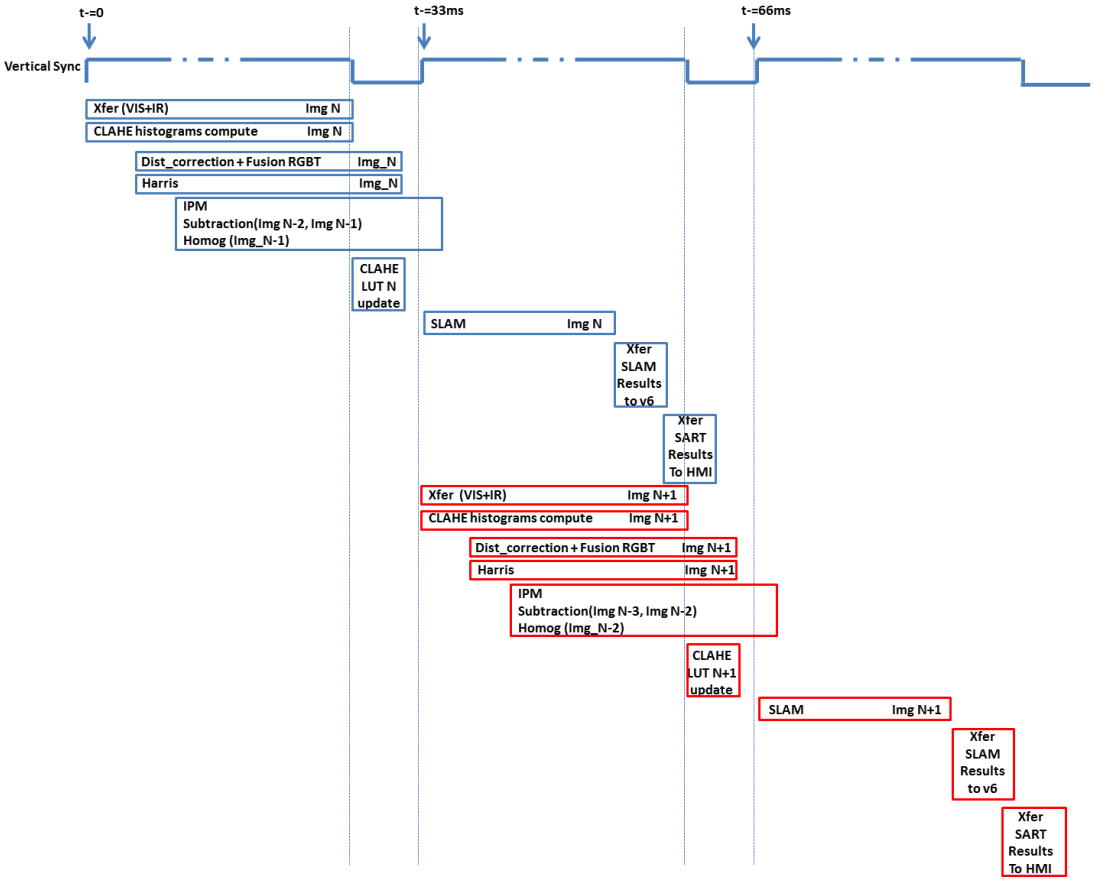

Figure 5.38: Timing representation of SART processing algorithms. - .

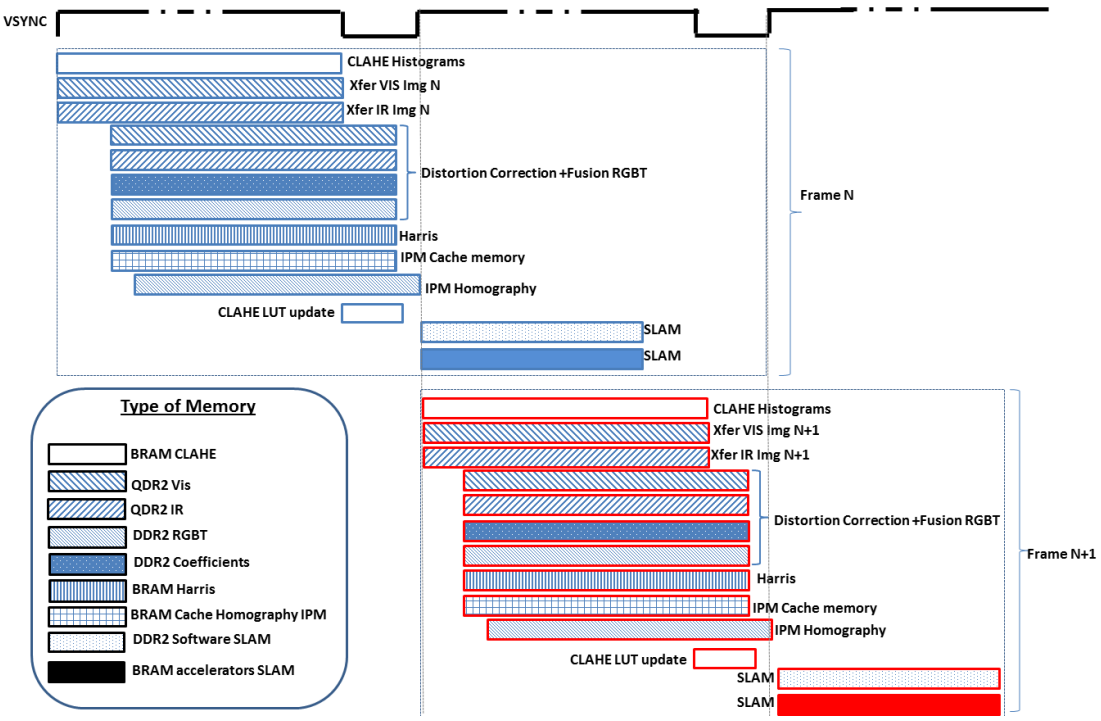

Figure 5.39: Timing representation ofMemory accesses SART - .

memory containing the Distortion Correction Coefficients, and one DDR2 memory used to store the output image (RGBT).

Harris points are detected from the RGBT image. Harris point detector uses a couple of BRAM memories used in a ping pong way to find the points. Simultaneously, a part of the image (RGBT) must be stored in a BRAM, this memory is used like a cache memory, to implement in real time the homographies. After the Cache memory has enough data, the Homography is computed on the fly and is stored in the same DDR2 RGBT. At the same time, the image I_n is read and subtracted with the homography of the image I_{n-1}.

During the vertical sync of the infrared image, the BRAM containing the histograms is swept to compute the transformation LUT that will be used to enhance the contrast of the image N+1.

Detection of interest points using the Harris detector is finished at the end of each image. At this moment, these points are sent to the Virtex 5 FPGA, which implements the SLAM algorithm. The software running the SLAM is executed in the frame N+1. The results of the SLAM algorithm are obtained at the end of the frame N+1, and at this moment they should be transmitted to the HMI.

5.9.1.5 Hardware prototyping

In this thesis we worked in the development of the LDF.

Figure 5.40 shows a block diagram of the LDF. The LDF is composed of sensors, processing elements and memories. The sensors used to satisfy the operational needs of SART are: Infrared Camera, Visible Camera, GPS, and IMU. The chosen cameras had a Long FOV and provide

high resolution images.

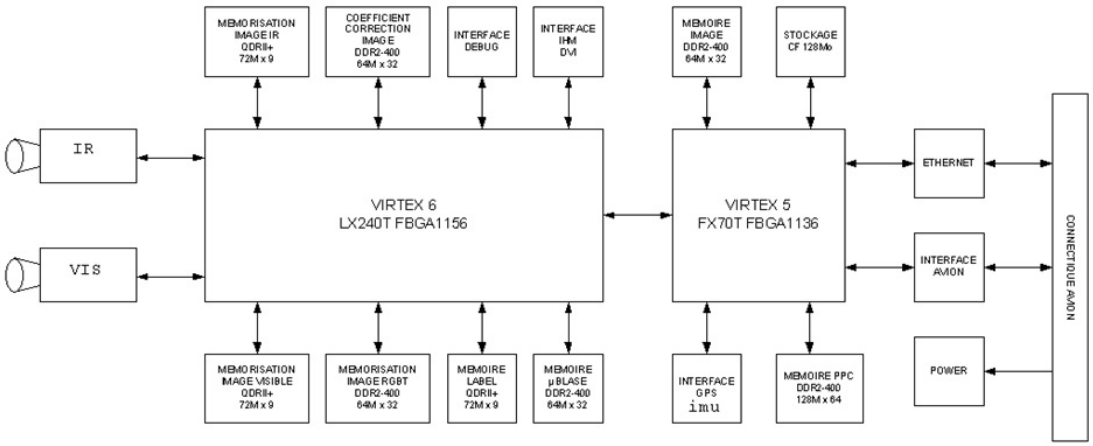

Figure 5.40: Block diagram SART System - The system is based on two FPGAs connected through the Gigabit transceivers.

All of the processing necessary for SART is implemented using two FPGAs: a Virtex 5 and a Virtex 6. The two FPGAs communicate together using a PCIe interface. On the Virtex 6 FPGA, all of the image processing is implemented, and the Virtex 5 implements the SLAM algorithm using the information coming from the Virtex 6 and the other sensors (GPS and IMU).

The system has SRAM memories used to implement all the processes that need pseudo-random access. Some other SDRAM memories were used to upload coefficients to correct distortion and implement image fusion. On the Virtex 6 FPGA, we also implemented obstacle detection using IPM.

Figure 5.41 shows a block diagram of the Inertial Measurement Unit (IMU) chosen for SART. The SBG200 [89] was chosen because it was compatible with the requirements of the project (absolute maximum ratings, operation frequency, temperature, etc). This IMU includes Gyroscopes, Accelerometers and Magnetometers. All of the information is internally processed by the IMU using a Kalman Filter, which provides an estimation of the orientation and the acceleration used by the SLAM algorithm. All this processes are done automatically by the IMU chosen.

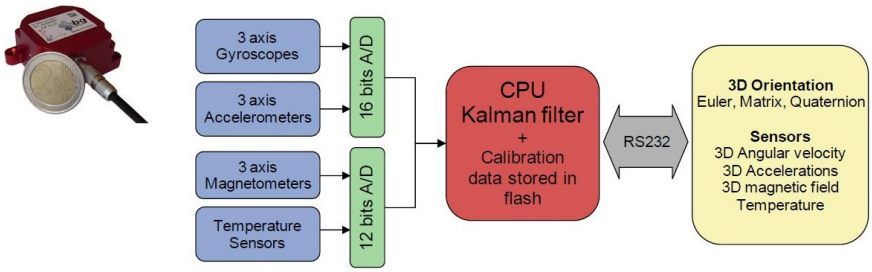

Figure 5.41: SART Inertial Measurement Unit block diagram - Taken from [89]

5.9.1.6 Board design

A board with two FPGAs was developed to support a full multi-spectral vision-based SLAM system. A Virtex-5 FPGA with an integrated PowerPC will compute the SLAM algorithm using algebraic accelerators (matrix multiplier, matrix subtraction), while a Virtex 6 FPGA will implement the pixel level image processing algorithms (Distortion correction, Image enhancement, Interest point detection, Correlations, detection de gradients, perspective transformation). The board under development has been designed upon the experience of using the ML507 and ML605 kits in the development process.

Because of the big distortion present in the cameras, internal memory of the FPGA was not enough to store the needed lines to implement the correction. Two QDR2 memories were added to the system in order to be able to correct important distortions. The QDR memory is a SRAM memory that allows four accesses in one clock period.

In order to be able to support higher frame rates, higher resolution, speed-up some modules, and move to a smallest FPGA, QDR2 (SRAM) memories were added to the board.

The design using QDR2 memories was simulated using Modelsim; Use of QDR2 allows for greater improvements of memory bandwidth because of the random memory accesses needed in the algorithm. Without using the SDRAM memories, the max supported distortion is limited, and is not possible to implement it for color images and infrared images simultaneously.

The final conceived board was routed in a twenty layer PCB. In this thesis, we worked in the definition of the architecture, component selection, interfaces definitions and pin assignment definition. The PCB layout was done by the company Latecoere.

When designing the board some considerations must be taken into account to have the best performance. When working with big FPGAs the size of the routes becomes considerable, and it could increase power consumption, reduce performance or even cause the disfunction of the design. This means that making a good pin-out choice is essential in the design using FPGAs.

Some others considerations for pin-out selection must be done like consequence of physical constraints, for example in the case of the memory of the PowerPC, the pins must be connected in a predefined way because they physically exist this way into the die, otherwise the performance of the PowerPC is reduced.

To choose the memories pin-out, MIG was used. For the other non critical pins (UART GPS, IMU, GPIO's, etc), we let the Place y route tool to choose them automatically in order to let the tool do this in the easiest way. Over-constraining a design can have negative effects. After all pin-out was defined, a Design Rule Check tool was used to verify the coherence of and compatibility of voltages and signal standards.

The communication between the two FPGAs is done using a PLB2PCIe bridge. In this configuration, the Virtex 6 is the Root Complex and the Virtex 5 is the endpoint. The PLB2PCIe bridge allows the FPGA to share a memory space, facilitating the communication between the Virtex 6 and the Virtex 5 domain.

Additional TEMAC to TEMAC connections were connected in order to have a backup channel in the case that the routing of the board would present problems with the PCIe link. The use of UDP does not allow the user to share the BUS in the same natural way it is done using the PLB2PCIe bridge. An additional TEMAC was connected to implement the communication with the Console of Maintenance and the HMI module.

5.9.1.7 System Validation

Two development kits with the same FPGAs components were used to validate all of the algorithms.

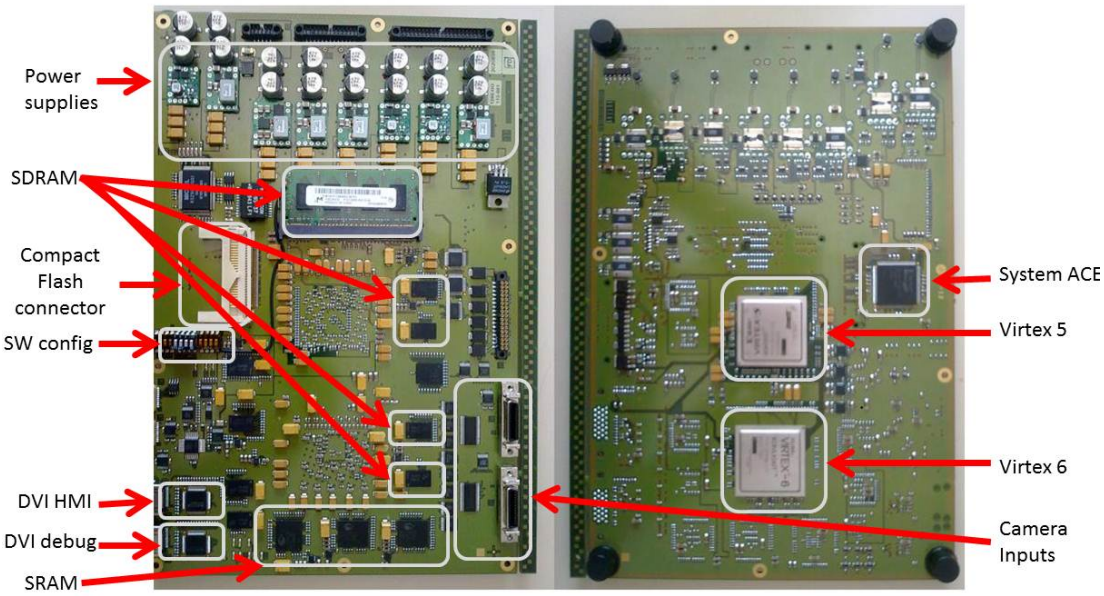

Figure 5.42: LDF SART Board - The board has the two FPGAs connected together through the Gigabit transceivers.

To validate the system, the cameras were installed on the roof of a vehicle, and the data from sensors were acquired simultaneously in a machine. The images were taken in the Blagnac airport with the agreement of the airport authorities and the DGAC (Direction Gnrale de l'Aviation Civile); our colleagues from LATECOERE were in charge for the administrative discussions (which took a lot of time and energy!) and of the organization (vehicle rental and instrumentation). Figure 5.43 shows the vehicle used to do acquisitions.

The acquired data was stored in a personal computer. Then, through the PCIe interface of the development kit, we injected the data to test the algorithms with real data. An API to manage PCIe transferences was developed using FIFOs, this API writes images directly to the memory of the development board. Then, images stored in the SDRAM memory were read by a module that simulates the camera signals, and generate the stimulus for the modules implemented in this thesis. This way we tested in the laboratory all the algorithms using real data.

5.10 Platforms used

Many of the algorithms and principles presented in this thesis were tested using different development kits. Image processing algorithms were mainly tested using the ML605 board [90]. Infrared and visible images were injected into the development kits using high communication speed protocols (PCIe and Ethernet) for prototyping. It allowed us to test the algorithms under the same scenes and verify the repeatability of the algorithm results. Two camera link custom boards with FPGA Mezzanine Cards (FMC) interface were developed for the SART project in order to test the algorithms in situ.

In this thesis we also used the ML507 (52) development kit for the implementation of the

Figure 5.43: SART demonstration Car - .

embedded SLAM algorithm and other communication protocol tests needed for the SART project. In this context, we worked with infrared and visible images acquired at 30fps with resolutions of 1024×768 and 960×720 respectively, cameras designed by the FLIR ATS and the NIT companies.

All of the algorithms were tested in software and then evaluated using development kits. After implementing them in the development kits, the conception of a board was done to satisfy the needs of the SART project.

Figure 5.44 shows a block diagram of the components present in the development kits used to develop the system.

Figure 5.45 shows the platform used to develop the SLAM accelerator. On the right, a black box contains the infrared and visible cameras connected to the ML605 development board with our FMC boards. The visible and infrared images with a sinusoidal transformation are shown on the screen.

A board to interface the SART cameras to the FMC ML605 development kit connector was developed. In the MML605 development kit, there are two FMC ports, one High Performance Connector (HPC) and one Low Performance Connector (LPC). Even if using just one FMC connector would be enough to connect both cameras, we preferred to develop two times the same board, and connect each of them to each of the FMCs. Because CAMLINK uses very high speed signals, some considerations with respect to impedance matching and good pin-out selection had to be done.

Figure 5.46 shows the platform used for the COMMROB project. In the upper part there is a laptop PC which is used to display the output images and to read the odometry from the SCOUT robot.

Figure 5.47 shows the hardware used to implement the disparity map in the PICASSO project. This Hardware was developed by the company Delta Technologies SudOuest.

Figure 5.48 shows the Stratix 3 development kit and the cameras used to develop the camera belt for the COMMROB project.

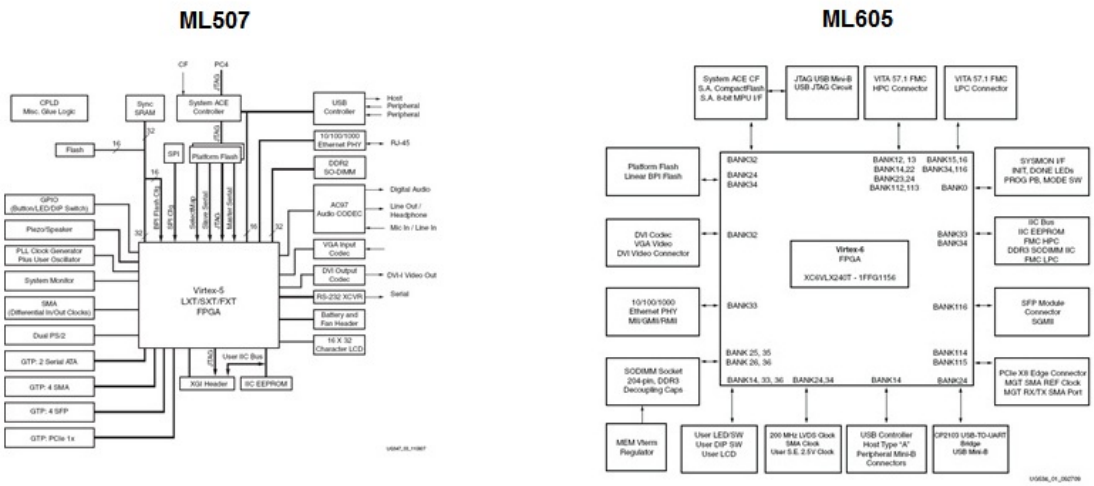

Figure 5.44: ML507 and ML605 Block diagrams - On the left the ML507 block diagram, this development kit includes a Virtex 5 FPGA (with PowerPC processor), a DDR2 Memory, Compact Flash memory, PCIe connectors, Ethernet connectors, etc. On the left, the ML605 Block diagram, this development kit includes a Virtex 6 FPGA, a DDR3 Memory, Compact Flash memory, PCIe connectors, Ethernet connectors, etc.

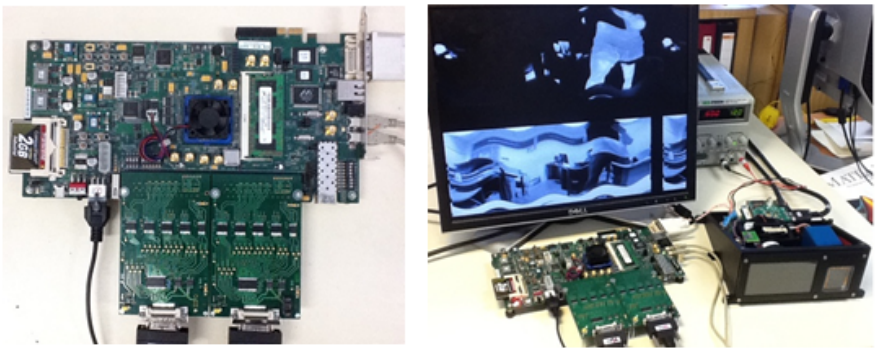

Figure 5.45: Hardware Platform used to develop SLAM Accelerator - On the left, the ML605 development kit, with the custom FMC boards to connect the cameras. On the right, the platform with the camera and a screen where we visualize the output images.

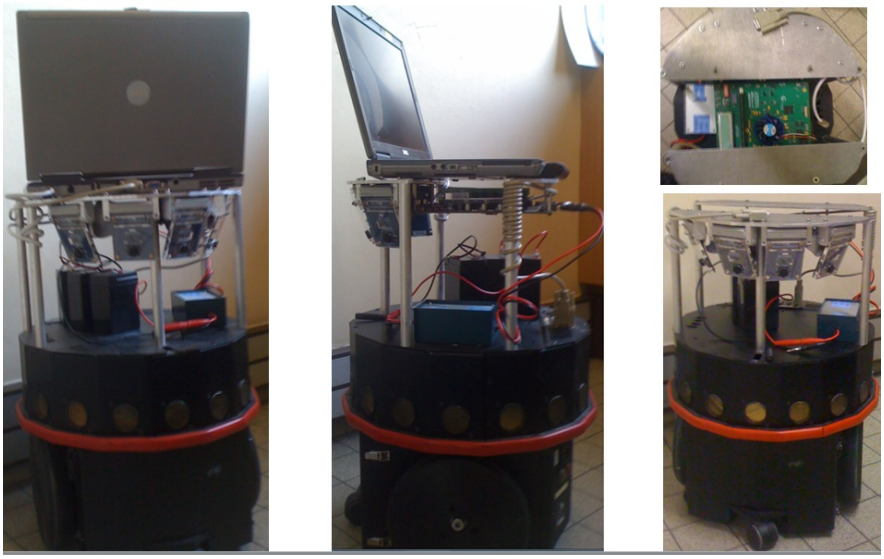

Figure 5.46: Commrob platform - The platform was mounted in a Scout robot. We installed a Stratix 3 development kit connected with four Terasic cameras. The result images were displayed in a computer.

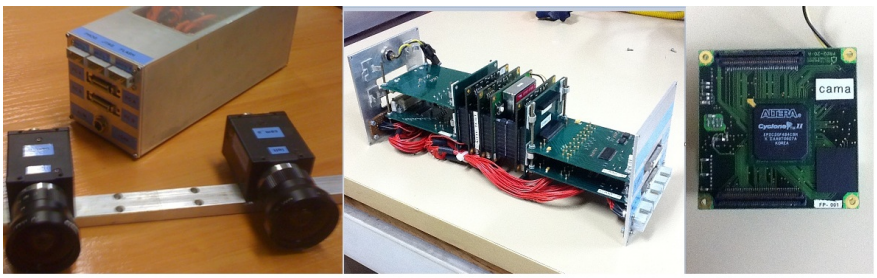

Figure 5.47: Picasso Hardware - Custom boards developed for the Picasso Hardware. The system is based in three Cyclone 2 FPGA

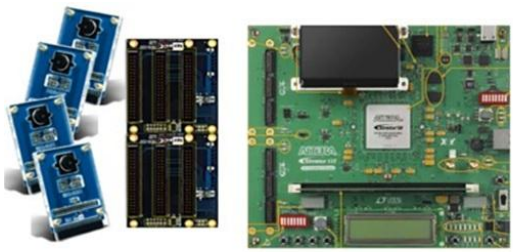

Figure 5.48: Camera Belt hardware platform -

6

Conclusion

The general topic of our thesis concerned both the evaluation of existing vision algorithms to cope with ADAS applications (obstacle detection, vehicle localization), and the design of dedicated architectures able to satisfy hard constraints coming from industrial requirements (limited resources, minimal cost...] and from the application itself (robustness, real time, acceptanbility...). This problematic is often names *Adequacy between Algorithms and Architectures*: it has been studied for number of years, moreover by computer scientists, proposing methodologies for the co-design, for the certification, the formal verification that a proposed architecture satisfies all constraints.

We studied mainly Advanced Driver Assistance Systems, designed to interact with an operator that needs information as soon as possible; a delay of some milliseconds can induce fatal results. Also, for the comfort of the operator, 30 FPS is the minimal frequency for the display functions. This imposes a constraint to process in real-time a lot of data.

In this thesis we exploited the inherent parallelism of an FPGA to propose an efficient co-design solution for multiple algorithms; hardware accelerations were done to optimize the execution of time-critical tasks. Let us recall our contributions before presenting some future works.

6.1 Developed Algorithms and Architectures

For complex systems with many processes running in parallel, methodologies based on Petri Nets can be used to make choices when distributing tasks between the hardware and software components of the system. In this thesis, we did not exploit tools like this; we based our choices on profiling tools only.

In order to speed-up algorithm execution, different dedicated architectures with hardware accelerators were proposed to accelerate the critical parts of these algorithms. Many functions were developed completely using hardware because of the increase in reliability of hardware systems with respect to the software implementation.

Pipelining techniques used in the proposed architectures allowed us to respect the real-time constraint of the application. Pipelining has the disadvantage to increment the latency of the modules, but in the other hand, it allows higher operating frequencies.

The use of high level tools enable us to validate the system at different design steps and helped to design relevant test-benches with adequate coverage. System Generator provides high

level functionalities of Matlab such as image reading/writing, graph plotting, random number generation, among others.

The use of the PCIe bus facilitates the development process by allowing us to run a co-designed system using a conventional PC. With this approach, the algorithms can be deported to the FPGA progressively.

With the new development flow, using HLS tools, the power of the FPGA can be easily exploited to develop an application. The design of architectures using the new FPGAs with a lot of resources begins to be unmanageable without using HLS tools.

The new Extensible Processing Platform (EPP) [91], recently available in Xilinx, could replace the architecture for the SART system proposed in this thesis, even this powerful platform could improve the results. However, this platform has only been available since the summer of this year.

One big difficulty when working with FPGAs is the technical skills needed to develop using these kinds of platforms since different FPGA providers have had different development methodologies and tools. During this thesis, two different FPGA platforms of two different vendors were used, it complicated the development tasks because of the time needed to learn the specific tools of each provider. In the last FPGAs of the main providers, the methodologies are beginning to converge thanks to the adoption of a common architecture based on the AMBA bus; it would facilitate the task of the designer.

Using multiple spectral sources (infrared and visible) allows for the improvement of the driving assistance systems giving it robustness against harsh visibility conditions. The HDR infrared images must be pre-processed in an efficient way in order to extract the useful information and to reduce the embedded resources needed.

In this thesis, we studied the problem of multispectral cameras and we used homographies to implement the fusion of multispectral sources. With the solution proposed here, ground planes can be aligned, but it does not give a perfect result, because of the ghost effects generated by the objects outside of this plane. Recently, new cameras including infrared and visible cameras perfectly aligned using a prism-based mechanism are marketed by JAI company.

The use of tables containing the calibration information of the cameras allows for the implementation of excellent distortion correction that can be used in camera with very long FOV. The architecture presented here can be used, for example, in fish eye cameras.

In the distortion correction algorithm, to improve the quality of the output image, bilinear interpolations are used to compute the value of the undistorted pixels. The use of the interpolation avoids the apparition of artifacts in the undistorted image.

The distortion correction algorithm implemented in this thesis was used in two projects: PICASSO and SART. In PICASSO, the stereo vision algorithm was enhanced with the distortion correction. A nearly perfect alignment of the camera is easier to accomplish using an architecture like the one proposed in this thesis compared to using the mechanical system shown in figure 2.12.

The homographic transformation is a pixel-level operation widely used in computer vision that exposes a lot of potential pipeline. Homography needs an important amount of computation, thus requiring powerful resources to be performed in real-time. Architecture such as GPUs can take advantage of such pipeline but does not meet the power requirements of an embedded platform. An FPGA allows for designing an application-specific computing platform to meet the performance requirements of the target application.

We showed a fully pipelined design to compute homographies. We presented two different architectures that allow for the implementation of the homography, to use a SDRAM memory instead of a SRAM one. Using a SDRAM instead of a SRAM memory reduces total system cost.

Memory requirements can lead to cost increases when using SRAM, but using cheaper SDRAM can slow down the performance due to out of order memory access. To target a cost effective solution, we have optimized the homography implementation using a Cache management strategy, designed to allow the use of SDRAM memory without degrading the performances.

For obstacle detection, we evaluated two different approaches; one based on texture characteristics learned offline and the other based on perspective. A fusion between multiple detection obstacles can create more robust detection systems.

We implemented the obstacle detection using a multi-camera system and a texture algorithm already generated in the works of [11]. Their algorithm was used to create occupation maps using four cameras.

Obstacle detection algorithms based on textures present a good solution for an indoor application. However, in an outdoor application where the conditions can change drastically, this could generate many false alarms.

For the perspective approach, in IPM, the most expensive part of the algorithm in terms of computer resources is the homography. Another expensive part of the algorithm is the extensive use of memory and the way that memory is used.

The first homography architecture presented (without cache memory) allows for the implementation of any homography with an execution time constraint. The time constraints limited the operation frequency and it was due to the random memory accesses nature of the homography algorithm.

The second homography architecture presented using a cache memory has the constraint that the max delta between the non-transformed coordinates and the transformed coordinates supported is proportional to the cache memory size. For the MIPM, this constraint limits the maximum camera movement between two frames. This maximum camera movement can be known from the characteristics of the robot movement model and the camera parameters, which allows to size the cache memory.

The advantage of SIPM, with respect to the MIPM, is that no odometry is necessary. In SIPM, the odometry information is replaced by the knowledge about the position between the cameras, which is fixed and known. Using a table to compute SIPM allows for encoding the lens distortion correction and the homography simultaneously with no extra cost. The disadvantage of the SIPM is that a pre-processing algorithm should be used because images come from different cameras and use sensors with different dynamic responses.

IPM has a big disadvantage in aircraft applications. The presence of many obstacles that are not in the ground plane, such as the wings of an aircraft, generate a detection, but the distance from the object will not be perceived where it should be.

The obstacle detection using MIPM presents an inconvenient effect with respect to the shadow of the robot, which will be detected as an obstacle. Perspective algorithms, such as stereo or IPM, can be used to provide information about the obstacles around a vehicle.

IPM algorithm can be evaluated using multispectral sources such as infrared cameras. Using infrared and visible sources simultaneously improves performance and can overcome notable issues of the algorithm like the susceptibility to static and moving shadows. MIPM is sensitive to the vehicle shadow; the vehicle's shadow moves constantly, and even if the shadow is in the ground plane its position cannot be aligned between different acquisitions with the IPM transformation.

In future implementations, IPM can be improved in many ways, for example, using a background model instead of the subtraction, implementing post-processing algorithms such as morphological operations (Erode and dilate) and executing the IPM algorithm in function of the speed of the robot (if the robot does not move, no subtraction is done; if the robot increments its speed, the subtraction will be implemented at a higher frame rate).

For the moment, we have evaluated mono and stereo IPM, but the algorithm could be extended to a system with more than two cameras.

We presented an accelerator that speeds-up the image processing of the vision-based SLAM and the resulting speed-up will allow us to run the SLAM algorithm on a low speed general purpose processor (PPC, ARM9 ...).

Creating an accelerator working like front-end of the algorithm, liberates the processor to implement algorithms that are very expensive when executed using a processor only approach.

The architecture conceived for the front-end accelerator is very versatile and could be used to implement other image vision algorithms like Stereo inverse perspective mapping [34], BI-cam SLAM [43], etc...

In the SLAM process, the coordinates of a predicted pixel must be corrected with the known distortion parameters of the camera. In typical SLAM implementations, a quadratic model is used to correct the distortion. In our application, because of the high distortion of the cameras, such a distortion model is not sufficient. We corrected the entire image using a table computed offline. With this, we do not have to implement extra corrections when doing point projections for the SLAM algorithm. Undistorted images were also necessary for other algorithms such as IPM.

The use of a processor with an FPGA represents a good solution for SLAM in an embedded system. EKF SLAM based on filtering denotes a sufficient trade-off in computer complexity for applications using big maps. The acceleration of the SLAM algorithm is focused on the optimization of the operation matrix management and the image processing algorithms.

6.2 Some future works

At first, concerning the methodology, due to the industrial context of the SART project, and also due to the limited delay of a thesis, we did not contribute on methodologies; we only used existing tools, provided by FPGA companies or by general software platform (Matlab). A first difficult, but promising future work could consist in revisiting our work in order to evaluate how high-level tools could improve our results; a difficulty here is the very fact market evolution for these development tools, the time required to learn the new proposed methodologies... which often require competences both in software and in hardware.

Now let us present some short-term perspectives, mainly inspired by the current results obtained for the SART project. One of the main objectives of the SART project was to increment the precision of the current driving assistance systems, currently based only in GPS information giving an error of about 8 meters. Implementing hardware accelerators to apply the SLAM algorithm allows for incrementing the precision of the position estimation. If more landmarks can be processed, a better estimation of the position can be built.

In the first prototype developed for SART, the first algorithm implemented into Infrared images was the Histogram Equalization (HE). We decided to implement the HE as the first algorithm because in order to reduce the dynamic range to 8-bits, thus reducing the resources needed to implement the post-processing algorithms such as the distortion correction. The distortion correction was implemented using only the internal RAM of the FPGA in order to satisfy the real-time operation constraint. However, implementing the CLAHE as the first algorithm imposes disadvantages, because the image quality, especially the sharpness, could be altered by the interpolation operations. Moreover, let us recall that there were black spots in the corner of the IR images that were coming from the lens and from the protected window set in front of the lens; this optical configuration reduced the efficiency of the CLAHE algorithm.

Another option (not possible to be implemented during this thesis, because of the delays

in the manufacturing process of the custom board) was to implement the correction distortion of the cameras first using the 16 bits of the IR image. From the undistorted images with full dynamic range, we could have computed the gradients directly using the entire dynamic range of the infrared sensor. At the same time, for the implementation of CLAHE, we would have had better results in some regions of the images because the black spots would have been eliminated before implementing CLAHE.

For the SART project, we decided to implement the NUC when the system is initialized while no NUC was performed during the execution of the algorithms. In a future improvement, the NUC should be implemented periodically and a special strategy should be used to do not affect the algorithms working with the IR images. With the current implementation, if a NUC is implemented, the camera shutter is closed during some images; the Tracking or Active Search algorithms can diverge or give poor results when disturbed by the NUC process. In the case of SLAM, landmarks will be initialized everywhere, the active search will give incorrect results, consequently the algorithm may diverge. So considering the use of infrared cameras, a future work could concern the NUC process, made directly on the scene images, without closing the shutter.

Considering obstacle detection, we limited our research on methods adapted for navigation of vehicles on a flat ground. In our research team in LAAS, several PhD students must study in the next period, methods based on classification, designed in order to detect objects of known classes; the appearance of these objects, described by descriptors ligne HOG (Histograms of Oriented Gradients), BoW (Bags of Words) ..., must be previously learnt from a large data base of images. These methods must be evaluated for aircraft applications, using dedicated learning steps, and moreover, architectures muts be studied in order to execute these demanding algorithms in real time.

A custom board based on two FPGAs was designed for the SART project. This board has many resources that will be used only during the development process and after a simpler prototype can be built. In a future upgrade of the board, a discrete processor connected to a single FPGA can provide a good solution. It would reduce the board price and the complexity of the system.

To show the driving assistance information to the pilot in SART two different partners developed on the one side the IHM (Human Machine Interface) and on the other side, the LDF (Sensory Data Processing, SLAM, Obstacle detction); the LDF was completely implemented in an FPGA, and the IHM in a personal computer. The entire process could have been done in the FPGA without the necessity of the implementation of a communication link DVI between these two modules.

When implementing the fusion of the images in the COMMROB application, there were some common areas between the FOV of the cameras; we generated tables that chose just one of the pixels coming from the cameras. An improvement to this architecture could read the two pixels and generate a better result, and why not, use stereo vision here, to compute the depth.

Birds eye views generated in the COMMROB application during this thesis, are now used by carmakers. At the moment, to our knowledge, it has not been exploited for aircrafts applications. In this field it can be very useful to provide a visual aid to the pilot of these large machines. In aircraft, the wingspan depends of the aircraft model; in a bird's eye view image, the pilot can identify if obstacles can collide with the airplane wings.

References

[1] Mike Santarini. **Driver Assistance Revs Up On Xilinx FPGA Platforms**. *Xcell Journal Issue 66*, Four Quarter 2008. 2

[2] Shorin Kyo, S. Okazaki, T. Koga, and F. Hidano. **A 100 GOPS in-vehicle vision processor for pre-crash safety systems based on a ring connected 128 4-Way VLIW processing elements**. In *2008 IEEE Symposium on VLSI Circuits*, pages 28 –29, June 2008. 2

[3] **Safety Recommendations Display**. http://www.ntsb.gov/safetyrecs/private/QueryPage.aspx. 3

[4] Airbus. **Getting to grips with CAT II / CAT III operations**. *Flight Operations Support and Line Assistance and AWO Interdirectorate Group*, October 2001. 4

[5] D. J. Jobson G. A. Woodell S. D. Harrah G. D. Hines, Z. Rahman. **Real-time Enhanced Vision System**. *Enhanced and Synthetic Vision 2005, Proc. SPIE 5802*, 2005. 4, 5

[6] Latecoere and Airbus. **External and Taxi Aid Camera System**. *US Patent*, 2000. 4

[7] MEGGITT-Securaplane. 4, 5

[8] Debora A.P. Hersman. **Safety Recommendation.** *National Transportation Safety Board.*, September 4, 2012. 5

[9] Honeywell. **TCAS II/ACAS II. Collision Avoidance System Users Manual**. *ACS-5059. Rev 5*, 02/2000. 4

[10] A.NAOULOU. **Architectures pour la stereovision passive dense temps reel : application a la stereo-endoscopie**. LAAS Reports 06531, LAAS, 194p., 2006-09-08. 6, 23, 83

[11] M. Devy, M.I. Manzano, J.L. Boizard, P. Lacroix, W. Filali, and J.Y. Fourniols. **Integrated subsystem for Obstacle detection from a belt of micro-cameras**. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1 –6, June 2009. 6, 24, 79, 139

[12] M.IBARRA MANZANO. **Vision multi-camra pour la dtection d'obstacles sur un robot de service: des algorithmes un systme intgr**. LAAS Reports 11838, LAAS, 154p., 2012-03-26. 6, 110

[13] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003. 11, 27

[14] D. Scaramuzza and R. Siegwart. **Appearance-Guided Monocular Omnidirectional Visual Odometry for Outdoor Ground Vehicles**. *IEEE Transactions on Robotics*, **24**(5):1015 –1026, October 2008. 15

[15] Lejing Wang, Joerg Traub, Sandro Michael Heining, Selim Benhimane, Ekkehard Euler, Rainer Graumann, and Nassir Navab. **Long Bone X-ray Image Stitching Using C-arm Motion Estimation**. In Hans-Peter Meinzer, Thomas Martin Deserno, Heinz Handels, and Thomas Tolxdorff, editors, *Bildverarbeitung fr die Medizin 2009*, Informatik aktuell, pages 202–206. Springer Berlin Heidelberg, 2009. 15

[16] Z. Yaniv and L. Joskowicz. **Long Bone Panoramas from Fluoroscopic X-ray Images**. In Wiro Niessen and Max Viergever, editors, *Medical Image Computing and Computer-Assisted Intervention MICCAI 2001*, **2208** of *Lecture Notes in Computer Science*, pages 1193–1194. Springer Berlin / Heidelberg, 2001. 15

[17] R. Laganiere, H. Hajjdiab, and A. Mitiche. **Visual reconstruction of ground plane obstacles in a sparse view robot environment**. *Graphical models*, **68**(3):282–293. 15

[18] A. Morfopolous, B. Metz, C. Villalpando, L. Matthies, and N. Serrano. **Implementation of pin point landing vision components in an FPGA system**. In *Aerospace Conference, 2011 IEEE*, pages 1 –9, March 2011. 16

[19] M.DEVY J.L.BOIZARD W.FILALI, D.BOTERO GALEANO. **SOPC components for real time image processing: rectification, distortion correction and homography**. *IEEE Workshop on Electronics, Control, Measurement and Signals.*, Juin 2011. 16

[20] C.L.D.A. Mai, M.T.T. Nguyen, and N.M. Kwok. **A modified Unsharp Masking method using Particle Swarm Optimization**. In *2011 4th International Congress on Image and Signal Processing (CISP)*, **2**, pages 646 –650, October 2011. 19

[21] Ali M. Reza. **Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement**. *J. VLSI Signal Process. Syst.*, **38**(1):3544, August 2004. 20

[22] Volker Schatz. **Low-latency histogram equalization for infrared image sequences: a hardware implementation**. *Journal of Real-Time Image Processing*, June 2011. 20

[23] H.G. Nguyen and J.Y. Laisne. **Obstacle detection using bi-spectrum CCD camera and image processing**. In *Intelligent Vehicles '92 Symposium., Proceedings of the*, pages 42 –50, July 1992. 21

[24] Y. Fang, K. Yamada, Y. Ninomiya, B. Horn, and I. Masaki. **Comparison between infrared-image-based and visible-image-based approaches for pedestrian detection**. In *IEEE Intelligent Vehicles Symposium, 2003. Proceedings*, pages 505– 510. IEEE, June 2003. 21

[25] Renaud Pteri and Ondej iler. **Object Tracking using Joint Visible and Thermal Infrared Video Sequences**. Technical report, 2009. 22

[26] S. J Krotosky and M. M Trivedi. **On Color-, Infrared-, and Multimodal-Stereo Approaches to Pedestrian Detection**. *IEEE Transactions on Intelligent Transportation Systems*, **8**(4):619–629, December 2007. 22

[27] James M. Suiter and Mary Beth Lapis. **Multispectral image fusion for the small aircraft transportation system**. 22

[28] O. Demuynck and J.L. Lazaro. **An Efficient Approach Technique for Dynamical Infrared/Visible Images Fusion**. In *IEEE International Symposium on Intelligent Signal Processing, 2007. WISP 2007*, pages 1 –6, October 2007. 22

[29] M.A. Ibarra-Manzano, D.-L. Almanza-Ojeda, M. Devy, J.-L. Boizard, and J.-Y. Fourniols. **Stereo Vision Algorithm Implementation in FPGA Using Census Transform for Effective Resource Optimization**. In *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, 2009. DSD '09*, pages 799 –805, August 2009. 23

[30] Manhua Liu, Jianchao Yao, Hui Zhao, and Kim-Hui Yap. **Learning-Based Image Ground Segmentation Using Multiple Cues**. In *2nd International Congress on Image and Signal Processing, 2009. CISP '09*, pages 1 –5, October 2009. 24

[31] J.J. Little H.P. Mallot, H.H. Bulthoff and S. Bohrer. **Inverse perspective mapping simplifies optical flow computation and obstacle detection**. In *Biological Cybernetics*, **1**, page 64(3), July 1991. 25

[32] Massimo Bertozzi, Alberto Broggi, Alessandra Fascioli, and Ra Fascioli. *Stereo Inverse Perspective Mapping: Theory and Applications*, **8**. 1998. 25

[33] N. Simond and M. Parent. **Obstacle detection from IPM and super-homography**. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 4283 –4288, November 2007. 25

[34] D. A.B Galeano, M. Devy, J. L Boizard, and W. Filali. **Real-time architecture on FPGA for obstacle detection using inverse perspective mapping**. In *2011 18th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 788–791. IEEE, December 2011. 25, 140

[35] Gang Yi Jiang, Tae Young Choi, Suk Kyo Hong, Jae Wook Bae, and Byung Suk Song. **Lane and obstacle detection based on fast inverse perspective mapping algorithm**. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, **4**, pages 2969 –2974 vol.4, 2000. 25

[36] C Harris and M Stephens. **A Combined Corner and Edge Detection**. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988. 27

[37] Jianbo Shi and C. Tomasi. **Good features to track**. In *, 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94*, pages 593–600. IEEE, June 1994. 27

[38] Merwan BIREM and François BERRY. **Hardware Architecture for Visual Feature Extraction**. *SCaBot'12 - 1st Workshop on Smart Camera for Robotic Application*, Octobre, 2012. 28

[39] H. Yasuura, N. Takagi, and S. Yajima. **The Parallel Enumeration Sorting Scheme for VLSI**. *IEEE Transactions on Computers*, **C-31**(12):1192 –1201, December 1982. 28

[40] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. **BRIEF: Computing a Local Binary Descriptor Very Fast**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **34**(7):1281 –1298, July 2012. 29

[41] J. Civera, O. G Grasa, A. J Davison, and J. M.M Montiel. **1-point RANSAC for EKF-based Structure from Motion**. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009*, pages 3498–3504. IEEE, October 2009. 30

[42] A. J Davison. **Real-time simultaneous localisation and mapping with a single camera**. In *Ninth IEEE International Conference on Computer Vision, 2003. Proceedings*, pages 1403–1410 vol.2. IEEE, October 2003. 30

[43] J. Sola, A. Monin, and M. Devy. **BiCamSLAM: Two times mono is more than stereo**. In *2007 IEEE International Conference on Robotics and Automation*, pages 4795–4800. IEEE, April 2007. 30, 140

[44] H. Strasdat, J.M.M. Montiel, and A.J. Davison. **Real-time monocular SLAM: Why filter?** In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2657 –2664, May 2010. 30

[45] Gabe Sibley, Gaurav Sukhatme, and Larry Matthies. **Visual Sliding Window SLAM with Application to Planetary Landers**. 30

[46] V. Bonato, E. Marques, and G. A Constantinides. **A Floating-Point Extended Kalman Filter Implementation for Autonomous Mobile Robots**. In *International Conference on Field Programmable Logic and Applications, 2007. FPL 2007*, pages 576–579. IEEE, August 2007. 30

[47] Grigorios Mingas, Emmanouil Tsardoulias, and Loukas Petrou. % bf An FPGA implementation of the SMG-SLAM algorithm. *Microprocessors and Microsystems*, **36**(3):190–204, May 2012. 30

[48] A. J Davison. **Active search for real-time vision**. In *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005*, **1**, pages 66– 73 Vol. 1. IEEE, October 2005. 32

[49] A. J Davison, I. D Reid, N. D Molton, and O. Stasse. **MonoSLAM: Real-Time Single Camera SLAM**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **29**(6):1052–1067, June 2007. 32, 98

[50] J.SOLA J.M.CODOL N.MANSARD S.LACROIX M.DEVY C.ROUSSILLON, A.GONZALEZ. **RT-SLAM: A generic and real-time visual SLAM implementation**. *International Conference on Computer Vision Systems (ICVS'2011), Sophia Antipolis (France), 20-22 Septembre 2011, pp.31-40*, 2011. 32

[51] Prabhuram Gopalan Xin Wu and Greg Lara. **Xilinx Next Generation 28 nm FPGA Technology Overview**. *WP312 (v1.1)*, March 26, 2011. 35

[52] Altera Corporation. **Introducing Innovations at 28 nm to Move Beyond Moores Law**. *White paper, WP-01125-1.2*, June, 2012. 35

[53] Zhiyi Yang, Yating Zhu, and Yong Pu. **Parallel Image Processing Based on CUDA**. In *Computer Science and Software Engineering, 2008 International Conference on*, **3**, pages 198 –201, December 2008. 37

[54] S. Asano, T. Maruyama, and Y. Yamaguchi. **Performance comparison of FPGA, GPU and CPU in image processing**. In *International Conference on Field Programmable Logic and Applications, 2009. FPL 2009*, pages 126 –131, September 2009. 37

[55] Benyi Shi, Sihai Chen, Feifei Huang, Cheng Wang, and Kun Bi. **The Parallel Processing Based on CUDA for Convolution Filter FDK Reconstruction of CT**. In *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*, pages 149 –153, December 2010. 37

[56] Hee Kong Phoon, M. Yap, and Chuan Khye Chai. **A Highly Compatible Architecture Design for Optimum FPGA to Structured-ASIC Migration**. In *IEEE International Conference on Semiconductor Electronics, 2006. ICSE '06*, pages 506 –510, December 2006. 37

[57] Xilinx. **Virtex-6 FPGA Configurable Logic Block User Guide**. *UG364 (v1.2)*, February 3, 2012. 38, 39

[58] Xilinx. **Virtex-6 FPGA Extended Overview**. *Xilinx Product Marketing*, March 22, 2009. 39

[59] Xilinx. **Virtex-6 FPGA DSP48E1 Slice User Guide**. *UG369 (v1.3)*, February 14, 2011. 40

[60] Xilinx. **Virtex-5 FPGA RocketIO GTP Transceiver User Guide**. *UG196 (v2.1)*, December 3, 2009. 41

[61] Inc. Berkeley Design Technology. **High-Level Synthesis Tools for Xilinx FPGAs**. *Berkeley Design Technology, Inc.*, 20xx. 40

[62] Xilinx. **Virtex-6 FPGA DSP48E1 Slice**. *User Guide, UG369 (v1.3)*, February 14, 2011. 40

[63] Hans-Peter Rosinger. **Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel**. *XAPP529 (v1.3)*, May 12, 2004. 42

[64] Xilinx. **Timing Closure User Guide**. *UG612 (v 13.3)*, October 19, 2011. 44, 59

[65] Xilinx. **Embedded Processor Block in Virtex-5 FPGAs Reference Guide**. *UG200 (v1.8)*, February 24, 2010. 46, 52

[66] Xilinx. **LogiCORE IP Multi-Port Memory Controller (MPMC) Product Specification**. *DS643 (v6.03.a)*, March 1, 2011. 48, 49

[67] Xilinx. **Memory Interface Solutions User Guide**. *DS530 v3.0*, September 21, 2010. 48

[68] Xilinx. **Logicore IP Muli-Port Memory Controller**. *V6.03.1 ds643*, 20xx. 50

[69] Xilinx. **Memory Interface Solutions**. *User Guide, UG086 (v3.6)*, September 21, 2010. 50

[70] Xilinx. **MicroBlaze Processor Reference Guide**. *UG081 (v9.0)*. 51

[71] Altera. **Nios II Processor Reference Handbook**. *NII5V1-11.0*, May, 2011. 53

[72] Altera corporation. **Avalon Interface Specifications**. *v11.0*, May, 2011. 54

[73] J. Cong, Bin Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Zhiru Zhang. **High-Level Synthesis for FPGAs: From Prototyping to Deployment**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **30**(4):473 –491, April 2011. 55

[74] Kazutoshi Wakabayashi. **C-based behavioral synthesis and verification analysis on industrial design examples**. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, ASP-DAC '04, page 344348, Piscataway, NJ, USA, 2004. IEEE Press. 55

[75] Shakeel Jeeawoody. **New Tools Take the Pain out of FPGA Synthesis**. *Xcell Journal*, Second Quarter 2012. 56

[76] Xilinx. **ChipScope Pro 12.1 Software and Cores**. *UG029 (v12.1)*, April 19, 2010. 58

[77] Altera Corporation. **Design Debugging Using the SignalTap II Logic Analyzer**. *Quartus II Handbook Version 12.0*, June, 2012. 58

[78] Xilinx. **Timing Closure User Guide**. *UG612 (v 13.3)*, October 19, 2011. 59

[79] Altera. **Ethernet and the NicheStack TCP/IP Stack - Nios II Edition**. *NII52013-11.0.0*, May 2011. 60

[80] Xilinx. **lwIP Library**. *v2.00.a*, January 8, 2007. 60, 124

[81] Xilinx. **LogiCORE IP Divider Generator. Product Specification**. *UG086 (v3.6)*, March 1, 2011. 72

[82] Inc. Nomadic Technologies. **Nomad Scout User's Manual**. *Part number : DOC00004*, July 12, 1999. 83

[83] JAI. **Digital Monochrome Quad Speed CMOS Progressive Scan Camera CV-A33CL Operation Manual**. *Version 1.0 Revision A*. 83

[84] Cypress Semiconductors. **EZ-USB FX2 Technical Reference Manual**. *Version 2.1.*, 2000, 2001. 84

[85] Xilinx. **System Generator for DSP Getting Started Guide**. *UG639 (v 13.1)*, March 1, 2011. 104

[86] Mario-Alberto Ibarra-Manzano and Dora-Luz Almanza-Ojeda. **An FPGA Implementation for Texture Analysis Considering the Real-Time Requirements of Vision-Based Systems**. In Andreas Koch, Ram Krishnamurthy, John McAllister, Roger Woods, and Tarek El-Ghazawi, editors, *Reconfigurable Computing: Architectures, Tools and Applications*, **6578** of *Lecture Notes in Computer Science*, pages 110–117. Springer Berlin Heidelberg, 2011. 110, 113, 114

[87] Terasic. **TRDB D5M 5 Mega Pixel Digital Camera Development Kit**. *Document Version 1.2*, August 10, 2010. 113

[88] Altera. **Stratix III 3SL150 Development Board Reference Manual**. *Version 1.4*, November 2008. 113

[89] SBG Systems. **IG-500A Sub-miniature AHRS User Manual**. *Revision: 8 19.*, November 2009. 130

[90] Xilinx. **ML605 Hardware User Guide UG534 (v1.7)**. *UG534 (v1.7)*, June 19, 2012. 132

[91] Sandeep Dutta Brad Taylor Ralph Wittig Vidya Rajagopalan, Vamsi Boppana. **An Extensible Processing Platform Family**. *Xilinx Zynq-7000 EPP*, August 18, 2011. 138