Intelligent IP Camera

An FPGA Motion Detection Implementation

DAT6, Spring 2008.

Nicolas Cothereau
Guillaume Delaite
Edouard Gourdin

Aalborg University
Department of Computer Science

Aalborg University Department of Computer Science

TITLE:
Intelligent IP Camera
An FPGA Motion Detection Implementation

THEME:
Distributed Systems
and Semantics
Control Systems

PROJECT PERIOD:
01/02/2008-31/05/2008

PROJECT GROUP:
d605a

GROUP MEMBERS:
Nicolas Cothereau
Guillaume Delaite
Edouard Gourdin

SUPERVISORS:
Alexandre David
Yannick Le Moullec

NUMBER OF COPIES: 2

NUMBER OF PAGES: 94

CONCLUDED: 26-05-2008

SYNOPSIS:

This Report deals with an intelligent camera implementation. This camera is linked with a Altera FPGA platform (DE2 Board) where a motion detection algorithm is implemented. A video is recorded when movements are detected. It is then stored on a SD-Card and accessible on an Ethernet network. Hardware and software co-design is studied to implement the motion detection algorithm on a NIOS II softcore processor, with hardware acceleration. The project is composed of background analysis, design model study, system analysis and design, motion detection algorithm analysis and design, implementation of the proposed solution, and testing and experiments of this implementation. The video capture and transmission are successfully implemented. The motion detection algorithm is also implemented, but using images from the desktop file system. Storage and recording parts are not implemented due to memory chip problems. As a conclusion, the optimization of video surveillance by using FPGA seems to be possible, but with a custom and optimized platform instead of a 'standard one' (i.e. produced by a manufacturer).

key words: Master Thesis, Motion Detection, Altera, FGPA, Softcore Processor, Algorithm, Background Subtraction, Thresholding, HW/SW Co-design, Design Model.

This report is the result of the work done during the SSE4 (Dat6) semester of the Cand.Scient study at Department of Computer Science, Aalborg University. This project was done in the research area of Distributed Systems and Semantics and Control Systems. It constitutes the complete work toward a master thesis.

We would like to thank our supervisors, Yannick le Moullec and Alexandre David for supervising this project and the many helpful comments made throughout the project. We would also like to thank Rasmus Abildgren for his precious and helpful insights into use of Design Trotter.

The document is organised as follows: **Part I** is the main report, with the explanations and details of the proposed solution. **Part II** regroups the appendices and the bibliography of this report. The organization of each part is detailed in the introduction of each.

This thesis has been revised on the 26th of may, 2008. Some corrections and changes were made to the published edition.

Nicolas Cothereau

Guillaume Delaite

Edouard Gourdin

Summary

This Report deals with an intelligent camera implementation. This camera is linked with a ALTERA Field Programmable Gate Array (FPGA) platform on a DE2 Board where a motion detection algorithm is implemented. The main goal of this project is to analyse the feasibility of optimizing video surveillance with an FPGA.

The aim is therefore to create a prototype to do several function, with some requirements. A room is monitored and the video captured by the camera sensor is displayed on a VGA screen. The video is recorded only when motions are detected. Then, the video is stored on a SD-Card and accessible on an Ethernet network. Hardware and software co-design is studied to implement the motion detection algorithm on a NIOS II softcore processor, with hardware acceleration.

The project is composed of background analysis which details the main context of the project and details the problem to be answered in this report. Then, a design model study is proposed to enhance the analysis, design and implementation of the project. The system analysis and design and the motion detection algorithm analysis and design are successively given. Finally, the implementation of the proposed solution and the testing and experiments parts are detailed.

The video capture and transmission are successfully implemented. Those parts are implemented in full hardware, using VERILOG programming language. The motion detection algorithm implemented is a background subtraction algorithm. As the recording and storage blocks are not successfully implemented, frames stored on the desktop file system are used to test the algorithm. It is fully implemented in software, using ANSI-C programming language. Storage and recording parts are not implemented due to memory chip problems.

As a conclusion, the optimization of video surveillance by using FPGA seems to be possible, but with a custom and optimized platform instead of a "standard one" (i.e. produced by a manufacturer). In facts, the memory chip of the selected board is an 8 MB SDRAM chip, while for this project two 256 MB SDRAM would have been better.

Contents

List of Figures

List of Tables

Part I

Main Report

Chapter 1

Introduction

This chapter presents the overall context of this project. Firstly, a short background description of surveillance systems and video surveillance is given. Then, the problem statement of the project is formulated as a question. The methodology used for answering the question is subsequently introduced: goals and tasks are defined step-by-step. Finally, the organization of Part I is detailed.

1.1 Surveillance systems

Surveillance is the monitoring of behaviour; a surveillance system is a system designed to process and monitor the behaviour of people, objects or processes within a given system for conformity to expected/desired norms in trusted systems for security/social control. It can be either secrete or evident. Although the word surveillance literally means “watching over”, the term is often used for all forms of observation or monitoring, not just visual observation [Uni89]. Nevertheless, the all-seeing “eye in the sky” is still a general icon of surveillance. Surveillance in many modern cities and buildings often uses closed-circuit television cameras.

From the beginning of human civilization, there has always been a need for surveillance systems. The reasons are numerous: borders guarding, raiders, spies and thieves protections. The Sun Tzu’s *Art of War*, written more than 2,000 years ago, explain how to use spies against enemies. During thousands of years, there were only one kind of surveillance system with two kinds of actors (or creatures): human beings and dogs. It was more or less efficient depending on the hour of the day. However, electronics and Information Technologies (IT) gave the surveillance new possibilities, making it more efficient and more robust.

1.2 Video Camera and Intelligence

From their invention a century ago, video cameras have been early associated with surveillance systems (e.g. during World War I). Now that cameras are more and more affordable, surveillance systems using them have been generalized and spread all over the world. It is because they are answering a need. It consists in placing video camera in a public or private area to visualize or record in a central room all the people flows. Its aim is to prevent thefts, attacks, frauds and to manage the incidents and the sways of the crowd. As the information is centralized, it allows the owner of a system to save money by employing less people and being more efficient, as a man cannot watch every part of the space at the same time whereas the cameras can [Cus03].

After the use of the video cameras, the other main progresses were mobile video cameras, tape recording and more recently intelligent video cameras. In fact, when one records all the information captured by a video camera, it leads to get a lot of tapes. Therefore, one needs to get room to store those data. Hence, it costs money for having a record mainly consisting of actions of no interest instead of a record of useful sequences (robbery, vandalism, etc.).

Intelligence in video camera is where the stress is put nowadays: with this kind of video camera, the recorded data is smaller. In fact, thanks to the implementation of an image processing algorithm, the video is recorded and stored only when needed (motion detected, face recognized, etc.). The memory usage of the video can also be reduced by using compression methods. More over, using new technology avoid to use tape but other storage media, like data card (CompacFlash card, SD/XD-card, etc.), which can contain more information and are smaller.

1.3 Problem statement

As explain previously, the use of an image processing algorithm allows to store only useful data and save space. Therefore, the main aim of this project is to answer this question:
Is it possible to implement an intelligent IP video surveillance camera on an Field Programmable Gate Array (FPGA) platform to optimize video surveillance?

For this project, the project group would like to connect a camera with an FPGA . This FPGA processes the pictures captured by a camera sensor, used as a video camera, with an image processing algorithm and then make it available on a web-server, connected on a local Ethernet network. The context of the experiments is:

- A fixed video camera;
- A given room to monitor;
- Start recording only when a motion is detected in this room.

The requirements can be described as follow:
First of all, the architecture has to be based on an FPGA (more details can be found in Chapter 3).
Secondly, the Hardware/Software Co-Desisgn methodology should be used to implement the algorithms. *"Hardware/Software Co-Desisgn tries to increase the predictability of embedded system design by providing analysis methods that tell designers if a system meets its performance, power, and size goals and synthesis methods that*

let researchers and designers rapidly evaluate many potential design methodologies" [Wol03]. A more detailed definition is given in Chapter 3. The constraints are defined and detailed in Chapter 3.

1.4 Goals

The main goal is to get the full implementation of the project on an FPGA to answer to the question. Which means building a prototype system to monitor a room, to process data captured, to record the useful part of the video and finally to allow accesses to this video through the Ethernet network, using the Internet Protocol (IP). To achieve this, the project is split in several parts described bellow.

1.4.1 Selection and simulation of the event detection algorithm

This part consists in exploring the different methods to process images. Then, it allows the selection of one algorithm which fulfil our requirements. Once the algorithm is selected, it needs to be profiled to extract useful information to get the best implementation. Therefore, a first draft using C code is to be implemented, then all metrics (parallelism, best suitable architecture, etc. [ASR+05]) are to be calculated to finally get the profile of the algorithm for the implementation of the final version of the algorithm.

1.4.2 Transmission chain

The aim of this part is to capture a video with the camera sensor and display it on a screen. This is the second step to achieve our final goal. In fact, this implementation step allows the extraction of data from the video camera and allows the verification of those data with a display. Please refer to Figure 1.1 (on page 5) to get the block diagram of this part.

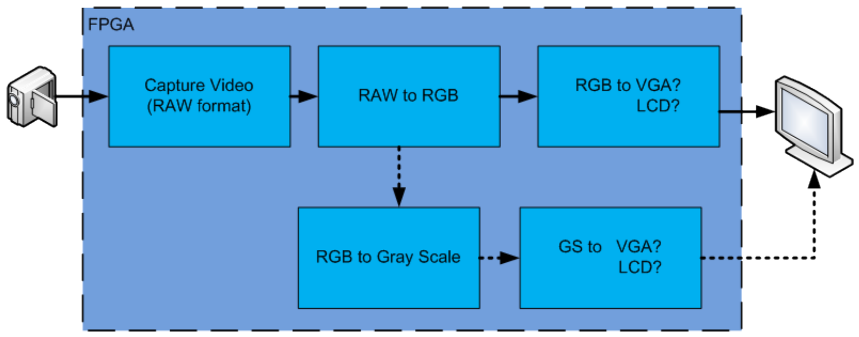


Figure 1.1: Transmission chain block diagram: the implementation of this step is very simple as it only needs a few processing: the conversions from raw data to RGB and the grey-scaled one. The decomposition of the different parts of this system in blocks allow the connection of future other blocks to enhance the system easily.

1.4.3 Recording and Storage Chain

The aim of this part is to add a recording function to the previously described system. To verify this and make this function useful, an access function is to be added. This allows the user to see what has been stored through a web-server. Please see Figure 1.2 (on page 6) to get the block diagram of this part.

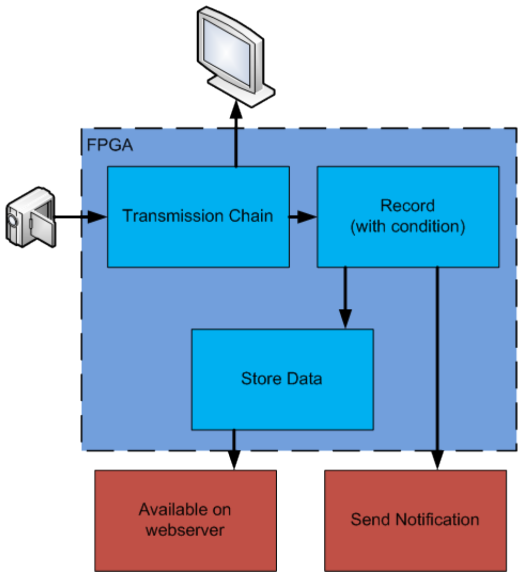


Figure 1.2: Recording chain block diagram: the new blocks are added upon the previous architecture. There are used to enhance the previous system and to avoid a complete redefinition of the system. As previously explain, this new system can be enhanced easily.

1.4.4 Intelligent Camera

Once the previous step implemented, the next step is to trigger those events when a motion is detected. Therefore, the implementation of an event detection algorithm is done during this phase. This part is the core of this project, as the previous steps are preparing steps and following steps are optimizations and feature additions. Please, see Figure 1.3 (on page 6) to get the block diagram of this part.

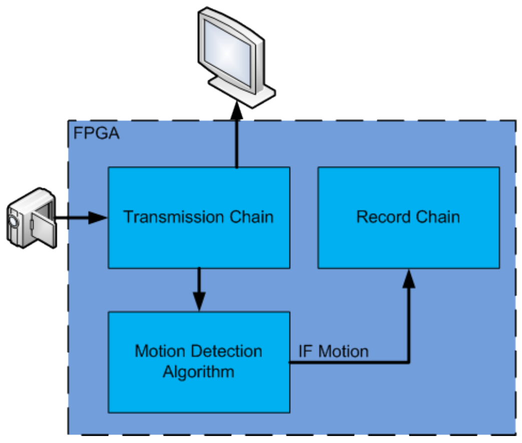


Figure 1.3: Intelligent video camera block diagram: upon the previous implementation are added the blocks of the algorithm.

1.4.5 Optimized Intelligent Camera

This step is for adding an important feature considering the storage space. In fact, implementing a video compression algorithm to improve the storage is highly desirable. It adds complexity to the project design, but also reduces the disk space usage on the SD-Card and therefore the storage costs of the proposed solution. Please refer to Figure 1.4 (on page 7) to get the block diagram of this part.

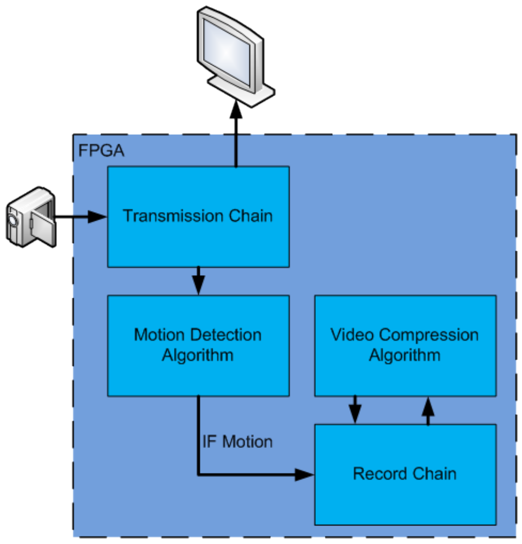


Figure 1.4: Optimized intelligent video camera block diagram: after adding the first algorithm, a second one is to be implemented, concerning the video compression to save more space.

1.4.6 Complete Intelligent IP Camera

This step mainly consists in testing and validating the complete system. In fact, the complete chain of execution is to be implemented to verify that all components of the system work and communicate together as planned. Please, see Figure 1.5 on page 7 to get the block diagram of this part.

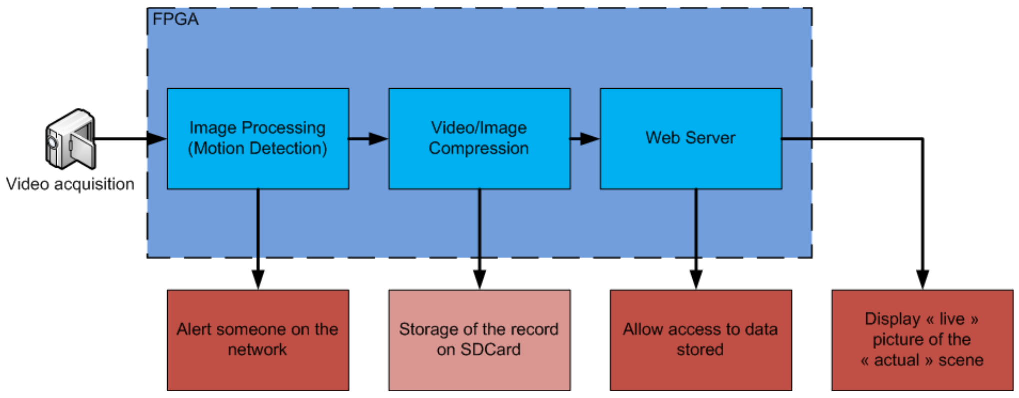

FPGA
Video acquisition
Image Processing (Motion Detection)
Video/Image Compression
Web Server
Alert someone on the network
Storage of the record on SDCard
Allow access to data stored
Display « live » picture of the « actual » scene

Figure 1.5: Global block diagram of the project: all blocks are added to build up the full system to answer the hypothesis. There are also some external event to perform.

1.5 Tasks

This section describes all the required tasks to implement the previous step-by-step presentation of the project. The following described tasks do not follow the previous order, as the implementation of certain tasks is more critical for the step-by-step implementation. Figure 2, page 88, presents the GANTT diagram of this project, using the following tasks to plan the project schedule.

1.5.1 Capture image with the camera sensor

This part is essential as it is the first step of the global process. It simply consists of acquiring the image with the camera sensor and made it available to the rest of the system.

Study

The aim of this task is to understand how the camera sensor acquires and sends data through its inputs and outputs.

Implementation

The aim is to implement the data acquisition from the sensor.

Test

The aim is to verify that the implementation is working according to the requirements (detailed in Chapter 3) to detect potential problems and correct them.

1.5.2 Display image on a screen

This part is a feature and a validation test for the implementation. In fact, it allows the user to see what is currently being captured by the camera sensor.

Choice of the display

Nowadays, it exists several way to display on a given screen (VGA, LCD, etc.). Therefore, the study and selection of one of those types has to be done for the implementation.

Implementation

The aim is to implement the controller for the selected display screen type.

Test

The aim is to verify that the implementation is working according to the requirements (detailed in Chapter 3) to detect potential problems and correct them.

1.5.3 Record captured video

This part consist of recording the video, to make it available for compression and storage.

Study

The study consists in understanding how to record a video from the sensor.

Implementation

The aim is to implement the recording functionality in the system by storing several frames on the memory.

Test

The aim is to verify that the implementation is working according to the requirements (detailed in Chapter 3) to detect potential problems and correct them.

1.5.4 Store recorded video

This part is one of the major goal of the project: the storage of data, previously recorded. Also, the reduction of the medium size (memory card instead of tape) is another way to reduce the size of the storage.

Storage choice: SRAM? SDRAM? SD-Card? Other?

Nowadays, there is a lot of available memory types. The aim of this task is to study ans select the one that will fulfil our requirements.

Implementation

The aim is to implement the storage functionality in the system. It therefore requires to get a controller to access the selected memory card and create the blocks to use this controller.

Test

The aim is to verify that the implementation is working according to the requirements (detailed in Chapter 3) to detect potential problems and correct them.

1.5.5 Access recorded video

The aim of this task is to build a mechanism for providing the user with a convenient access to the data stored.

Study

The aim is to study how to make a data stored available on an Ethernet network.

Implementation of a softcore processor

As the Ethernet protocol is easier to implement at a higher abstraction level, implementing a softcore processor on the FPGA helps using the Ethernet controller of the system. A soft microprocessor (also called softcore microprocessor or a soft processor) is a microprocessor core that can be wholly implemented using logic synthesis. It can be implemented via different semiconductor devices containing programmable logic (e.g., FPGA , CPLD). A more detailed definition of a softcore processor is given in Chapter 3.

Implementation of a web-server

Once the softcore processor implemented, the web-server executing on it makes the data available on the network.

Test

The aim is to verify that the implementation is working according to the requirements (detailed in Chapter 3) to detect potential problems and correct them.

1.5.6 Event detection algorithm

The aim is to record only useful data. Therefore, the implementation of an event detection algorithm is one of the main goal of this project.

Study

As it exists various event detection algorithms, the goal of this task is to study them and select one to implement.

Calculation of metrics

Once the algorithm selected, it has to be profiled (to get the full understanding of the algorithm) in order to be implemented it in the best way.

Hardware/Software co-design

With the previous task done, the algorithm is partitioned between hardware and software parts to have the best implementation.

Implementation

The aim is to implement the algorithm according to the previous results. That means, use the draft c-code algorithm and adapt it to the softcore processor.

Optimization

Once the algorithm implemented, it may requires optimization of some parts, such as bottlenecks, to have a faster implementation. The main point of this optimization is the execution time. Other optimization, such as power, area, etc., are not considered.

Test

The aim is to verify that the implementation is working according to the requirements (detailed in Chapter 3) to detect potential problems and correct them.

1.5.7 Video compression

When the video is recorded and stored, it would be beneficial to optimize the space used by it. To do so, a video compression algorithm can be used. It will enhance one of the project's goals: saving space for data storage.

Study

As it exists various video compression algorithms, we have to study and select one of them. In facts, as explained in Chapter 2, depending on the requirements of the final video, the range of possible algorithms is quite wide.

Calculation of metrics

Once the algorithm selected, it has to be profiled in order to be implemented it in the best way.

Hardware/Software Co-Design

With the previous task done, the algorithm is partitioned between hardware and software parts to have the best implementation.

Implementation

The aim is to implement the algorithm according to the previous results.

Test

The aim is to verify that the implementation is working according to the requirements (detailed in Chapter 3) to detect potential problems and correct them.

1.6 Structure of the document

The rest of the report is organised as follows: **Chapter 2** is an overview of the project using a meta-model. **Chapter 3** is the system analysis and design, while the **Chapter 4** details the analysis and design of the motion detection algorithm. **Chapter 5** deals with the actual implementation by describing each step. **Chapter 6** is about the testing and give the main results of our experiments. Finally the **Chapter 7** conclude this report and propose several future work.

Chapter 2

Design Models

This chapter describes how the use of design models can facilitate the process of fulfilling the goals of the project. Firstly the A^3 paradigm is explained. Then, the Rugby Meta-Model is introduced. Finally, the details of the actual description of the project using the Rugby Meta-Model and A^3 paradigm are provided. Both of these models are used because they are complementary for a complete implementation, as explained in this chapter.

2.1 Description of the A^3 paradigm

The acronym A^3 stands for Application-Algorithm-Architecture (as explain in [tea07]) which corresponds to the three fundamental domains of the hardware design process. The A^3 model is used to describe the design flow, its basic stages and transitions between them. Figure 2.1 (on page 14) presents the model with generic references in each of the three domains. The three domains and corresponding design flow stages are as follows:

- Application domain: characterizes the system from a purely behavioural point of view. System analysis in this domain can be expressed by the phrase: "*What are the system's functionalities, properties and requirements?*". In the design process this corresponds to a high-level functional specification of the system as well as establishing design constraints.
  A transition to the next domain is achieved by detailed functional analysis leading to selection of specific algorithms.

- Algorithm domain: concerns algorithms used to realize specific functionalities of the system. System analysis in this domain can be expressed by the phrase: "*How are system functionalities realized?*". As most of the time numerous algorithms are exploited in a single application, system design tasks in the algorithmic domain are usually related to the evaluation of multiple algorithms. This can be done either by analysis of theoretical models or performing numerical simulations. Such evaluation leads to the verification of algorithm correctness as well as obtaining measures of complexity, robustness and various other metrics. This eventually leads to select a set of algorithms most suitable for realizing the application.

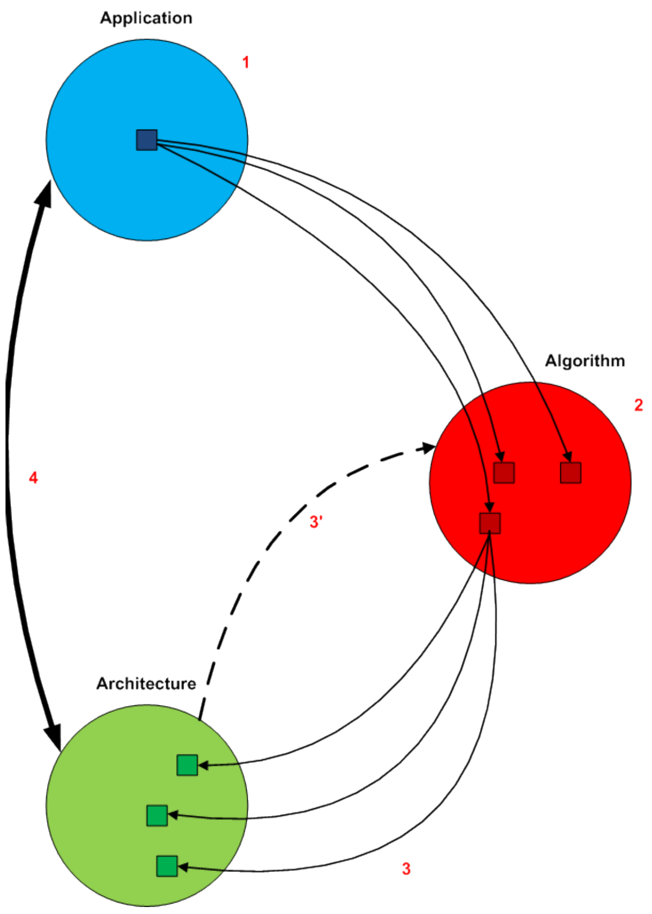


Figure 2.1: A^3 paradigm design flow: the first step is to define the purpose of the application (1). The second one is to explore and simulate the algorithms that fulfil the requirements of this application (2). Then, the third step is to select the best architecture to implement those algorithms (3). The implementation on an architecture can lead to modification to the algorithm (3'). And last but not least, the fourth step is to check that the actual implementation fulfil the requirements of the application (4).

A transition to the third design domain corresponds to the process of Design Space Exploration (DSE) of which purpose is to map the selected algorithms onto an hardware or software architecture which matches most application requirements and constraints.

- Architecture domain: Characterizes type, number and organization of Processing Elements (PE) used for system implementation as well as determines the scheme of mapping selected algorithms to respective PEs. System analysis in this domain can be expressed by the phrase: "*How are the algorithms implemented, what hardware/software resources are used and how are various hardware/software modules cooperating?*". The architecture domain is strongly related to the final stage of the design process, namely system implementation. In this section of the design flow, algorithms selected for realization of system functionalities are mapped to specific hardware/software structures with consideration of requirements and constraints from the application domain. Both architecture selection and algorithm mapping are performed with consideration of various metrics such as performance, hardware/software complexity, code size, cost, power consumption and development time. The DSE process can also involve a feedback path to the algorithm domain leading to re-specifying the set of algorithms used for realizing system functionalities. Such situation can occur when the algorithms selected originally are either infeasible or inefficient to implement on any available architecture resulting in a conflict between system requirements and availability of design solutions.

"Successful system implementation enables to characterize the application in terms of both qualitative and quantitative metrics such as application feasibility, achievable performance, hardware requirements and cost. In order to achieve this, a complete system design flow is followed as described in the following section ."[Por07]

The A^3 design flow summary:

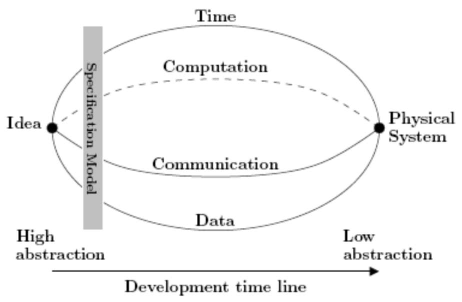
1. Select the algorithm(s) for the application
2. Simulate the algorithm(s)
3. Select the architecture to implement the algorithm(s)
4. Model the algorithm(s) on the architecture
5. Design Space Exploration
6. HW/SW Co-design
7. Compare results of the implementation to the requirements

As the A^3 paradigm is very generic, it is complemented by the use of a more detailed and precise design model, the Rugby Meta-Model. In fact, as explain in the next section, the Rugby Meta-Model is closer to the implementation and is very designed for hardware and software co-design. Moreover, the A^3 is a paradigm, which means mostly a way of thinking or designing a system. The Rugby is more an implementation model to follow from the beginning of the project to its end.

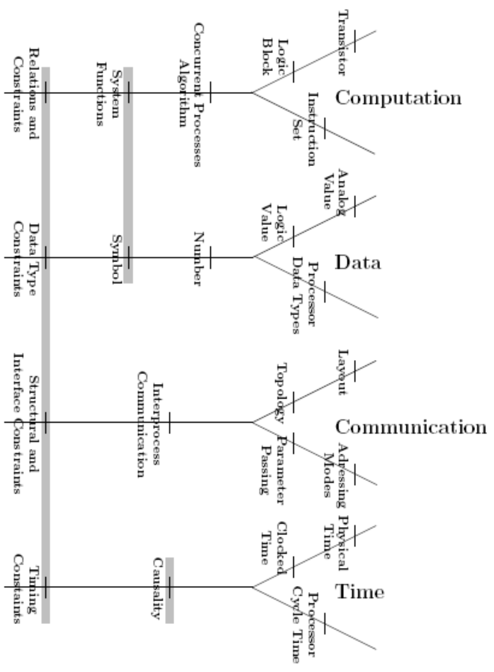
2.2 Description of the Rugby Meta-Model

Models like the Y-chart [KG83] & [Gaj88] and the Rugby Meta-Model [HJK99] make it possible to represent the abstraction levels and domains of the design model under consideration and the transitions between the abstraction levels when moving from one model to the next.

The Y-chart model has three domains: Structural, Behavioural and Physical. However, it has no explicit representation for time, data and communication. Especially a representation for communication is necessary in heterogeneous systems having several processing elements like for example a GPP and an ASIC communicating. The Rugby Meta-Model (as described in [HJK00]) on the other hand has



(a) The Rugby meta model with the four domains: Communication, Data, Computation and Time. The system start with an idea and ends with a physical system. The development time progresses from right to left



(b) Elaboration of the four domains of the Rugby meta model. The level of abstraction decrease and the development time increase from left to right. At some stage in the development of the system, a decision about the Hardware & Software partitioning is made. From that point, the development of the system is split into two.

Figure 2.2: The Rugby meta model and elaboration of the four domains. As an example the specification model of the methodology of a system design language is shown in the Rugby meta model together with the level of abstraction in the four domains [HJK00].

explicit representations for both time and communication, since it has four domains, namely Communication, Data, Computation and Time. It has therefore been chosen to utilize the evaluation properties of the Rugby meta model to visualize the abstraction levels of the different models in the design. The Rugby Meta Model is shown in Figure 2.2(a) (on page 16) and an elaboration of the four domains is shown in Figure 2.2(b) (on page 16). As an example the specification model of the methodology of a system design language has been represented by the Rugby Meta-Model. The specification model is at an early stage of the design process, hence it is on a high level of abstraction. As a system has a number of abstraction levels, Figure 2.3, on page 17 shows the different typical abstraction layers of a generic system. Those abstraction levels are used in the Rugby Meta Model.

Having described the problems of designing a system for a given application and the means to solve the problem in a structured manner, the last section of this chapter introduces how those terms are used in this project.

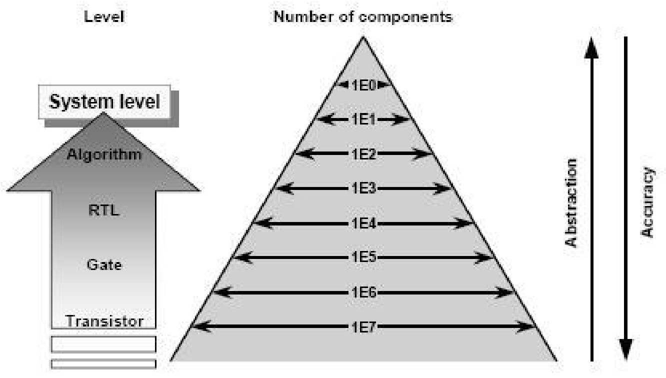


Figure 2.3: The different typical abstraction levels of a generic system. It goes from the top, with the highest abstraction layer, to the bottom with the lowest [Döm03].

2.3 This project described using the design models

In this section, the A^3 paradigm is applied to this project and then the Rugby Meta-Model is applied. For each of those two design models, each domain is detailed.

2.3.1 The A^3 applied to our project

This section is the application of the section 2.1 to this project. For the A^3 paradigm, Figure 2.4 (on page 18) is the specific flow of the A^3 paradigm applied to this project. The application of each domain is given in the following paragraphs.

Application

The goal is to make an intelligent IP surveillance video camera. It implies that the system must be secure and available all the time for the surveillance. There is an “IP part” because the video camera is connected to an Ethernet network (through the Ethernet adapter), using the IP protocol. This feature provides the owner of the system a fast and convenient access to data. The other main part is the “intelligence”. As previously explained, only useful data are stored. This implies using an image processing algorithm to record only video when something happens. As a feature, what is being captured is also displayed. It provides the validation of the recording and storage parts.

One of the constraints to be met is to be as much as possible in a real-time system. Which means that the system built is a real-time system, but with soft timing condition. A second point is to have data stored during the process in the correct format to be useful for the following steps of the process (i.e. in raw from the camera sensor, in RGB & grey scaled for the algorithm, etc.). An other one is the security of the system. In fact, as it is a surveillance system, the data stored need to be secured when they are written on the final memory block, and only the system should be allowed to write data on the selected storage medium.

For further explanation and details, please refer to Chapter 3, on page 23.

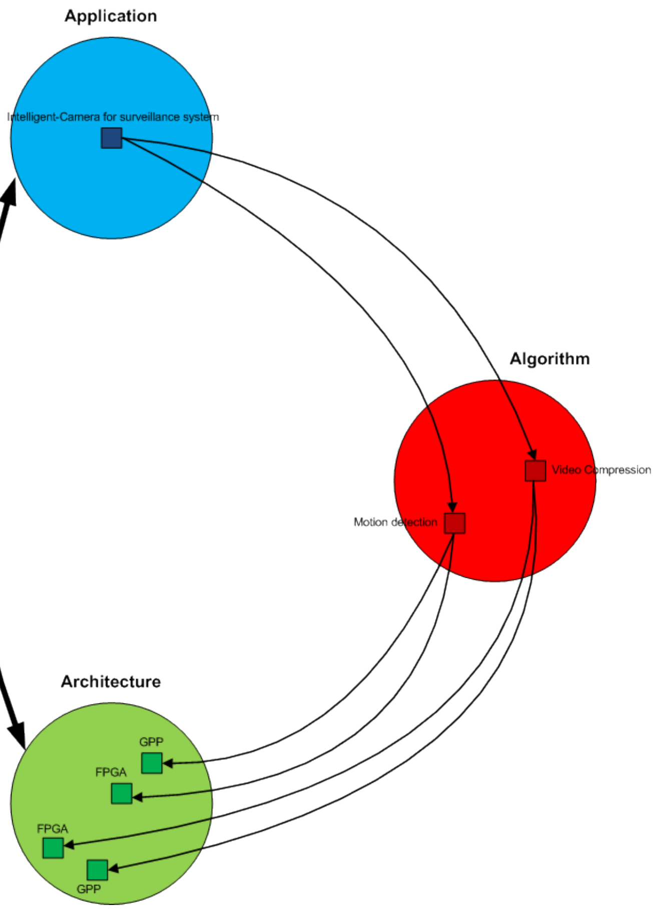


Figure 2.4: A^3 paradigm applied to this project: there is one application (intelligent camera for video surveillance) using two algorithms (one for event detection and one for video compression). For each of those algorithms, the architecture is based on a General Purpose Processor (GPP) and on an FPGA . The GPP in this project is a softcore processor implemented on the FPGA .

Algorithm

In order to meet one of the main functionalities of the system, the "intelligence" part, an image processing algorithm is to be implemented. It exists several algorithms and several ways to implement those algorithms. The main categories of event detection algorithms are motion detection, face recognition and shape recognition. A motion detection algorithm was chosen because it seems to cost less in development time, its complexity is rather simple and more accurate for this project than shape or face recognition algorithms. More over, the implementation of this algorithm is just to verify the feasibility of the proposal solution. Also, the algorithm is used to detect any kind of event occurring in the monitored room. Therefore, face recognition and shape recognition are too specific to be used. The main method for motion detection is to make a background subtraction methodology. The motion detection algorithms are presented in Chapter 4 (page 43), along with the details of the choice of algorithm.

Another functionality is to compress the final stored video to save more space. This can be done by implementing a video compression algorithm. Again, there are many algorithms and implementation methods to explore. The main algorithms are based on different constraints or requirements. The first requirements is the quality of the final compressed video. The other one is the disk space usage. In fact, those requirements are antagonistic. Therefore, one can decide that the final video needs to be the smallest possible (in terms of disk space usage), but it implies that the quality has to be very low. Between those two extrema, there is a wide range of algorithms that try to fit the best optimum solution, to have the best quality for the smaller disk space usage.

Architecture

It exists mainly three kinds of architectures: embedded General Purpose Processors (GPPs), Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs). For the selected algorithms, the main characteristics of those architecture (pros and cons) are given in Table 2.3.1, on page 19. Typically, embedded GPPs are the most common processors, offering a lot of flexibility, DSPs are mainly used in repetitive signal processing application (e.g. number crunching) and FPGAs offer parallelism and reconfigurability and are more and more used.

Processors	Pros	Cons
embedded GPP	speed, development time, floating point support	limited parallelism (depending on the number of core)
DSP	some parallelism, fixed or floating point support	specific for signal processing calculation
FPGA	massive parallelism, reconfigurability	more difficult to implement than software (time consuming), floating point is very expensive

Table 2.1: Pros and cons of processors available for this project: each of those have there own capabilities. The FPGA have also the possibility to include DSP elements.

For further explanation and details about the selected architecture, please refer to Chapter 3, on page 23. For the main details about the actual implementation on the selected architecture, please refer to Chapter 5, on page 55.

2.3.2 The Rugby Meta-Model applied to our project

The application of each domain is given in the following paragraphs. The global meta-model applied to this project can be seen in Figure 2.5, on page 20.

Computation

The computation domain starts from the relations and constraints level to go to the transistor level, for the hardware part, and the instruction set, for the software part. For this project, the first level of abstraction is described in Chapter 3, on page

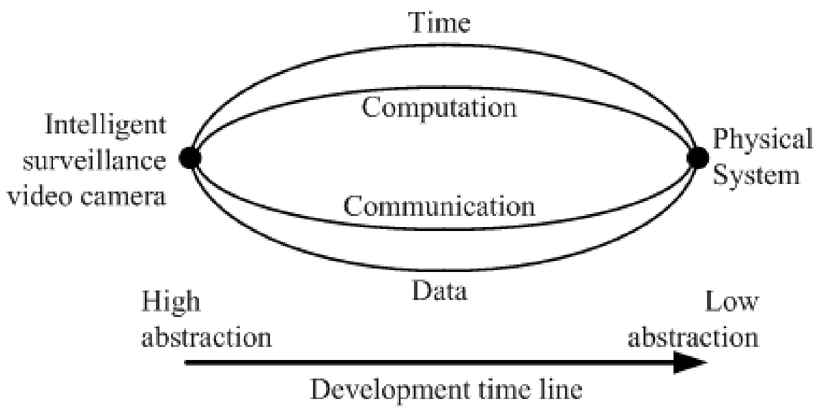


Figure 2.5: Rugby meta-model applied to our project.

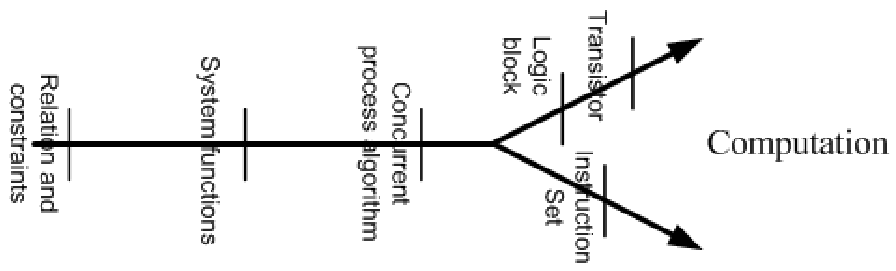


Figure 2.6: The Computation domain applied to our project.

23. The system functions of the project are described in the following chapter. For the hardware part, the abstraction level stops at the logic blocks level as an already existing FPGA is used. Concerning the software part, as a softcore processor is used (please refer to Chapter 1, page 3), the instruction set is the one of this processor. The diagram of this domain is shown on Figure 2.6, on page 20.

Communication

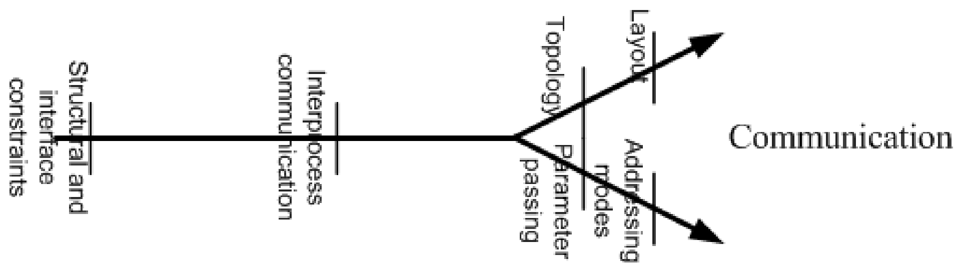

Structural and interface constraints
Interprocess communication
Topology
Layout
Parameter passing
Addressing modes
Communication

Figure 2.7: The Communication domain applied to our project.

The communication domain starts with the structural and interface constraints level to finish at the layout level, for the hardware part, and the addressing nodes, for the software part. For this project, the first level of abstraction is described in Chapter 3, on page 23. The interprocess communication is core part of the project as the algorithms used require a lot of data. Also, memory accesses are very important because the internal memory of an FPGA chip is too small for image processing. Therefore, the better the processes communicate, the fewer time is spend accessing the memory. Concerning the parameter passing, for the software, and the topology of the hardware implementation, the following chapter describes how the problems have been solved. The diagram of this domain is shown on Figure 2.7, on page 20.

Data

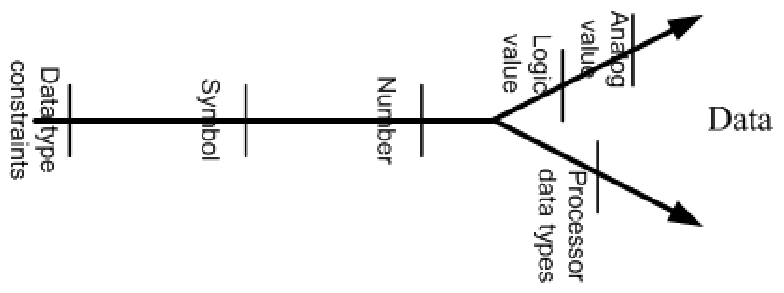


Figure 2.8: The Data domain applied to our project.

The Data domain starts with the data constraints level to finish at the analog value, for the hardware part, and the processor data types for the software part. For this project,the first level of abstraction is described in Chapter 3, on page 23. The symbols and number used are described in the following chapters, as it was very important to define them well to improve and enhance the communication domain. The analog value abstraction level was only used for testing and experiments purpose. For the hardware implementation, only logic values were used. For the software part, the processor data types used were those of the Nios II softcore processor. The diagram of this domain is shown on Figure 2.8, on page 21.

Time

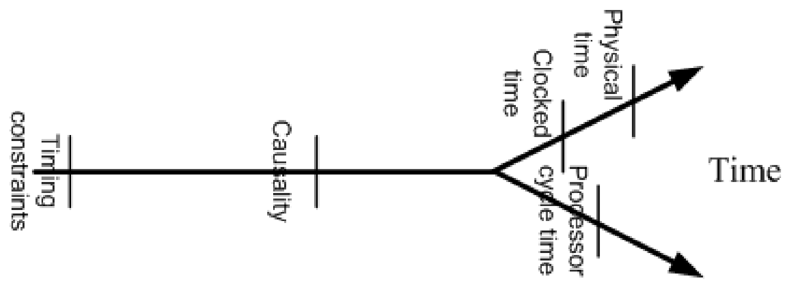


Figure 2.9: The Time domain applied to our project.

The time domain starts with the timing constraints level to finish at the physical time, for the hardware part, and the processor cycle time for the software part. For this project,the first level of abstraction is described in Chapter 3, on page 23. The main point to be noted is that the frequency of the Nios II softcore processor is 100 MHz. The physical time level of abstraction was only used for testing and experiments purpose. The diagram of this domain is shown on Figure 2.9, on page 21.

Chapter 3

System Description

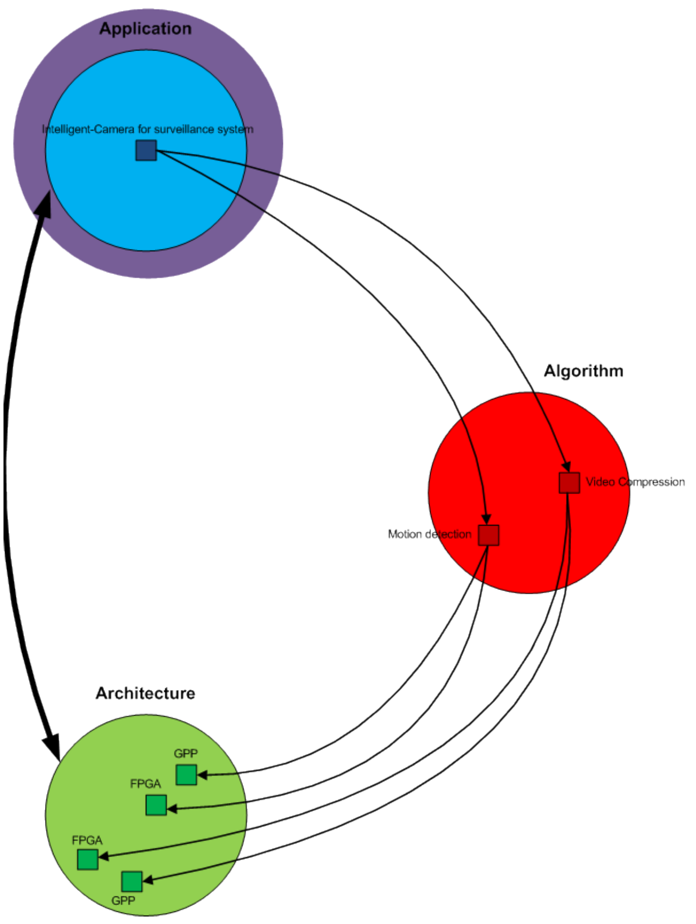
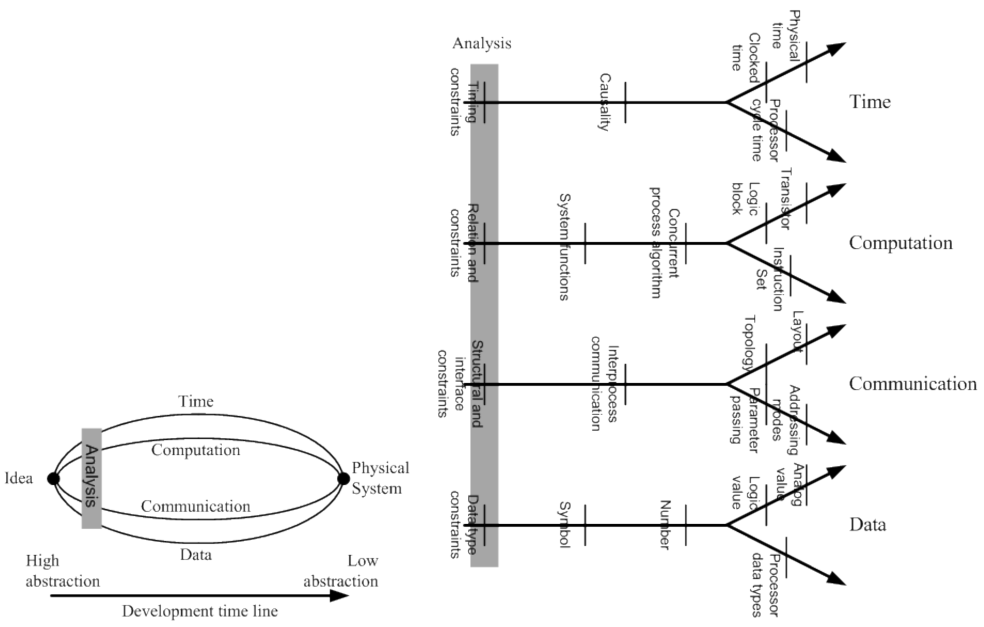


Figure 3.1: Location in the A^3 paradigm, highlighted by the purple circle.

This Chapter presents the system analysis and the system design. To do so, a first section explains the constraints of this project. Then, it gives the needs required by this project. The platforms and programming languages are then shortly reviewed, before dealing with the softcore processors (defined in 3.2.2) and embedded



(a) Global location (b) Elaborated location

Figure 3.2: Location in the Rugby Meta-Model, highlighted by the grey bars on (a) and (b).

OS (defined in 3.2.4) possibilities. Finally, the last section details and explains the choices made to meet those constraints and requirements.

3.1 System constraints analysis

This section gives an overview of the different constraints of the system.

3.1.1 Use of FPGA

FPGA definition

The first constraint is the choice of a technology. In this project, one of the goals is to try to evaluate the feasibility of implementing a motion detection algorithm onto a FPGA platform.

FPGA stands for Field-Programmable Gate Array and refers to a semiconductor device containing programmable logic components called "logic blocks" including memories and programmable interconnects ([PT05]).The designer can program the logic blocks and the interconnects like a one-chip programmable breadboard.

In the design flow of a new product, the early designs are using FPGAs then migrated into a fixed version that more resembles an ASIC. ASIC means Application-Specific Integrated Circuit and refers to an integrated circuit customized for a particular use. To be configured, the FPGA has to be described by a logic circuit diagram or a source code using a hardware description language (defined in 3.5.1).

Table 3.1.1 (page 25) summarize the pros and the cons of the FPGAs compared to ASICs.

Pros	Cons
Time-to-market shorter than ASICs	Slower than ASICs
Development cost cheaper than ASICs	More expensive for mass productions than ASICs
Lower non-recurring engineering costs than ASICs	More power consuming than ASICs
Re-programmable in the field	Difficulty to program FPGA platform
Performance/power relative to standard processors and DSPs	

Table 3.1: Pros and cons of FPGA compared to ASIC: despite their lower performances compared to ASICs, FPGAs are more flexible and easy to develop

The requirements specified during the early design step of the project are the following:

- Fast processor for processing parallel tasks
- Support/documentation/examples available and up to date
- Free of charges or not expensive price
- Input/output for the camera sensor
- USB wire for programming/debugging the FPGA
- Possibility to add a softcore processor
- Possibility to add an embedded OS
- SD card or MMC slot to add extra memory (SD: Secure Digital, MMC: Multimedia card)

3.1.2 Time constraint

The time constraint follows two different axis. On the one hand, the image processing part must be fast enough to start the recording as soon as a motion is detected. On the other hand, the storage process must be fast enough to store a picture after another picture without freezing. Those two aspects depend on the platform used and the efficiency of the algorithm. This time constraint is related to the data constraint because the quickness of execution of the image processing depend on the temporary storage of pictures.

3.1.3 Data constraints

For motion to be detected, pictures need to be processed and analysed. It already underlines two kinds of constraints. The first one is that the system needs a memory with fast access for the process to be in real time (or at least as close to). The other constraint is about the type of the data : is it possible to perform the algorithm on the raw data or do this raw data need to be transformed into a known picture type. After the detection of a movement, when the system starts to record, the video must be stored. It means that the pictures has to be stored quickly enough as they are arriving (less than 0.1s).

As a consequence: the system must have a memory sized big enough to store a decent amount of video recorded. The data stored must fit to the performances of the platform and need not to be too important. Then, when a video is recorded, it need to be available in the FPGA memories(internal or external) for the external users of the system. The success of the motion detection and the storage of the pictures and videos are dependent to the flow of pictures and their dimensions.

3.1.4 Other constraints

Last but not least, some other constraints need to be mentioned. One concerns the security. Indeed, as this motion detection camera could be used as a surveillance system, it needs to be secured. One other aspects concerns the power consumption. This aspect is mainly depending on use context of the system. This aspect is out of the scope of this report because not all the FPGA have been optimized for power consumption and the choice of FPGAs are available for the system was quite limited to those available in the AAU embedded lab (Terasic DE2 with Altera Cyclone II or Stratixand Celoxica RC203 with Xilinx Virtex).

The constraints applied to this system can change deeply depending on the final purposes. Indeed, the performances required and the algorithms needed are not the same depending if the objective is to detect a movement, to count peoples or to recognize forms.

3.2 Sketching the initial architecture

3.2.1 Hardware/software co-design

One of the key notion in this project is Hardware/Software Co-Desisgn [Wol03] which is used through all the development phases. This concept has been created a little more than a decade ago in response to the raise of the embedded systems, the complexity and the heterogeneity of those systems. As an example, already in 1998, 98% of the processors were found in embedded systems. Embedded systems are single-functionned, tightly constrainted, reactive and realtime systems. When designing a embbedded system, the partitioning between hardware and software is a crucial point. Hardware has better performances but is more difficult to code contrary to software.

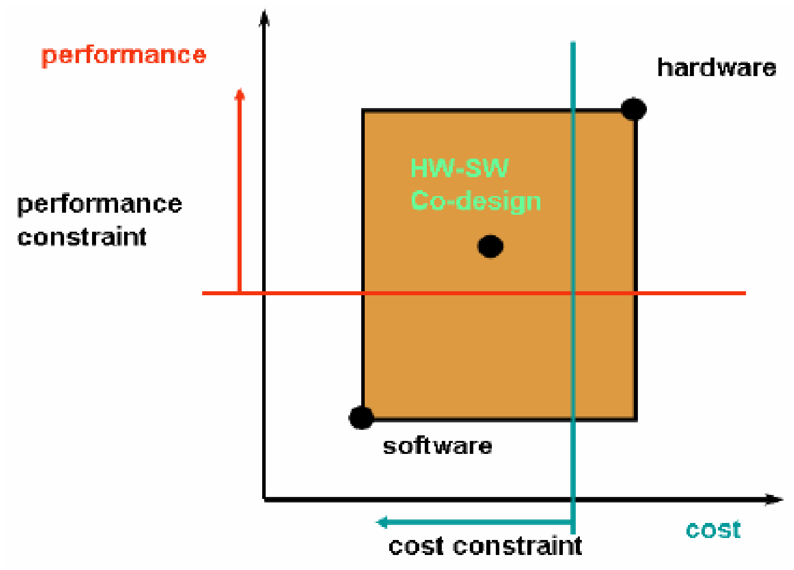


Figure 3.3: Hardware/Software Co-Design to reach the constraints of performances and costs [Koc].

Figure 3.3, page 27 [Koc], enlightens that to deal with time and performance constraints, a designer has to make choices about partitioning his algorithm between hardware and software part. This partitioning and tasks scheduling are the two main dimensions of the Co-Design. Hardware/Software Co-Desisgn can be summarized in the following way:
"Hardware/Software Co-Desisgn provides a way of customizing the hardware and the software architectures to complement one another in ways which improve system functionality, performance, reliability, survivability and cost/effectiveness." [SFL85]

Hardware/Software Co-Desisgn was used in this project to save some time for development and to make the system more efficient and faster because both hardware and software development have been used in this project and separate them as late as possible in the conception cycle, aiming at unifying the specification that includes both hardware and software. The reasons and the partitioning used in this project are described in the appropriate section A.

3.2.2 softcore processor definition

A softcore processor is a processor implemented into a reprogrammable system like a FPGA . It is a System on Programmable Chip or SoPC.

A softcore processor is a very flexible architecture, it can be reconfigured at any time, contrary to a hardcore processor whose core has it own non- reprogrammable chip. A softcore processor can be adapted to the material constraints (performances, resources, power consumption,...). However, softcore processors performances are inferior to the hardcore ones but a softcore processor is easier to maintain and can implemented into an ASIC (Application Specific Integrated Circuit).

3.2.3 The need for a softcore processor

Using a softcore processor has many advantages:

- Higher level of abstraction (the softcore processor transforms the developer's software code into hardware language)
- Possibility to use higher level languages as C like languages
- Better management of the memories
- With a softcore processor , the conception is going from hardware to software
- Easily reconfigurable compared to other processors

3.2.4 The need for an embedded OS

The following definitions are necessary to clarify what is an embedded OS.

The software component of a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer is defined as an operating system. This Operating System (OS) acts as an host for application programs that are executed on the machine. As an host, one of the purposes of an operating system is to handle the details of the operation of the hardware.

An embedded operating system is an OS for embedded systems. These OS are specialized, compact and efficient. They do not have all the functions the non-embedded computer have because they do not need them. The embedded OS are designed to be operated on real-time operating systems.

The advantages given by the embedded OS are:

- Higher abstraction level (possibility to use object-oriented languages like python)
- Better memory management
- Better network management
- Some web server are already coded and available
- Possibility to do parallelism (most processors does not allow software parallelism)
- Better management of the TCP/IP stack

Once a softcore processor has been implemented on the board, another step could be the implementation of an embedded OS. The available embedded OS are depending on the platform used. Like for the softcore processors, some of them can be free and/or open-sources but none of them has been created by the companies building the platforms (XILINX or ALTERA).

3.3 Possible options to satisfy the system constraints

In the following sections are detailed all the feasible solutions that can be explored in all the relevant domains:

- FPGA platforms
- Programming languages softcore processor
- Embedded OS

After these presentations the options selected for this project are shown in 3.8.

3.4 FPGA platforms

Here is list of the FPGA platforms available in the embedded laboratory of AAU that could be used for this project. They are supposed to fit the constraints declared in 3.1: resources, input/output for camera sensor, possibility to add a softcore processor or an embedded OS and so on.

3.4.1 Altera Cyclone II on Terasic DE2 board

This is a common solution using classic education board with a medium sized FPGA ([Cor06a]). It is supposed to represent the middle point between complexity, performances. One of the advantages of this kind of board are the examples provided with it and the ones which can be found on the Internet. The relevant DE2 main characteristics are :

- Altera Cyclone II processor (2C35) with 35000 LEs
- 4 MegaBytes SDRAM memory
- 512 kiloBytes SRAM memory
- 8 MegaBytes flash memory
- VGA output
- Ethernet Connectivity
- softcore processor Nios II II created by Altera for its FPGAS

3.4.2 Altera Stratix

The Stratixdevelopment board ([Cor08]) is a better version of the previous one: faster FPGA & larger memory. The main differences are :

- The Stratixprocessor (EP1S10) with 10570 LEs
- The SRAM memory is 1 MegaBytes
- The SDRAM memory is 16 MegaBytes

3.4.3 Xilinx Virtex II on Celoxica RC203 board

Xilinx is the other company that is leading the FPGA market. Xilinx proposes also main development boards similar to the Altera ones. The one available in the embedded laboratory is the Celoxica RC203 featuring Xilinx Virtex II FPGA ([Cor06b]). It has the following key features:

- Xilinx XC2V3000-4 Virtex II processor
- 2 Banks of 2Mbyte each SRAM (4Mb total)
- Video In/Out
- Audio In/Out
- Smartmedia socket
- 10/100 Ethernet
- Parallel Port
- RS232 port
- PS2 Keyboard and Mouse ports

3.5 Programming and simulation languages

In this section are described the different programming and simulation languages that can answer the needs of the project. Those languages can be differentiated into categories because all can not be used in the same conditions and for the same goals. The Hardware Description Language (HDL, 3.5.1) are the languages directly available on the FPGA platform. Then there are the System Description Language which do not also need any added layer on the FPGA platform. After that are the high level languages which needs a softcore processor or an embedded OS to work. The conclusions and the choices made for this project concerning the programming languages are presented in section 3.8

3.5.1 Hardware Description Languages (HDL)

Definition

In electronics, an Hardware Description Language or HDL is a computer language for formal description of electronic circuits. So it can describe an operation made by a circuit, its design and organization. It can also be simulated and tested to verify this operation.

A Hardware Description Language is written in standard text-based and describes the behaviour circuit structure of an electronic system (for example FPGA). Differing from software programming languages, an HDL's syntax and semantics take in account the time and the concurrency between processes. The languages describing the circuit connectivity between a classified hierarchy of blocks are netlist languages.

Executable specifications for hardware can be written with HDLs. The simulation program takes also the time in account and allows the programmers to model a piece of hardware before it is created physically. With the hardware descriptions, a software program called a synthesizer can infer hardware logic operations from the language statements and produce an equivalent netlist of generic hardware primitives to implement the specified behaviour on the platform.

Designing a system in HDL is generally much harder and more time consuming than writing the equivalent program in a C like language. To solve this difficulty, there has been much work done on automatic conversion of C code into HDL.

Verilog

Verilog ([TDMP85]) is a Hardware Description Language used to model electronic systems like FPGAs and ASICs. The language is designed for the design, verification, and implementation of analog, digital or mixed-signal circuits at various levels of abstraction. The main characteristics of this language are :

- Syntax similar to the C programming language
- Case-sensitive
- Preprocessor with C tools
- The major control flow keywords, such as "if" and "while", are similar to C
- The formatting mechanism and language operators are also similar to C

"*One of the key concept is the non strictly sequential execution of statements. A* Verilog *design consists of a hierarchy of modules. Modules are defined with a set of input, output, and bidirectional ports. Internally, a module contains a list of wires and registers. Concurrent and sequential statements define the behaviour of the module by defining the relationships between the ports, wires, and registers. Sequential statements are executed in sequential order within the block. But all concurrent statements and all begin/end blocks in the design are executed in parallel, qualifying* Verilog *as a dataflow language.*" [Wikd]

VHDL

VHDL (VHSIC Hardware Description Language)([Air83]) is commonly used as a design-entry language for field-programmable gate arrays and application-specific integrated circuits in electronic design automation of digital circuits. VHDL was originally developed at the best of the US Department of Defence in order to document the behaviour of the ASICs that supplier companies were including in equipment. That is to say, VHDL was developed as an alternative to huge, complex manuals which were subject to implementation-specific details.

Handel-C

Even if like the previous languages Handel-C ([Cel05]) is made to enable the compilation of programs into synchronous hardware (FPGA and ASICs), its case is a bit different. Handel-C is more a programming language aimed at compiling high level algorithms directly into gate level hardware than a HDL. It is a subset of C, with non-standard extensions to control hardware instantiation and parallelism. It contains all the necessary features to describe complex algorithms.

3.5.2 System Design Languages (SDL)

SDL

A System Description Language (SDL) is more than a HDL because it can be used for designing the whole system, unlike HDL which are used only for coding blocks. Concerning their characteristics, the SDLs are:

- Made for System-on-Chip design
- Used to move to Higher levels of abstraction
- Used to design a complete system

The programming of the System Description Language is made on the development computers. Then, after a co-design step, the code is divided between hardware and software by the compiler and coder. That is one of the main advantage of this category of languages: hardware and software, all the system is coded with a same language which is not low level (time saving and productivity increasing).

The main SDLs with their main characteristics are describing in the following subsections.

System-C

Unlike the VHDL or Verilog languages which are only material description languages, System-C is higher level because it allows the design of systems at a behavioural level. System-C is not really a complete, new language but more an ensemble of C++ classes introducing the necessary concepts for designing material (e.g. the concurrency concept) ([CG05]). System-C is able to design material systems, softwares, mixed. System-C is adapted to conception of SoC (System on Chip) systems. It also allows to simulate and to create an implementable representation.

In System-C , the system is a hierarchy of objects, i.e. modules imbricated and/or processes. These modules are communicating by channels establishing links between ports of different modules.

Impulse-C

Impulse-C ([PT05]) is based on standard ANSI-C supporting parallel programming and provides a software level of abstraction in particular for the development of applications targeting FPGA devices. The advantages of the ANSI C for programming are :

- It supports standard C development tools
- It supports multi-process partitioning
- It is compatible with a wide range range of FPGA-based computing platforms

Impulse-C is as software-to-hardware compiler:

- It optimizes C code for parallelism
- It generates HDL, ready for FGPGA synthesis
- It also generates hardware/software interfaces

Spec-C

Spec-C ([Döm03] is another SDL, also created a Irvine University (California) like System-C but it is considered as less general as its main concurrent System-C . Its main characteristics are:

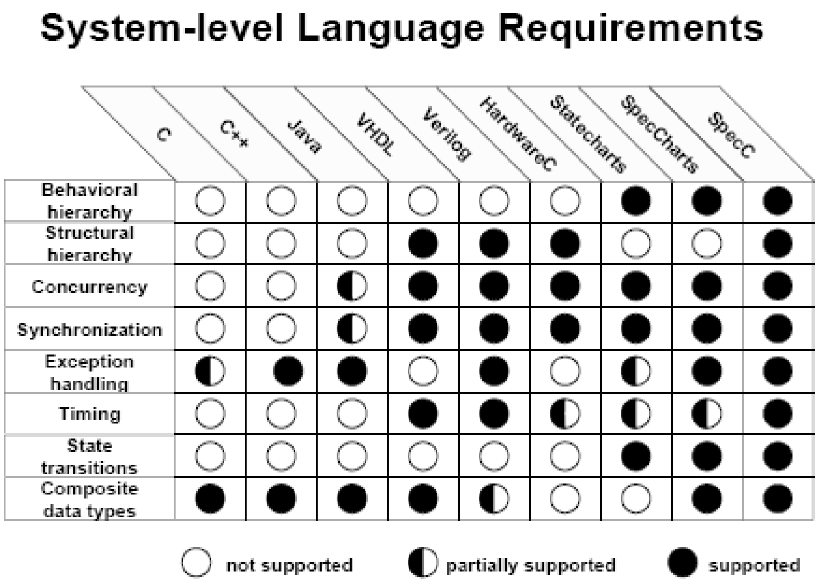
- Its execution behaviour (Validation and simulation)
- Its synthesis behaviour (possibility to implement in software and hardware)
- Its modularity (Hierarchy)
- It supports all the embedded systems concepts
- It is a real language, not only a class library

Matlab and Simulink

Matlab is both a programming language and a development environment([inc08c]). It is used for numeric calculations, algorithmic and project development. In the domains of the FPGA , Matlab is used with Simulink as a multi-domain simulation platform. It is also a dynamic systems design platform. Simulink allows design, simulation, implementation and control of communication and signal analysis systems.

The Simulink environment can design a system, simulate its behaviour, break up the design before its implementation. Simulink can simulate numeric, analogical or mixed components.

A summary of a comparison between all languages able to be used in FPGA programming is found in Figure 3.4, on page 34.



System-level Language Requirements

	C	C++	Java	VHDL	Verilog	HardwareC	Statecharts	SpecCharts	SpecC
Behavioral hierarchy	○	○	○	○	○	○	●	●	●
Structural hierarchy	○	○	○	●	●	●	○	○	●
Concurrency	○	○	◐	●	●	●	●	●	●
Synchronization	○	○	◐	●	●	●	●	●	●
Exception handling	◐	●	●	○	●	○	◐	●	●
Timing	○	○	○	●	●	◐	◐	◐	●
State transitions	○	○	○	○	○	○	●	●	●
Composite data types	●	●	●	●	◐	○	○	●	●

○ not supported ◐ partially supported ● supported

Figure 3.4: System-level language requirements: comparison between the SPEC-C and other languages possibly used in FPGA [Döm03].

3.5.3 High Level Languages

High Level Language definition

A short definition of a high-level programming language may be that this kind of language is more abstract, easier to use, or more portable across platforms than low-level languages. High level languages are supposed to make complex programming simpler and low level languages make more efficient code. They consume more memory, have a larger binary size and are slower to execute. In fact the difference between low and high level languages could be very relative, originally, C language was considered as high level language and now depending on the context, it can be also considered as a low level language because it still allows memory to be accessed by address, and provides direct access to the assembler level.

C Language

C is a general-purpose programming language with the following features ([Com07]):

- Block structured
- Procedural
- Imperative
- Allowing low-level access to memory
- Requiring minimal run-time support

These languages are available once a softcore processor has been implemented into the FPGA platform. Table 3.5.3, page 35 gives an overview of the possibilities of c language.

Pros	Cons
based on open standards	no object-oriented concepts
use few concepts to ease the programming	no error handling mechanism
influenced numerous recent languages such as C++/Java, PHP	no standard garbage collector
allow or code softwares not needing any support (libraries, virtual machine)	buffer overflow breach
Has a predictable comportment (ram memory)	difficulty to code portable code

Table 3.2: Pros and cons of the C language

C++ Language

C++ ([Wika]) is a general-purpose programming language, compatible as much as possible with the C language. Its main features and differences compared to C are:

- Statically typed
- Multi-paradigm
- Not platform specific
- No need for sophisticated programming environment
- Exception handling
- Templates

The key concept of the C++ language is the object oriented programming. It allows classification, encapsulation, classes composition, classes association, heritage, abstraction and genericity.

Python Language

Python ([fou06]) is a multi-paradigm, interpreted language for structured imperative and object-oriented programming. Python is a general-purpose, very high-level programming language. Python has also a large variety of libraries and its core syntax is quite simple.

Its design philosophy emphasizes programmer productivity and code readability. Python's core syntax and semantics are minimalist, while the standard library is large and comprehensive. Python has a automatic management of the memory (garbage collector) and a exception handling system. This language can be available once an embedded OS has been implemented into the FPGA platform. On most of the embedded OS , there is a python interpretor included. Compared to other high level languages, a key advantage of python is the fact that it does not need any virtual machine.

3.6 Softcore processors

This section presents the different softcore processors that can be used on the available FPGA platforms. There are two distinct possibilities: the official branded ones and the open source ones. Depending on the platform used, some official softcore processors can be free of charge. As the choice is quite narrow, all the possibilities have been studied and are displayed in the following subsections.

3.6.1 Nios II

The Nios II softcores (I and II) [Cor07] are softcore processors owned by Altera . They fit for Altera FPGA , are well documented and numerous existing IP-Bloc can be implemented into it. The Nios II softcore processor is based on a RISC 32 bits Cores and Avalon bus. The implementation of the Nios II into the FPGA is done via Quartus([Inc08b]). The development of the core and its components (ip-Blocs) is done via SOPC Builder(citesopc). The development of the software is done under Nios II IDE (based on Eclipse, [Inc08a]) using C language. These three softwares are the development tools provided by Altera to develop applications on FPGA .

3.6.2 Microblaze

Microblaze [Inc07] is the softcore processor developed by Xilinx for its FPGA platform Virtex. Its main features are:

- It is based on RISC architecture
- Both instruction and data words are 32 bits
- Many aspects can configured cache size, bus-interfaces and embedded peripherals
- Speed up to 210 MHz on the Virtex-5 FPGA family
- Configurable 3-stage or 5-stage pipeline
- Very customizable for designing custom-built software.

The Xilinx Embedded Development Kit (EDK) contains:

- The MicroBlaze core
- Peripheral cores
- Software development tools:
  - GNU C Compiler (GNU: Gnu's Not Unix is an OS based exclusively free and open sources softwares)
  - GNU Debugger
  - Eclipse IDE

3.6.3 SPARC and LEON

SPARC (Scalable Processor ARChitecture) is brand owned par SPARC International Incorporation and stands for an open microprocessor architecture. It is constructed on a RISC model with a pipeline and as few features or op-codes as possible. This architecture is supporting 64 bits data and memories addresses. SPARC designs also a microprocessors family. SPARC is an architecture which specifications are free. A completely free of cost processor using SPARC is LEON.

LEON [GHC07] is an open source 32 bits RISC processor, SPARC compatible. Its main features are:

- 7 levels pipeline
- Data cache and instructions separated
- Configurable cache
- MMU
- Bus interface AMBA-2.0
- Works until 125MHz on FPGA

3.6.4 Open softcore processor

The other softcore processor are concentrated on the Website www.opencores.com. There are some numerous projects but only a few documentation and no IP-Blocs. These projects can be designed for Altera , Xilinx or other architectures. Here are two examples of interesting projects found on this website.

DDR SDRAM Controller Core

The DDR SDRAM project ([Win06]) provides a controller to access standard memory devices like DDR SRAM. This controller manages the initialization, the auto refresh and other commands such like of course READ and WRITE. The controller has been designed for the XILINX Virtex II.

SD/MMC Bootloader

The SD/MMC Bootloader project ([Lae07]) manages configuration and bootstrapping of FPGAs. This bootloader control SecureDigital (SD) cards or MultiMediaCards (MMC) which are operated in SPI mode (avoiding dedicated implementations). This bootloader is designed initially for the Xilinx Spartan IIe on BurchED's B5-X300 board. Many different sized, different brands SD and MMC cars have tested successfully but not all.

3.7 Embedded OS

This section will present the different embedded OS that can be used on the available FPGA platforms.

3.7.1 uCLinux

uCLinux means "MicroController Linux". It is a fork of the Linux Kernel designed for Microcontrollers without MMU (memory management unit) but now it covers more processor architectures still without MMU like FPGA platforms.

3.7.2 eCos

Ecos is used to design softwares with strong constraints of response time and reactivity. Ecos is very configurable, allowing good run-time performances optimised hardware resource footprint.At the beginning developed by Cygnus, Ecos is now a free software developed by a community ensuring on-going technical innovation and platform support.

3.7.3 MicroC/OS-II

MicroC/OS-II [Lab02](commonly written μC/OS-II or uC/OS-II) is a low-cost real-time OS for microprocessors. It is now extended to other platforms. It is written in ANSI-C for maximum portability. The society, Micrium Incorporation, developing this OS also created lots of middlewares for it like uC/CAN, uC/TCP-IP, uC/FS, uC/GUI, uC/MOD-BUS, uC/LCD, uC/USB for example. MicroC/OS-II is suitable for use in safety critical embedded systems such as aviation, medical systems and nuclear installations.

3.8 Choices made to meet the constraints and requirements

After defining all the different options available for implementing the project, some options have been selected to fit the requirements and constraints. In the following subsections are presented the chosen options in the following categories:

- FPGA platform
- Camera sensor
- softcore processor
- Programming languages
- Embedded OS

3.8.1 FPGA platform

The chosen FPGA platform is the Altera DE2 Board with Cyclone II processor. The reasons of this choice are:

- A camera sensor has been designed by Terasic for this platform (1.3Mega Pixel Digital Camera Development Package), and is available in the embedded system lab.
- The examples' availability on how to use the camera

- A modular Processor and synthesisable one (softcore processor) available with suited tools and softwares.
- The presence of a socket for the camera sensor.
- It is an education board meaning that it should be well documented and with supports.
- JTAG cables (for FPGA programming and softcore processor debuging)
- The examples' availability on how to use the camera

For the previooulsy enumerated reasons, this FPGA platform fits the requirements in terms of possibilities, space and time constraints.

3.8.2 Camera sensor

The camera sensor selected for this project is the TRDB-DC2 produced by TERASIC ([Tec06]). It is a 1.3 mega pixel camera sensor and is part of the "1.3Mega Pixel Digital Camera Development Package". Its main interesting features are:

- The quality is fitting the requirements: resolution is sufficient for the targeted surveillance application
- Wires and connectors are fitting the ones of the FPGA platform
- The availability of examples using the camera and the FPGA platform

3.8.3 Programming languages

Currently VERILOG and C language are the two languages used in this project. The choice of these two languages has been done within the flow of the Hardware/-Software Co-Desisgn . These two languages are corresponding to the partitioning. The VERILOG is used for the acquisition chain which is the hardware part of the project, while the C language is used thanks to the NIOS II (3.8.4)for the motion detection part (software).

They were two main reasons for the choice of the VERILOG . The first one was the fact that the camera sensor examples were coded only in VERILOG . The other one is the similarity between the VERILOG and the C languages compared to the VHDL (same syntax,similarity for the loop mechanisms).
The main reasons for the choice of the C language are:

- It is the language available without any configuration compared to C++language when using a softcore processor
- The project team had already some knowledge about C programming.
- It introduces a limited overhead compared to other high level languages
- It provides an easier way to handle memory management, while still allowing some low-level access to the memory.

- It is a portable language, the algorithm can be developed in C on desktop computer (profiled, debugged) and then ported onto the targeted platform
- Most of the tools used to developed onto the targeted platform are common with desktop development tools (Eclipse IDE and the GNU toolchain including GNU C Compiler, GNU Debugger)

Concerning the SDLs, it has decided not to use them for some kinds of reasons:

- For the HANDEL-C , some license problems have been encountered
- For the SPEC-C , it was impossible to make the compiler work in the Linux development computers
- For the IMPULSE-C , the researches made shown that this language is too specific, with less documentation and less examples as the previous, it appears having no more special feature compared to the previous ones for the project.

Moreover, the most important reasons not to use these languages are more general. All these languages needs some learning time to become efficient while programming with them. After that, all the tools needs compilers that cannot be more reliable than the ones provided by the society building the FPGA platform and/or the softcore processor and sometimes do not work the the SPEC-C compiler with desktop computers.

Last but not least, for the same reasons as in the last paragraph, there were no needs of python because it required a embedded OS which is harder to implement compared to a softcore processor and require a much higher overhead in term of resources' use. Python special features are not needed in this project and as for every new programming language, there is also the problem of the learning curve.

3.8.4 softcore processor

As the project was progressing, the need for a softcore processor has became clearer. Indeed, after a long time of work on VERILOG language with the camera sensor and the memories, the conclusion was that it was not possible to realize the project within the meeting deadlines using this language. Even with the FPGA examples provided by ALTERA , they were still some functions whose code was not available, for example, the stack function for the memories (SDRAM, SRAM). The second main reason was the lack of documentation concerning the needed functions, especially the ones involved into the memories management. This concerns also examples, lot of the ALTERA examples on their CD were not working on the chosen board. Moreover the free examples of flash memory management using VERILOG language that can be found on internet are also incomplete or bugged.

But this decision to use software is not only due to the disadvantages of programming without softcore processor . As said before using a softcore processor allows to use C/C++ and other object-oriented languages and to work at a higher level of abstraction. On the other hand, it needs another way to think the architecture of the system. Using a softcore processor means using hardware/software co-design methodology.

After some researches, the conclusion that the open cores did not fit the project requirements because they is a lack of support/manuals. They are also not designed for the chosen FPGA platform and do not have some needed features as GPIO management (General Purpose Input Ouput) for example (The opencore projects were not targeting the DE2 platform, were not up to date or were not using the required functions).

Here are the reasons to use the Nios II :

- It is the softcore processor dedicated to the selected FPGA platform, both Altera products
- It is supported by both the Quartus II software and SOPC builder software from Altera which should provide a thigh and easier integration into the platform, which is relevant given the time frame.
- It allows to use C and C++ languages
- It is documented and there are some information examples and forums on the internet.
- It has the 32 bits instruction cache like desktop computers.
- It uses the development chain GNU based on Eclipse (A project providing a universal open-source tool-set for development) due to an ANSI-C implementation on the Nios II .
- the resources (Cyclone II, SRAM, SDRAM, FLASH memories, LCD screen and so on) and the architectures seemed to be sufficient enough for the project
- possibility to accelerate some functions (C2H technology) : a tool provided as part of Altera tools chain which aims to facilitate the hardware optimisation of C function by creating the required component directly in HDL. This tool is supposed to ease Hardware/Software Co-Desisgn mapping process.
- System designers can create their own custom peripherals that can be integrated within Nios II processor systems. For performance-critical systems that spend most CPU cycles executing a specific section of code, it is a common technique to create a custom peripheral that implements the same function in hardware.

So the Nios II , another Altera product is well documented and fits the project's requirements. This softcore processor is the official one for the DE2 Altera board and even with that security, there are still some cost consuming not-so-documented bugs.

3.8.5 Embedded OS

For this project, no embedded OS has been integrated. The reasons are quite simple. The main advantage of an embedded OS is the possibility to use higher level languages but there is no need to use these languages, neither for the acquisition chain nor for the motion detection algorithm. So integrating a embedded OS would have been less useful and would have added complexity and probably overhead which could have slowed done the whole application [Lu, Section IV : Results]. Furthermore, implementing a embedded OS on the FPGA is a long, and complex task with a not negligible error rate.

Chapter 4

Motion Detection Algorithm

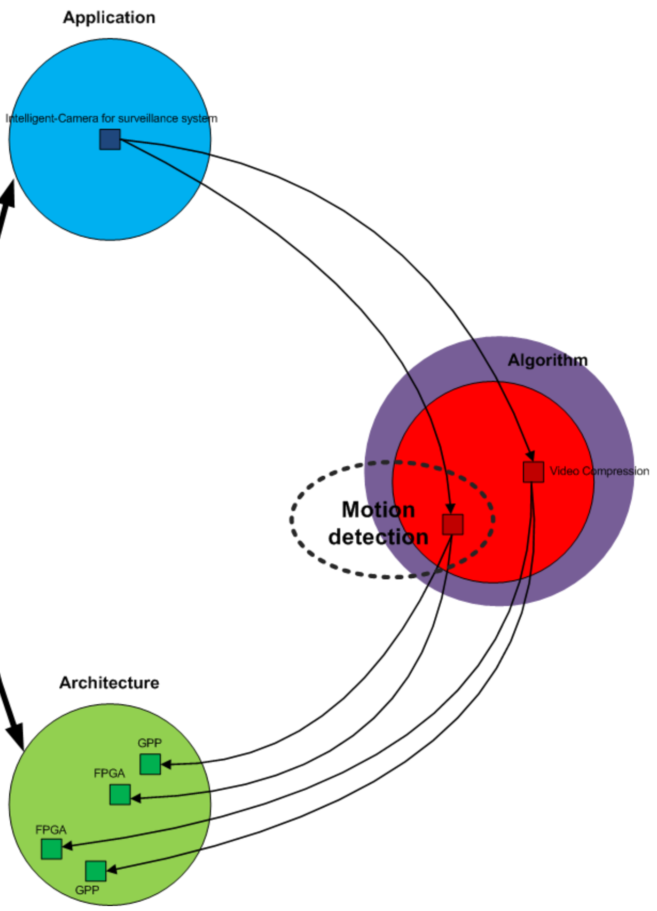
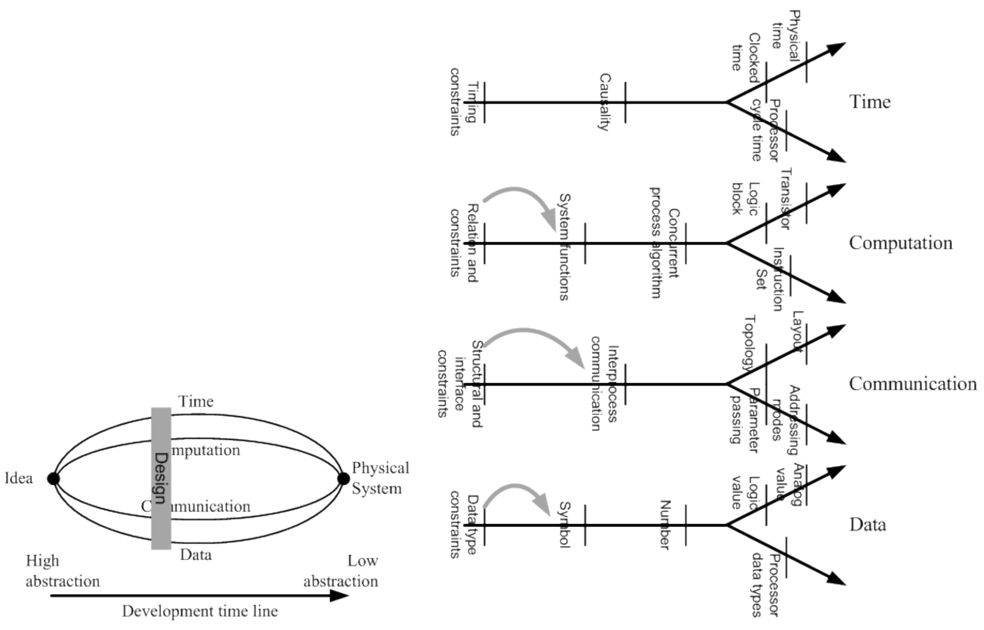


Figure 4.1: Location in the A^3 model highlighted by the purple circle, the algorithm studied in this chapter is encircled by the doted line

The purpose of this chapter is to provide an overview of the motion detection algorithm development.



(a) Global location (b) Elaborated location

Figure 4.2: Location in the Rugby Meta-Model.

4.1 Definition

As previously stated, one of the project's context is a fixed video camera observing dynamic events in a scene. This fact has been taken into account while exploring the broad variety of motion detection algorithms one can find in the literature. Background subtraction method is a common used approach for detecting moving objects in videos from static cameras [Pic04a].
Common applications of image differencing include object tracking, vehicle surveillance systems, and interframe data compression. There are also many examples of its use for analysing satellite images to measure land erosion, deforestation, urban growth,crop development and for analysing medical images to measure cell distribution.

The main idea behind this approach is that of detecting the moving objects from the difference between the current frame and a reference frame, often called the background image, or background model. As a basic, the background image must be a representation of the scene with no moving objects and must be kept regularly updated so as to adapt to the varying luminance conditions and geometry settings.

Therefore the foundation of the algorithm can be summarized by the following equation (4.1):

$$|Frame_i - Background_i| > Threshold \tag{4.1}$$

where $Frame_i$ is the current frame captured by the video camera, $Background_i$ is the actual background and $Threshold$ is the threshold level differentiating "motion pixels" from "background pixels"
Reference images can be generated by a variety of methods, e.g. on a background

image acquired during a period of relative inactivity within the scene or from a temporally adjacent image from a dynamic sequence.

4.2 Analysis

To answer the problem statement (**Is it possible to implement an intelligent IP video surveillance camera on an FPGA platform to optimize video surveillance?**), the research of a proper algorithm able to solve the intelligent part of the system was focused on the following requirement:

- Simple Method : If an apparently "straightforward" method is already hard enough to implement, a more elaborated one could prevent to answer the question in the given time frame.
- Low memory requirement : The chosen FPGA platform (ALTERA DE2 Board with CYCLONE II processor) has a limited embedded available memory.

A more thorough investigation of the Motion Detection algorithm can be found in the review like [Pic04a] or [CGPP03].

The first method reviewed is the background as the *average* (*a single value (as a mean or median) that summarizes or represents the general significance of a set of unequal values*) or the *median* (*a median is described as the number separating the higher half of a sample, a population, or a probability distribution, from the lower half.*) of the previous n frames. This method is rather fast, but very memory consuming due to the memory requirement needed to calculate the average of those n frames. The memory requirement is $n * size(frame)$, the more frame is used to calculate the average the more the algorithm is going to be precise but the more memory it will use.

The second algorithm studied is the background as the *running average*(*A series of successive averages of a defined number of variables*). This method can be calculated by the equation 4.2:

$$B_i + 1 = \alpha * F_i + (1 - \alpha) * B_i \tag{4.2}$$

Where B_i is the current background, F_i is the current frame and α is the learning rate (typically 0.05 [Pic04b]) This way the memory requirement is lower than the previous algorithm 4.2.

Another method is based on the pixel's recent history to create the background. Such history is often:

- just the previous n frames
- a weighted average where recent frames have higher weight

The background model is calculated as a chronological average from the pixel's history. This history is often based on an histogram (*a mapping m_i that counts the number of observations that fall into various disjoint categories*).

4.3 Design

Frame differencing is a particularly efficient and sensitive method for detecting grey level changes between images which are co-registered. Motion is detected by differencing a reference and the “current” image frame and applying a certain Threshold to this difference (4.3).

$$|Frame_i - Frame_{i-1}| \geq Th \tag{4.3}$$

where $Frame_i$ is the current frame captured by the video camera, $Frame_{i-1}$ is previous captured frame and Th is the threshold level differentiating “motion pixels” from “background pixels”

The frame differencing algorithm may be sub-divided into three parts [Pic04b]:

1. the generation of a suitable reference or background
2. the arithmetic subtraction operation
3. the selection (and application) of a suitable threshold

4.3.1 Background Generation

The first task is then to generate the background image $B_{x,y}$ from a sequence of frames $I^t_{x,y}$ which may contain moving objects. $I^t_{x,y}$ being the pixel value at the x, y location in captured frame at a given time t and $B^t_{x,y}$ the extracted background. The FPGA DE2 Board used embeds a limited amount of available memory to use, so the detection of the background must not be very memory consuming. Therefore, one of the least memory consuming way of achieving the background’s creation is to estimate the background from the previous frame (4.4).

$$B_i = Frame_{i-1} \tag{4.4}$$

The following step is to transform both the background $B_{x,y}$ and the current frame $I_{x,y}$ into their greyscale version. As both of those image are coded in a 8 bits RGB colours we use the following greyscale conversation (4.5):

$$Grey_{x,y} = \frac{(299 \cdot Red_{x,y} + 587 \cdot Blue_{x,y} + 114 \cdot Green_{x,y})}{1000} \tag{4.5}$$

Then each of the 8 bit coded colour is replaced by the new grey value. The selected coefficients are from the 601st recommendation of the CIE (International Commission on Illumination)[oI]
Subsequently, the *arithmetic subtraction operation* can be performed as follow (4.6):

$$D_{x,y} = |I_{x,y} - B_{x,y}| \tag{4.6}$$

of which the result $D_{x,y}$ is the absolute difference pixel value between $I_{x,y}$ and $B_{x,y}$ previously defined.
At this point the two first step of the frame differencing algorithm are done.

4.3.2 Thresholding

Thresholding is the simplest method of image segmentation. Individual pixels in a greyscale frame are marked as “object” pixels if their value is greater than some threshold value and as “background” pixels otherwise.
The frame differencing algorithm is really sensitive ([Pic04b] and [Ros95, Section 1]) about the threshold *Th* used to differentiate “object” pixels from “background” pixels.
The proper value of the threshold is dependent on the scene, possibly fluctuating camera levels, as well as viewing conditions (e.g. illumination) which may vary over time. A too high *Th* could inhibit the detection of a really subtle change and hence create a false negative. On the contrary, a too low *Th* could lead to detect noise as a movement and then create false positive.
As the motion detection algorithm relies heavily on a correct threshold, an automatic thresholding appears as a good choice due to its adaptive property during runtime thus preventing a fixed threshold becoming irrelevant. The Automatic thresholding *Th*of difference images algorithm used is described in [IDT95, Chapter 2.2]: First, the difference image $D_{x,y}$ is analysed to determine the median MED (4.7):

$$MED = med_{x,y \in F} D_{x,y} \tag{4.7}$$

Then the median absolute deviation MAD is computed (4.8):

$$MAD = med_{x,y \in F} |D_{x,y} - MED| \tag{4.8}$$

Assuming that less than half the image is in motion, the median should correspond to typical noise values, and a suitable threshold should correspond to the value obtained by the following calculation (4.9) based on [IDT95, Chapter 2.2]:

$$Th = MED + 3 * 1.4826 * MAD \tag{4.9}$$

where 1.4826 is a normalisation factor with respect to a Gaussian distribution.

Now that the threshold *Th* is obtained, the last step is to apply it to the difference frame $D_{x,y}$ to obtain the foreground $F_{x,y}$ binary image from the background (4.10).

$$F_{x,y} = \begin{cases} 1 & if \quad D_{x,y} \geq Th \\ 0 & else \end{cases} \tag{4.10}$$

1 meaning $F_{x,y}$ is a foreground pixel hence part of a “moving” object; 0 meaning $F_{x,y}$ is a background pixel.

4.3.3 Selecting an Efficient Median Search Algorithm

The median of N numerical values can be defined by:

- The median of a list of N values is found by sorting the input array in increasing order, and taking the middle value.
- The median of a list of N values has the property that in the list there are as many greater as smaller values than this element.

The concept is fairly simple to understand but its implementation can be computing intensive and therefore the selection of a fast Median Search Algorithm is necessary. From the review of fast median search algorithm [Dev98], the selected C implementation is the a non-recursive search based on [Wir]. It was selected for different reasons:

- One of the fastest considering the benchmark done in [Dev98, Section 5]
- Non recursive : a recursive algorithm is not easily mapped into hardware for accelerating its computation

4.4 Profiling

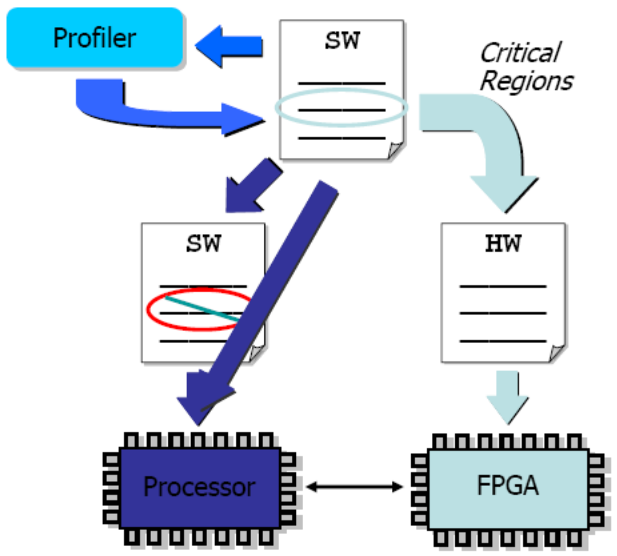


Figure 4.3: The profiler analyses the application. The results hightlight the critical part of the application. This results can guide the Hardware/Software Co-Desisgn , hence potentially map the critical parts onto Hardware. [Mou08]

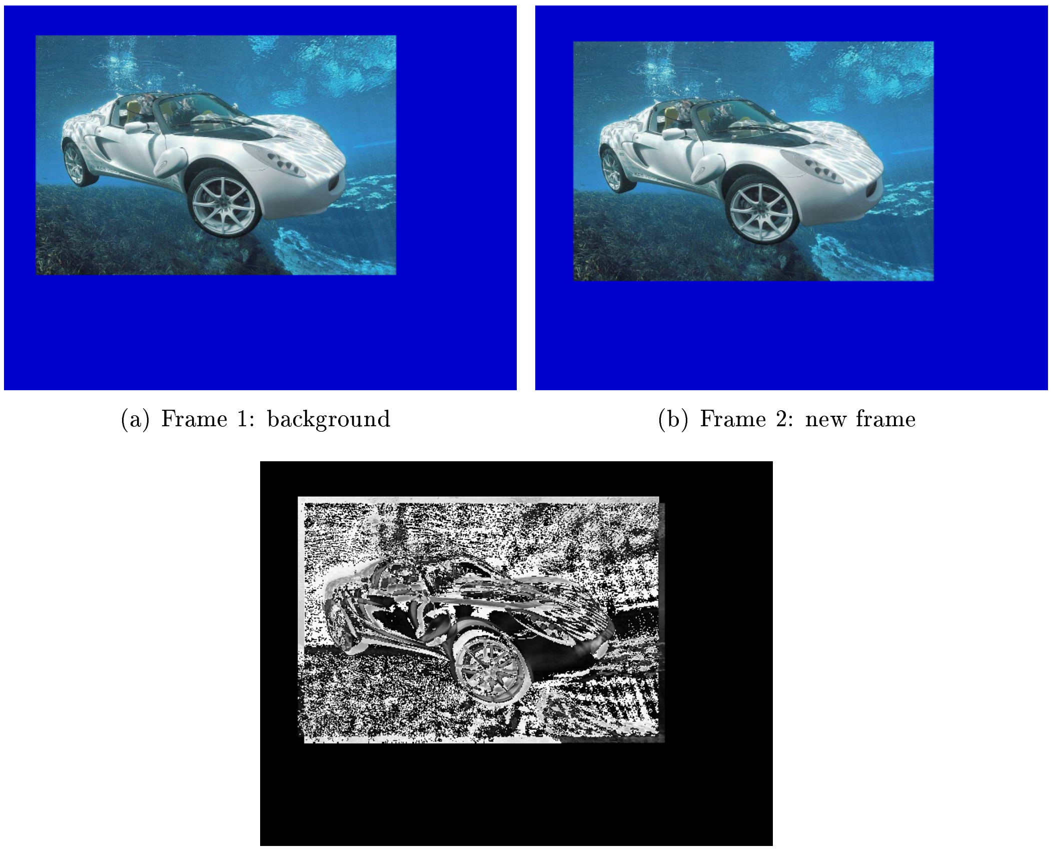
"performance analysis (more commonly profiling): investigation of a program's behaviour using information gathered as the program runs, i.e. it is a form of dynamic program analysis, as opposed to static code analysis." [Wikb]
The purpose of profiling the Motion Detection algorithm is to discover bottlenecks and to determine which parts of the program simulating the chosen algorithm to optimize for speed, memory etc..

Profiling is a part of the Hardware/Software Co-Desisgn (4.3). The GNU Gprof profiler [GKM] used to analyse the C version of the algorithm can output both the flat profiles (computation of the average call times, from the calls) and the call-graph (showing call times, frequencies of the functions and also the call-chains involved based on the callee) of the application.

A software written in C, based on the previously designed algorithm as been profiled on a desktop computer (Intel Pentium 4 @2.80Ghz - 1GB RAM) running Ubuntu 8.04. A simplified version of the application call graph can be found in Appendix 1.

Those two pictures have been used to test this algorithm and have the following characteristics:

- Resolution: 640x480 pixels;
- Colour depth: 24 bits;
- Moving element size: 450x200 pixels.


(a) Frame 1: background

(b) Frame 2: new frame

(c) Result of motion detection

Figure 4.4: Results of motion detection experiments: frame (a) is considered as the background frame, while frame (b) is the "Current" frame. The frame difference algorithm is used to detect motion. (c) is the result of the comparison: black pixels are when no motion is detected and grey pixels enlightens the motion.

The table 4.4 shows the timing profiles results obtained after running the algorithm on the two pictures 4.4(a) and 4.4(a) producing 4.4(c):

The time spent in each functions are not shown because they are not relevant due to the overhead introduced by the profiler allowing it to trace all the functions calls and the times spent in them. However the time overall percentage spent in each functions is more relevant, it's highlighting the critical part, in term of execution

Function	Called (#)	% Time
BMP_Load_Data_24b	2	42.76 (≈ 21 each)
BMP_Save_File	1	16.60
BMP_GreyScaleconversion	2	12.36 (≈ 6 each)
MD_AutomaticThreshold	1	10.89
↪ kth_smallest(Median Search Algorithm)	2	9.93 (≈ 5 each)
MD_Background_Substitution	1	7.43
BMP_Read_Header	2	0
BMP_Copy_Header	1	0

Table 4.1: Application profiled Results (% Time) sorted by the total amount of time spent in each function and its children, The important results are highlighted in red

time, of the application. And therefore can undercover bottleneck in the algorithm used.

Some part of the results found in Table 4.4 are not are not really pertinent, those parts (in black) concern some portion of the C code used only to load and save BMP Files on a Desktop computer. Those portions of code represent roughly 60% of the overall execution time. The interesting parts (in red) related to Algorithm previously defined (4.3) took approximately 40% of the computation time.
The table 4.4 shows the percentage of execution time of each important function from those 40% indexed on a 100% basis. Those results show that the Greyscale

Function	Called (#)	% Time
BMP_GreyScaleconversion	2	30.43 (≈ 15 each)
MD_AutomaticThreshold	1	26.82
↪ kth_smallest(Median Search Algorithm)	2	24.45 (≈ 12 each)
MD_Background_Substitution	1	18.30

Table 4.2: Algorithm profiled Results (% Time) sorted by the total amount of time spent in each function and its children

conversion and the Automatic Thresholding are the heaviest computation tasks with respectively ≈ 30% and ≈ 52% of the execution time. The Median Search Algorithm (kth_smallest in the Table, used twice for the Automatic Thresholding) consumes alone ≈ 12% of the execution time. The Background Substitution is relatively less processing intensive with "only" ≈ 18%.
Those result by themselves are a good hints for highlighting where the algorithm could be optimized, but they cannot indicate which part of the algorithm could be mapped onto hardware, accelerating them. However those results correlated with relevant metrics could help the Hardware/Software Co-Desisgn process by preselecting part of the algorithm which could benefit to be mapped onto hardware.

4.5 Metrics

"Metrics: A software metric is a measure of some property of a piece of software or its specifications."[Wikc]
In this section, the aim is to use metrics and tools based on those metrics to guide the Design Space Exploration and Hardware/Software Co-Desisgn during the implementation of the chosen algorithm, hence to map the computational part of the algorithm on to the components of the system architecture.
The tool use to analyse the algorithm and look through it against some specific metrics is the design framework called *Design Trotter* . Design Trotter is a set of co-operative tools which aim at guiding embedded system designers early in the design flow by means of design space exploration [MDA+05].

Design Trotter is an academic tool which is still in development with a certain numbers of issues and limitations. The framework employs fast list-based scheduling and heuristic methods to explore the search space so that feasible and near-optimal solutions can be found rapidly and then guides the developer in the hardware / software partition phase.

4.5.1 Chosen Metrics

The choice of relevant metrics has to be made to guide the designer and the synthesis tool towards an efficient application architecture matching.
Design Trotter is based on different metrics which are defined in [DTm03] and [AAS+07].

γ'

γ' is especially relevant to measure the degree of inherent parallelism in the algorithm. γ' can be calculated for a function by the formulae 4.11 as in [AAS+07, Section 3].

$$\gamma' = 1 - \frac{Nb\ operations}{CriticalPath} \tag{4.11}$$

A high γ' indicate an optimization potential for speed-up and consumption reduction when mapped to an highly parallelised architecture.

MOM

Memory Orientation Metric : *MOM*
MOM can be calculated for a function by the formulae 4.12 as in [DTm03, Section 3.3].

$$MOM = \frac{Nb\ global\ memory\ accesses}{Nb\ processing\ operations + Nb\ global\ memory\ accesses} \tag{4.12}$$

MOM indicates the frequency of memory accesses in a graph. *MOM* values are normalized in the [0;1] interval. When *MOM* gets closer to 1 the function is dominated by memory accesses. Hence in case of hard time constraints, high performance memories are required (large bandwidth, dual-port memory) as well as an efficient use of memory hierarchy and data locality.

COM

Control Orientation Metric : COM
COM can be calculated for a function by the formulae 4.13 as in [DTm03, Section 3.4].

$$COM = \frac{Nb\ controls}{Nb\ processings + Nb\ memory\ accesses + Nb\ controls} \tag{4.13}$$

It indicates the appearance frequency of control operations (i.e., tests that cannot be eliminated during compilation). COM values are normalized in the [0;1] interval. When COM gets closer to 1 the function is dominated by controls.

4.5.2 Results

The main idea is now to apply those metrics to the chosen algorithm correlated with the result of the profiling step.

Greyscale Conversion

The greyscale conversion (*BMP_ GreyScaleConversion* in 4.4) of both the background and the current frame is the first function to be studied. The Table 4.5.2 is the result of the calculation of those three metrics using Design Trotter .

Metric	Result
γ'	0.5
MOM	0.7
COM	0.0

Table 4.3: Metrics Results (γ', MOM, COM) for the Greyscale Conversion using Design Trotter

The Greyscale Conversion is quite dependant on memory accesses, as the MOM points it with a 0.7 result. It is something pretty easy to foresee because this step is fundamentally accessing data in memory, process them and then store them back. This implies that the memory accesses should be optimised in the architecture to avoid memory bottlenecks.
More important, γ' result of 0.5 shows that it this step could benefit from a parallelised architecture. Even more when one can consider that both of the greyascale conversion could be carried out in parallel without even thinking about optimizing the inner self algorithm because the data manipulated are not the same. Those results outline another facts, this portion of the algorithm contain almost no control operation ($COM = 0$).

Background Substitution

The background substitution (*MD_ Background_ Substitution* in 4.4) is the second function to be studied. The Table 4.5.2 is the result of the calculation of those three metrics using Design Trotter .

Metric	Result
γ'	0.656
MOM	0.656
COM	0.0

Table 4.4: Metrics Results (γ', MOM, COM) for the Background Substitution using Design Trotter

The Background Substitution is quite dependant on memory accesses, as the MOM points it with a 0.656 result. As previously stated for the Greyscale Conversion it was something almost predictable because this step is fundamentally accessing data in memory, process them and then store them back. This implies that the memory accesses should be optimised in the architecture to avoid memory bottlenecks.
Those results outline another facts, this portion of the algorithm contain almost no control operation ($COM = 0$). However on this particular part of the algorithm, as the γ' result of almost 0.7 highlights it, could really benefit from a parallelised architecture. It even more true, when the fastest architecture and scheduling (in term of number of cycles) proposed by Design Trotter is composed by 4 ALUs.

Median Search algorithm

The Median Search algorithm (*kth_ smallest* in 4.4) is the third function to be studied. The Table 4.5.2 is the result of the calculation of those three metrics using Design Trotter .

Metric	Result
γ'	0.357
MOM	0.786
COM	0.0

Table 4.5: Metrics Results (γ', MOM, COM) for the Median Search algorithm Design Trotter

The Median Search algorithm is heavily dependant on memory accesses, as the MOM underlines it with roughly *0.8* MOM result. This is even more important given that this part of the Thresholding part of the algorithm is repeated twice. One more time, This implies that the memory accesses should be optimised in the architecture to avoid memory bottlenecks therefore slowing down the execution speed of the overall application.
There is less inherent parallelism is this part of the Frame Differencing algorithm as the 0.35 γ' result indicates it.

Automatic Threshold

The Automatic Threshold (*MD_ AutomaticThreshold* in 4.4) is the last function to be studied. It is important to state that for this part, the Median Search algorithm has not been analysed during the metrics calculation due to its results already expressed.

The Table 4.5.2 is the result of the calculation of those three metrics using Design Trotter .

Metric	Result
γ'	0.25
MOM	0.75
COM	0.083

Table 4.6: Metrics Results (γ', MOM, COM) for the Automatic Threshold (without the Median Search algorithm included)Design Trotter

The Automatic Threshold is heavily dependant on memory accesses, as the MOM underlines it with roughly *0.8 MOM* result. Automatic Threshold could not really benefit to be parallelised regarding to its γ' result of 0.25. This part of the application is slightly more control oriented (a COM result of 0.083) than the previous studied part. This is due to the 4.10 portion of the algorithm.

4.5.3 Conclusion

Design Trotter is a tool especially useful to give an insight on the parallelism level of different part of the algorithm studied. However, Design Trotter is still undergoing development and has a rather limited implementation of the C language (no "while" statement supported for instance). It is far from being straightforward to translate the C version of an algorithm to the "C version for Design Trotter ". There is many restriction on how to write it for Design Trotter to be able to handle and analyse it. Therefore, it takes some time to rewrite an existing source code into one that Design Trotter can analyse.

However results given by the tools are appreciable indications to help the designers and developers to make their choices. For instance, the overall result tends to show that the algorithm designed could benefit from a parallelised architecture like FPGA , given that from all the functions studied two could be fairly parallelised especially one which is used twice during the course of execution.

Moreover a specific targeted processor, with multiple ALU or specific custom hardware function (such as the Nios II for instance) could help to speed up the execution time or cycle used to process the algorithm.
Apparently the Frame Differencing algorithm is not really control oriented with most of the COM metric results close to 0 or equal to 0. The results obtained through the use of Design Trotter using the chosen Metrics seemed to prove that the System Architecture chosen in 3 is a relevant choice.

Chapter 5

Implementation

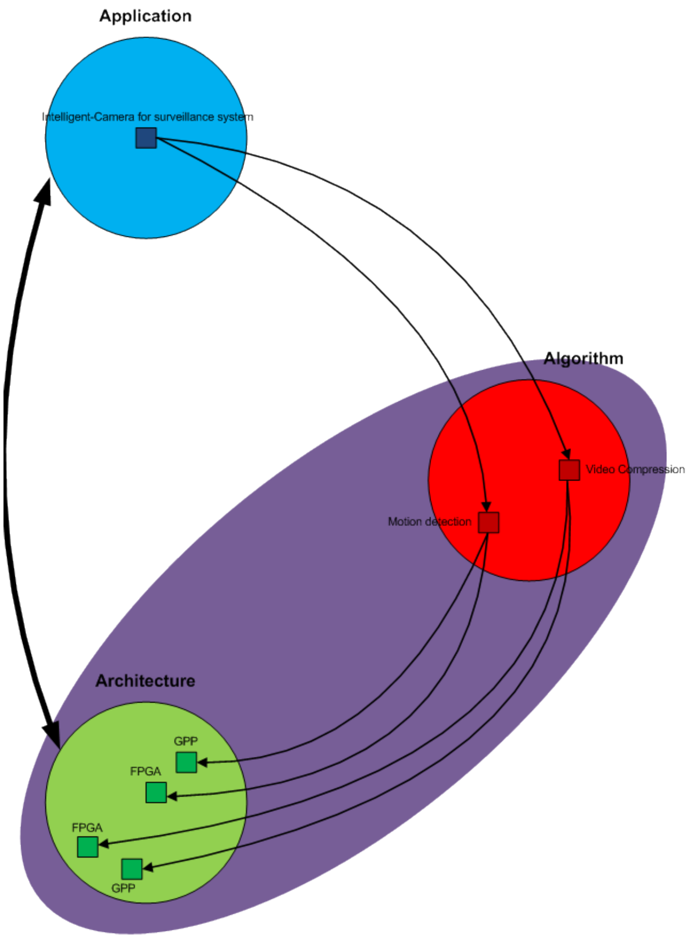
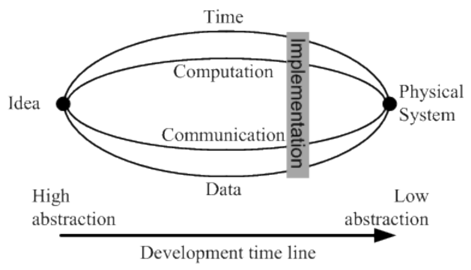
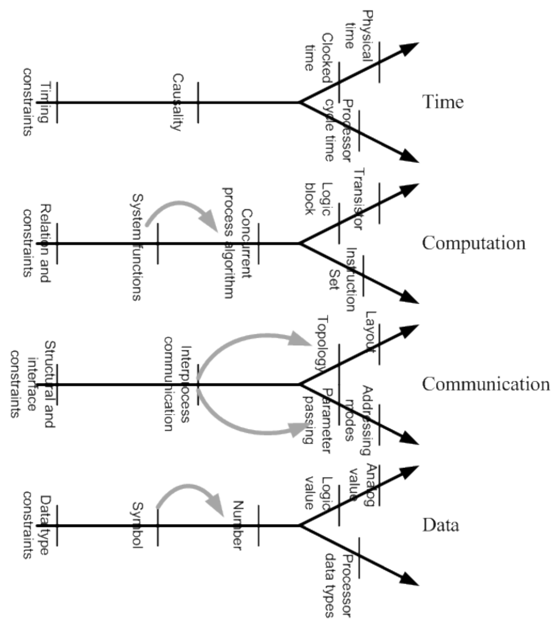


Figure 5.1: Location in the A^3 paradigm, highlighted by the purple circle.

In this chapter, the application implementation is described and detailed. Each section outlines the implementation of one of the goals described in the Chapter 1. Therefore, this chapter is a step-by-step description of the implementation of the project. The first section describes the blocks with a “black-box approach”: the main



(a) Global location



(b) Elaborated location

Figure 5.2: Location in the Rugby Meta-Model, highlighted by the grey bar on (a) and grey arrows on (b).

functionalities of the block are given before given its inputs of outputs and their goal. The source codes of the different hardware blocks and the algorithm (the Nios II version) are on the CD-Rom.

5.1 Transmission chain

The first step of the application is the implementation of a video camera on the DE2 FPGA platform. It is implemented in full hardware, using Verilog as the hardware language. This is because, firstly, the camera sensor is connected to the DE2 Board through General Purpose Input/Output (GPIO) pins, which requires a low level access and because, secondly, there were already some existing IP-block to use the camera sensor in Verilog . The block diagram of this step can be see in Figure 5.3 on page 57. To see the configuration of the GPIO pins of the camera sensor, please refer to Figure 5.4, on page 58.

5.1.1 Image capture

This section is the description of the main blocks involved in the image capture from the camera sensor.

CCD_Capture

The RTL diagram of this block can be see on Figure 5.5 on page 59. This RTL view is presented in the report because it is simple enough to be readable and to fill in a A4 page. RTL diagrams of the other blocks are in the CD-Rom.

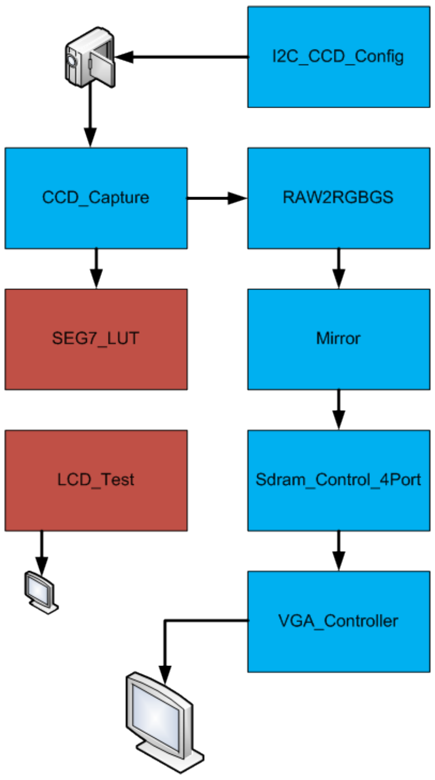


Figure 5.3: Simplified block diagram of the transmission chain: blue blocks are the required blocks, while red blocks describes the features. The camera sensor is controlled by an Inter Integrated Circuit Bus (I2C) command to select the exposure time, and send data to a block which translate those for the rest of the circuit. The other parts are the steps of the transmission, from raw data to the VGA display.

The aim of this block is to get raw data from the camera sensor and transfer them to the other parts of the circuit. It has several inputs and outputs (Existing Block):

- Inputs:
  - iCLK: main clock.
  - iRST: main reset signal.
  - iStart: start signal, which begin the information from the camera sensor to be process.
  - iEnd: end signal, which stop the information from the camera sensor to be process.
  - iFVAL: valid frame of pixel if 1, invalid if 0.
  - iLVAL: valid line of pixel if 1, invalid if 0.
  - iDATA: raw data from the web-cam.
- Outputs:
  - oDVAL: validate the pixel.
  - oDATA: return the raw data of the actual pixel.

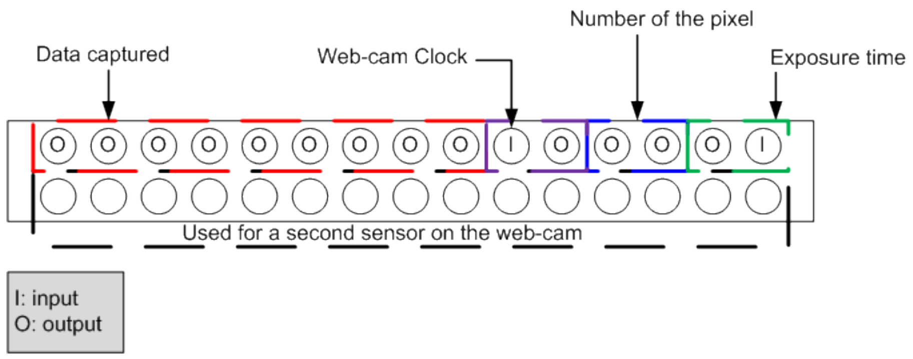


Figure 5.4: Camera sensor GPIO organization: one sensor requires only 15 pins.

- oX_Cont: return the X coordinate of the actual pixel.
- oY_Cont: return the Y coordinate of the actual pixel.
- oFrame_Cont: return the number of frame captured.

RAW2RGBGS

The aim of this block is to translate raw data from the camera sensor to three-colour data Red, Green and Blue (RGB). The second functionality of this block is to calculate the grey value of each pixel, using equation (5.1):

$$Grey = \frac{(299 \cdot Red + 587 \cdot Blue + 114 \cdot Green)}{1000} \tag{5.1}$$

It is used with integer numbers because it is very expensive to use float numbers in hardware, as it requires more logic elements and therefore bigger area. The selected coefficients are from the 601st recommendation of the CIE (International Commission on Illumination)[oI]. The conversion of a RGB picture to a grey-scaled one is the first step of the background subtraction algorithm. It is also possible to do it in software.

The block has several inputs and outputs (Modified Block):

- Inputs:
  - iCLK: main clock.
  - iRST: main reset signal.
  - iDVAL: validate the pixel.
  - iDATA: raw data from the block CCD_Capture.
  - iX_Cont: return the X coordinate of the actual pixel.
  - iY_Cont: return the Y coordinate of the actual pixel.
- Outputs:
  - oDVAL: validate the pixel.
  - oRed: red value of the pixel.
  - oBlue: blue value of the pixel.
  - oGreen: green value of the pixel.
  - oGrey: grey value of the pixel.

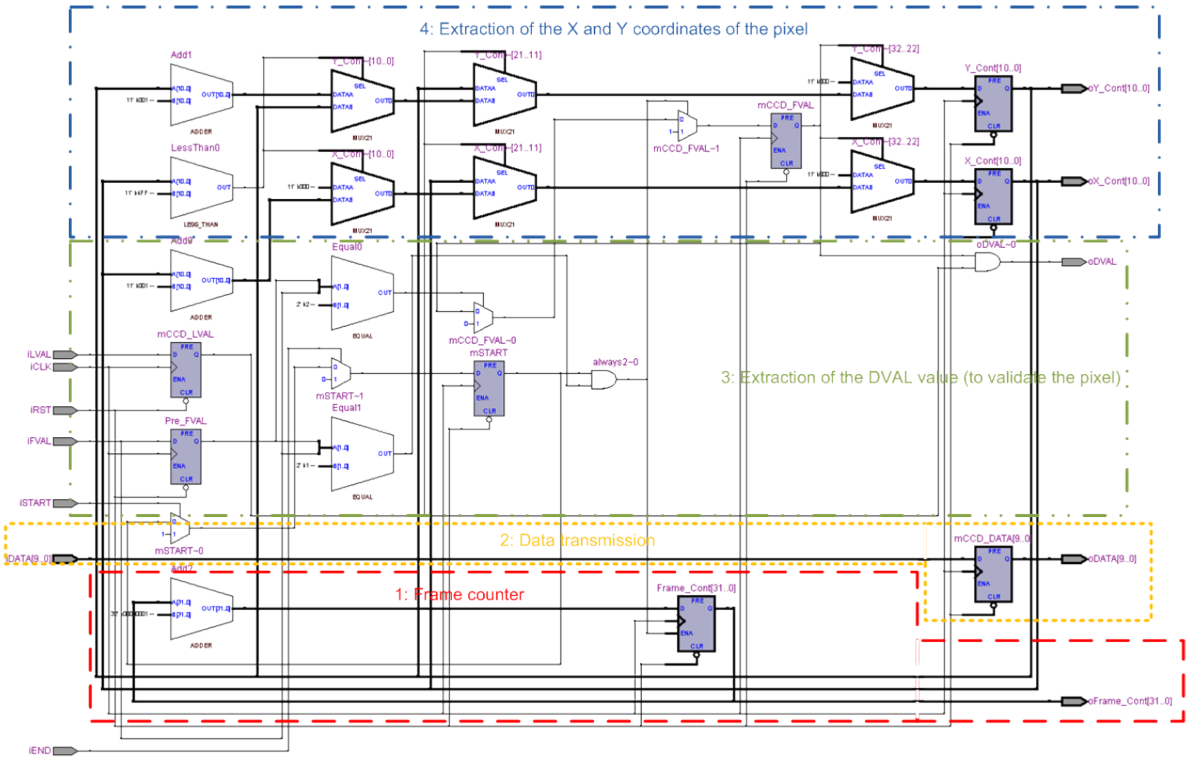


Figure 5.5: RTL view of the CCD_Capture block: in this block, from bottom to top, (1) is the logical elements involved in the frame counter, (2) enlightens the logic involved in the transmission of data (the pixel colour), (3) regroups the elements used to extract the DVAL value of the current pixel and, finally, (4) regroups the elements that allows to extract both the X and Y coordinates of the pixel.

Mirror_Col

The main purpose of this block is to prepare and format information to be stored on the SDRRAM. Therefore, it gets data from the RAW2RGBGS block and sends them in the Sdram_Control_4Port block. The number of the pixel is stored separately, while the colour values are stored by splitting the green value in two and sharing the memory space for two colours (Red and the five Green least significant bits & Blue and the five Green most significant bits). It is because the SDRAM block has two storage stack of 16 bits. It has several inputs and outputs (Modified Block):

- Inputs:
  - iCCD_DVAL: validate the pixel.
  - iCCD_PIXCLK: clock of the camera sensor.
  - iRST: main reset signal.
  - iCCD_R: red value of the pixel.
  - iCCD_G: blue value of the pixel.
  - iCCD_B: green value of the pixel.
  - iCCD_Gr: grey value of the pixel.

- Outputs:
  - oCCD_DVAL: validate the pixel.
  - oCCD_R: red value of the pixel.
  - oCCD_G: blue value of the pixel.
  - oCCD_B: green value of the pixel.
  - oCCD_Gr: grey value of the pixel.

5.1.2 Display Image on a Screen

This section is the description of the main blocks involved in the display of a video.

Sdram_Control_4Port

The aim of this block is to get an access to the SDRAM of the board. In this block, all information for a picture are stored temporarily. It consists of four First In First Out (FIFO) stacks to optimize the writing and the reading of data. It has several inputs and outputs (Existing Block):

- Inputs:
  - REF_CLK: main clock.
  - RESET_N: main reset.
  - WR1_DATA: data input.
  - WR1: write request.
  - WR1_ADDR: write start address.
  - WR1_MAX_ADDR: write maximum address.
  - WR1_LENGTH: write length.
  - WR1_LOAD: write register load & stack1 clear.
  - WR1_CLK: write stack1 clock.
  - WR2_DATA: data input.
  - WR2: write request.
  - WR2_ADDR: write start address.
  - WR2_MAX_ADDR: write maximum address.
  - WR2_LENGTH: write length.
  - WR2_LOAD: write register load & stack2 clear.
  - WR2_CLK: write stack2 clock.
  - RD1: read request.
  - RD1_ADDR: read start address.
  - RD1_MAX_ADDR: read maximum address.
  - RD1_LENGTH: read length.
  - RD1_LOAD: read register load & stack1 clear.
  - RD1_CLK: read stack1 clock.

  - RD2: read request.
  - RD2_ADDR: read start address.
  - RD2_MAX_ADDR: read maximum address.
  - RD2_LENGTH: read length.
  - RD2_LOAD: read register load & stack2 clear.
  - RD2_CLK: read stack2 clock.
  - DQ: SDRAM data bus.

- Outputs:
  - WR1_FULL: write stack1 full.
  - WR1_USE: write stack1 used.
  - WR2_FULL: write stack2 full.
  - WR2_USE: write stack2 used.
  - RD1_DATA: data output.
  - RD1_EMPTY: read stack1 empty.
  - RD1_USE: read stack1 used.
  - RD2_DATA: data output.
  - RD2_EMPTY: read stack2 empty.
  - RD2_USE: read stack2 used.
  - SA: SDRAM address output.
  - BA: SDRAM bank address.
  - CS_N: SDRAM chip selects.
  - CKE: SDRAM clock enable.
  - RAS_N: SDRAM row address strobe.
  - CAS_N: SDRAM column address strobe.
  - WE_N: SDRAM write enable.
  - DQM: SDRAM data mask lines.
  - SDR_CLK: SDRAM clock.

VGA_Controller

The aim of this block is to display data on a VGA-Screen. Therefore it gets data from the SD-Ram and encode them for VGA standard. If one wants to display grey scaled picture, the VGA display only needs to get the same value of grey on the three colours. The Switch allows the user to select the “colour” to be displayed: RGB image or Grey-scaled image. The oRequest signal is sent by the screen to get the information to display. Firstly, synchronization data is sent to prepare the screen. Then, Red, Green and Blue values are sent to the VGA-screen. It has several inputs and outputs (Modified Block):

- Inputs:

  - iCLK: main clock.
  - iRST: main reset signal.
  - iRed: red value of the pixel.
  - iBlue: blue value of the pixel.
  - iGreen: green value of the pixel.
  - Switch: allow the user to select the "color" to display (RGB or Grey Scale).

- Outputs:
  - oRequest: send request signal to the VGA-Screen.
  - VGA_BLANK: use for VGA synchronization.
  - VGA_H_SYNC: use for VGA synchronization.
  - VGA_SYNC(GND): use for VGA synchronization.
  - VGA_V_SYNC: use for VGA synchronization.
  - VGA_R: VGA red value of the pixel.
  - VGA_G: VGA green value of the pixel.
  - VGA_B: VGA blue value of the pixel.

5.1.3 Other Features

This section is the description of the blocks providing nice but not critical features.

SEG7_Controller

The aim of this block is to display the number of frames captured on the 7-segment blocks of the board. It splits the frame number (written on 32 bits) in eight part of 4 bits (for each display). The number shown on the 7-segment displays is written in hexadecimal. It has one input and several outputs (Existing Block):

- Inputs:
  - iDIG: number of captured frame.
- Outputs:
  - SEG0 to SEG7: the eight available 7-segment displays.

I2C_CCD_Config

The purpose of this block is to control and configure the exposure time of the camera sensor. It allows the user, through the switches, to change this time and, therefore, to enlighten the image in case it is too dark. This block use the I2C protocol to communicate the new exposure time of the sensor. I2C is a multi-master serial computer bus that is used to attach low-speed peripherals to a motherboard, embedded system or cellphone. It has several inputs and one output (Existing Block):

- Inputs:
  - I2C_SDAT: actual exposure time.

  - iCLK: main clock.
  - iRST: main reset.
  - iExposure: value of selected exposure time.

- Outputs:
  - I2C_SCLK: new value of the exposure time.

LCD_TEST

The aim of this block is to display information on the LCD screen of the DE2 Board. It has several inputs and outputs (Created Block):

- Inputs:
  - iCLK: main clock.
  - iSwitchN: activate the LCD screen.
  - iRST_N: main reset.
- Outputs:
  - oTestLED: activate one LED when LCD is activated.
  - LCD_DATA: data sent to the LCD.
  - LCD_RW: read (0) or write (1) mode.
  - LCD_EN: enable the LCD.
  - LCD_RS: flush the screen.

5.2 Recording and storage chain

This section describes the second goal of the project: to record and store a video captured by the camera sensor. In Chapter 1, the softcore processor is involved in the web-server and algorithms parts. But, the need of a softcore processor comes earlier. In facts, it appears to be very hard to use both the SD-Card and the SD-Ram only with Verilog blocks. Therefore, the implementation of the Nios II processor is described in this section instead of the next one, as written previously.

5.2.1 Implementation of the Nios II

In order to access the memory block of the board, and later implement the algorithm in C/C++ , a convenient solution is to use a softcore processor. As an Altera DE2 Board is used, a Nios II processor is used as the softcore processor. But a custom version has to be created to satisfied the possibilities of the board used for this project. To do so, the Nios II manual was used to have a better understanding of the different modules [Cor07]. Figure 5.6, on page 64 shows some of the modules accessible for the Nios II softcore processor, while Figure 5.7, on page 64 details the block diagram of the Nios II core.

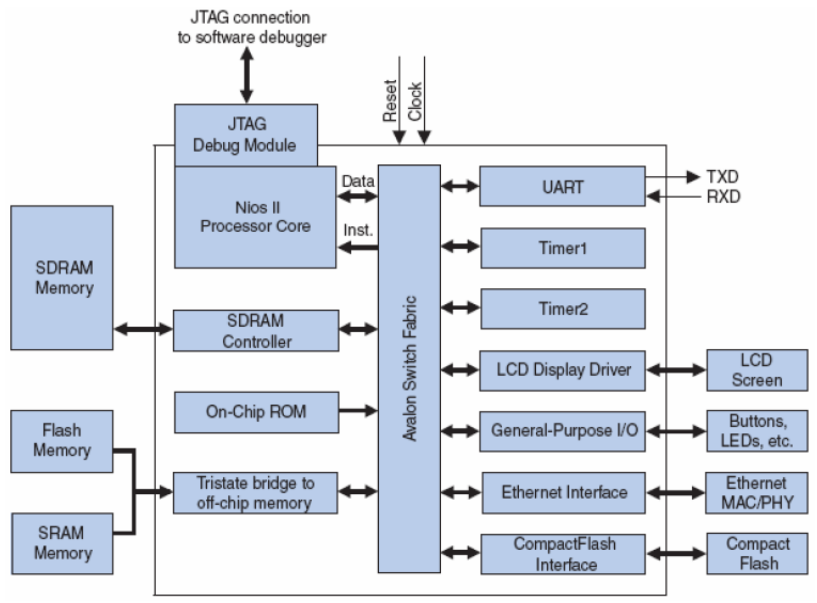


Figure 5.6: Generic Nios II diagram: it includes several components around the Nios core processor [Cor07].

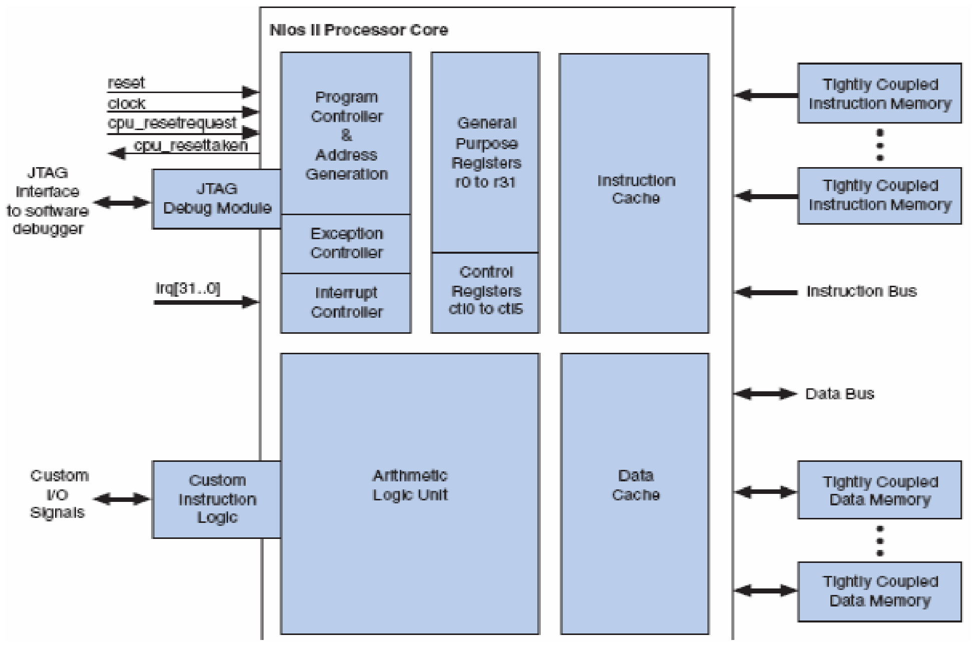

Nios II Processor Core
reset
clock
cpu_resetrequest
cpu_resettaken
JTAG interface to software debugger
JTAG Debug Module
irq[31..0]
Custom I/O Signals
Custom Instruction Logic
Program Controller & Address Generation
Exception Controller
Interrupt Controller
General Purpose Registers r0 to r31
Control Registers ctl0 to ctl5
Instruction Cache
Arithmetic Logic Unit
Data Cache
Tightly Coupled Instruction Memory
Tightly Coupled Instruction Memory
Instruction Bus
Data Bus
Tightly Coupled Data Memory
Tightly Coupled Data Memory

Figure 5.7: Generic Nios II core block diagram: it describes the components included in the Nios core [Cor07].

In order to create this custom version of the Nios II processor, one needs to follow several steps:

1. Create a Quartus II project
2. Create the Nios II using SOPC Builder
3. Make the pins plan of the FPGA (i.e. associate each pin of the board to inputs and/or outputs)
4. Add the fitting modules on the Nios II (i.e. VGA Controller, memory controller, etc.)
5. Verify that it works (i.e. build and run a test program on the Nios II)

The custom version used in this project contains the following modules:

- Nios standard core, with JTAG debug level 2 (RISC 32 bits processor @100 MHz).
- SRAM controller.
- LED red & green parallel outputs.
- JTAG Module.
- Tri-state Avalon bridge to control flash memory.
- Flash memory controller.
- LCD controller.
- Button & switch parallel inputs.
- DMA9000 controller (Ethernet port of the DE2 Board).
- 7-segments controller.
- SD-Card controller (3-bits inputs/outputs).
- Various parallel inputs/outputs (PIO) for the processor to communicate with the rest of the system.

A JTAG debug level 2 was used. It means that one can debug the program on the Nios II directly, and use some breakpoints and view some variables values. There are five level of debug, form 0 to 5. Each level allows more variables values to be viewed and more breakpoints to be put, but requires more logic elements.

5.2.2 Record captured video

The first part of this block is the memory block. In fact, memory is very important in this step, as it requires to store temporarily a lot of frame captured by the webcam. The first problem that appeared is that one frame is around 1 MB:

$$\frac{640 \cdot 480 \cdot 24}{8} = 921600 Bytes = 900 kB \tag{5.2}$$

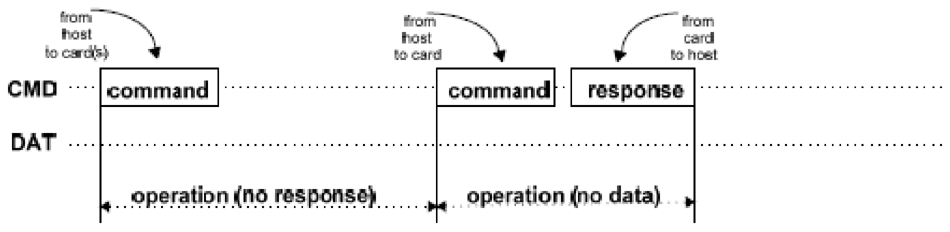
(8 bits per colour * 3 colours (red, green and blue) = 24 bits per pixel, a frame is 640x480 pixels, 1 bytes = 8 bits & 1kB = 1024B.)

The available temporary memory on the board is 8 MB for the SDRAM, which means no more than 8 images simultaneously. It is quite a problem to capture a video, running 24 frame per second while processing 3 images simultaneously with the processor (the background image, the actual image and the temporary grey-scaled image). It means that the processor need to access the SD-Card around 3 times per second. It is impossible to have such a speed to access this kind of memory.

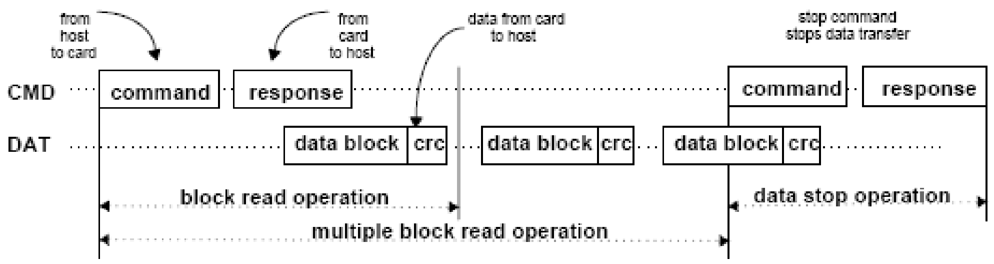
5.2.3 Store recorded video

As the previous block is nearly impossible to make with the available development board, this part cannot be either done nor tested. If the SDRAM chip was around 256 MB, a video could have been recorded and processed in the same time. To have a better processing, the optimal solution should have used two chips of 256 MB, one as the processor RAM, one for the hardware blocks to record the video.

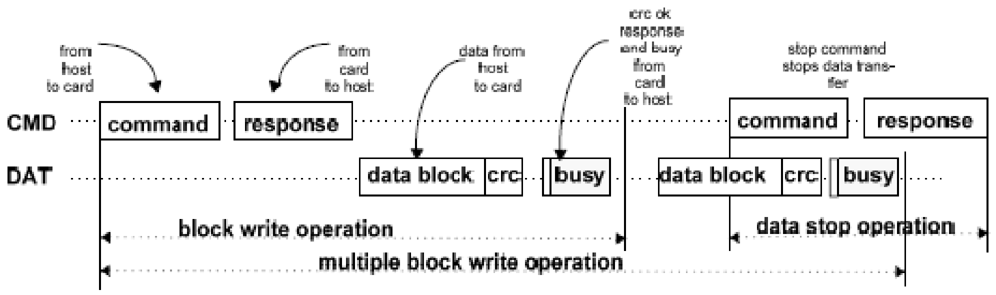
5.2.4 Access recorded video



(a) No-data operation



(b) Read-block operation



(c) Write-block operation

Figure 5.8: Different operation on the SD-Card: those diagrams enlighten the use of a 2-bits communication channel between the FPGA and the SD-Card [Ass06].

The way to use the SD-Card with the FPGA is to get a 1-bit access to the SD-Card. The SD-Card controller module of the custom of the Nios II used in this project has a 3-bits output: one bit for the clock, one bit for the command and one bit for the data. Figure 5.8(a) shows a no-data operation with the SD-Card, Figure 5.8(b) shows a basic read-block operation and finally Figure 5.8(c) shows a basic write-block operation (those figures are on page 66).

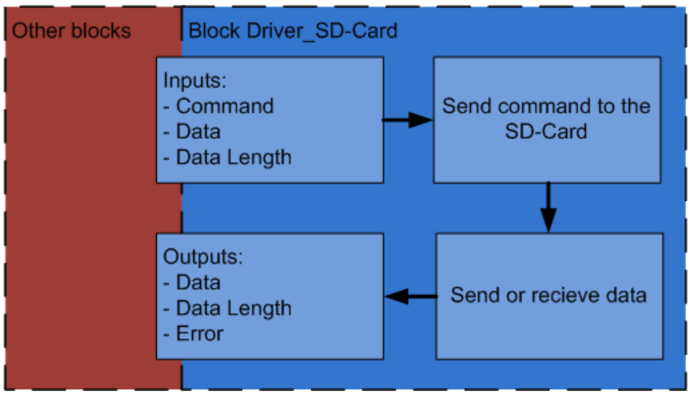


Figure 5.9: Block diagram of the SD-Card custom driver

There are no existing IP-block to use easily the SD-Card. The time to develop a specific custom driver to use it during the project was too long. Moreover, the aim of this project was to implement an intelligent surveillance video camera, using the SD-Card as a tool to store data, not to develop a custom SD-Card driver. This part was not done, but can be considered as a long term future work. In fact, this block should contained the main commands to send to the SD-Card and an input to send the data. Figure 5.9, on page 67, briefly describes how the block should look like to fit in the system.

5.3 Intelligent camera

This section describes mainly the implementation of the motion detection algorithm. A first draft was made to make all measurements in Chapter 4. The final version, detailed below is the draft adapted to the specificities of the Nios II processor and using the inputs and outputs available to it.

5.3.1 Motion detection algorithm

The algorithm was in c-code for being studied. From this working draft, a Nios II version of the code was written. The main points that have been changed concerned:

- strings: the Nios II does not recognised char * type as a string but char[x] (with x the number of character);
- files path: the draft used desktop path to access file. With the use of the host file system, those paths had to be changed (as explained in Chapter 6, page 69);

- small bugs.

For more details about the algorithm, please see Chapter 4 for the analysis and design and Chapter 6 for the results of tests and experiments.

Although the aim of studying the algorithm by calculating metrics and profiling it to get an idea of the hardware and software co-design, the algorithm is fully implemented in software. The first reason is that the tool, C2H, provided in ALTERA development software, is very hard to use. The second reason is that, if the code to be accelerated does not fulfil some requirements, there can be no acceleration. The final reason is that the main bottlenecks of the algorithm are memory accesses, and this cannot be accelerated, except by adding other memory chips.

5.4 Non implemented blocks

As the recording and storage parts were not fully implemented, the following blocks were not implemented. It is however possible to implement them by using the previously proposed solution, which is described in Chapter 7.

- IP camera: this part should has presents the implementation of a web-server on the softcore processor. As it is conditioned by the implementation of the SD-Card driver, because the aim of this block is to access data stored into the SD-Card, the web-server is not implemented.
- Optimized intelligent camera: this part of the project should have presents the implementation of the video compression algorithm. As no videos are stored by the system, it is impossible to implement this feature.
- Complete intelligent IP camera: this block should have presents the implementation of all the blocks together. As not all the blocks are working or implemented, the implementation of this part is not done.

Chapter 6

Testing & Experiments

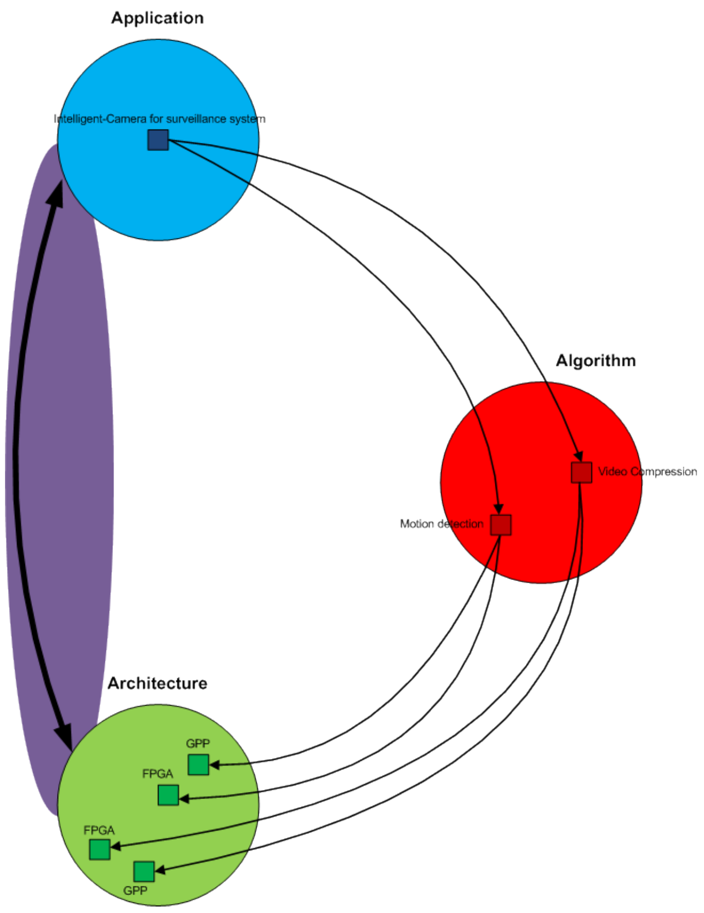
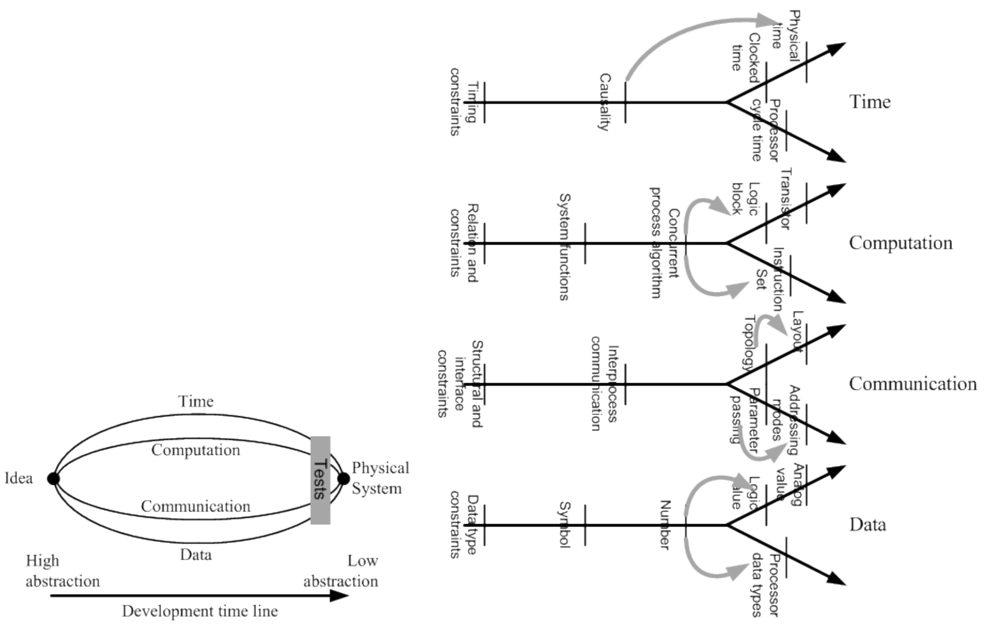


Figure 6.1: Location in the A^3 paradigm, highlighted by the purple circle

The first section of this chapter deals with the testing scenarios used to validate the implementation, to be sure to have a solid fully functional basis on which the implementation of other blocks can be followed through. The second section details the different experiments made to ensure that the implementation meets the



(a) Global location (b) Elaborated location

Figure 6.2: Location in the Rugby Meta-Model, highlighted by the grey bar on (a) and grey arrows on (b).

requirements and constraints defined in Chapter 1 and Chapter 3 (mainly the speed criteria).

6.1 Testing

This section presents the main test scenarios used to verify that the actual implementation is functioning as expected. The section is therefore mainly organized as the previous chapter, with each step of the implementation tested one after another.

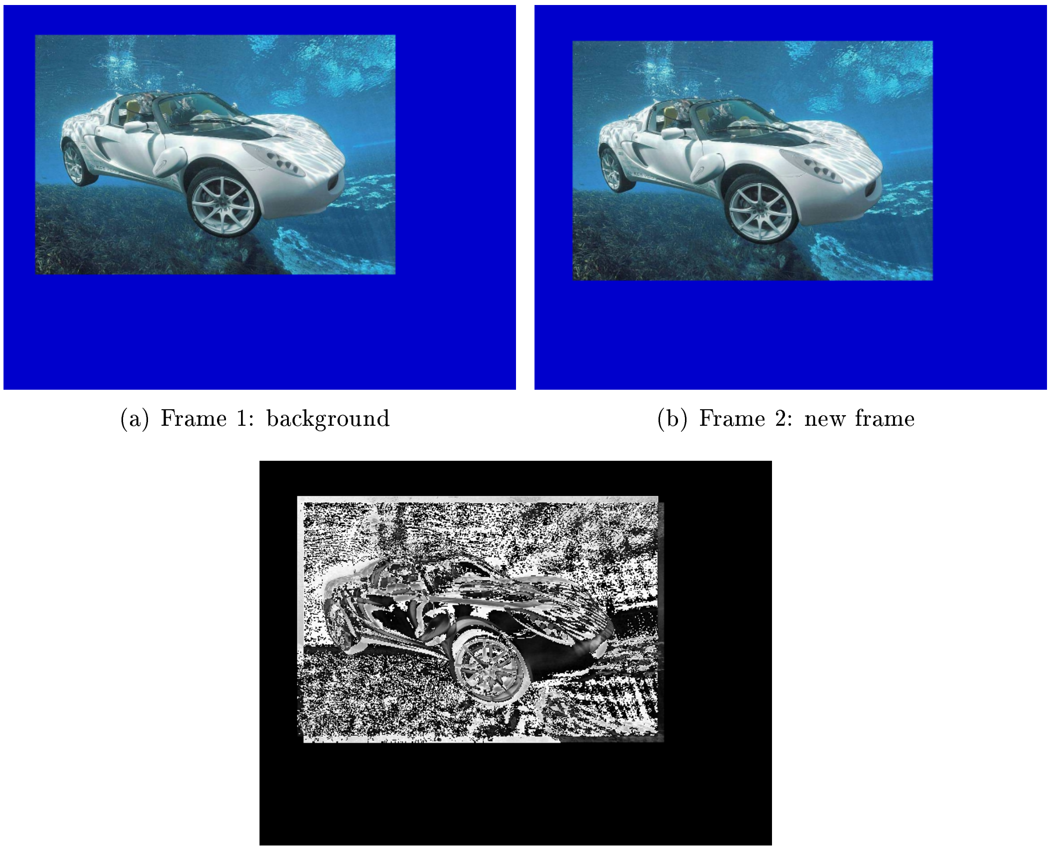
6.1.1 Transmission chain testing

The first step of the implementation needs to function as required for the project to continue. The test scenario used here is quite straightforward, as the main functionalities to be tested are the capture of a picture with a video camera and the display of this picture on a VGA screen. The scenario consists in capturing a picture with the camera sensor and displaying it on the VGA screen.

6.1.2 Motion detection algorithm testing

In order to validate the motion detection algorithm, the first draft was tested on a desktop computer, to be profiled and analysed with different metrics to calculate the algorithm's complexity for instance.(please refer to Chapter 4). Then, once the draft validated and the Nios II implemented, the final version of the algorithm is to be tested. The images used to test this algorithm both on the desktop computer and on the Nios II are the same. Those two pictures have the following characteristics:

- Resolution: 640x480 pixels;
- Colour depth: 24 bits;
- Moving element size: 450x300 pixels.



(a) Frame 1: background

(b) Frame 2: new frame

(c) Result of motion detection

Figure 6.3: Results of motion detection experiments: frame (a) is considered as the background frame, while frame (b) is the current frame. It is subtracted by the background to detect motion. (c) is the result of the comparison: black pixels are when no motion is detected and grey pixels enlightens the motion.

The results obtained are satisfactory, as the algorithm is able to detect a motion. The result of the test, running on the Nios II is shown in Figure 6.3 (page 71). This example shows that the algorithm running on the Nios II has the same behaviour as when running on a desktop computer.

6.2 Experiments

This section presents the main experiments done to verify that the implementation satisfies the requirements and constraints of the system.

6.2.1 Transmission chain testing

During this phase, several experiments were done to understand the existing blocks to adapt them to the project requirements and constraints in a first experiment. Then, after having modified the blocks, an experiment was performed to observe the data acquisition and to validate the tests results.

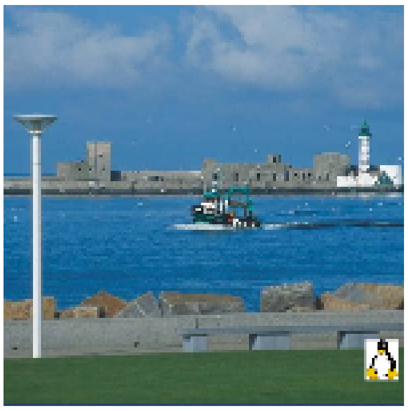
6.2.2 Motion detection algorithm experiments

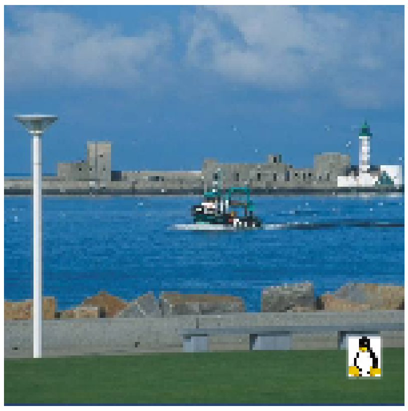
The experiments scenario is quite simple: compare two by two a set of 30 pictures. Those pictures have the following characteristics:

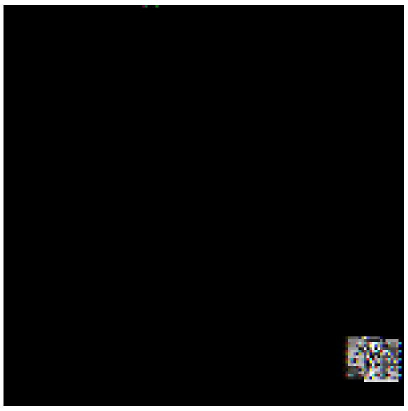
- Resolution: 150x150 pixels;
- Colour depth: 24 bits;
- Moving element size: 15x15 pixels (in Figure 6.4, the small "penguin" picture).

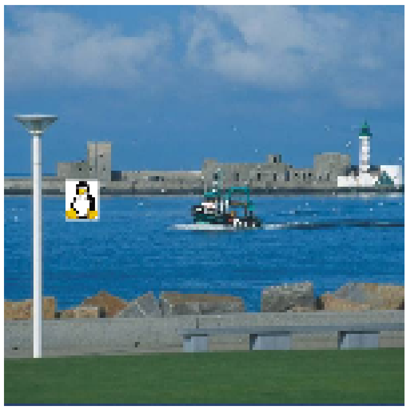
The pictures are located on the host computer, using the "host file system" possibility of the Nios II IDE, to facilitate results extraction and image storage on the FPGA.

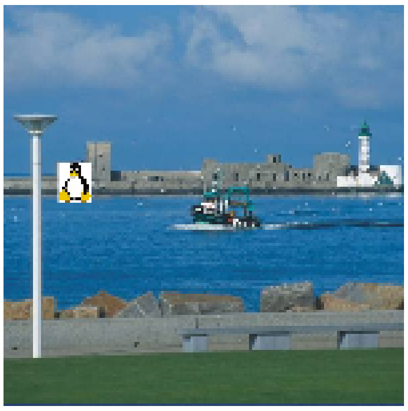
The results obtained are satisfactory, as the algorithm is able to detect a motion on a quite difficult background. Examples of the results are shown in Figure 6.4 (page 73). As the frames are subtracted, when no motion is detected, the result is black pixels. However, if a motion is detected, the result displays the pixels (in grey-scale value) considered as different form the background thus detected as moving objects. Those examples underline the complexity of the background. They also show that the resulted pictures only display the pixels in motion.


(a) Frame 1: background 1


(b) Frame 2: new frame 1


(c) Result of motion detection 1


(d) Frame 3: background 2


(e) Frame 2: new frame 2

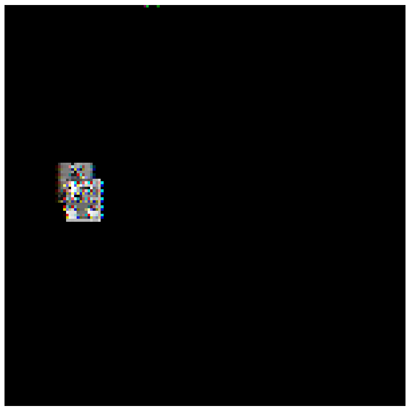

(f) Result of motion detection 2

Figure 6.4: Results of motion detection experiments: frames (a) and (d) are considered as background because they are the "old" frames. Frames (b) and (e) are the current frames, and are subtracted by the background to detect motion. Frames (c) and (f) are the results of the comparisons: black pixels are when no motion is detected and grey pixels enlighten the motion.

Chapter 7

Conclusion and Future Work

The first section of this chapter presents the conclusions of the project. The second section lists possible future work on this project, both for short- and long-term perspectives.

7.1 Conclusion

The first part of the conclusion is a reminder of the problem statement defined in the first chapter. Then the work done to attempt answering this question is summarized. Finally, a final answer is given and a possible way to answer differently to this problem is proposed.

7.1.1 What has been done to answer the problem

As defined in Chapter 1, the problem statement of this project is:
Is it possible to implement an intelligent IP video surveillance camera on a Field Programmable Gate Array (FPGA) platform to optimize video surveillance?

The following paragraph summarizes the main issues encountered during this project and then the main fully operational blocks that were implemented are described.

Problems encountered

The main technical problem encountered concerns memory. The first point is that it is impossible to give a read (or write) access of a single memory chip to two (or more) different blocks. That prevents conflicts on this specific chip. But that also forbid the softcore processor to access the memory where hardware blocks write pictures captured by the sensor. The second point is that the resolution of the sensor is quite big (640x480 pixels) and colour values of pixels are stored not compressed. This means that the memory chip must be bigger that the one used to record more than a few picture (less than ten now). Those memory problems have been a crucial concerns during the whole developpement.

A second technical problems is the SD-Card access. In facts, it requires to develop a custom driver from the SD-Card specification. Which means wasting a lot of time to study and understand this specification before implement and testing a driver just to use a tool.

An important point which became a problem is that even if there are Intellectual Property (IP) blocks for some parts of the system, it is neither easy nor straightforward to implement them together and get a final working prototype. In facts, each IP blocks has its own architecture requirements. Therefore, if one want to lower the development time by using IP blocks, most of the time one did not succeed by wasting time trying to fit the selected IP block to the desired system. The knowledge required to ease those IP blocks integration and communication is far from being negligible.

Another problem encountered is the skills required to use development tools and languages used during this project. In facts, the tools are quite specific and are dependant of the selected platform (Altera or Xilinx). This require to make choice very early during the project, often without the most complete view of it. Moreover, the learning curve is very steep in the beginning, the developers and designers have to learn how to use the provided tools and become efficient while using new languages. This learning curve should be minimized during early design choice, mostly because those difficulties reveal themselves along the project progresses. But it still stays high, as hardware and software co-design implies mixing different languages and tools together, and therefore understanding the communication between those. Which is a core issue as experienced with the memory management problem. Especially for people coming from a more software oriented development field and who are not new to the Hardware/Software Co-Desisgn concept. Moreover some of the documentations that could have been helpful have been published a few weeks before the end of this project, making them nearly impossible to benefit from them.

Operational blocks

Even if the previous problems did not allow the project to be fully finished, the following blocks are functioning according to the requirements and constraints previously defined. In facts, the transmission chain, the fundamental block, upon which the rest of the implementation is done, work perfectly.

The softcore processor block is also completely working. Moreover, some components can be added in this block to enhance the possibilities. With the softcore processor running, the motion detection algorithm is also fully implemented.

As detailed in Chapter 5 (page 55), studies about recording, storage and SD-Card have also been done. The specifications of constraints of the system and the detailed algorithm analysis and design also provide a clear methodology to continue this project.

7.1.2 Answer to the problem

The answer to the problem statement seems to be **"yes"**.It is presumably possible to implement an intelligent IP video surveillance camera on a FPGA platform to optimize video surveillance. Several points highlight this. In facts, with another development platform and a larger time frame, the main problem encountered could have been solved very The next paragraph presents a possible solution. Last but not least, the implementation of the algorithm on the FPGA is functioning. Therefore, recording and storing data only when an event happens is possible and "just" a step further. So optimizing video surveillance seems to be possible by using FPGA platform.

7.1.3 Solution proposal for a complete operational system

As explain above, neither of the accessible development board fitted perfectly the system design. A solution to achieve a prototype to show that using an FPGA can enhance a video surveillance camera is to build a custom development board. As a matter of facts, this custom board should present the following characteristics:

- Connectors:
  - VGA;
  - Ethernet;
  - JTAG;
  - GPIO;
  - Switches;
  - Buttons;
  - SD-Card controller.
- Chips:
  - FPGA (the Cyclone II was powerful enough);
  - SDRAM: 2x256 MegaBytes (one chip for the Nios II part and one for the hardware part);

7.2 Future works

This section give a non exhaustive list of possible future work. Short term future works are presented first, then followed by the long term ones.

7.2.1 Short term future works

This section presents the short terms future works to continue this project.

A first possibility could be to study and implement the SD-Card driver (or controller) especially with the new data availabe on the ALTERA website concerning this particular point. It allows data to be read and write from the NIOS II on an SD-Card and, therefore, avoids using host file system solution to test and experiments the algorithm. Which should also speed up the algorithm processing speed by avoiding the latency and overhead added by using the debug mode of the NIOS II and the JTAG cable.

A second short term improvement should be the implementation of some part of the Motion Detection algorithm directly into hardware function available for the NIOS II . The use of the C2H tools could be planed for instance.

A third short term possibility is to implement the web-server to allow a user to access data faster than through the JTAG connection. It also enhance the implementation of the SD-Card controller.

7.2.2 Long term future works

This section presents the long terms future works to improve this project

Recording and storing chain

As this part of the project require a specific development platform, it is considered as a long term future work. Moreover, this implementation could underline the answer to the problem given in the previous section. In fact, to implement this chain, it requires first to develop a custom development platform, as described in Section 7.1.3.

Video compression algorithm

A goal of this project was to implement a video compression algorithm. This allows the owner to save more space and money by reducing the number of memory card required to store surveillance data. A possible future work could be to implement this algorithm upon the video surveillance camera. This part can enlighten the advantages of having a platform based on an FPGA rather on a single processor: having two algorithms (motion detection and video compression) running on one chip also reduce data treatment costs.

Power consumption

Another constraint could be the power consumption as surveillance systems could be use in no-electricity environment. This constraint is not so relevant because even for surveillance systems, they are mostly plugged to a electricity source. Moreover, the constraint will depend mainly of the platform used. As a future work, one idea could be to reduce the power consumption of the global system. It can be done either by reducing or optimized the area used by the system on the FPGA , but also by studying the possibility of an ASICs implementation with a fully working prototype on FPGA development platform (including all chips in one ASIC: memories, FPGA , etc.).

Part II

Appendices & References

Glossary

This appendix presents several definitions.

A Hardware/software co-design definition

One of the key notion in this project is the hardware and software co-design [Wol03]. This concept has been created a little more than a decade ago in response to the raise of the embedded systems, the complexity and the heterogeneity of those systems. As an example, already in 1998, 98% of the processors were found in embedded systems. Embedded systems are single-functionned, tightly constrainted, reactive and realtime systems. When designing a embbedded system, the partitionning between hardware and software is a crucial point. Hardware has better performances but is more difficult to code contrary to software.

Figure 3.3, on page 27, enlightens that to deal with time and performance constraints, a designer has to make choices about partitionning. This partitionning and scheduling are the 2 main dimensions of the Co-Design. Software-hardware Co-Design can be summarized in the following way.
"Software/hardware co-design provides a way of customizing the hardware and the software architectures to complement one another in ways which improve system functionality, performance, reliability, survivability and cost/effectiveness."[SFL85]

B OS

It is the software component of a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer is called an operating system. The Operating System (OS) acts as an host for application programs that are running on the machine. As a host, one of the purposes of an operating system is to handle the details of the operation of the hardware.

C embedded OS

An embedded operating system is an OS for embedded systems. These OS are specialized, compact and efficient. They don't have all the functions the non-embedded computer have because they don't need them. The embedded OS are designed to operate as real-time operating systems.

D Hardware Description Language definition

In electronics, an Hardware Description Language or HDL is a computer language for formal description of electronic circuits. It can describe an operation made by a circuit, its design and organization. It can also simulate and tests to verify this operation.

A Hardware Description Language is written in standard text-based and describes the temporal behaviour circuit structure of an electronic system (for example FPGA). differing from the software programming languages, an HDL's syntax and semantics take in account the time and the concurrency between processes. The languages describing the circuit connectivity between a classified hierarchy of blocks are netlist languages.

Executable specifications for hardware can be written with HDLs. The simulation program as it is takes also the time in account allow the programmers to model a piece of hardware before it is created physically. With the hardware descriptions a software program called a synthesizer can infer hardware logic operations from the language statements and produce an equivalent netlist of generic hardware primitives to implement the specified behaviour on the platform.

As designing a system in HDL is generally much harder and more time consuming than writing the equivalent program in a C like language. To solve this difficulty, there has been much work done on automatic conversion of C code into HDL.

E System Description Language definition

A System Description Language (SDL)is more than a HDL because it can be used for designing the whole system, unlike HDL which are used only for coding blocks. Concerning their characteristics, the SDLs are:

- Made for System-on-Chip design
- Used to move to Higher levels of abstraction
- Used to design a complete system -> to be completed

F high-level programming language definition

A short definition of a high-level programming language is that they may be more abstract, easier to use or more portable across platforms than low-level languages.

High level languages are supposed to make complex programming simpler and low level languages make more efficient code. They consume more memory, have a larger binary size and are slower at the execution.

In fact the difference between low and high level languages could be very relative, originally, C language was considered as high level language and now depending on the context, it can be also considered as a low level language because it still allows memory to be accessed by address, and provides direct access to the assembly level

G softcore processor definition

A softcore processor processor is a processor implemented in to a reprogrammable system like a FPGA . It is a System on Programmable Chip or SoPC.

the softcore processor is a very flexible architecture, it can be reconfigured at any time, contrary to a hardcore processor whose core has it own non- reprogrammable chip. A softcore processor can be adapted to the material constraints (performances, resources, power consumption,...). However, softcore processor performances are inferior to the hardcore ones but a softcore processor is easier to maintain and can implemented into an ASIC (Application Specific Integrated Circuit).

H Embedded FPGA definition

FPGA means field-programmable gate array and stands for a semiconductor device containing programmable logic components called "logic blocks" including memories, and programmable interconnects.The designer can program the logic blocs and the interconnects like a one-chip programmable breadboard.

In the design flow of a new product, the early designs are using FPGAs then migrated into a fixed version that more resembles an ASIC. ASIC means application-specific integrated circuit and stands for an an integrated circuit customized for a particular use. To be configured, the FPGA has to be described by a logic circuit diagram or or a source code using a hardware description language (defined in another paragraph).

Table H, page 83, summarizes the pros and the cons of the FPGAs.

Pros	Cons
Time-to-market shorter than ASICs	Slower than ASICs
Development cost cheaper than ASICs	More expensive for mass productions than ASICs
lower non-recurring engineering costs than ASICs	More power consuming than ASICs
re-programmable in the field	

Table 1: Pros and cons of FPGA : despite its lower performances compared to ASICs, FPGAs are more flexible and easy to develop

I Other acronyms

I.1 SD definition

SD stands for Secure Card, a type of memory card.

I.2 MMC definition

MMC stands for Multimedia Card, a type of memory card.

I.3 ASIC definition

ASIC stands for Application Specific Integrated Circuit.

I.4 SoPC definition

SoPC stands for System on Programmable Chip.

Algorithm simplified call graph

Here is the simplified call graph of the software written in C, based on the algorithm used in the project, ran on a desktop computer.

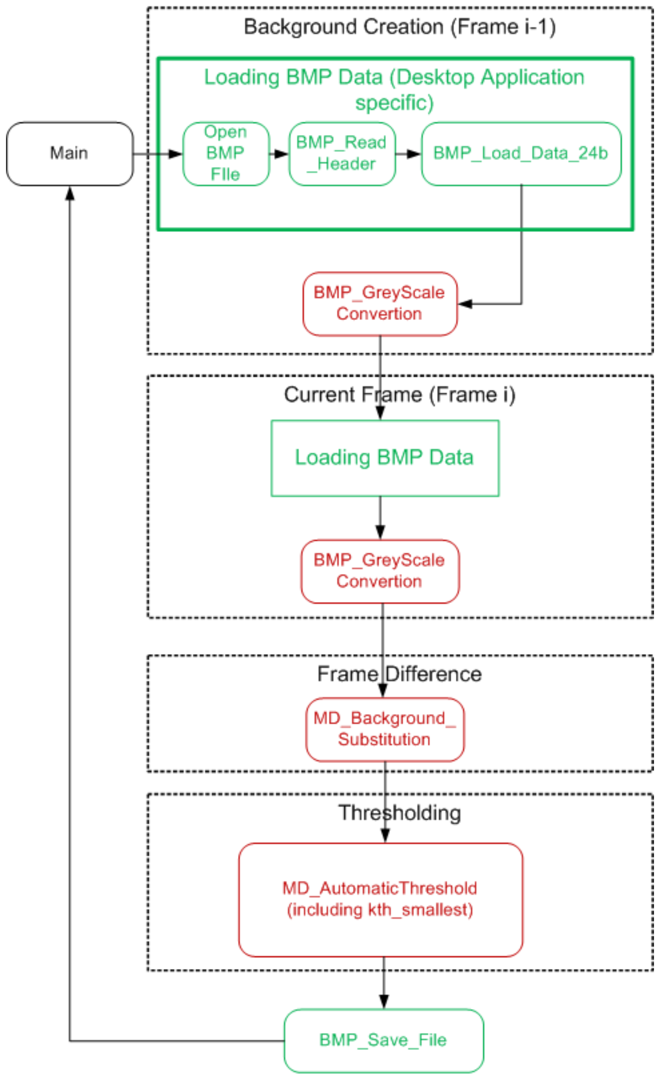
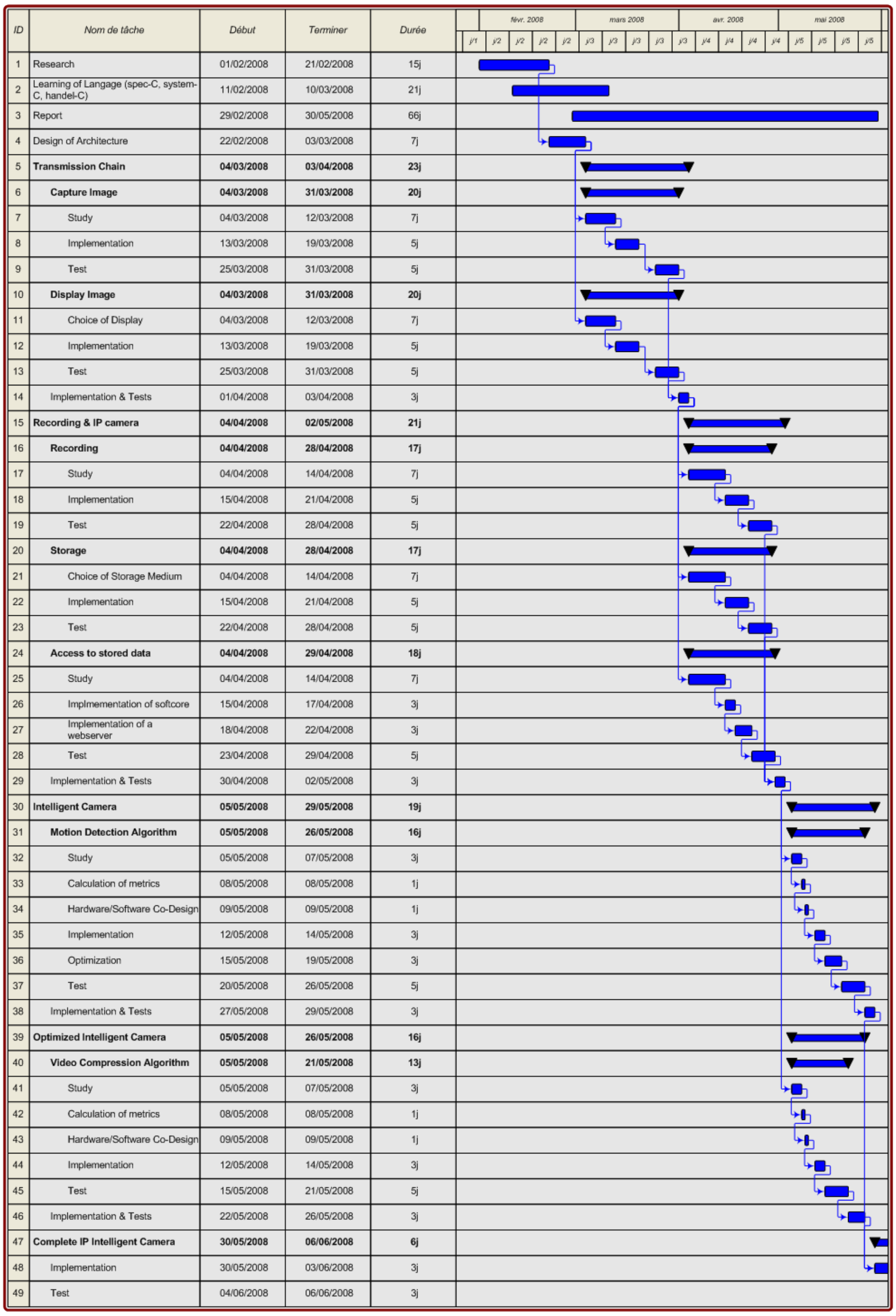


Figure 1: Simplified call graph of the C version of the algorithm: The green boxes are Desktop version specific (used to load and save BMP files) and the red boxes are the algorithm parts implemented in c

Time plan

The time plan of the project is Figure 2 on page 88.



ID	Nom de tâche	Début	Terminer	Durée
1	Research	01/02/2008	21/02/2008	15j
2	Learning of Langage (spec-C, system-C, handel-C)	11/02/2008	10/03/2008	21j
3	Report	29/02/2008	30/05/2008	66j
4	Design of Architecture	22/02/2008	03/03/2008	7j
5	**Transmission Chain**	**04/03/2008**	**03/04/2008**	**23j**
6	**Capture Image**	**04/03/2008**	**31/03/2008**	**20j**
7	Study	04/03/2008	12/03/2008	7j
8	Implementation	13/03/2008	19/03/2008	5j
9	Test	25/03/2008	31/03/2008	5j
10	**Display Image**	**04/03/2008**	**31/03/2008**	**20j**
11	Choice of Display	04/03/2008	12/03/2008	7j
12	Implementation	13/03/2008	19/03/2008	5j
13	Test	25/03/2008	31/03/2008	5j
14	Implementation & Tests	01/04/2008	03/04/2008	3j
15	**Recording & IP camera**	**04/04/2008**	**02/05/2008**	**21j**
16	**Recording**	**04/04/2008**	**28/04/2008**	**17j**
17	Study	04/04/2008	14/04/2008	7j
18	Implementation	15/04/2008	21/04/2008	5j
19	Test	22/04/2008	28/04/2008	5j
20	**Storage**	**04/04/2008**	**28/04/2008**	**17j**
21	Choice of Storage Medium	04/04/2008	14/04/2008	7j
22	Implementation	15/04/2008	21/04/2008	5j
23	Test	22/04/2008	28/04/2008	5j
24	**Access to stored data**	**04/04/2008**	**29/04/2008**	**18j**
25	Study	04/04/2008	14/04/2008	7j
26	Implmementation of softcore	15/04/2008	17/04/2008	3j
27	Implementation of a webserver	18/04/2008	22/04/2008	3j
28	Test	23/04/2008	29/04/2008	5j
29	Implementation & Tests	30/04/2008	02/05/2008	3j
30	**Intelligent Camera**	**05/05/2008**	**29/05/2008**	**19j**
31	**Motion Detection Algorithm**	**05/05/2008**	**26/05/2008**	**16j**
32	Study	05/05/2008	07/05/2008	3j
33	Calculation of metrics	08/05/2008	08/05/2008	1j
34	Hardware/Software Co-Design	09/05/2008	09/05/2008	1j
35	Implementation	12/05/2008	14/05/2008	3j
36	Optimization	15/05/2008	19/05/2008	3j
37	Test	20/05/2008	26/05/2008	5j
38	Implementation & Tests	27/05/2008	29/05/2008	3j
39	**Optimized Intelligent Camera**	**05/05/2008**	**26/05/2008**	**16j**
40	**Video Compression Algorithm**	**05/05/2008**	**21/05/2008**	**13j**
41	Study	05/05/2008	07/05/2008	3j
42	Calculation of metrics	08/05/2008	08/05/2008	1j
43	Hardware/Software Co-Design	09/05/2008	09/05/2008	1j
44	Implementation	12/05/2008	14/05/2008	3j
45	Test	15/05/2008	21/05/2008	5j
46	Implementation & Tests	22/05/2008	26/05/2008	3j
47	**Complete IP Intelligent Camera**	**30/05/2008**	**06/06/2008**	**6j**
48	Implementation	30/05/2008	03/06/2008	3j
49	Test	04/06/2008	06/06/2008	3j

Figure 2: GANTT Diagram of the Project

CD-Rom content

The CD-Rom of the report contains the following items:

- Electronic version of the present report;
- RTL view of all hardware blocks;
- Set of 30 pictures and results used in the algorithm experiment;
- Algorithm source code (Nios II version);
- Hardware blocks source code.

References

[AAS⁺07] Rasmus Abildgren, Aleksandras, Saramentovas†, Paulius Ruzgys†, Peter Koch, and Yannick Le Moullec. Algorithm-architecture affinity – parallelism changes the picture. 2007.

[Air83] United States Airforce. *VHSIC Hardware Description Language* , 1983.

[ASR⁺05] Rasmus Abildgren, Aleksandras Saramentovas, Paulius Ruzgys, Peter Koch, and Yannick Le Moullec. Algorithm-architecture affinity – parallelism changes the picture. *Aalborg University*, 2005.

[Ass06] Technical Committee SD Card Association. *SD Specifications, Part 1: Physical Layer, Simplified Specification*, 2006.

[Cel05] Celoxica. *Handel-C Language Reference Manual*, 2005.

[CG05] Cadence and Mentor Graphic. *Open SystemC Language Reference Manual*, 2005.

[CGPP03] Rita Cucchiara, Costantino Grana, Massimo Piccardi, and Andrea Prati. Detecting moving objects, ghosts and shadows in video streams. 2003.

[Com07] International Standard Comittee. *Programming language C* , 2007.

[Cor06a] Altera Corporation. *DE2 Development and Education Board User manual 1.4 (Nios II, Cyclone II)*, 2006.

[Cor06b] Xilinx Corporation. *RC203 development and education board Datasheet*, 2006.

[Cor07] Altera Corporation. *Nios II Processor Reference Handbook*, 2007.

[Cor08] Altera Corporation. *Stratix Device Handbook*, 2008.

[Cus03] Maurice Cusson. La vidéosurveillance : les raisons de ses succès et de ses échecs. *http://www.criminologie.com/cusson/cusvideo.pdf*, 2003. written in French, last visit date: April 2008.

[Dev98] Nicolas Devillard. Fast median search: an ansi c implementation. 1998.

[Döm03] Rainer Dömer. The SpecC Language. 2003.

[DTm03] *Multi-Granularity Metrics for the Era of Strongly Personalized SOCs*, 2003. Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03).

[fou06] Python Software foundation. *Python reference manual* , 2006.

[Gaj88] Daniel D. Gajski. *Silicon Compilation.* Addison-Wesley Publishing Company, 1988.

[GHC07] Jiri Gaisler, Sandi Habinc, and Edvin Catovic. *GRLIB IP librairy User's manual*, 2007.

[GKM] Susan L. Graham, Peter B. Kessler, and Marshall K. McKusick. gprof: a Call Graph Execution Profiler.

[HJK99] Ahmed Hemani, Axel Jantsch, and Shashi Kumar. The rugby model: A framework for the study of modeling, analysis, and synthesis concepts in electronic systems. *Proceedings of Design Automation and Test in Europe (DATE)*, 1999.

[HJK00] Ahmed Hemani, Axel Jantsch, and Shashi Kumar. The rugby metamodel. *Electronic System Design Lab*, 2000.

[IDT95] *Image difference threshold strategies and shadow detection*, 1995. Proceedings of the 6th British Machine Vision Conference.

[Inc07] Xilinx Incorporation. *MicroBlaze Processor reference Guide*, 2007.

[Inc08a] Altera Incorporation. *Nios II Software Developer's Handbook*, 2008.

[Inc08b] Altera Incorporation. *Quartus II Development Software Handbook v8.0*, 2008.

[inc08c] The MathWorks incorporation. *Matlab 7 Desktop Tools and Development Environment*, 2008.

[KG83] Robert H. Kuhn and Daniel D. Gajski. Guest editor's introduction: New vlsi tools. *IEEE Computer, pages 11-14*, 1983.

[Koc] Peter Koch. Hardware/Software Codesign constraints.

[Lab02] Jean J. Labrosse. *MicroC/OS-II.* CMP Books, 2002.

[Lae07] Arnim Laeuger. Sd/mmc bootloader. *http://www.opencores.org/projects.cgi/web/spi_boot/overview*, 2007. last visit date: April 2008.

[Lu] Ping-Hong Lu. Light source motion tracking (using terasic de2 board). http://instruct1.cit.cornell.edu/courses/ece576/FinalProjects/f2007/pl328/pl328/index : Visited 22/05/2008.

[MDA+05] Yannick Le Moullec, Jean-Philippe Diguet, Nader Ben Amor, Thierry Gourdeaux, and Jean-Luc Philippe. Algorithmic-level Specification and Characterization of Embedded Multimedia Applications with Design Trotter. *Journal of VLSI Signal Processing 42*, 2005.

[Mou08] Yannick Le Moullec. ASPI8-S2-11 : HW/SW Co-design - MM5 : Automated HW/SW partitioning and metrics for HW/SW partitioning Slides, 2008.

[oI] International Commission on Illumination. Itu-r recommendation bt.601-2. *Encoding parameters of digital television for studios.*

[Pic04a] Massimo Piccardi. Background subtraction techniques: A review. 2004 IEEE International Conference on Systems, Man and Cybernetics, IEEE, 2004.

[Pic04b] Massimo Piccardi. Background subtraction techniques: A review. The ARC Centre of Excellence for Autonomous Systems (CAS) Faculty of Engineering, University of Technology (UTS), Sydney, 2004.

[Por07] Maciej Portalski. Hardware aspects of fixed relay station design for ofdm(a) based wireless relay networks. Master's thesis, Aalborg University, Department of Electronics Systems, 2007.

[PT05] David Pellerin and Scott Thibault. *Practical FPGA Programming in C.* CMP Books, 2005.

[Ros95] Paul L. Rosin. Thresholding for change detection. 1995.

[SFL85] Connie U. Smith, Geoffrey A. Frank, and John L. An architecture design and assessment system for software/hardware codesign. 1985.

[TDMP85] Thomas, Donalda, Moorby, and Phillip. *The Verilog Hardware Description Language*, 1985.

[tea07] "Teaching team". Applied signal processing and implementation, introduction slides. 2007.

[Tec06] Terasic Technologies. *TRDB_DC2 Camera Development Package User Guide*, 2006.

[Uni89] Oxford University. Oxford english dictionary. 1989.

[Wika] Wikipedia.org. *C++ Programming.* http://en.wikipedia.org/wiki/C%2B%2B : Visited 22/05/2008.

[Wikb] Wikipedia.org. Performance analysis. http://en.wikipedia.org/wiki/Performance_analysis : Last Visited 22/05/2008.

[Wikc] Wikipedia.org. Software metric. http://en.wikipedia.org/wiki/Software_metric : Visited 22/05/2008.

[Wikd] Wikipedia.org. Verilog (internet). http://en.wikipedia.org/wiki/Verilog : Visited 22/05/2008.

[Win06] Hermann Winkler. Ddr sdram controller core. *http://www.opencores.org/projects.cgi/web/ddr_sdr/overview*, 2006. last visit date: April 2008.

[Wir] Niklaus Wirth. *Algorithms + Data structures = Programs.*

[Wol03] Wayne Wolf. A Decade of Hardware/Software Codesign. *IEEE*, 2003.