MULTIPROCESSING SYSTEMS DEVELOPMENT FOR IMPLEMENTATION OF APPLICATIONS

Doctoral Dissertation

Calliope-Louisa Sotiropoulou, M.Sc.

ARISTOTLE UNIVERSITY OF THESSALONIKI

FACULTY OF SCIENCE

SCHOOL OF PHYSICS

December 2014

ΑΝΑΠΤΥΞΗ ΠΟΛΥΕΠΕΞΕΡΓΑΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΓΙΑ ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΩΝ

Διδακτορική Διατριβή

Καλλιόπη-Λουίζα Σωτηροπούλου, M.Sc.

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΕΙΟ ΘΕΣΣΑΛΟΝΙΚΙΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΦΥΣΙΚΗΣ

Δεκέμβριος 2014

"You mustn't be afraid
to dream a little bigger, darling…"

Eames, from Inception

Dedicated to my husband, my parents and to all creative minds…

Calliope-Louisa Sotiropoulou

Aristotle University of Thessaloniki

ABSTRACT

MULTIPROCESSING SYSTEMS DEVELOPMENT FOR IMPLEMENTATION OF APPLICATIONS

This dissertation presents the author's research in the field of multiprocessing systems for implementation of applications. The main objective is the identification of the optimum heterogeneous multiprocessing design architecture for a given application. This objective is tackled though two different approaches: a) formulation of a theoretical model that can be used to optimize each system according to the application's specification and platform's characteristics and b) direct approach through finding the optimum architecture for two characteristic and high performance applications with hard real-time requirements.

The developed theoretical model formulation is based on Integer Linear Programming and can be used to identify the optimum architecture and task assignment for a given application and a specific platform. The optimization can be executed with a variety of objective functions that express the main characteristics of a system, such as performance and usage of hardware resources, and additional characteristics, such as usage of memory resources, power and temperature. Objective functions can even be combined with varying importance weights if necessary. Emphasis was laid on the efficient usage of hardware resources by studying the correlation of performance increase and increase in resource usage. The introduction of memory resources, power and temperature did not add significant complication to the initial model. The final model can be used for heterogeneous MPSoC optimization early in the design development stage.

The first application used to implement a working example of an optimized MPSoC system is a real-time machine vision system for real-time flow detection on microfluidic Lab-on-Chips. The system is designed to follow a 60 fps camera with 1 Mpixel resolution. To achieve this performance a highly parallel implementation was designed. The system is robust against changing lighting conditions and small displacements. High performance modules for center of mass and median calculations were designed, and additionally an application specific alarm point

detection module. These modules are generic and easily adjustable to various image processing applications. Additionally, bit accurate simulation was used to specify the precision required for the edge detection preprocessing step. The system achieves the required performance by a significant margin and comparison with previous machine vision implementations on various platforms demonstrates that the system has the best performance for the same or bigger video frame resolutions.

The second application used as a working example is a high performance 2D pixel clustering implementation for streaming data. The implementation was originally designed to be used in the ATLAS Fast TracKer processor, an upgrade for the Trigger and Data Acquisition system of the ATLAS detector. The system uses a moving window technique that exploits the 2D fabric of the FPGA to reduce looping of data for cluster identification. In addition, it is fully generic and modular, and can be implemented with multiple clustering engines working in parallel. A critical factor is the avoidance of all possible bottlenecks in data throughput. Therefore special provision was taken and exploration was executed to prevent backpressure to previous elements in the processing chain with the use of appropriate buffering techniques. The system achieves the required performance of processing data with 40 MHz input rate by using 4 parallel engines. Comparisons demonstrate that previous implementations would require 64 times more resources to achieve equivalent performance.

Concluding, this thesis presents a twofold approach to multiprocessing system design. First, a theoretical model that can be used to optimize system architecture and application task assignment at an early design stage was developed and, second, hands on optimized MPSoC implementations for two computationally intensive applications were presented. The systems presented are optimized for parallelism, memory access, buffering, so that performance requirements are met but also no bottlenecks can hinder data throughput.

ΠΕΡΙΛΗΨΗ

ΑΝΑΠΤΥΞΗ ΠΟΛΥΕΠΕΞΕΡΓΑΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΓΙΑ ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΩΝ

Το αντικείμενο αυτής της διδακτορικής διατριβής είναι η ανάπτυξη πολυεπεξεργαστικών συστημάτων για υλοποίηση εφαρμογών. Ο βασικός στόχος της διατριβής είναι ο εντοπισμός του βέλτιστου ετερογενούς πολυεπεξεργαστικού συστήματος για την υλοποίηση κάθε συγκεκριμένης εφαρμογής. Ο στόχος αυτός επιτυγχάνεται μέσω δύο διαφορετικών προσεγγίσεων: α) διατύπωση ενός θεωρητικού μοντέλου που μπορεί να χρησιμοποιηθεί για τη βελτιστοποίηση κάθε συστήματος σύμφωνα με τις απαιτήσεις κάθε εφαρμογής και β) ευθεία αντιμετώπιση του προβλήματος με ανεύρεση της βέλτιστης αρχιτεκτονικής για δύο χαρακτηριστικές εφαρμογές υψηλών απαιτήσεων με ανελαστικές προδιαγραφές απόκρισης σε πραγματικό χρόνο.

Το θεωρητικό μοντέλο που αναπτύχθηκε στηρίζεται στο Γραμμικό Προγραμματισμό Ακεραίων και μπορεί να χρησιμοποιηθεί για να εντοπίσει τη βέλτιστη αρχιτεκτονική αλλά και τη βέλτιστη ανάθεση εργασιών σε κάθε επεξεργαστική μονάδα για κάθε εφαρμογή και συγκεκριμένη πλατφόρμα υλοποίησης. Η βελτιστοποίηση μπορεί να εκτελεστεί με μια ποικιλία από συναρτήσεις στόχου (objective functions) που εκφράζουν τα βασικά χαρακτηριστικά του συστήματος όπως επίδοση και χρήση πόρων, αλλά και επιπρόσθετα χαρακτηριστικά όπως η χρήση πόρων μνήμης, ισχύς και θερμοκρασία. Συναρτήσεις στόχου μπορούν να συνδυαστούν σε μία ενιαία συνάρτηση με διαφορετικό βαθμό σημασίας για την καθεμία. Δόθηκε έμφαση στην αποδοτική χρήση των πόρων του συστήματος μελετώντας τη συσχέτιση ανάμεσα στη βελτίωση των επιδόσεων και την αύξηση στη χρήση πόρων. Η εισαγωγή παραμέτρων όπως η χρήση πόρων μνήμης, ισχύος και θερμοκρασίας έγινε χωρίς σημαντική αύξηση στην πολυπλοκότητα του αρχικού μοντέλου. Το τελικό μοντέλο μπορεί να χρησιμοποιηθεί για τη βελτιστοποίηση ετερογενών πολυεπεξεργαστικών συστημάτων ακόμα και σε αρχικό στάδιο της διαδικασίας ανάπτυξης του συστήματος.

Η πρώτη εφαρμογή που χρησιμοποιήθηκε για την υλοποίηση ενός λειτουργικού πολυεπεξεργστικού συστήματος είναι ένα σύστημα μηχανικής όρασης με απαιτήσεις απόκρισης σε πραγματικό χρόνο για την ανίχνευση ροών σε μικρορροϊκά Lab-on-Chip. Το σύστημα έχει σχεδιαστεί για να ακολουθεί κάμερα με ταχύτητα 60 καρέ το δευτερόλεπτο και ανάλυση 1 Mpixel. Για να επιτευχθεί αυτή η επίδοση σχεδιάστηκε ένα σύστημα με υψηλή παραλληλία. Το σύστημα μένει ανεπηρέαστο από τις αλλαγές φωτεινότητας και τις πιθανές μικρομετακινήσεις. Υποσυστήματα υψηλών επιδόσεων σχεδιάστηκαν για τον υπολογισμό του κέντρου βάρους, αλλά και του διαμέσου, καθώς και ένα σύστημα που ανιχνεύει σημεία ενδιαφέροντος πάνω στις ροές. Τα υποσυστήματα αυτά είναι πλήρως διαμορφώσιμα και μπορούν να προσαρμοστούν σε ποικίλες εφαρμογές επεξεργασίας εικόνας. Επιπροσθέτως, προσομοίωση με ακρίβεια bit χρησιμοποιήθηκε για τον καθορισμό της απαιτούμενης ακρίβειας στο στάδιο προεπεξεργασίας ανίχνευσης ακμών. Το σύστημα επιτυγχάνει με άνεση τις απαιτούμενες επιδόσεις. Σύγκριση με προηγούμενες υλοποιήσεις αντίστοιχων συστημάτων μηχανικής όρασης σε διάφορες πλατφόρμες αποδεικνύουν ότι το σύστημά μας έχει την καλύτερη επίδοση για ίδια ή και μεγαλύτερη ανάλυση κάμερας.

Η δεύτερη εφαρμογή που χρησιμοποιήθηκε ως λειτουργικό παράδειγμα είναι μια υλοποίηση ενός δισδιάστατου αλγορίθμου συσταδοποίησης εικονοστοιχείων (2D pixel clustering) υψηλών απαιτήσεων. Η υλοποίηση αυτή αρχικά σχεδιάστηκε για τον επεξεργαστή Fast TracKer του ATLAS, μία αναβάθμιση του συστήματος δειγματοληψίας του ανιχνευτή ATLAS. Το σύστημα χρησιμοποιεί μία τεχνική μετακινούμενου παραθύρου ανίχνευσης που εκμεταλλεύεται τη δισδιάστατη δομή του FPGA για την αποφυγή των πολλών κυκλικών μετακινήσεων των δεδομένων. Επιπλέον, είναι πλήρως παραμετροποιημένη και μπορεί να υλοποιηθεί με πολλές μηχανές συσταδοποίησης να εργάζονται παράλληλα. Ένα κρίσιμο στοιχείο στην υλοποίηση είναι η αποφυγή οποιασδήποτε ανάσχεσης στην ανεμπόδιστη διέλευση δεδομένων, γι’ αυτό και δόθηκε ιδιαίτερη έμφαση στην απρόσκοπτη επεξεργασία δεδομένων και στη χρήση των απαιτούμενων αποθηκευτικών στοιχείων. Το σύστημα επιτυγχάνει τις απαιτούμενες επιδόσεις των 40 MHz ρυθμού επεξεργασίας δεδομένων εισόδου με χρήση 4 παραλλήλων μηχανών συσταδοποίησης. Συγκρίσεις

με προηγούμενες υλοποιήσεις αποδεικνύουν ότι θα απαιτούνταν χρήση πόρων 64 φορές περισσότερων για την επίτευξη αντίστοιχων επιδόσεων.

Συμπερασματικά, η διατριβή αυτή παρουσιάζει μια διπλή προσέγγιση στην ανάπτυξη πολυεπεξεργαστικών συστημάτων. Πρώτα ένα θεωρητικό μοντέλο το οποίο μπορεί να χρησιμοποιηθεί για τη βελτιστοποίηση των αρχιτεκτονικών των συστημάτων και την ανάθεση εργασιών στις επεξεργαστικές μονάδες ακόμα και νωρίς στη διαδικασία ανάπτυξης του συστήματος. Στη συνέχεια, δύο βελτιστοποιημένες υλοποιήσεις πολυεπεξεργαστικών συστημάτων για εφαρμογές υψηλών απαιτήσεων. Τα συστήματα που παρουσιάζονται είναι βελτιστοποιημένα ως προς την παραλληλία, τις προσβάσεις στη μνήμη και τις μνήμες προσωρινής αποθήκευσης, έτσι ώστε τα συστήματα να επιτυγχάνουν τις απαιτούμενες προδιαγραφές και να υπάρχει διαρκής απρόσκοπτη διέλευση των δεδομένων.

Acknowledgments

This dissertation reports the research results produced by the author during the years 2009-2014 as a PhD Candidate in the Electronics Laboratory of the School of Physics in Aristotle University of Thessaloniki.

Research for a PhD dissertation is a long and mentally challenging process and I am using this opportunity to express my gratitude to everyone who supported me throughout the course of my PhD.

First I want to express my special appreciation and thanks to my supervisor Associate Professor Spyridon Nikolaidis. Prof. Nikolaidis has been my supervisor as an undergraduate, post-graduate and PhD student. He has put great faith and support in my abilities and with his guidance I grew not only as a researcher but as a person as well. I also want to thank the other two members of my advisory committee, Professor T. Laopoulos (Vice Rector - AUTH) and Assistant Professor G. Theodoridis (University of Patras), for their continuous supervision and support during my thesis.

I must also thank Professor S. Siskos for his help and support and for participating in the review committee. My appreciation is also expressed for E. Nikolaidis and all the personnel from the Electronics Laboratory for their help all this years. In addition, I would like to thank the review committee members Professor A. Hatzopoulos (AUTH), Associate Professor D. Soudris (NTUA) and Associate Professor G. Sirakoulis (DUTH). I would like to especially thank Assistant Professor K. Kordas (AUTH) for giving me the opportunity to work on the FTK-IAPP project.

During this thesis I had great help from my fellow PhD candidates and friends in AUTH, C. Gentsos, L. Voudouris and S. Gkaitatzis. I must also thank T. Izawa, T. Mitani and Dr. N. Kimura from Waseda University (Japan) for their help with testing and debugging the Pixel Clustering system at CERN. I must extend my appreciation also to Micro2Gen Ltd. and especially to Dr. A. Demiris for our great collaboration during the Corallia-LoC project.

A great thank you to Dr. P. Giannetti (Director of Research – INFN Pisa) and Prof. M. Dell' Orso (Pisa University) for the trust and faith they put in my abilities and for giving me the opportunity to work in creative research tasks. Their enthusiasm and dedication to the FTK project fueled my work. A special thank you goes to Dr. A. Annovi (INFN Pisa – ATLAS FTK Project Leader) for his constructive input, his continuous feedback, his guidance and his commitment to the FTK project. My deepest appreciation goes to the FTK group in Pisa University, M. Piendibene, S. Citraro, P. Luciano and D. Magalotti, and CAEN SpA for making me feel at home during my stay there and making my time in the FTK lab as creative and fun as possible.

Special thank you and love goes to my parents and close family for teaching me to appreciate science and education from a very young age and for supporting me in all my choices. My deepest love and appreciation is dedicated to my husband for his understanding, patience and support, for understanding how my mind works and knowing when to bring me down to earth. Without him as a pillar I wouldn't have been able to finalize this thesis.

The author,

Calliope-Louisa Sotiropoulou

Thessaloniki, 10/11/2014

Table of Contents

List of Figures

List of Tables

List of Equations

Nomenclature

2D	Two Dimensional
AM	Associative Memory
ASIC	Application Specific Integrated Circuit
ATCA	Advanced Telecommunications Computing Architecture
ATLAS	A Toroidal LHC Apparatus
CAM	Content Addressable Memory
CCN	Completely Connected Network
CERN	European Organization for Nuclear Research (Conseil Européenne pour la Recherche Nucléaire)
CMS	Compact Muon Solenoid
CoM	Center of Mass
COP	Constrained Optimization Problem
CPU	Central Processing Unit
DF	FTK Data Formatter
DSP	Digital Signal Processor
EF	Event Filter
FE	Front End chip
FPGA	Field Programmable Gate Array
fps	Frames per Second
FSL	Fast Simplex Links
FTK	Fast TracKer Processor
FTK_IM	FTK Input Mezzanine
GPU	Graphics Processing Unit
HiPEAC	High Performance and Embedded Architecture and Compilation
HLT	High Level Trigger
IBL	Insertable B-Layer
ILP	Integer Linear Programming
LHC	Large Hadron Collider
LoC	Lab On Chip
LVL1	Level 1
MPSoC	Multiprocessing System on Chip
MSE	Mean Squared Error
NMS	Non Maximum Suppression
RISC	Reduced Instruction Set Computer
ROD	Read Out Driver
RoI	Region of Interest
S-LINK	Serial Link
SoC	System On Chip
TDAQ	Trigger and Data Acquisition
TeV	Tera electron Volts
ToT	Time Over Threshold
VHDL	VLSI Hardware Descritpion Language
VLSI	Very Large Scale Integration

Chapter 1

Introduction

1.1. Embedded Systems

Modern embedded systems are no longer used for simple “computing” tasks. They are the most pervasive technology in our lives, used in almost all devices that involve a certain level of technology: telephones, cars, televisions, cameras, all kinds of home appliances, security systems, and in devices that require high performance for data processing such as surveillance cameras, medical imaging, tracking for high energy physics, astronomical imaging, DNA sequencing etc. Many of the tasks at hand for these new generation embedded systems require high performance processing and high data throughput with increased algorithmic complexity. To implement such powerful systems complicated hardware design is required. However, the constant scaling of process technology leads to an ever increasing cost of designing and developing Very Large Scale Integrated Circuits (VLSI circuits).

To cope with the increasing demands of the current application domains embedded systems need to have high processing power which in turn leads to high power consumption. In Figure 1.1 the key challenges in mobile device industry are presented [1]. The step function demonstrates the gain in cellular transmission over the change of protocols over time. This gain follows Shannon’s law [3] for algorithmic complication. It can be seen that the transmission performance is

doubled in 8.5 months. Moore's Law [2] demonstrates that 18 months are required to double the number of transistors and therefore double processor performance. Additionally, 10 years are required for batteries to double their energy density and 12 years are required to double memory access time. It is obvious that the gaps in the evolution time between the different technologies create obstacles to the development and commercialization of systems and products in the communications field. Mobile devices industry is a very popular application field with constant development and evolution and very indicative of the bottlenecks that appear in the development of high performance embedded systems. It is obvious that an alternative approach to embedded system design is essential to cope with the bottlenecks in technology evolution.

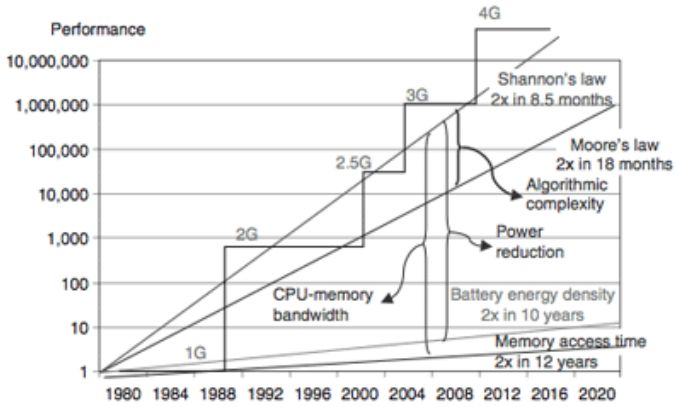


Figure 1.1. Processing Requirements Combined with Moore's Law, Battery Capacity and Memory Access Time

The answer to the increasing cost of developing high performance embedded systems on ASIC technology is the use of flexible multiprocessing systems. An additional advantage is the implementation of such architectures on reconfigurable devices such as FPGAs. With increasing levels of integration, it is now feasible to integrate heterogeneous systems entirely on a single chip. The design of such systems is complex, and the complexity is increasing with the technology evolution. In parallel, mapping applications on such systems follows a parallel complexity curve [4].

1.2. Multiprocessing Systems

Multiprocessing systems are not a new concept. The first multiprocessing systems were systems that used multiple processors that were in fact several separate CPUs [5]. What has advanced significantly over the past decade is the number of processing elements that can be integrated on a single device (ASIC, FPGA, GPU etc.) that are now defined as a multiprocessing system.

Multiprocessing systems can be *homogeneous* or *heterogeneous*. *Homogeneous* systems use identical processors with the exact same characteristics: architecture, clock speed, memory structure and size etc. *Heterogeneous* systems have processing elements that differ; they can be general processors, DSPs, hardware accelerators, or even general processors with different characteristics depending on the application's needs. Identical MPSoCs are common in computing systems that are designed for general use or use for a broad field of applications: e.g. processors for desktop computers [6], for mobile devices [7], even space grade processors [8]. *Heterogeneous* MPSoCs target application specific fields exactly because they are optimized for specific applications. Heterogeneous MPSoCs require more development time but are better suited performance wise and consumption wize to the application they are designed for. Heterogeneous MPSoCs are the new holy grail of embedded systems.

1.3. The Five Laws of Embedded Systems

There are two fundamental laws that govern the evolution of integrated circuits and electronics in general: Moore's law and Amdahl's law. Three additional laws that govern software development have also been suggested by N. Myhrvold. As a whole, these five laws describe the evolution of computing systems in general.

1.3.1. Moore's Law

Moore's law derives from the observation that throughout the history of computing hardware the number of transistors in dense integrated circuits doubles every two years (Figure 1.2). The exponential law is often adjusted to chip performance that, as attributed to Intel executive David House, is said to double over

eighteen months since the transistors do not only increase in number, but are faster as well.

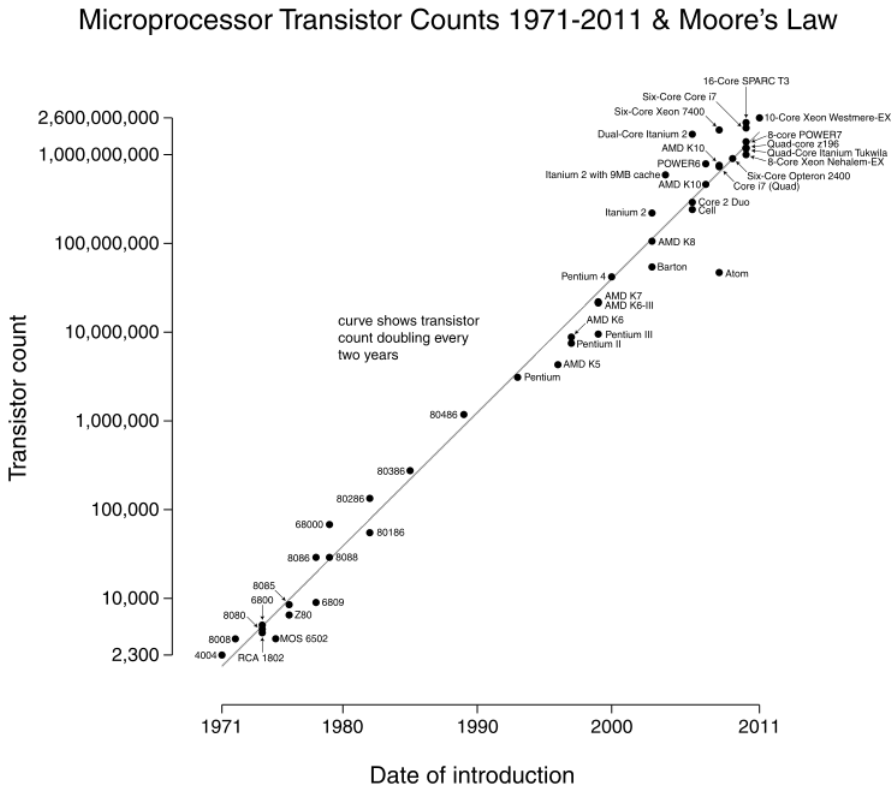


Figure 1.2. Moore's Law

Moore's law has been the driving force of electronics evolution ever since it was first introduced. However, it is bound to reach its limit due to technological limitations. The size of transistors cannot be reduced indefinitely. Already technology is facing the quantum limitations of transistor miniaturization. The solution is the multi-gate design approach (such as FinFET technology [9]). It is obvious that the trend in integrated circuit evolution is towards multiprocessing systems (Figure 1.2). The reason behind this trend is that increasing operational clock speeds is no longer a viable solution to increase system performance. Faster clock speed means more power consumption and more heat dissipation. These are even bigger bottlenecks than technology itself. Placing more cores per chip is the answer to performance. But being able to extract the performance from a multicore

system is a challenge on its own right. Part of this challenge is best described by Amdahl's law next.

1.3.2. Amdahl's Law of Diminishing Returns

Amdahl's Law is the statement that the speedup in parallel processing of an algorithm is always limited by the fraction of the problem that must be executed sequentially. The mathematical formulation of the law is presented in Equation 1.1. *TSerial* is the total sequential execution time, *TParallel* is the total parallel execution time and *N* is the number of parallel processors. *SpeedUp* factor is always smaller than the number of parallel processors, since *TSerial* can never be zero.

$$SpeedUp = \frac{TSerial + TParallel}{TSerial + \frac{TParallel}{N}}$$

Equation 1.1. Amdahl's Law

In Figure 1.3 the correlation between the fraction of sequential execution time and speedup is very vividly demonstrated. When 50 % of the execution time is sequential, speedup cannot be increased further than 2 irrespective of the number of parallel processors used. There is always an upper bound of speedup that can be achieved in an algorithm and it is dependent only the sequential time fraction.

It should be noted however that there are no naturally occurring parallelisms in algorithms. Parallelism can only occur when all internal dependencies are removed and it is a complicated task to fulfill. Additionally, properly assigning these tasks to processing elements to achieve parallelization is not a straightforward process as there is much more to processing time to take into account, such as memory sharing, power, message passing delays and more.

In [10] Cassidy and Andreou raise a number of interesting questions about the number of processors that are efficient and effective per application, the organization of local memories and the architectural organization of the multiprocessing systems that optimize performance. They combine two targets, speed and energy, in one objective function to find the optimal multiprocessing

organization. Defining an objective function is one of the keys to optimizing the performance of a parallel implementation.

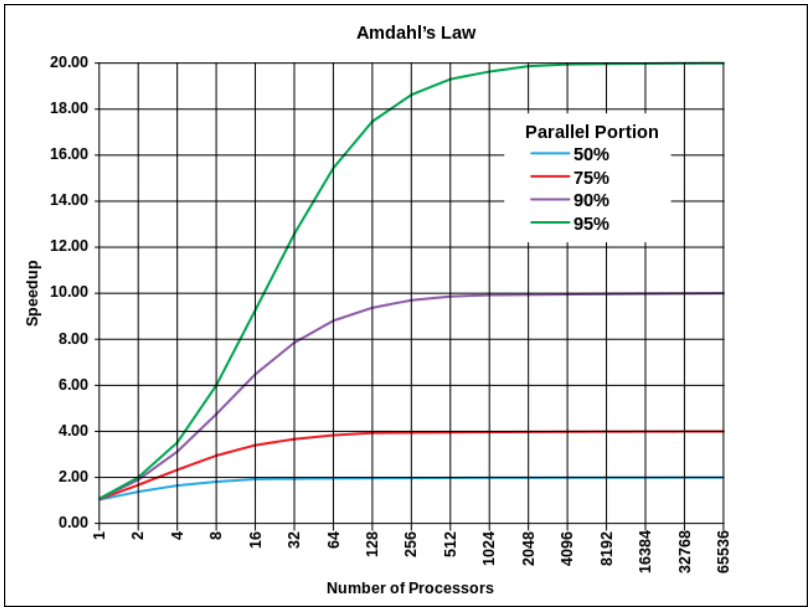


Figure 1.3. Parallel Execution SpeedUp Based on Amdahl's Law

1.3.3. Myhrvold's three fundamental laws of software

Nathan Myhrvold was the CTO of Microsoft when in 1997 ACM conference made a well-known presentation about the next 50 years of software where he introduced the "three laws of software" [11]. The three laws of software where articulated in very simple words but grasped the evolution of software development for the years to come:

First Law - Software is a gas: Software expands to fit the "container" it is in. The "container" in the software notion is the available hardware. Therefore, no matter how powerful the available hardware is, software will always become complex and demanding enough to require all available resources.

Second Law – Software grows until it becomes limited by Moore's law: The second law is complementary to the first law. Initial software growth is rapid, but then it is limited by the available hardware.

Third Law – Software growth makes Moore's law possible: Software growth is the reason that urges people to buy new hardware. It is the economic motivator of hardware development and the reason why hardware becomes more powerful for the same price and not cheaper.

These three laws describe the motivation behind hardware performance improvement. As the current trend is that more money is spend on application development than actual hardware development, it is clear that the most cost efficient strategy for hardware improvement will win the market.

The limits of Dennard Scaling [12] are slowly reached (transistor miniaturization is reaching its technology limits) and the "silver bullet" to increase performance comes from parallelism in the design. Parallelism can be implemented in all system levels: from coarse grain implementations in general processing elements, to fine grain implementations in task specific hardware accelerators. Exploring all available options to optimize for performance in different implementation platforms is the key to a successful design.

1.4. ASICs vs. FPGAs

ASICs are devices custom build for a particular design. The ASIC technology has advanced substantially during the last decades and now popular CPUs, such as the Intel Core-i7 use 22nm 3D technology transistor design [13].

Field Programmable Gate Arrays (FPGAs) are programmable semiconductor devices that are based around a matrix of Configurable Logic Blocks (CLBs) connected through programmable interconnects (Figure 1.4).

ASIC designs are by definition lower power than equivalent FPGA designs. They can achieve better performances and if a large number of units are needed, they have smaller cost per unit. FPGAs are more flexible, design can be changed late in the development process, they are reusable, and they have smaller time to market, cheaper tools and low cost per unit if small number of units is required. Each technology has advantages and disadvantages that are defined by the application and its specifications.

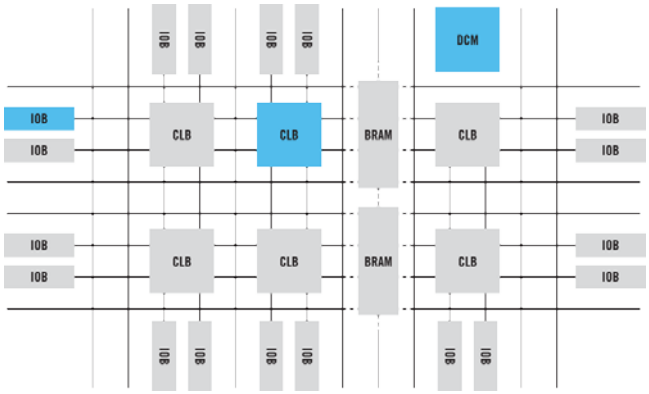


Figure 1.4. FPGA Block Structure (©Xilinx Inc.)

FPGA technology, however, is advancing rapidly. Xilinx and Altera are already producing high intensity and ultra high performance FPGA devices, closing even more the gap to ASIC technology. Xilinx's Ultrascale devices use TSMC 16 nm FinFET technology, doubling the capacity of the current largest Virtex-7 device. Altera is producing Stratix 10 series using Intel's 14 nm Tri-gate process [15]. Both vendors offer a complete heterogeneous solution: Xilinx offers an Utlrascale MPSoC Zynq solution [14] and Altera couples with ARM-Cortex quad core processors. The future holds even bigger challenges as FPGA vendors plan to combine FPGAs and ASICs on a single die, producing the so called "multi-package technology" ([16]-[18]). The demands of massive data centers, the so called "big-data" problem are now the market and technology drives: fast networking requirements and ultra fast processing of substantial amounts of data call for hardware accelerations that ASIC technology alone can no longer provide. Heterogeneous MPSoCs are the solution big IT companies look for and hardware design needs to be adaptable, generic and portable to cope with such demands.

1.5. Thesis Contribution

This thesis comes to explore the process used for homogeneous and heterogeneous MPSoC optimization. The objective is the final implementation of a highly optimized MPSoC architecture for a given application field.

Applications in signal, image, and video processing require large computing power and have real-time performance requirements. The hard specifications for such applications call for embedded implementations as opposed to general-purpose computing elements. ASIC design has evolved significantly over the past decades but the best solution to find more computational power is to switch to multiprocessing system on chip design.

One of the challenging issues in design of embedded multiprocessors is managing communication and synchronization overhead between the heterogeneous processing elements. Optimized task assignment on the various processing elements is critical and requires significant effort. More and more high performance reconfigurable devices are available in the market and gain popularity in the industry day by day. The FPGA market is booming with new generation devices and ultrascale technology. Adding this reconfigurability characteristic of the devices in the parameters that need to be defined for the development of an MPSoC system, it is understood that the optimization problem becomes a very complicated one.

Tools and models have been developed for various characteristics of the application specific MPSoCs. Even though design space exploration of parallel systems and the related automated tools have existed for some time now, the complexity of the parameters is increasing continuously. New parameters are added in the required system exploration such as memory requirements and energy consumption. The importance of such approach is evident by the inclusion of design exploration tools in the HiPEAC Roadmap 2015 [19] that targets the vision of embedded systems research until 2020.

In this thesis the MPSoC optimization problem is tackled from two different points of view: a) formulation of a theoretical model that can be used to optimize each system according to the application's specification and platform's characteristics and b) direct approach through finding the optimum architecture for two characteristic and high performance applications with hard real-time requirements.

The first step in this research is the extensive design space exploration of MPSoC on reconfigurable devices (FPGAs). MicroBlaze soft processors are used as processing elements because they offer a large number of reconfigurable parameters (local/external memory, caches, various bus interfaces etc.). Data and task level parallelism is extensively explored as well as the correlation between increase of performance and hardware resource usage for different architectural approaches. As the main focus is signal and image processing, JPEG was chosen as a typical algorithm for this application field. A new parameter called *Hardware Efficiency* is introduced to associate area increase with performance. This new parameter quantifies how "efficiently" hardware resources are used in every implementation.

The hands on experience gained lead to the development of an Integer Linear Programming (ILP) model for automated design space exploration of hybrid MPSoC systems. Hybrid and heterogeneous MPSoC systems use different types of processing elements: general-purpose processors, hardware accelerators, DSPs etc. The ILP model incorporates the parameters of internal and external memory resources, as well as the option of resource sharing. Objective functions for area, performance and memory usage can be used to optimize the system according to the application and the device needs. The model is extended to include power and temperature estimations. The formulation is appropriately adapted for temperature estimation for each processing unit of a hybrid FPGA MPSoC as well as estimation for the mean device temperature at regular time intervals. The inclusion of the new variables does not impose significant overhead to the previous formulation. The new model provides the designers with a tool for optimization based on a variety of parameters early in the system development time.

A direct approach in MPSoC optimization was used for two characteristic high performance applications as working examples:

The first one is a Machine Vision implementation for real-time flow detection on microfluidic Lab-On-Chips (LoCs). The implementation will be integrated in a movable Point-of-Care system, therefore special provisions had to be made so that the system is robust against changing lighting conditions and small LoC

displacements. The system uses a chip frame detection step to identify the device's bounding box and calculate all flows by reference to the bounding box's upper left corner. The architecture uses extensive parallelism to achieve the required performance to follow a 60 fps 1 Mpixel camera. As machine vision implementations are very commonly used in industrial and other applications, this implementation is a very good example of a high performance system for this particular application field of multimedia and image processing.

For the development of the system, the degree of parallelism required was explored in correlation with the application requirements. In addition, a bit accurate simulation of the edge detection module was developed to be used for exploration on the designs characteristics to approach the given specifications as well as system verification. High performance generic modules were designed for center of mass and median calculation, and an alarm point detection module. All these modules have the parallelism required by the system and are flexible enough to be adapted for different image processing implementations. The performance of the system is compared with previously implemented similar algorithms in various platforms to prove the performance gains of the implementation. The machine vision system is significantly faster for the same or bigger video frame resolutions.

A working prototype of the machine vision system was produced as a proof of concept.

The second application is a real-time 2D pixel clustering implementation. This application is a very characteristic image processing algorithm used for early stage date reduction and localization purposes. The 2D pixel clustering implementation was originally developed for the Fast TracKer processor of the ATLAS detector at CERN, but the design is generic enough to be adaptable to various image processing applications. The ATLAS detector presents the challenge of hard real time requirements for data processing (40 MHz data input rate) and in addition the design must be flexible to be used not only by the previously installed Pixel Detector Layers but the newly installed Insertable B-Layer that has a maximum input data rate of 100 MHz. Previsions are taken so that the system can be used in future upgrades of the ATLAS detector after the next LHC shutdown (further

increase in data input rates). The proposed application is very good example for high-throughput data streaming applications.

Taking all these requirements into consideration a 2D pixel clustering system for FPGAs was designed. The implemented innovative algorithm takes full advantage of the 2D FPGA fabric to avoid looping of data as much as possible for clustering identification using a moving window approach. The system is highly modular and the degree of parallelism can change by means of a single constant change in the design libraries. In High Energy Physics applications, unlike most common multimedia applications, data cannot be dropped in case a data throughput bottleneck appears in the system. Therefore, extensive design space exploration of the architecture was executed to not only achieve the required performance but to avoid all possibilities of backpressure applied by the pixel clustering module to previous computing elements in the ATLAS Trigger and Data Acquisition chain. The resulting implementation is innovative and fully parametric and achieves much better performance than equivalent approaches in the relevant literature. Comparison with previously implemented clustering algorithms proves the performance improvement gained by the newly proposed implementation. 64 times more resources would be required from older implementations to achieve equivalent performance.

1.6. Thesis Organization

This thesis is organized in three chapters. Each chapter presents work developed and completed independently but under the scope of multiprocessing systems development for implementation of applications.

The relevant references used in this thesis can be found after the conclusions chapter (Chapter 5).

In Chapter 2 the Integer Linear Model formulation for heterogeneous MPSoC optimization is presented. The chapter begins with an introduction, the thesis contribution and presentation of the relevant literature. An introduction to Integer Linear Programming, the chosen JPEG algorithm and a brief description of the characteristics of MicroBlaze soft processor follow. Sections 2.6 to 2.8 describe the

design space exploration process of the MicroBlaze based reconfigurable MPSoCs. In 2.9 the Integer Linear Programming model formulation is introduced and in 2.10 the proposed extension to include memory occupation is explained. A task graph example is presented in 2.11. The power and temperature extension of the ILP model is described in 2.12 and an implementation on a similar task graph is presented in 2.13. The chapter finishes with the conclusions.

Chapter 3 contains the description of the work included in the Machine Vision developed for real-time flow detection on microfluidic LoCs. This chapter also begins with an introduction and thesis contribution. In 3.2 a small introduction in mcrofluidic technology and experiments is included and related work from the literature follows in 3.3. In 3.4 we have the system specifications and in the following section the proposed system is described. The exploration process is explained in 3.6 and more specifics on the edge detection preprocessing step in 3.7. The chip frame detection stage is briefly introduced in 3.8 and the flow detection stage in 3.9. 3.9.1 is a detailed description of the center of mass calculation module and 3.9.2 of the median calculation module. The alarm point detection module is described in 3.9.3. Section 3.10 presents the finalized system core and the testing results can be seen in 3.11. The chapter ends with the conclusions and future work.

Chapter 4 is the work on the 2D pixel clustering implementation for the ATLAS Fast TracKer. Again the chapter begins with an introduction and the thesis contribution and then the related work follows. Section 4.3 contains the necessary background information on the application: the ATLAS detector, the ATLAS pixel detector, the FTK system and the clustering problem itself are described to allow the reader to understand the properties of the problem. The extensive description of the 2D pixel clustering implementation begins in 4.4. In sections 4.4.1 - 4.4.3 the three fundamental modules, the hit decoder, grid clustering module and centroid calculation module, are described. Section 4.5 describes the necessary additions made to make the system parallel with a generic number of parallel engines. Two extra modules were added, the parallel data distributor module and the data merger module, described in 4.5.1 and 4.5.2 respectively. The process of functional verification and design space exploration is illustrated in 4.6 Sections 4.7 to 4.9 contain the testing results of the implementation and a comparison with a previous

clustering approach is reported in 4.10. Statistics that derive from the ATLAS experiment and the nature of the pixel detector are presented in 4.11, while the hardware testing process is illustrated in 4.12. The chapter ends with future developments and conclusions. Extra information on the design and the hardware testing process is available in Appendices A and B after the publications.

The last chapter (Chapter 5) of this thesis presents the conclusions from the presented work. It reviews all the derived results from this dissertation and presents targets for future research.

Chapter 2

Exploration and Modeling of MPSoC Performance

2.1. Introduction

In recent years embedded applications present a constant increase in complexity and computational intensity. These applications demand the development of systems that are powerful enough to cope with the applications' characteristics as well as efficient enough to do so with the minimum possible cost. The cost of such systems derives from multiple factors, such as development time and time to market and of course manufacturing costs.

In addition, modern multimedia, signal processing and communication applications rely on standards that change continuously and therefore the applications' algorithms need to be constantly updated. As a result the systems developed for such applications need to have inherent flexibility. The answer to such demands comes from FPGA multiprocessing systems, where the power from the multiprocessors can be combined with the reconfigurability of the device.

All the latest advancements in the FPGA technology have made the devices available larger and more powerful, with a variety of memory architectures offered on both the device itself and the device board as external components. The

combination of these elements has directed the designers to the study and implementation of FPGA based multiprocessing platforms. These architectures offer high computational power and extreme design flexibility by providing configurable characteristics such as the number of processors, memory size and type, interconnection structures and buses and more. However, the vast amount of configurable parameters in such systems pose a great challenge to the designers and increase the time and effort needed for the final implementation of the system, thus eliminating one of the FPGA's greatest advantages, the limited time needed from design to prototyping.

In addition, as FPGA logic densities have increased substantially and so has their performance, but this has also lead to the undesired effect of increased power dissipation, increased device temperatures and the generation of device hotspots during operation time. Such phenomena can cause penalties in performance by reducing operational speed and lead to a greater power consumption due to increased leakage currents. Great fluctuations in device temperature also reduce reliability and cause device aging. In order to tackle such phenomena designers direct their efforts into finding the optimum architecture mapping and application scheduling which reduces thermal implications.

The solution proposed is the formulation of different design models, mostly of integer linear programming (ILP) that identify the optimal hybrid MPSoC design for each application, taking into consideration the constraints set by the designer. These constraints can be based on performance, availability of resources, power consumption, operating temperature or a combination of the above with different weight factors. The target is the design optimization that suits the application needs as well as the implementation platform of choice.

2.1.1. Thesis Contribution

The importance of system optimization is known to every experienced embedded systems designer. However, the design space exploration that is necessary is a tedious and time consuming process. Significant research effort has been invested by various groups for the limitation of the design space exploration time, by trying to formulate the problem using ILP and developing automated tools

for the design, synthesis and implementation of these systems. However, insufficient attention has been paid on the impact of the different memory architectures on the designs. There are also efforts on developing power and temperature estimation models, but a combined model that a designer can fine tune to demand has not been presented yet.

The initial effort of the presented research was to gain experience by a hands on design space exploration of FPGA MPSoC architectures. A very popular image processing algorithm (JPEG) was used and the platform of choice was Xilinx FPGAs. In total 15 system architectures were studied with 20 different types of algorithm partitioning. Emphasis was laid on using different types of parallelism (data/task) and different types of memory configurations (external memory/local memory and combined).

The design space of multiprocessing FPGA platforms was studied, taking into account the number of processors, data-level and task-level parallelism, the different interconnection strategies and adding on top of these parameters the different memory architectures offered. The intention was to formulate the complete design space exploration problem, including the parameters of internal and external memory resources, as well as the possibility of resource sharing. Such a model is proposed in this thesis. It was developed using Integer Linear Programming (ILP). The proposed model:

- automatically generates the desired architecture for a given application task graph
- takes into consideration the instruction and data memory needed for each microprocessor when a task is mapped for execution
- supports objective functions for area, time and memory usage.

The same ILP model was then extended to include power formulation that is used for temperature estimation for each processing unit of a hybrid FPGA MPSoC (soft processors and hardware accelerators) as well as the mean device temperature at regular time intervals. The motivation was the incorporation of the power/thermal formulation into the existing ILP model that optimizes the FPGA system implementation area and the application's execution time in order to provide

a complete solution for design space exploration. With this work, power and temperature constraints are added to the model and thus present a system modeling formulation that produces an estimation of the most important system characteristics (area, time, memory usage, power, temperature) at a very early system development stage. The application mapping procedure takes into account a temperature threshold for all system components. Appropriate task scheduling is selected to decrease power dissipation concentration and therefore decrease the possibility of hotspot generation.

The necessary variables and constraints for the presented model can be produced through an appropriately designed parser and the model itself can be used through open source and commercial ILP solvers. The proposed design space exploration and model formulation from this research where published in [21] and [22].

2.2. Related Work

The problem of the design space exploration of MPSoC systems has been addressed by a number of research groups. The ESPAM toolchain [23] generates MPSoC platforms from high level description languages and maps them on FPGAs. However, it uses a one to one mapping algorithm which makes resource sharing impossible. Resource sharing through the use of e.g. external memories results in more efficient architecture design and is taken into account in the presented design space exploration.

The Eclipse work [24] defines a scalable architecture template for designing stream-oriented MPSoCs. However, their template is limited only to task-level parallelism exploitation.

In [25] a design space exploration for the JPEG algorithm is offered for implementation on an OSCAR type multiprocessor architecture. Even though data-level parallelism is partly explored in this paper, there are no additional memory interfaces offered or direct comparison of implementations with or without resource sharing.

The work in [26] is the one more closely related to the design space exploration concept in this thesis. It presents an automated system for design space exploration for a JPEG implementation on a Xilinx FPGA. However, their exploration is limited to pipelined implementations of the algorithm and data-level parallelism is not yet explored. In addition, the impact of shared or non-shared memory resources and/or use of on board memory versus external memory are not taken into account. The work in this thesis comes to fill this void and offer a more complete design space exploration for a better visualization of the variety of MPSoC implementations offered on an FPGA.

Different research groups have proposed ILP models that lay emphasis on various characteristics of the MPSoC architectures. In [27] Cho et al. present an ILP model that is specific for shared communication primitives. In [28] Yang et al. proposes a model that takes advantage of task, data level and temporal parallelism. However, the number of processors is set as an input and the exploration is used solely to define the scheduling and the communication primitives. The works presented in [29] and [30] are developed for ASIP multiprocessing systems, with their design flexibility limited to the ability to configure the number of the processors as well as the configurations of the ASIP processors to the specific tasks. Kadayif et al. [31] produces an energy aware ILP model. This model uses performance and energy constraints, however, it doesn't explore different bus and memory architectures but merely the number of processors. The work of [32] suggests a model for pipelined applications. It is limited though to a specific type of communication bus and a shared memory architecture.

The ILP model proposed in this thesis follows the structure of the one introduced in [33]. In that, Theodoridis et al. proposed an ILP model for multiprocessing systems with ASICs, FPGAs and microprocessors. The data memory of the system is shared and only the configuration memory needed for the FPGA reconfiguration is taken into account as a constraint. The model proposed in this thesis is configured for hybrid FPGA MPSoC implementations, which include both soft processors (such as MicroBlaze) and hardware accelerators and it also includes three types of buses, shared buses such as the PLB bus, private buses such as the LMB bus and FIFOs. A similar approach based on the same model was also

presented by Wu et. al. in [34]. However, memory usage, for data and instructions, was not taken into consideration.

Different approaches have also been proposed for power and temperature management in single and multiprocessor systems. In [40] Kumar et. al. propose a thermal-aware scheduling algorithm for a single processor system. The work in [41] and [42] present power and temperature management through scheduling for MPSoC. In [41] emphasis is layed on dependability issues because of temperature fluctuations, while [42] targets Dynamic Voltage and Frequency Scaling. However, they are both limited to homogenous MPSoC. Bhoj and Bhatia [43] demonstrate a method for a more uniform temperature distribution on an FPGA, but the application is restricted to hardware FPGA implementations (no soft-processors are taken into consideration). [44], [45] and [46] present three different ILP systems for temperature management. The ILP of [44] targets behavioral synthesis for ASIC design. Coskun et. al. [45] rely on OS-level scheduling for MPSoCs. Mohanty et. al. [46] proposes a system which exploits datapath scheduling for a single processor.

The model proposed in this thesis allows a design space exploration with a variety of objective functions that express the main characteristics of a system, such as performance and usage of hardware resources, and additional characteristics, such as usage of memory resources, power and temperature. To the best of our knowledge a model that combines all the above characteristics has never been proposed before.

2.3. Introduction to Integer Linear Programming

The formulation used for the description of the proposed model is Integer Linear Programming. A very good introduction to Integer Linear Programming is provided in the “Applied Integer Programming” book by Chen, Batson and Dang [47]. Linear Programming (LP) is a class of constraint optimization problems (COP) in which we seek to find a set of values for continuous variables $(x_1, x_2, \ldots, x_n)$ that maximizes or minimizes an objective function z, while satisfying a set of linear constraints with the following format:

Maximize: $z = \sum_j c_j x_j$

Subject to: $\sum_j a_{ij} x_j \le b_i \qquad (i = 1, 2, \ldots, m)$

$x_{ij} \ge 0 \qquad (j = 1, 2, \ldots, n)$

Equation 2.1. Linear Programming Constraint Format

An Integer Linear Programming (ILP) problem is an LP problem where at least one of the variables is restricted to integer values. An ILP problem in which integer variables are restricted to be 0 and 1 is called a binary integer programming problem.

Combinatorial Optimization Problems are discrete problems where a solution in a finite set of solutions is sought and this solution maximizes or minimizes and objective function. COPs are very tightly related to ILPs, and most COPs can by formulated as ILPs and/or binary IPs. A very well known example of an ILP formulated COP is the *travelling salesman problem* [48]. ILPs are solved using a variety of algorithms, the most widely implemented one being the Simplex algorithm. The Simplex algorithm is used by most commercial and open source ILP solvers such as LP-solve [39], IBM CPLEX and others. Hardware implementations of the Simplex algorithm has also been proposed by Bayliss et al. [49] and Mittal et al. [50] and have also been developed by AUTH-eLAB in the form of a post-graduate thesis [51] (co-supervised under this PhD Thesis development).

The design space exploration model for Hybrid MPSoCs can be easily formulated as an ILP problem as it is clearly an optimization problem where the objective function needs to be defined according to each application's and platform's needs.

2.4. Algorithm Selection

In order to explore the capabilities of different multiprocessor architectures implemented onto an FPGA, the JPEG decoder was chosen as an algorithm that is a widely used streaming multimedia application and presents different types of

parallelism. The source used is the Powerstone Benchmark JPEG decoder, a popular benchmark for characterization of different processing systems.

The Powerstone JPEG decoder consists of four stages: the 1-D DC prediction stage, the Entropy Decoder, the DeQuantization and the two dimensional IDCT stage. Each stage processes an 8 x 8 pixel block at execution time. There is no data dependence among 8 x 8 blocks in JPEG decoding except in the 1-D DC prediction stage. This allows the exploration of different partitioning approaches of the algorithm, taking advantage of both task-level and data-level parallelism. The software itself is clearly written and requiring less effort to partition.

2.5. MicroBlaze Soft Processor

The design space exploration requires a flexible and reconfigurable implementation platform and FPGAs were the solution. To implement several different MPSoC architectural scenarios a soft processor is needed that can be implemented multiple times on an FPGA device and has several different parametric characteristics. Since our platform is a Xilinx FPGA, MicroBlaze [36], the soft processor developed by Xilinx was the best choice.

The MicroBlaze embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx FPGAs. The MicroBlaze processor has some fixed characteristics, but offers a great degree of flexibility, including some configurable characteristics in its architecture, a variety of memory structures (use of cache and size, access to external memory via a specific interface etc.) and the ability to communicate with other hard processors (PowerPC), soft processors (MicroBlaze) and customized co-processors with a variety of buses (AXI, PLB, FSL etc.) In Figure 2.1 the MicroBlaze core block diagram is presented. The shadowed blocks are the ones that can be configured or opted out from the implementation.

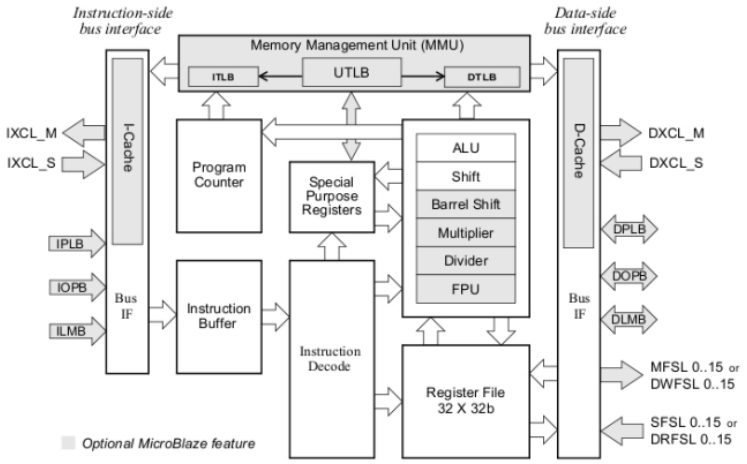


Figure 2.1. MicroBlaze Core Block Diagram (© Xilinx Inc.)

The MicroBlaze processor is offered as an IP core integrated in the Xilinx Tools and can be easily configured with the use of wizard interfaces. It can be optimized for area or performance. The number of MicroBlaze processors that can be implemented in a device is only limited by the available resources.

2.6. Architecture Styles

The starting point of all design space explorations is the definition of a comparison reference. The comparison reference is set by profiling the execution time of the application on a single MicroBlaze processor system. The design space exploration was expanded to systems with up to four MicroBlaze processors. Different architectures are implemented to take advantage of data-level parallelism or/and task-level parallelism. The bottleneck in the design space exploration of such systems lies in the limited memory throughput and the unavailability of finer task-level parallelism in the algorithm itself. Through the exploration process it was demonstrated by the performance results that there is no expected gain or possibility of exploitation of data/task-level parallelism by expanding our systems with more than four processors. The limit was set to four processors because, as the previous work presented in [23] it was found that the existing levels of parallelism within the existing code could not profit for further addition of resources.

The MicroBlaze processors are interconnected through Fast Simplex Links (FSL) [37], a Xilinx developed inter-processor bus that is implemented as a FIFO and can also serve as a data buffer. When data-level parallelism is used, the FSL FIFO depth is set to 16 and it is only used for basic synchronization between the processors (a processor is defined as a "master" and controls the operation of the system). When task-level parallelism is used, the FSL FIFO depth is either set to 16 when only the pointer for the pixel block is sent through the bus (use of external memory), or set to 64 (8 x 8) when the whole pixel block data is propagated through the FSL (use of local memory per processor). It must be noted that the size of 16 words was imposed as a minimum by Xilinx Tools since a FIFO cannot be implemented by the Core Generator with less than 16 words size. In reality 2 words depth would be sufficient.

Apart from the different number of processors the design space exploration includes two different memory architectures with three different memory access approaches. The first memory architecture uses external memory (DDR2, offered by the Xilinx development board used, the xupv5-lx110t) via the component MPMC. In this case each MicroBlaze is implemented with 2kb of instruction cache and 4kb of data cache and an 8kb local BRAM. The whole address space of the external memory is cacheable for each processor. In this particular architecture two different memory access approaches are explored: In the first both the data and the instructions for each processor are stored in the DDR2 and the local BRAM is used only for the processor heap and stack. In this case, when task-level parallelism is used, only the pointer of the processed pixel block is propagated to the next processor, thus reducing the delay due to the FSL bus. In the second approach each 8 x 8 pixel block is downloaded to the local BRAM prior to processing. In this case the whole pixel block data is propagated through the FSL when task-level parallelism is used. For this memory architecture data-level parallelism is also examined, as well as a combination of both.

For the second memory architecture each MicroBlaze is implemented with only one local BRAM of 128kb, and therefore only internal FPGA memory is used. Since there is no shared memory or common address space for the processors only task-level parallelism is examined.

In total 15 system architectures were studied with 20 different types of algorithm partitioning, by exploiting data and task level parallelism.

2.7. Case Study

The different processing systems were developed using the Xilinx EDK tool. The algorithm partitioning and implementation was done using the Xilinx SDK tool. The performance measurements were made by integrating a timer for each processing element of the system. The timers used were XPS timer that is an IP core provided by Xilinx. Each XPS timer has two independent timers with configurable counter widths and interrupts, event generation, and event capture capabilities. The XPS timer counts cycles of the system clock. Timer values were called before and after each measured task and the difference was used as execution cycles count.

The first two implementations studied were a single MicroBlaze system with external memory and a single MicroBlaze System with internal memory for the purpose of acquiring information for the execution time of each stage. The results are presented in TABLE 2.1. As we see in TABLE 2.1 the results presented for both architectures are similar and the most computationally intensive stage is, as expected, the 2D-IDCT. The performance similarity is expected as the system with external memory also uses cache for the processor, therefore the impact of the delay in data propagation between stages was insignificant.

TABLE 2.1. Execution Time of JPEG Stages on the Reference System

Single MB	**JPEG Decoding Stage**			
	DC prediction	***Entropy Decoding***	***DeQuantization***	***2D-IDCT***
DDR2	3,81%	28,83%	10,81%	48,69%
BRAM	4,11%	29,92%	8,50%	51,00%

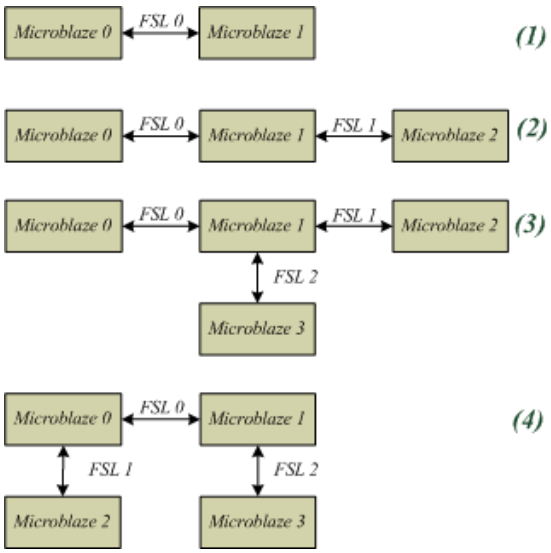


Figure 2.2. Explored MPSoC Architectures

The different types of multiprocessor interconnections used in the design space exploration are demonstrated in Figure 2.2. As said in the previous section the first memory architecture was used to explore both task-level and data-level parallelism. In the case were only the external memory (DDR2) is used, the FSL buses have a FIFO depth of 16. When data-level parallelism is used (architectures (1), (2) and (3) in Figure 2.2) one MicroBlaze serves as a master and the others serve as slaves. In the master MicroBlaze the execution of the 1-D DC prediction algorithm for the whole image is executed, since the data dependence does not allow exploitation of data-level parallelism in this stage. However, the measurements taken from the reference system (TABLE 2.1) clearly show that this does not affect the performance significantly (less than 4.2 % of total execution time).After this stage is finished the master MicroBlaze divides the image data in equal parts and sends the corresponding pointers to the slave processors. When task-level parallelism is used (architectures (1), (2) and (4)) the algorithm is partitioned in the following manner: In the dual MicroBlaze system, the 2D-IDCT is executed on a separate processor. In system architecture (2) two different algorithm partitioning approaches are used: in implementation (a) the DeQuantization stage is executed in MicroBlaze 1 and the 2D-IDCT in MicroBlaze 2 and in implementation (b) the DeQuantization stage is executed in MicroBlaze 0 and then each pass of the 1D-IDCT is executed separately in MicroBlaze 1 and then MicroBlaze 2. In system (4) the DC

prediction and the Entropy Decoding is executed in MicroBlaze 2, the DeQuantization in MicroBlaze 1 and each pass of the 1D-IDCT in MicroBlaze 1 and MicroBlaze 3. For architecture (4) another option is offered: a combination of both data and task-level parallelism. In this option MicroBlaze 0 serves as a master and MicroBlaze 1 serves as a slave. The DC stage for the whole image is executed in MicroBlaze 0 and then data-level parallelism is exploited. For half the image data Entropy Decoding and DeQuantization is executed in MicroBlaze 0 and through task-level parallelism the 2D-IDCT stage is executed in MicroBlaze 2. For the other half of the image data the execution follows the same pattern from MicroBlaze 1 through to MicroBlaze 3.

For this architecture the local BRAM of each MicroBlaze is used for pipelining by downloading each 8 x 8 pixel block to the local BRAM prior to the execution of each stage. In this manner a small additional delay is imposed to the execution of the program by downloading the corresponding data from the external memory to the local BRAM, but the delay is compensated by avoiding the delay imposed by the simultaneous memory requests to the DDR2 of the different processors (MPMC component uses a round robin algorithm to define priority of memory access for the processing elements). All the types of data and task-level parallelism described in the previous paragraph are also implemented in this case. In this type of design space exploration when task-level parallelism is used the FSL FIFO depth is set to 64 since each pixel block data is propagated through this bus to the next processor. When data is transferred through the FSL bus one cycle is needed for the FIFO write and two cycles for the FIFO read, but the processor does not need to access the external memory for the pixel block data since they are stored in the local BRAM after FIFO read is completed.

For the second memory architecture each MicroBlaze uses an independent BRAM of 128kb and therefore no data-level parallelism can be implemented. Task-level parallelism is used instead. Therefore only the MicroBlaze architectures (1), (2) and (4) in Figure 2.2 with FSL links of FIFO depth 64 are implemented. The pixel block data is transferred through the FSL link since there is no common address space. The partitioning of the algorithm used is the same as for the task-level parallelism implementations of the previous architecture.

2.8. Experimental Results

In the design space study of multiprocessing systems on an FPGA a new parameter is introduced that is the use of different memory architectures. The set reference system for performance comparisons is one with a single MicroBlaze with the use of external memory. The equations Equation 2.2(1) and Equation 2.2(2) are used for calculating speed up and efficiency as in the work of Wu *et al*. in [26]. A new parameter is also introduced called *HW_efficiency* (hardware efficiency) to associate the area increase of the design with the speed up. The results associated with these three parameters are presented in Figure 2.3.

$$SpeedUp = \frac{execution\_time\_of\_multiprocessor}{execution\_time\_of\_single\_processor} \quad (1)$$

$$Efficiency = \frac{SpeedUp}{Number\_of\_Cores} \quad (2)$$

$$HW\_efficiency = \frac{SpeedUp}{Area\_Increase} \quad (3)$$

Equation 2.2. SpeedUp Formulation

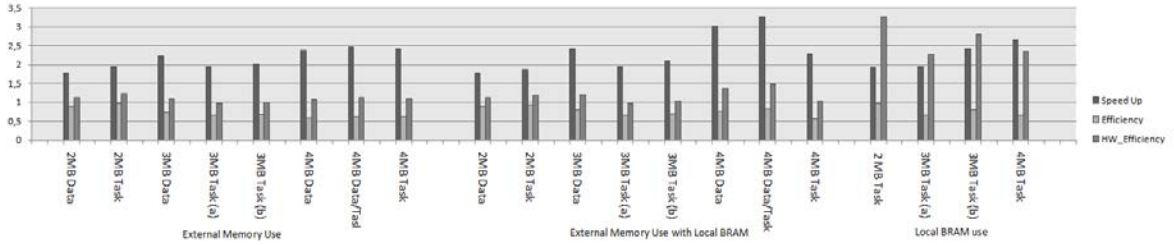


Figure 2.3. Design Space Exploration Results

It is clear that the greatest speed up is achieved by the 4 MicroBlaze circuit with external memory and use of the local BRAM. The highest speed up achieved is 3.27 by the system that uses a combination of data and task-level parallelism, demonstrating that the exploitation of data-level parallelism should not be omitted even in streaming applications such as JPEG. On the other hand, the most efficient implementation is the dual MicroBlaze system with task-level parallelism for all

three types of system implementations. When the new parameter hardware efficiency is taken into account, the systems that achieve the highest value only use internal BRAMs. This result was expected since in these systems there is actually a decrease in the area in comparison to the reference system. This is due to the fact that the instantiation of the MPMC, the Xilinx memory controller used for the communication with the external DDR2 memory, occupies a significant area of the total architecture of each system.

In the systems that use only external memory there is a limitation in the speed up that can be achieved by data-level parallelism because after the addition of the third processor the bottleneck stems from the simultaneous requests for memory access by the processors. This bottleneck is overcome by the use of both external and internal memories. In the systems where the 8 x 8 pixel blocks are downloaded to the local BRAM, data-level parallelism is better exploited since the simultaneous memory requests to the DDR2 by different processors are reduced. The additional one clock delay in the FSL FIFO read does not impose an overhead as big as the one from the data cache misses in the systems without using the local BRAM. It can also be pointed that by comparing the two different task-level parallelism implementations (a) and (b) using three MicroBlaze processors in both architectures, the greatest speed up is achieved by implementation (b) where the 2D-IDCT is partitioned in two 1D-IDCT stages, executed by separate processors. The reason is that the execution times of the processors are better balanced.

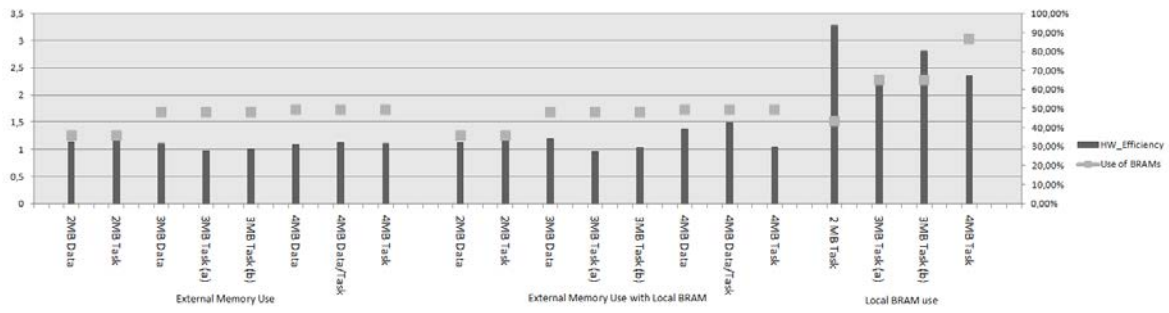


Figure 2.4. Design Space Use of BRAM Results

Figure 2.4 presents a diagram that demonstrates the hardware efficiency in correlation with the use of BRAMs on the board. It is obvious that the designs that use only local BRAM demonstrate great hardware efficiency (less area while achieving similar speed up) but this is achieved at the expense of the total BRAM block usage, except from the system with two MicroBlaze processors. As a general result the architectural approach with the best exploitation of the hardware, which means the higher hardware efficiency is the dual MicroBlaze system with local BRAM use. However, the applicability of such systems is questionable for high-throughput multimedia applications since the great demands for memory space cast the sole use of internal memories prohibitive.

2.9. Introducing a Task Assigning Model with a Memory Usage Parameter

2.9.1. The Proposed ILP Model

Little attention has been paid by previous FPGA MPSoC ILP approaches to the indispensable amount of memory the system's microprocessors need for instructions and data as well as hardware accelerator memories. Memory usage is especially important since modern applications have a need for large memory spaces. The important role of memory resources in the performance exploration and optimization of MPSoC was also demonstrated by the Case Study in 2.7. The goal of this model is to produce a real life model that tackles this weakness. Using the same approach as in 2.5, Xilinx FPGAs were the chosen platform and the MicroBlaze [35] as the system's microprocessing unit. The characteristics of the MPSoC that need specification by the ILP model were identified. However, these characteristics can be easily identified for various platforms and the same approach can be used for any FPGA and microprocessor type.

The application is imported in the model as a directed acyclic graph, a task graph, where every node is a task $TASK=\{t_1,t_2,....,t_i\}$ and every edge is a data transfer denoted as a queue $Q=\{q_1,q_2,....,q_l\}$. Each task's granularity can vary from a single operation or loop to a set of kernels or functions and can comprise of data reading, task executing and data writing procedures which are annotated in the following

form $t_i=\{r_0,r_1,...,r_n,e_i,w_0,w_1,...,w_m\}$. The system model comprises of a set of processing elements $PE=\{\mu B \cup HW\}$, which includes soft processors $\mu B=\{p_1,p_2,.....,p_k\}$ and hardware accelerators $HW=\{hw_1,hw_2,.....,hw_l\}$. A maximum number of available processing elements must be defined in the model (MaxPE). A set of communication buses $COM=\{U \cup L \cup F\}$ is also set, where $U=\{u_1,u_2,.....,u_m\}$ are shared buses (PLB buses), $L=\{l_1,l_2....,l_n\}$ are private buses (LMB buses) and $F=\{f_1,f_2,.....,f_j\}$ are FIFOs (FSL buses). These buses can be turned "on" or "off" as an option in the model depending on the explored system architectures. For example if a multi-MicroBlaze system is explored the following assumptions must be made:

- At least one shared bus must exist in every system implementation and every soft processor must have at least one private bus.
- When FIFOs (FSLs) are used for interconnection they are assumed to be "uni-directional" connections and the maximum possible number of existing FIFOs in every system is when the system is a Completely Connected Network (CCN). The number of connections for a CCN is $N(N-1)$ where N is the number of soft processing elements.

Each processing element and communication bus is assigned a predefined area constant $a\mu b_k,ahw_l,au_m,al_n,af_j$, the value of which is the number of FPGA slices required by each component. The total system area is defined as *A*, and a constant *A_MAX* is set as a constraint for the maximum system implementation area. For each task or communication procedure an integer variable ts_i is defined for the start time of the procedure and another integer variable tf_i for the finish time. The variable tf represents the completion time of the whole application on the architecture. A T_MAX time constant is set, which is the maximum allowed system operation time. The time a task needs to complete its operation on a processing element is a preset value set in the model as a $tZ_{i,k}$ constant, where i is the task and k the assigned processing element. All the time variables and constants are measured in time units. The basic decision variables and constraints of the model are presented in TABLE 2.2 and TABLE 2.3.

TABLE 2.2. ILP Model Decision Variables

Decision Variables
$X_{i,k}^{r,e,w}, Y_{i,k}^{r,e,w} = \begin{cases} 1 & r_i, e_i, w_i \text{ is mapped on processor k or} \\ & \text{hardware accelerator k} \\ 0 & \text{otherwise} \end{cases}$
$U_{i,l}^{r,w}, L_{i,l}^{r,w}, F_{i,l}^{r,w} = \begin{cases} 1 & r_i, w_i \text{ is mapped on } u_l, l_l, f_l \\ 0 & \text{otherwise} \end{cases}$
$CSX_k, CSY_k, CSU_l, CSL_l, CSF_l = \begin{cases} 1 & p_k, hw_k, u_l, l_l, f_l \text{ is used} \\ 0 & \text{otherwise} \end{cases}$

TABLE 2.3. ILP Model Basic Constraints

Constraints	Description
$\sum_k X_{i,k} + \sum_k Y_{i,k} = 1$	General Constraints
$\forall p_k \in PE, \forall r_i, e_i, w_i \in t_j, \forall t_j \in TASK$ $X_{i,k}^{r,e,w} \le CSX_k$ $\forall p_k \in PE, \forall r_i, e_i, w_i \in t_j, \forall t_j \in TASK$ $CSX_k \le \sum X_{i,k}^{r} + \sum X_{i,k}^{e} + \sum X_{i,k}^{w}$ $\forall \mu B, HW \in PE, \forall U, L, F \in COM$ $A = \sum CSX_k \times a\mu b_k + \sum CSY_l \times ahw_l +$ $+ \sum CSU_m \times au_m + \sum CSL_n \times al_n + \sum CSF_j \times af_j$ $A \le A\_MAX$	Area Constraints
$\forall p_k \in PE, \forall r_i, e_i, w_i \in t_j, \forall t_j \in TASK$ $tf_i \ge ts_i + tZ_{i,k}$ $tf \ge tf_i$ and $tf \le T\_MAX$	Time Constraints

2.10. Memory Usage

The ILP model is extended to incorporate memory usage and BRAM occupation by making the following assumptions for the architecture:

- Each microprocessor corresponds to one microprocessor memory which is used for instruction and data storage.
- Each microprocessor communicates with its dedicated memory though a private LMB.
- For each task mapped on a microprocessor the instructions must be preloaded to the instruction memory space.
- The data memory space of each microprocessor can be reused by each task mapped on this microprocessor.
- All microprocessor memories are instantiated on BRAMs.
- The option to instantiate hardware FIFOs on BRAMs is offered as well.

An integer variable MB is defined for the total BRAM occupation for the system architecture. A value of MB_MAX is set as the maximum number of available BRAMs that can be used by the developed system. For each processing unit a variable M_k is created which constitutes the memory space needed by the specific processor. A second variable MB_k is also defined for the number of BRAMs needed for the instantiation of this memory. For computational purposes a variable MS_k is defined which is the actual memory size for each processing unit. For each task mapped on a particular processing element (microprocessor or hardware accelerator) there is a preset value $mPD_{i,k}^{e}$ or $mHWD_{i,k}^{e}$ which constitutes the amount of data memory space needed for the execution. For the microprocessing units there is also a predefined value $mPI_{i,k}^{e}$ which is the instruction memory space needed for the application execution on the processor. Also the possibility of the hardware FIFOs F_l to be implemented on BRAMs is taken into consideration. In this case a preset value $mF_{i,l}^{r,w}$ is defined for each FSL instantiated as BRAMs.

The memory constraints are based on the following principle: ***the amount of memory space each microprocessor needs is the total number of instruction memory space needed by all the tasks mapped on this specific processor and the data memory space size of the task that requires the greatest data memory space***. The difference between instruction memory space and data memory space is

that the instructions for each task have to be preloaded on each processor, whilst data memory space can be reused.

The total memory space required by each processing unit is defined by constraint Equation 2.3(1). The supported memory sizes that can be used by a MicroBlaze processor are a power of 2 [38]. Therefore the actual memory size of each MicroBlaze is the smallest power of 2 number that is greater than M_k Equation 2.3(2).

$$\forall e_i \in TASK, \forall p_k, hw_k \in PE$$

$$M_k = \sum_{p_k \in P} X_{i,k} \times mPI^e_{i,k} + \\ + \max\left(X_{i,k} \times mPD^e_{i,k}\right) + \sum_{hw_k \in P} Y_{i,k} \times mHWD^e_{i,k} \quad (1)$$

$$\forall e_i \in TASK, \forall p_k \in PE$$

$$MS_k \geq M_k, MS_k \in \left\{2^n\right\}, n = \text{integer}, n \geq 0 \quad (2)$$

Equation 2.3. Memory Constraints

In addition there is a set rate between memory size and BRAM usage for each specific FPGA family. In Spartan-3 and Spartan-6 FPGAs 8k memories of 32bit width need 4 BRAM primitives (minimum requirement $MB_{\min}$), whilst in Virtex-5 and Virtex-6 FPGAs 4k memories of 32bit width need only one BRAM primitive. This rate is imported to the model as a constant C_{MB} Equation 2.4(3). The total number of BRAM primitives required by the implementation is defined as MB and it is constrained to be smaller than MB_MAX Equation 2.4(4).

$$\forall e_i \in TASK, \forall p_k \in PE$$

$$MB_k = C_{MB} \cdot MS_k \quad (3)$$

$$MB_k \geq MB_{\min}$$

$$\forall p_k, hw_k \in PE, \forall f_l \in COM$$

$$MB = \sum MB_k + \sum mF_{i,l} \quad (4)$$

$$MB \leq MB\_MAX$$

Equation 2.4. Memory Constraints

With the above addition our proposed ILP model offers the flexibility to be solved based on a variety of objective functions. It can be solved with a set area constraint A, set memory constraint MB and minimize the time tf (optimize performance),or set time constraint tf, set memory constraint MB and minimize area A (optimize area), or set time constraint tf, set area constraint A and minimize memory usage MB (optimize memory usage). These options offer extreme flexibility to the designer to explore and choose the optimum architecture for an application implementation.

2.11. Trial System (Memory Model)

In order to demonstrate the efficiency of the proposed ILP model, a trial system is presented. A random task graph (Figure 2.5) of adequate complexity is used as input for the model. It is assumed that there are 4 microprocessors available and 2 hardware accelerators, one assigned to task 2 and one to task 5. The time delays and memory requirements of each task are presented in TABLE 2.4. The area values for each one of the resource elements, MicroBlaze, dedicated HW task 2, dedicated HW task 5, PLB, LMB and FSL are set to 1500, 2000, 2200, 450, 20, 200 slices respectively. The device chosen is the Spartan-6 and therefore C_{MB} is set to 0.5. The experiment is performed on an Intel i3 computer with 4Gb RAM. The lp_solve linear programming solver [39] is used.

The initial ILP model (without memory variables and constraints) produces 650 variables and 1057 constraints. The ILP model with memory usage produces 670 variables and 1078 constraints. A series of ILP model implementations will be presented for different objective functions and constraints to demonstrate our model's flexibility.

The first implementation test is for optimizing performance. The objective function is minimizing tf. The constraints for area and memory usage are set to the maximum for the assumed processing units. The solver required 730sec to produce a solution. The minimum processing time achieved is 66 time units with an area of 10410 slices. The system produced includes 3 MicroBlaze processors, 2 hardware

accelerators, 3 LMB buses, 6 FSL buses and 1 PLB bus. It requires 96 BRAMs (192kB memory space).

The second implementation test is for optimizing memory usage. The objective function is minimizing MB. The constraint for time is set to 120 time units and the constraint for the area is set to the maximum available slices. The solver required 408 sec to produce a solution. The minimum BRAM usage achieved is 48 (96 kB memory space). This was achieved for execution time $tf = 115$ time units. The system produced includes 2 MicroBlaze processors, 2 LMB buses, 2 FSL buses and 1 PLB bus. It occupies 3890 slices.

The third implementation test a case of system fine tuning is presented. Tight constraints are set for both area and memory usage. Area is set to be less than 6000 slices and BRAM usage less than 96. The architecture with the best performance under the above constraints is wanted, therefore the objective is again minimizing tf . The solver required 312 sec to produce a solution. The minimum processing time achieved is 75 time units with an area of 5610 slices. The system produced includes 3 MicroBlaze processors, 3 LMB buses, 3 FSL buses and 1 PLB bus. It requires 80 BRAMs (160kB memory space). For this implementation the Gantt scheduling graph of the tasks mapped on the processing units which was produced by our model is presented in Figure 2.

TABLE 2.4. Time Delay and Memory Requirements

	MicroBlaze			***Hardware***	
Tasks	***t.u.***	***Instr. Mem. (kB)***	***Data mem. (kB)***	***t.u.***	***Data mem. (kB)***
E1	10	5	20		
E2	15	5	30	5	16
E3	5	5	20		
E4	10	5	30		
E5	20	10	30	8	16
E6	5	5	20		
E7	10	5	40		

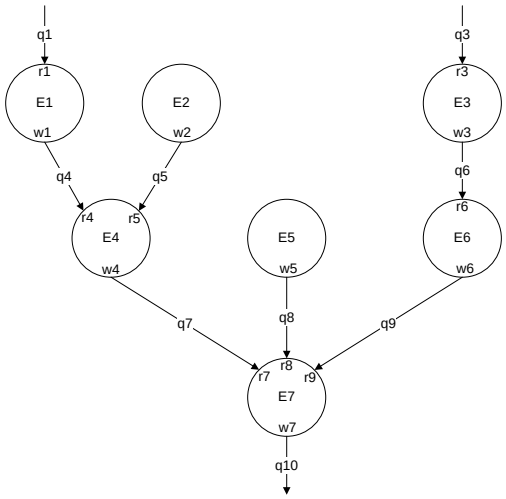


Figure 2.5. Application Task Graph

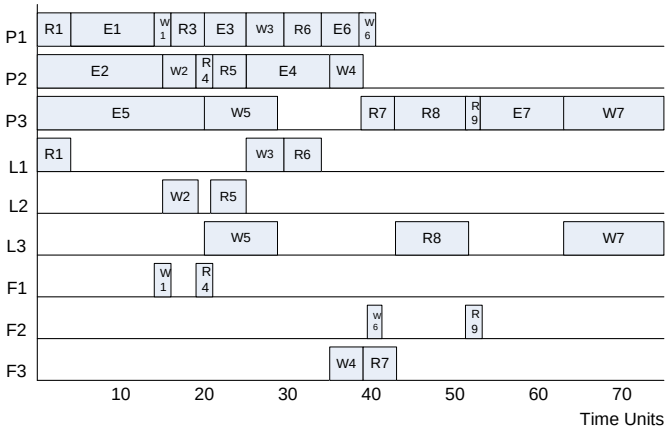


Figure 2.6. Gantt Scheduling Graph

2.12. Power and Temperature ILP model

The same model can be extended to include estimations for power and temperature. The core of the model remains the same but new decision variables are added to calculate power as a first step, and by using power estimation extract the execution time temperature.

2.12.1. Power and Temperature Estimation

The temperature of an FPGA device is estimated through the following formula [52]:

$$T_J \geq T_{Ambient} + \theta_{JA} \cdot P_{total}$$

Equation 2.5. FPGA Device Temperature

T_J is the junction temperature, θ_{JA} is the junction to ambient thermal resistance and it is provided by the FPGA specifications and P_{total} is the total applied power. The total applied power for a system can be estimated through appropriate development tools provided by major manufactures such as the Xilinx XPE [40] and the Altera PowerPlay Early Power Estimators [53] from early development stages. These tools can estimate static and dynamic power for the system and device temperature by taking into account early design information such as occupied area, clock frequency and estimated toggle rates. However, these values are of estimated average power for the complete system and cannot illustrate power and temperature fluctuation during operation or temperature diversity on the surface of the device.

The work in this thesis comes to increase the granularity of the provided power and temperature estimations by implementing a "sliding power observation window" for each implemented processing unit (Figure 2.7).

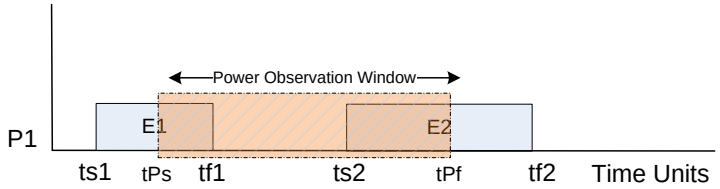


Figure 2.7. Sliding Power Observation Window Illustration

The principle of the "sliding power observation window" is that in order to calculate temperature through the formula in Equation 2.5 at a specific time interval

on operation time for a system we need to know the average applied power at that time interval. The following assumptions are made:

- The power consumption per task per processing unit is a known value.
- The thermal energy due to the power consumed before the observation window's starting point is already removed from the device and doesn't influence present temperature.
- The average power for a processing unit is the total energy consumed by the processing unit at a time interval divided by the total interval time.

For example in Figure 2.7 it can be seen that processing unit P1 executes parts of two tasks during the moving power window. If $Pd1$ and $Pd2$ are the dynamic power for tasks E1 and E2 respectively, and Ps the static power of the processing unit then the average power for this window is:

$$Pw1 = Ps + \frac{Pd1 \cdot (tf1 - tps) + Pd2 \cdot (tPf - ts2)}{tPf - tPs}$$

Equation 2.6. Average Power for Execution Window

The estimated temperature of the processing unit is:

$$Tw1 = T_{Ambient} + \theta_{JA} \cdot Pw1$$

Equation 2.7. Estimated Temperature of the Processing Unit

The system temperature T is calculated by the weighted sum of all processing units' temperatures [40].

$$T = \sum_{k} aw_k \cdot Tw_k \quad \forall k \in [1, \text{total number of processing units}]$$

$$aw_k = \frac{\text{processor k area}}{\text{total system area}}$$

Equation 2.8. System Temperature

The appropriate "sliding power observation window" size can be calculated through proper device characterization techniques which are beyond the scope of this work. For ease of use it is assumed that it must be $tPf - tSf \geq \max(\text{task execution time})$. At this stage of the model's development the impact of neighboring temperatures for a processing unit's temperature is not considered since routing information is not included in the model, but the impact of the area of each processing unit on the device is included in the weight factor of the system temperature.

2.12.2. Power and Temperature ILP Formulation

In Figure 2.8 the five different timing positions of a task relative to the power window can be observed. For the power calculation decision variables must be applied in order to include only the duration of the task that lies within the window interval in the estimation. As ILPs are NP-complete it is highly important to produce a formulation that generates the minimum amount of variables and constraints. Through experimentation the following formulation was chosen.

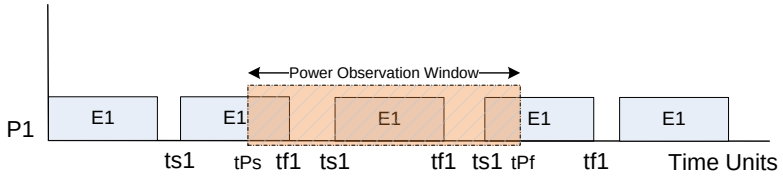


Figure 2.8. Task Position Relative to Power Observation Window

The task start and finish time must be compared with the start and finish time of the moving power window. To do so the following binary decision variables with an appropriate set of constraints are defined.

TABLE 2.5. Power Window Binary Decision Variables and Constraints

Variables	Value	Constraints
$ZPs1_{t_o,i}$	1: when $ts_i \geq tPs$ 0: otherwise	$ZPs1_{t_o,i} + ZPs2_{t_o,i} + ZPs3_{t_o,i} = 1$ $ts_i - tPs \cdot ZPs1_{t_o,1} \geq 0$ $tf_i(1 - ZPs1_{t_o,1}) - tPs \cdot ZPs2_{t_o,1} \geq 0$ $tPs(1 - ZPs2_{t_o,1}) - tf_i \cdot ZPs3_{t_o,1} \geq 0$
$ZPs2_{t_o,i}$	1: when $tf_i \geq tPs \geq ts_i$ 0: otherwise	
$ZPs3_{t_o,i}$	1: when $tPs \geq tf_i$ 0: otherwise	

$ZPf1_{t_o,i}$	1: when $ts_i \geq tPf$ 0: otherwise	$ZPf1_{t_o,i} + ZPf2_{t_o,i} + ZPf3_{t_o,i} = 1$ $ts_i - tPf \cdot ZPf1_{t_o,1} \geq 0$ $tf_i(1 - ZPf1_{t_o,1}) - tPs \cdot ZPf2_{t_o,1} \geq 0$ $tPf(1 - ZPf2_{t_o,1}) - tf_i \cdot ZPf3_{t_o,1} \geq 0$
$ZPf2_{t_o,i}$	1: when $tf_i \geq tPf \geq ts_i$ 0: otherwise	
$ZPf3_{t_o,i}$	1: when $tPf \geq tf_i$ 0: otherwise	

The power consumed by a task's operation is:

$$Pw_t = \frac{Pd_t \cdot (tf_t - tPs) \cdot ZPs2_{t_o,i} + Pd_t \cdot tZ_{i,k} \cdot ZPs1_{t_o,i} \cdot ZPf3_{t_o,i} + Pd_t \cdot (tPs - ts_t) \cdot ZPf2_{t_o,i}}{tPf - tPs}$$

Equation 2.9. Power Consumed by a Task's Operation

The total power consumed by a processing unit during the power window interval is:

$$Ptw_k = Ps_k \cdot CSX_k + \sum_i Pw_i \cdot X_{i,k} \quad \forall i \in TASK$$

Equation 2.10. Total Power Consumed by a Processing Unit (Power Window)

It is obvious that these constraints and formulas include multiplications of two model variables, one integer and one binary or two binary variables, which is a non linear form that cannot be included in an ILP model. Therefore these calculations need to be linearized. For this procedure a method proposed in [45] and [39] is used. Firstly a suitably large number must be chosen (T_MAX is suitable in this model) which will be a bound for the variables. For each $t_i \cdot ZP_i$ calculation a new variable $tZP_i = t_i \cdot ZP_i$ is defined. This variable must satisfy the following constraints in order to substitute the non linear multiplication:

$$tZP_i \le T\_MAX \cdot ZP_i$$
$$tZP_i - t_i + T\_MAX \cdot ZP_i \le T\_MAX$$
$$tZP_i - t_i - T\_MAX \cdot ZP_i \ge -T\_MAX$$

Equation 2.11. Constaints for Not Linear Multiplications

Thus, the multiplied variables are substituted by the new defined tZP_i variable.

A variable TPw_k is set for the estimated junction temperature of a processing unit during an observation window. Each TPw_k is defined as follows:

$$TPw_k = T_{Ambient} + \theta_{JA} \cdot Ptw_k$$

Equation 2.12.Estimated Junction Temperature

The system temperature T is calculated by calculating the weighted sum of all processors' temperatures. Since total system area is also a model variable and cannot be a divisor in a formulation, the weight factor is substituted in the following way, where approximate system area is the area calculated for the same constraints without the power and temperature ILP.

$$T = \sum_k aw_k \cdot TPw_k \quad \forall k \in PE, \qquad aw_k = \frac{\text{processor k area}}{\text{approximate system area}}$$

Equation 2.13. System Temperature

With this formulation constraints can be set for system temperature for every chosen observation window, as well as the temperature of each processing unit. By setting a constant $Temp\_MAX$ these constraints are defined as follows:

$$T \le Temp\_MAX \quad (1)$$
$$TPw_k \le Temp\_MAX, \ \forall k \in PE \quad (2)$$

Equation 2.14. Temperature Constraints

Constraint Equation 2.14(2) includes constraint Equation 2.14(1) by definition. These constraints adjust both architecture and scheduling in order to comply with the temperature thresholds and decrease the possibility of hotspot generation during operation time. If multiple observation windows are required then a new set of variables and constraints must be added for every new window.

2.13. Trial System Results (Power/Temperature Model)

The complication increase which the Power and Temperature estimation causes to the introduced ILP model is examined first. A basic three node graph is used and it is assumed that there are three microprocessors and a hardware accelerator available for implementation, as well as all three types of communication buses for the edges (shared, private and FIFO bus). The simple ILP model generates 122 variables and 165 constraints. The addition of power and temperature estimation increases these numbers to 179 variables and 265 constraints for calculation for one window. For every power observation window added, the increase of variables and constraints is linear. For two windows 236 variables and 383 constraints are generated and for three 336 variables and 492 constraints.

TABLE 2.6. Time Delay and Power Characteristics

	MicroBlaze			**Hardware**		
Tasks	**t.u.**	**Static Power**	**Dynamic Power**	**t.u.**	**Static Power**	**Dynamic Power**
E1	10	1	2			
E2	15	1	2	9	0.5	1
E3	5	1	1			
E4	10	1	2			
E5	20	1	4	12	1	2
E6	5	1	1			
E7	10	1	2			

In order to demonstrate the efficiency of the proposed ILP model, a trial system is presented. The same task graph as for the memory design space exploration is used (Figure 2.5). It is assumed that there are 3 microprocessors

available and 2 hardware accelerators, one assigned to task 2 and one to task 5. The task delays, static and dynamic power values for each task implemented on each processing unit are presented in TABLE 2.6. A power observation window of 25 time units duration is set. The θ_{JA} is considered to be $6.5C°/W$. The experiment is performed on an Intel i3 computer with 4Gb RAM.

TABLE 2.7. ILP Model Trial System Results

Const.	Area	tf	System	Pe1	Pe2	Pe3	Hw1	Hw2
min(tf)	11260	66	5.845 4.94	1.28 1.1	2.1 1	1 1	0.7 0.5	0.77 1.3
			56.77 56.42	58.31 57.14	63.64 56.5	56.6 56.6	54.54 53.25	55.00 58.64
$T\_MAX = 100$ min(Area)	3240	100	4.18 6.9	2.28 3	1.9 3.9	- -	- -	- -
			62.66 70.86	64.8 69.5	62.34 75.35	- -	- -	- -
$T\_MAX = 100$ $Temp\_MAX = 70$ min(Area)	3440	100	4.28 4.076	2.28 3.076	2 1	- -	- -	- -
			63.14 61.55	64.82 70	63 56.5	- -	- -	- -

The basic ILP model created for the optimization of this system generates 663 variables and 1070 constraints. By inserting two power observation windows, one from time units 0 to 25, and the second from 26 to 50 the number of variables and constraints increases to 930 and 1582 respectively. We test the model by setting various constraints to demonstrate how the system implementation is affected. The results are demonstrated in TABLE 2.7. In the System, PE and HW columns we present power and temperature for the first and second observation window. The power is measured in Watts and the temperature in C^o. Ambient temperature is set to $T_{ambient} = 50C^o$. We first set the model to execution time optimization with relaxed area constraints. As can be seen the system achieves a $tf = 66$ with area $A = 11260$, utilizing five of the available processing units, two hardware accelerators and three

microprocessors. We then set $T\_MAX = 100$ to allow for area optimization. As can be seen by the temperature values in the second observation window Pe2 reaches a temperature of $75.35\,C^{\circ}$. We set a temperature threshold for all processing units $Temp\_MAX = 70$ and rerun the model with the same time constraint. The result was an increase in area 200 slices. This was due to the rescheduling of the tasks on the two processing units to keep the temperature below the set threshold, which led to the addition of one extra FIFO bus for communication purposes. This model took 487.78 sec to be solved on the lp_solve ILP solver [39].

2.14. Conclusions

The objective of this research was the definition of a model formulation to execute efficient design space exploration for MPSoC systems at an early stage in the system design cycle. The first step for this work was a hands on experience on a design space exploration performed by developing each system separately and profiling the application on it. The chosen algorithm was JPEG as a commonly used image processing application. Data and task level parallelism was explored for different system architectures, as well as the correlation between performance increase and hardware resources usage. Twenty different system implementations with 3 different memory approaches and 4 different processor architectures were explored. The platform of choice was Xilinx FPGAs and as a processing element the MicroBlaze processor was chosen. A new parameter called *Hardware Efficiency* was introduced to associate area increase (resource usage increase) with performance. It was observed that a highest speed up of 3.27 was achieved for a 4 MicroBlaze system and a highest hardware efficiency of 3.27 was achieved for a dual MicroBlaze system with use of internal memory.

The next step was the introduction of an innovative ILP model for optimizing performance, area and memory usage of a hybrid FPGA-based MPSoC is presented. The model proposed introduces for the first time the distinct constraint of memory usage that is imported without a significant increase in the complexity of the model. Memory usage and BRAM occupation is a critical factor of modern MPSoC designs, which target high throughput applications. This addition offers extreme flexibility to

the designers who can now fine tune the system design according to the specific application needs. Different objective functions and constraints were explored. It has been demonstrated that the proposed ILP can be easily adapted to minimize execution time, memory usage and area. By changing objective functions, the different design parameters' values change significantly to double the value of a previous implementation.

The above model was extended to include temperature management. The concept of "sliding power observation window" was proposed for power estimation over time during system operation. The inclusion of the power estimation formulation to the previous model lead to a model that can optimize the system's most important parameters such as area, time, power and temperature. In addition the final model can decrease the possibility of hotspot generation through setting temperature thresholds for the processing units. A combination of the offered model constraints can produce the optimum system implementation for the designer's needs. One very important characteristic is that the inclusion of the new parameters does not impose significant overhead in the number of variables and constraints.

Future developments will include the automated model generation through a parsing tool that will accept as an input the application task graph and the system architecture parameters.

Chapter 3

MPSoC Implementation for Multimedia Applications

3.1. Introduction

Real-time multimedia applications require high performance processing systems to implement image processing algorithms. The computational demands call for implementations optimized to meet the applications' needs. The constant evolution of standards and the requirement for scalability present challenges that only reconfigurable platforms such as FPGAs can answer. More specifically now that FPGA fabric is ever denser and the performance that can be retrieved from such devices is ever more close to ASIC technology, FPGAs present an excellent solution with smaller development costs and significantly smaller time to market.

Machine vision implementations are now commonly used in industrial, biomedical and security applications. Their main role is real-time image-based inspection or analysis. A fundamental characteristic of such systems is the use of cameras for image acquisition. The resolution and speed of the camera is determined by the system's needs. The acquired images are then processed by a sequence of image processing algorithms to extract the required information. Machine vision systems have hard real-time response requirements. For several applications these requirements can be even critical. Therefore designing a Machine

Vision system with hard real-time requirements is a challenging task. Great effort is required to analyze the application, define the specifications, explore the possible implementation strategies, design, optimize, test and verify the system. All these challenges were met and tackled with the work of this thesis.

3.1.1. Thesis Contribution

The contribution of this thesis is the design of a complicated Machine Vision system that extensively exploits parallelism as well as the design of specific high performance parallel modules for image processing. The proposed Machine Vision system is integrated in a complex prototype system that includes microcontrollers, actuators and sensors. The system was designed in collaboration with Micro2Gen Ltd. Micro2Gen proposed a sequence of algorithms to be implemented and the AUTHeLAB team adapted them to be implementable on hardware. The machine vision system consists of an edge detection module, a bounding box detection module and the flow detection system.

For this thesis the bit-accurate simulation for the edge detection system was developed. With the available bit accurate simulation the necessary precision for the edge detection was defined, exploration for the architecture requirements was executed and the necessary parallelism was chosen.

In addition, high performance modules for the flow detection system were conceived and developed. More specifically a center of mass (CoM), an alarm point detection module and a median calculation module were designed and developed to be included in the machine vision system, but to be generic enough to be used in various image processing applications. The center of mass and alarm point detection modules are the cores of operation for the flow detection process. Center of mass calculation is also very commonly used in many image processing algorithms for data reduction (as suggested also in Chapter 4). Median calculation was introduced as an improved method for identifying flow coordinates. Median calculation module exploits parallelism in calculation and memory design. Both modules center of mass and median modules can be used in a variety of image processing applications, such as the one presented in Chapter 4.

The scientific results from the edge detection module implementation were published in [90] and [91]. The flow detection modules and the median calculation module were published in [96] and [142]. The complete machine vision system was published in [120] and [88].

The work of this research was partially funded by the research project “Lab-on-Chip” of the Fund for Hellenic Technology Clusters in microelectronics Framework.

3.2. Microfluidics

Before the Machine Vision system is described, the application field of microfluidic Lab-On-Chips (LoCs) must be introduced to demonstrate the motivation behind the application.

The advancements in biology and molecular diagnostics call for faster and cheaper technologies to execute analyses. This has lead to the use of microfluidics and lab-on-chip systems (Figure 3.1) becoming a very important factor in diagnostic procedures. In vitro microfluidic analysis on Lab-on-Chip devices is widely used for biomedical research and clinical diagnostics due to the advantages they offer. Lab-on-Chips are micromechanical devices (depending on the functionality incorporated they are often also called Micro Total Analysis Systems – μTAS [55]), which can integrate one or more biological laboratory analyses on a single chip area varying from a few square millimeters to a few square centimeters. Lab-on-chips are not restricted to microfluidics but can also include droplet based systems and microarrays. However the basic principle is the same: a combination of miniaturized analytical chemistry assays, microelectromechanical systems (MEMS), which are the integration of mechanical and electrical elements (e.g. logical circuits, sensors, or actuators) and miniaturized flow control devices (e.g. channels, pumps, mixers and valves).

In microfluidic LoCs (Figure 3.2), which are the devices of interest for the presented research, the volume of the microdevice’s channels is very small and the amount of necessary reagents and analytes can be maintained small as well. Lab-on-Chips can support different phases, such as DNA extraction, DNA amplification (e.g.

with the use of Polymerase-Chain Reaction, PCR [56]), and hybridization detection targeting identification of DNA polymorphisms with specially developed biosensors or optical means. An overview of microfluidic systems can be found in [57]. The advantages of LoC devices are:

- their small size,
- lower cost,
- small consumption of biological analytes,
- increased portability and
- ability to be integrated with other types of micro-analysis systems.

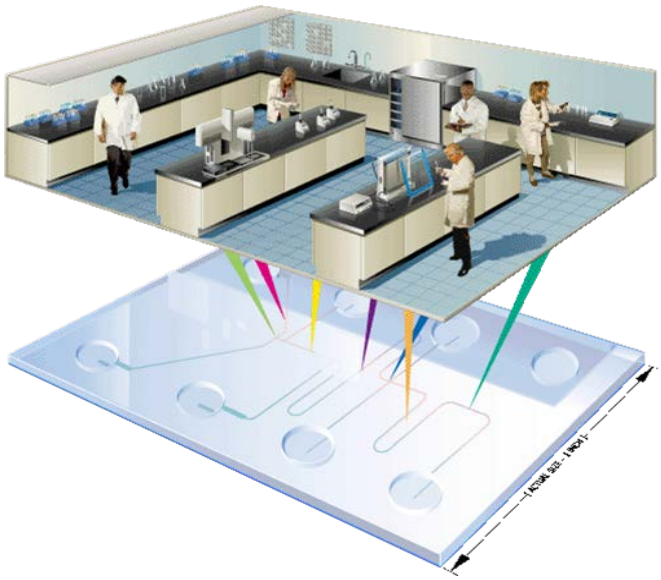


Figure 3.1. The Lab-On-Chip working principle [58]

LoC systems, especially disposable ones, are often controlled by off-chip control units, embedded in a dedicated instrument. Placing the control units off chip reduces substantially the cost per device. These control units receive the information sent both by on-chip and off-chip sensors to determine the direction of the flow, activate biological procedures and stop the experiment in case of an exception. A variety of sensors is used such as thermal, pneumatic, mechanical, electrical and optical [60].

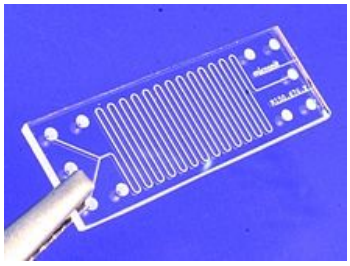


Figure 3.2. Simple Lab-On-Chip [59]

The flow within such a microsystem can be continuous, flow with different phases, or flow with droplets of fluid within another baseline fluid. In microsystems featuring a continuous flow model, typical flow control mechanisms are based on source-sink pairs detecting the flow of different liquids through pre-selected points on a chip design, such as infrared sensors, light barriers or even ultra-sound detectors [61]. In the research project "Lab-on-Chip" of the Fund for Hellenic Technology Clusters in Microelectronics Framework we have experimented with an alternative technique based on machine vision to cover the complete microsystem at all times and detect different flows in different channels. This approach introduces a set of advantages over previous approaches, such as continuous monitoring of the complete microsystem, resilience to changes, since the measurement setup is not affected by redesigns of the microsystem, avoidance of micro-mechanical alignment problems in case of disposable microsystems and finally lower costs.

Exploiting machine vision for LoC implementations requires real time response and precision. FPGAs have been exploited for both LoC control and machine vision implementations due to their high speed, ability to host systems on chip, low cost and small time-to-market.

3.3. Related work

Machine vision implementations for droplet based microfluidic chips are presented in [62] and [63]. These are software implementations (LabView) for droplet motion control where the machine vision system requirements are not as demanding as the proposed implementation, since they aim primarily at the

modeling of motion or deploy low-speed motion control of distinct objects (droplets). Other approaches address similar issues such as Dimalanta et al. [64] who describe a software approach for automatic detection and molecule analysis in large DNA molecule arrays. Such approaches focus mainly on the hybridization detection phase and usually do not cover a complete “bleed-to-read” lab-on-chip integrating multiple analysis phases and complex microfluidic structures. In the same category, Merola et al. [65] propose a digital holography method for specimen analysis. Batabyal et al. [66] suggest a method, which combines fluorescence microscopy and spectroscopy for ultra fast chemical process investigation in microfluidic chips. The image processing is again done by software in this approach and focuses on the temporal evolution of a chemical process much more than on the control of complex fluidic motion. Uvet et al. [67] propose a vision system for cell-based microfluidic applications. Kornaros [68] implements on FPGA a multi-core soft-processor system for LoC microarrays utilizing edge detection, but once again focusing mainly on the hybridization detection and completely omitting the fluidic control aspect. The MicroBlaze multiprocessing system used has sufficient performance to accommodate the sobel edge detection algorithm implemented in this method. Sapuppo et. al. [69] present an ad-hoc optical system to detect bubbles in microfluidic experiments for in vivo and in vitro systems. Their presented system is not a hardware implementation per se, but uses the AnaFocus ACE16kv2 FPP and the Altera Nios II Digital Microprocessor. The system achieves real time response but with a lower resolution demand. Our specifications called for a complete hardware implementation of the machine vision system.

The proposed Machine Vision system is an implementation of a sequence of appropriately adapted image processing algorithms: i) an edge detection implementation, ii) a bounding box detection implementation and iii) a flow detection module. Related work for the modules and work proposed in this thesis follows.

Edge detection is the first step in many computer vision algorithms. It is used to identify sharp discontinuities in an image, such as changes in luminosity or in the intensity due to changes in scene structure. Edge detection has been researched extensively. A lot of edge detector algorithms have been proposed, such as Robert

detector, Prewitt detector, Kirsch detector, Gauss-Laplace detector and Canny detector. Among all the above algorithms, Canny algorithm [70] is the most widely used due to its good performance and its ability to extract optimally edges even in images that are contaminated by Gaussian noise. Canny algorithm has the ability to achieve a low error-rate by eliminating almost all non-edges and improving the localization of all identified edges.

Because of its algorithmic efficiency and applicability many Canny implementations have been proposed. In [71] an implementation of a self-adapt threshold Canny algorithm is proposed. This design is FPGA based and intended for a mobile robot system. The results presented are for an Altera Cyclone FPGA and the highest frequency achieved is 27MHz, which result in 2.5ms computation time for a 360 x 280 grayscale image. In [72] an industrial implementation for ceramic tiles defect detection is presented, which defines the hysteresis thresholds with a histogram subtraction method. A Canny edge detection on NVIDIA CUDA is presented in [73], which takes advantage of the CUDA framework to implement the entire Canny algorithm on a GPU. It achieves a 10.92 ms computation time for a 1024 x 1024 image. In [74] there is an implementation of an adaptive edge-detection filter on an FPGA using a combination of hardware and software components proposed by Altera. In [75] a reconfigurable architecture and implementation of edge-detection using Handle-C is presented This is a pipelined design of a canny-like edge detection algorithm. It achieves a computation time of 4.2ms for a 256 x 256 grayscale image.

Center of mass implementations are common in image processing applications. They are mostly used as data reduction tools. Shi and Tsui in [76] describe an FPGA-based smart camera for gesture recognition. Center of mass calculations are used for feature extraction. Lu, Ren and Yu [77] use a kernel based center of mass for mean-shift tracking. Their implementation is of a more complicated function than required for the proposed machine vision in this thesis. Grull et al. [78] use center of mass for localization in microscopy image analysis, following a principle similar to ours. In [79] Curry, Morgan and Kilmartin present a Xilinx FPGA implementation for an object detection image classifier. In this

implementation the center of mass is calculated as a centroid on a Xilinx Virtex XCV400 FPGA.

Center of mass calculation can be insufficient if there is noise in the sample that can reduce precision in the result. This is the reason why using a median calculation module was also explored as a solution. The median filter is a very useful and important tool in image processing. This filter can be used for noise reduction (image smoothing) as a pre-processing step in most image processing algorithms, having the advantage that it distinctly preserves image edges, in the same principle as center of mass calculation. Median calculations are also commonly used for localization purposes.

The median value in a set of numbers is the numerical value separating the higher half of the numbers from the lower half. A median filter is a digital filtering technique where the incoming signal is run entry by entry, and each entry is replaced by the median value of its neighboring entries. The number of entries used to calculate the median is decided by a predefined window size. When the median filter is 2D the window used to define the median value is 2D as well. A common window size used is a 3 x 3 window. Example of such implementations on FPGAs are presented in [80] and [81]. In [80] a description of a generic implementation of such a filter is well described, while in [81] a more modern approach is presented, giving emphasis on the sorting algorithm used to find the median value. Fahmy, Cheunga and Luk present in [82] a weighted median approach optimized for an FPGA implementation on a Xilinx Virtex II FPGA. The same group in [83] applies the weighted median approach to a one-dimensional window.

This commonly used image processing algorithms were used as an application field for conducting design space exploration and determining the appropriate level of parallelism for the proposed machine vision system.

3.4. System Specifications

The machine vision system used as an application example for this thesis is an FPGA-based flow detection system for microfluidic LoCs, designed to identify and monitor flows in experiments where the moving faces of the flows are always visible

(laminar or plug flow). The flows are two-phased (there are two visible identifiable liquids inside the same flow), but the liquid remains homogeneous in the channel and the meniscus at the start and at the end of the liquid flow is visible to the camera. The target is to identify and track individual flows (front tracking) on the microfluidic chip with a real-time response, following a high-speed camera at 60fps with a 1 Mpixel resolution.

As with every implementation that requires real-time response, what determine the performance of the system are the given specifications. This is the most important factor that also defines the degree of parallelism used in the implementation. The detection through the proposed machine vision component relies both on the characteristics of the overall optical setup and the geometrical properties of the microsystem (Lab-On-Chip device). In the presented case microsystems up to 12,7 cm x 8,5 cm (standard microtiter-plate format) are used and the footage is captured using a 1:1.2/6mm varifocal lens at a distance of 10cm from the surface of the microsystem. For the capture various types of Microsystems were tested by Micro2Gen, both molded and micro-milled. The majority of the microsystems tested are based on biocompatible polymeric materials. The channels have different cross-sections, i.e. circular, rectangular or trapezoid. The distance between the two edges of the channels is 200μm (in case of circular channels this corresponds to the diameter, in case of rectangular channels to the sides). In this channel “width” the specified flow of 60μl/s corresponds to approximately 20 mm/s speed of the fluidic front captured by the camera. The 20 mm/s maximum speed of the flow defines the specifications for the flow detection implementation.

The machine vision system is integratable in different Point-of-Care systems, possibly used in non-ideal laboratory conditions. Therefore, the challenge is to create a system that can achieve real-time response while dealing with video noise, non-ideal lighting conditions (positioning of light sources in a way that they either cause reflections or yield a low contrast between the channels and the background) and Lab-on-Chip displacements (rotation, translation) during the video capture, while being portable and easy to integrate. These demands call for a single system-on-chip complete hardware implementation as opposed to one with multiple microprocessors or hardware/software co-design. This multiprocessing parallel

implementation was the first in the relevant literature for the specific application field.

The machine vision algorithm and software model for the machine vision system were introduced by Micro2gen and patented in [76] and subsequently implemented on hardware by Aristotle University of Thessaloniki after adaptation.

3.5. Proposed Machine Vision System

The goal of the proposed machine vision system is to capture the coordinates of the moving faces (menisci or fronts) of the flows. The experimental protocol for which the machine vision system was developed requires the channel to be empty before the liquid is inserted, therefore the face of the flow is clearly visible. In continuous flow systems the menisci between different fluids exhibiting different viscosity or color can also be easily identified, thus the system can be used for two-phase-microfluidic experimental protocols as well. The system also differentiates between the starting face or the "head" of the flow, and the ending face or the "tail" of the flow. Up to five different concurrent flows can be identified, differentiating both the head and the tail of each flow. The maximum number of flows is in accordance with the experimental protocol specifications and was limited in order to make a better use of the hardware resources. The number of flows also defined the degree of parallelism in the implementation. However, the hardware implementation was designed to be generic and modular and increasing the number of flows is a trade-off with the implementation area and can be adapted when different image processing application fields are targeted.

The machine vision system can also support user-defined points of interest called "alarm points". These are coordinates preset by the user where the experiment reaches a decision point (resulting in a change of flow direction, activation/deactivation of an actuator etc.). The machine vision system checks whether a flow face has reached an alarm point and notifies the experiment control module and the user interface.

Lab-On-Chip devices are produced by different manufacturers and have different characteristics. In order to allow flexibility in the use of a variety of LoC

devices the machine vision implementation is developed in such way that a specific design element (such as redundancy patterns on the chip etc.) is not a prerequisite for the machine vision system to work. Nevertheless, the system can only work with transparent microsystems, which is the majority currently. Most microsystems are currently manufactured on the basis of transparent polymeric materials.

A rigid background is required during the video capture for the flow monitoring. However, some miniscule movements are usually present in such setups due to mechanical pressure from the integrated valves and the external actuators, which are commonly used for pumping and routing. These micro-movements need to be compensated by the algorithm in order to have a precise flow coordinate detection.

In order to reduce the computational intensity of the flow detection algorithm an annotation of the LoC device is required. This procedure is done by an in-house developed software (developed by Micro2Gen), which allows the user to preset the LoC alarm points and the flows' starting coordinates (by reference to the chip's upper left corner). The flows' starting coordinates are used to reduce computational requirements. This information is imported by the machine vision system in form of an input file. The input file also includes general information about the experiment execution, which is essential for the machine vision operation, such as video resolution and frame rate. In order to correctly interpret and use this input file the machine vision system must detect the LoC's upper left corner, since all coordinates are in reference to this point. This is achieved by using a specially adapted chip frame detection algorithm. With the chip frame detection algorithm the machine vision system calculates the necessary reference point and at the same time reduces the memory accesses required for the flow detection process. This is achieved because in the majority of cases the LoC device doesn't occupy the whole video frame and consequently by detecting the chip frame area the system only needs to access the memory data of interest. Bearing in mind that the system was to be integrated with other memory access demanding cores, such as the frame grabber, video compression or execution control, it was essential to reduce the memory accesses as much as possible. In addition, the chip frame detection

processing stage is run on every frame in order to recalculate the reference point if a LoC displacement has occurred.

The machine vision algorithm and system architecture was optimized for FPGA implementation. Various techniques were used to improve the system's performance, such as parallelization and pipelining, memory space cropping, mathematical simplifications etc. The developed system is fully parametric and adjustable to operate in different noise levels, with different detection sensitivity at the different processing steps. Threshold values at different stages of the machine vision system are user defined parameters and can be changed at run-time.

3.6. Machine Vision Implementation and Design Space Exploration

As described before the machine vision system is following a high-speed camera at 60 fps. The camera input is transformed from RGB to YCbCr color space from which the machine vision system requires only the luminance "Y" component. Therefore, the machine vision system is designed for grayscale 8 bit pixel input data.

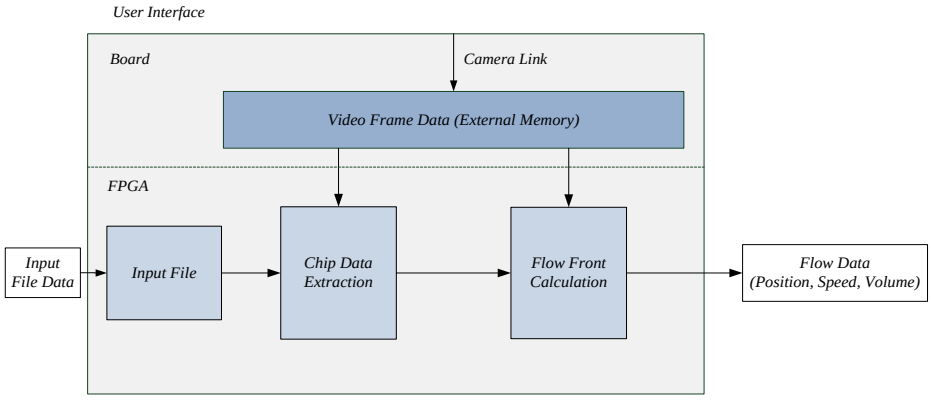


Figure 3.3. Machine Vision Top Level Block Diagram

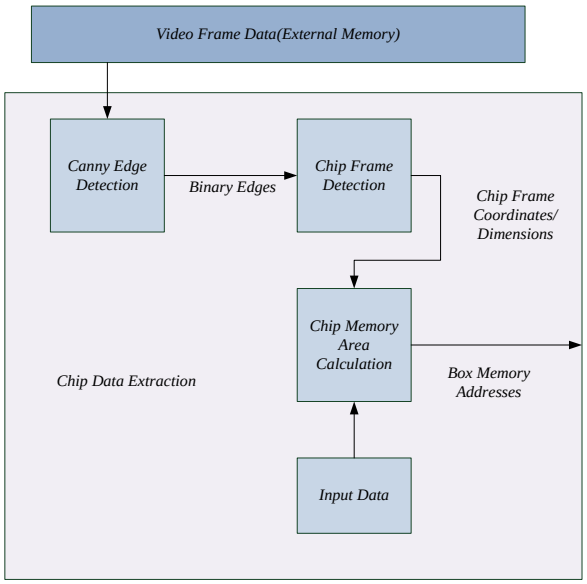


Figure 3.4. Machine Vision Chip Data Extraction Block Diagram

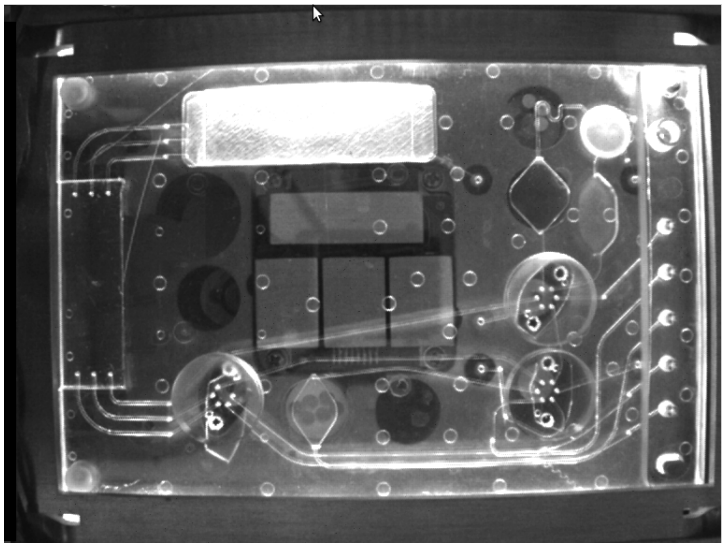


Figure 3.5. Grayscale Image Framce from a Microfluidic LoC

The proposed machine vision system depends on a chip frame detection stage. The chip frame detection stage is a customized version of the Hough transform. The Hough transform requires a binary input, produced by an edge detection pre-processing step. The Hough transform (chip frame detection) and the edge detection stage combined are the chip data extraction stage of the implementation (Figure 3.3 and Figure 3.4). The basic processing steps of the

machine vision system are: i) edge detection, ii) chip frame detection and iii) flow front calculation.

An indicative image frame of a microfluidic LoC used for the proposed Machine Vision implementation is presented in Figure 3.5.

3.7. Edge Detection

The first step of the machine vision computational sequence is an edge detection stage. Since the proposed machine vision system will be integrated in a system that will not be used in ideal laboratory conditions it is essential that measures to cope with the noise imposed by changes in video capture environment are taken. The Canny Edge detection algorithm [89] was implemented, because of its ability to improve localization of edges even in images highly distorted by noise.

3.7.1. Edge Detection Algorithm

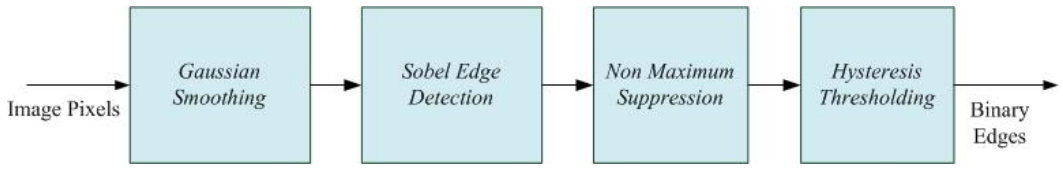


Figure 3.6. Canny Edge Detection Stages

The computational stages of Canny Edge detection algorithm are presented in Figure 3.6. The first stage is a smoothing filter which is essential for eliminating the image noise. The smoothing filter used is a Gaussian Convolution Equation 3.1(1), and it is implemented by using a mask Equation 3.1(2) which is sled over the image manipulating a square of pixels at a time (Figure 3.7). In the relevant literature implementations with 3 x 3, 5 x 5 and 7 x 7 masks can be found. The bigger the dimensions of the mask are, the lower the sensitivity of the detector to noise.

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \quad (1)$$

$$G=\frac{1}{159}\begin{bmatrix}2 & 4 & 5 & 4 & 2\\ 4 & 9 & 12 & 9 & 4\\ 5 & 12 & 15 & 12 & 5\\ 4 & 9 & 12 & 9 & 4\\ 2 & 4 & 5 & 4 & 2\end{bmatrix} \quad (2)$$

Equation 3.1. Gaussian Smoothing

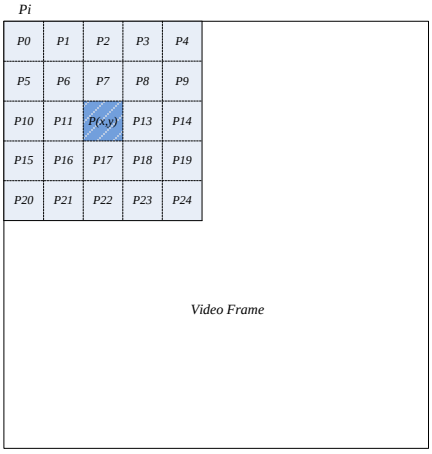


Figure 3.7. Gaussian Smoothing Area for Pixel P(x,y) for a 5 x 5 Convolution Mask

$$G_x'=\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} \quad G_y'=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \quad (3)$$

$$\left|G'\right|=\sqrt{G_x'^2+G_y'^2}\approx\left|G_x'\right|+\left|G_y'\right| \quad (4)$$

$$\theta=\arctan\left(\frac{\left|G'(y)\right|}{\left|G'(x)\right|}\right) \quad (5)$$

Equation 3.2. Sobel Edge Detection

The second stage of the Canny algorithm is calculation of the luminosity gradient of the image. It is executed by a convolution with the Sobel gradient operators, which are a pair of 3 x 3 convolution masks Equation 3.2(3), one for the calculation of the gradient in the x-direction (columns) and the other for the calculation of the gradient in the y-direction (rows). Gradient magnitude and orientation is calculated by using the equations Equation 3.2(4),(5).

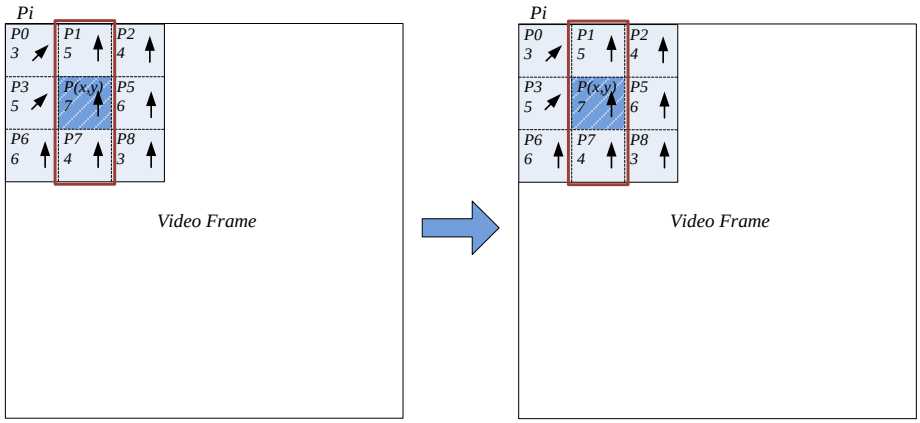


Figure 3.8. Non Maximum Suppression: P(x,y) magnitude is higher in the orientation of the gradient

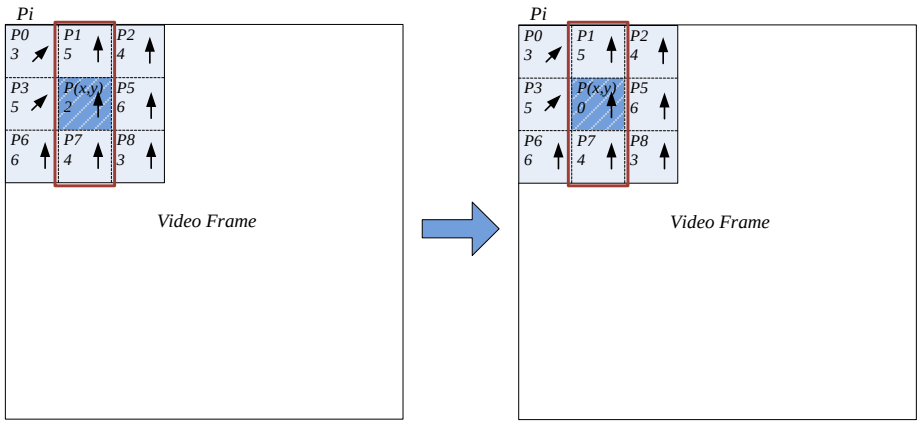


Figure 3.9. Non Maximum Suppression: P(x,y) magnitude is suppressed

Nom maximum suppression is a procedure which improves the localization of the edges by suppressing the values of the pixels whose magnitude is not maximum on the orientation of the gradient. In Figure 3.8 the case where the magnitude of the calculated pixel value is higher on the orientation of the gradient and therefore kept can be seen. In Figure 3.9 however, the value is not the highest and therefore on the next computational step it is suppressed.

Finally due to the difficulty to set a global gradient intensity threshold to pick the correct edges of an image only, hysteresis thresholding is used. In this procedure the magnitudes of the remaining edge values are compared with two threshold values, high threshold Th_h and low threshold Th_l. All pixels with values greater than Th_h are considered to be definite edges. All pixels with values lower than Th_l are considered non edges, and the rest are possible edges. Hysteresis is the procedure where all possible edges are suppressed, unless there is a definite path from this pixel to a pixel that is defined as a certain edge, which includes only possible and certain edges. In Figure 3.10 the hysteresis process is demonstrated: if value “2” is a “definite edge”, value “1” a “possible edge” and value “0” a “non edge” then pixel P(x,y) is transformed from “possible edge” to a “definite edge” because of the direct proximity with pixel P3.

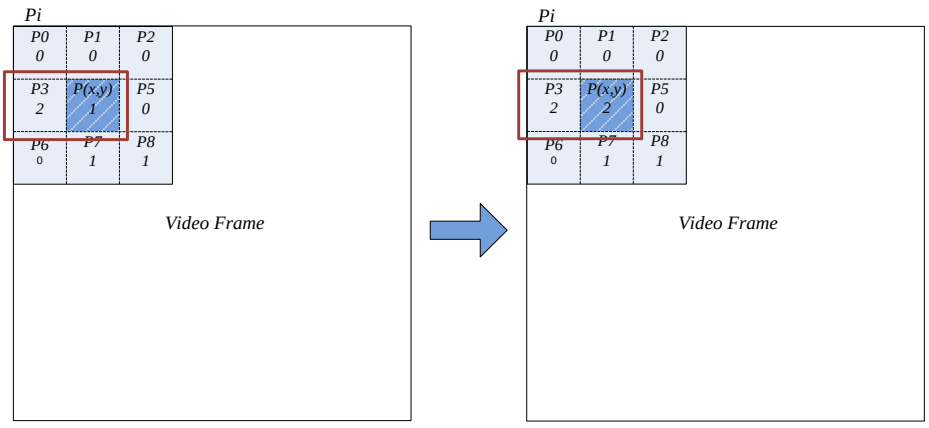


Figure 3.10. Hysteresis Process

3.7.2. Exploration

As part of the design space exploration process a bit-accurate simulation of the Canny Edge Detection process was designed in software (C code). The target was the generation of a simulation platform for verification of the implementation and parameter setting. Code::Blocks [85] open source IDE was used for the development and MinGW [86] compiler for compilation.

The target was to have a bit-accurate software implementation for the hardware architecture as well as a software program that is easily portable to other platforms, such as the MicroBlaze soft-processor to be used for further research [87]. Therefore use of external image/video processing libraries such as OpenCV was avoided. Output of every computational stage of the implementation could be grabbed and used for comparison (Figure 3.11). According to the specifications, 8 bit grayscale images are used for input.

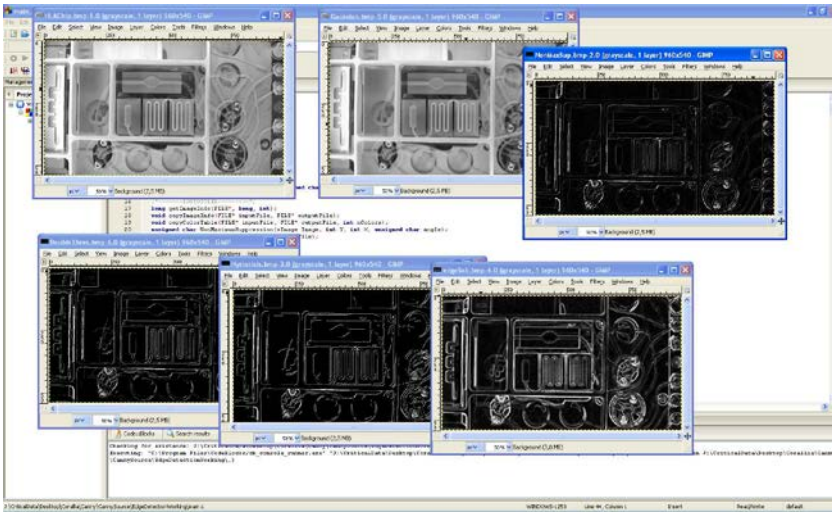


Figure 3.11. Code::Blocks IDE with the Outputs of All Canny Edge Detection Computational Stages

The absence of external libraries lead to the generation of structures and functions necessary to use .bmp files as input:

- sImage: struct {int rows; int cols; unsigned char* data;} sImage structure that stores image dimensions (columns, rows) and image data
- getImageInfo: long getImageInfo(FILE*, long, int);
  function that reads the bmp file header and stores the information in variables
- copyImageInfo: void copyImageInfo(FILE* inputFile, FILE* outputFile);
  function that copies the input header file to the output header file
- copyColorTable: void copyColorTable(FILE* inputFile, FILE* outputFile, int nColors);
  function that copies the color array of the input bmp file to the output files

In the computational stages where convolution is required a neighborhood of pixels must be used for calculating the value of the current pixel (8 when the mask size is 3 x 3, 24 when the mask size is 5 x 5 and 48 when the mask size is 7 x 7). This poses a problem when the currently calculated pixel is in such position on the image frame that a number of the required neighboring pixels are placed outside the image frame (non existent). There are two approaches in the relevant literature: either the value of the nearest pixel on the edge of the frame is used as the value of the pixels outside the image frame, either the pixels outside the image frame are considered to have value “0” (black color). For the given application using the second approach proved to be more efficient as the pixels near the edge of the image frame are almost always outside the LoC device and therefore have a very high probability to be black.

With the use of the above simulation software thorough investigation was executed for the definition of the various hardware parameters. For the demands of the machine vision system a 5 x 5 convolution mask for Gaussian smoothing was chosen as sufficient, with the option of three different standard deviation values (minimum σ=1, nominal $\sigma = 1.4$, maximum $\sigma = 2.5$) for noise level adjustability.

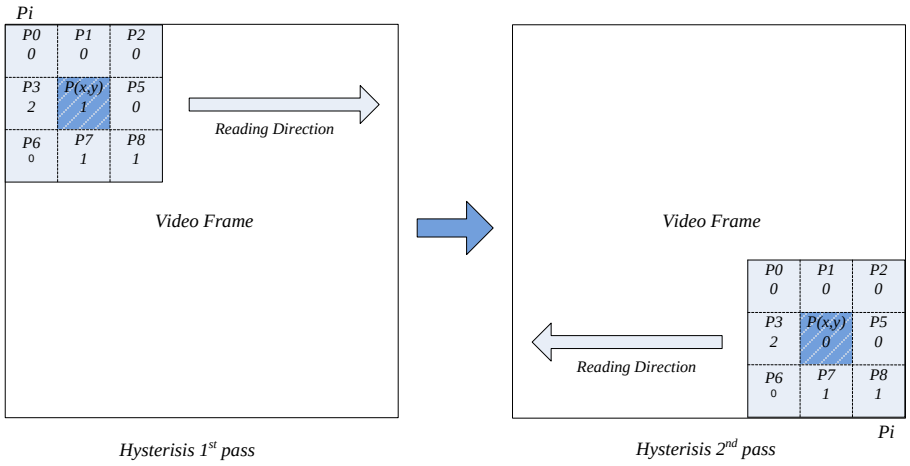
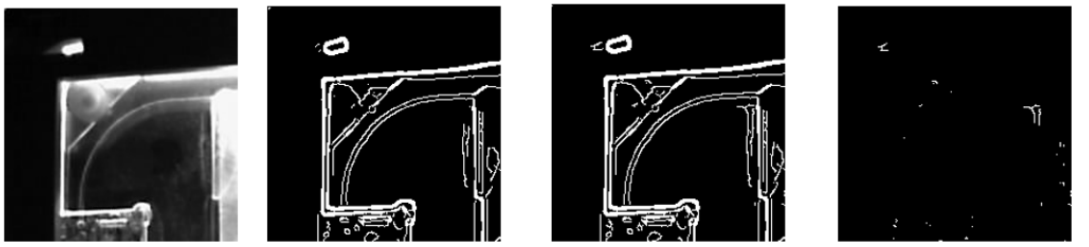


Figure 3.12. Reversing Image Frame for Second Hysteresis Pass

The original Canny Edge detection algorithm requires multiple passes of the Hysteresis stage until all "possible edges" are either suppressed or transformed to "definite edges". This approach however poses a significant problem to a possible hardware implementation: execution time is completely data dependent. This would be unacceptable for the proposed implementation with hard real-time requirements. Therefore a different approach was chosen: using two hysteresis passes on a single frame, but the second hysteresis pass would be on a reversed order of pixels (Figure 3.12). The double hysteresis pass approach was used in the preliminary version of the machine vision system [88].

After the completion of the first design version, exploration was executed for reducing used resources to allow for more space for integrating more functionality for the future. A critical value was the use of memory resources, since reversing the direction of hysteresis required buffering a whole image frame. By using a substantial number of input videos, an average MSE of 4.5 was calculated by comparing the single pass images to the double pass images. However, when the pixels of interest are reduced to the edges that are used in the calculation of the chip frame detection (the edges which result into lines near the horizontal and vertical axis of the frame) then the average MSE is reduced to less than 1. This value does not interfere with the chip frame detection stage and a single pass of the hysteresis stage

was deemed sufficient for satisfactory non-edge suppression. Figure 3.13 demonstrates Canny Edge Detection results for single and double hysteresis passes on the upper left corner of the input frame. This region includes the application critical reference point. In Figure 3.13 (d), which is the absolute difference of the two passes, it is clearly demonstrated that the resulting images of the two different implementations differ mostly in pixels unrelated to the chip frame. Consequently, all the hardware related to the second pass was removed resulting in a lighter edge detection implementation. The output of the hysteresis pass is a binary image with well defined edges, ready to be used by the chip frame detection stage.



a) Input grayscale video frame (upper left corner of the captured frame - including reference point)

b) Edge detection with single hysteresis pass

c) Edge detection with double hysteresis pass

d) Absolute difference of two passes

Figure 3.13. Canny Edge Detection Results of the Upper Left Chip Corner Area with Single and Double Hysteresis Passes and the Absolute Difference of the Two (parameters: σ=1.4, High Threshold=80, Low Threshold=40)

3.7.3. Implementation

The given specifications for the Machine Vision response time is that it requires to follow a camera at 60 fps, resulting in 16.6 ms to process each 1 Mpixel image frame. From early estimations in the design development a computational time of at least 13 ms was allowed for the iterative chip frame detection implementation [94]. Therefore it was essential to fit the edge detection step in 2 ms or less.

The performance requirements led to a 4-pixel parallel computation implementation with pipelining. The implemented parallelism also reduces the required memory accesses. The Gaussian blur stage 5 x 5 convolution requires the contents of a neighborhood of 25 pixels. For a 4-pixel parallel computation 25 x 4 =

100 pixels would be necessary. By appropriate pixel grouping (elimination of overlapped pixels) the number of pixels is reduced to 40, as demonstrated in [90]. The same principles are applied to the Sobel edge detection and Non Maximum Suppression stages. Both these stages require a neighborhood of 3 x 3 = 9 pixels for each pixel value computation. A four pixel parallel computation would require 4 x 9 = 36 pixels, but by appropriate overlapping the pixels necessary are reduced to 18 [90], [91]. The imposed parallelism leads to a maximum execution time of 1.3 ms for a 1 Mpixel image size.

3.8. Chip Frame Detection

One of the algorithmic innovations of the proposed machine vision system is the use of a Chip Frame Detection step to simplify the flow detection process (by detecting a set reference point) and to cope with the possible LoC displacements. The Chip Frame Detection identifies the smallest bounding box that encloses the LoC in the video frame by using an adaptation of the Hough transform, as described by Duda and Hart [92] and implemented by Chen et. al. [93]. The implemented version in the Machine Vision is based on the work of Voudouris et. al. in [94].

The implementation follows the parallelism decided in the Edge Detection stage as it is essential to keep pixel data throughput steady. The frame of the LoC will be a quadrilateral in the general case and what is calculated by the module is the least bounding rectangle. The outputs of the module are the dimensions and the upper left corner coordinates of this bounding box.

3.9. Flow Identification

The final and most important step in the machine vision system is the flow identification. The flow identification concept is based on a patented algorithm by Micro2Gen Ltd. [76]. Adaptations were made to make hardware implementation possible within the performance specifications. Each flow is characterized by its “head” and “tail”. The flow identification step detects the coordinates of either the head or both head and tail per flow and provides the information to the experiment’s control unit and user interface. In addition, points of interest called

“alarm points” are set by the user. When a flow’s head or tail crosses an alarm point, a signal is raised to inform the user along with an index indicating the particular alarm point. The identification of these points in a 1 Mpixel frame would be a very time consuming process if the whole frame data were to be examined. Most implementations in the relevant literature use computationally intensive image processing algorithms, such as connected components analysis to identify the blobs of the flow in the image frame. Such implementations are resource hungry and unsuitable for hardware implementations. Therefore, certain adaptations were necessary in order to minimize the computations in a limited area of interest.

The flow identification step receives information on the coordinates of the upper left corner of the chip in the video frame and the size of the chip’s bounding box from the Chip Frame Detection stage. The experimental region of interest is restricted within the boundaries of this box, therefore only data within this area is required from the memory to detect the flows. Thus, the number of memory accesses necessary is reduced. The initial coordinates or points of entry (by reference to the chip’s upper left corner) of each flow are also provided by the experiment’s input file, together with the alarm points’ coordinates. Therefore the areas around these coordinates are defined as regions of interest, since a flow can only appear near these coordinates (by allowing some error margin). A detection “window” is defined around these coordinates which is a square within which a change in fluid position is expected. The size of a detection “window” is user adjustable with a maximum of 32 x 32 pixels. Only the area within these windows needs to be examined for flow detection and henceforth system performance is increased and area requirements are decreased. It is assumed that the flow’s tail doesn’t enter the chip (initial coordinate) until the head has moved at a distance at least equal to the window dimension in order to avoid conflicts from overlapping windows. Since the channels are investigated from a distance of 10 to 20 cm with a normal lens and have a width down to 200 μm, when a region of up to 32 x 32 is used, it is not expected a fluidic pluck to be smaller than this size. This detection window size can detect flow speed of up to 20 mm/sec in this setup. The detection windows are defined by the coordinates (x,y) of their upper left corner and their dimension (width and height). The window dimension is a function of the fluid’s

speed, the minimum microfluidic channel distance and the maximum channel width. The dimension is a user adjustable parameter, in order to achieve maximum precision.

At the start of every frame the window data is loaded in two internal RAMs, separate for heads and tails. For flow positions near the chip edges the detection window is adjusted to a rectangle shape and gets cropped. The pixel data is stored to the internal memory and simultaneously compared to the data from the previous frame. The absolute difference of the two pixel values is compared to a threshold value, and results to a 1bit output ("active pixel") indicating if there was a significant difference. In Figure 3.14 an example of data for such a detection window can be seen. The pixel data in d) is calculated from the absolute difference of two windows of the same size captured around a flow front in two subsequent frames (Figure 3.14 b) and c)). If the total number of "active pixels" exceeds another set threshold value called "binary threshold" the machine vision system assumes that there was a change in flow position (absolute difference threshold and binary threshold are both user set parameters for detection sensitivity). The crescent shape of the flow front and the luminance changes in the flow result in a distribution of binary pixels around the actual flow front coordinates. Two approaches were used to identify the flow front position: the first one was to calculate the barycenter (center of mass) of the binary pixels. The barycenter $\boldsymbol{r}_c$ is defined by:

$$\boldsymbol{r_c} = \frac{1}{N_{wp}} \cdot \sum_{N_{wp}-1}^{0} \boldsymbol{r_i},$$

Equation 3.3. Barycenter Calculation

Where $\boldsymbol{r}_i$ the coordinates of each "active pixel", $N_{wp} = \sum_{N-1}^{0} p_i$, $p_i = \begin{cases} 0 & \text{inactive pixel} \\ 1 & \text{active pixel} \end{cases}$, N_{wp} the total number of active pixels and N the total number of pixels in the window.

The second approach was to calculate the median value of the active pixels within the detection window (3.9.2).

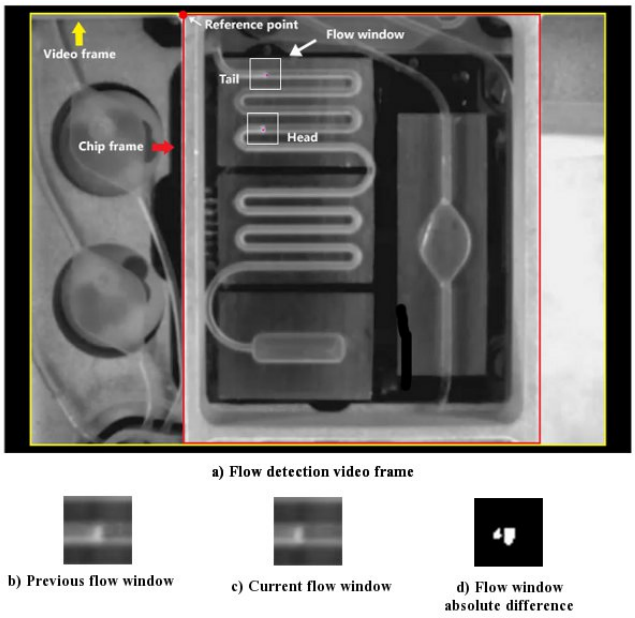


Figure 3.14. Example Video Frame of the Machine Vision System

If a flow movement was detected then the detection window is readjusted around the new coordinates. The data from the new windows are fetched and stored in a separate internal memory called previous frame memory. If a flow wasn't detected then no data needs to be fetched. The current frame data is dropped and the next frame data will be compared to the data from two frames back already stored in the previous frame data memory. The same procedure is repeated until a flow movement is detected. When a flow movement is detected, the new flow coordinates are compared with the alarm point coordinates. If the flow is within a Euclidian distance approximation (a user defined parameter) of one alarm point then an alarm indicator is raised, together with the alarm point identification index number. A detailed flowchart of the operation can be found in [96].

Following the principle of the whole machine vision design, flow identification also uses a four pixel parallel computation with a fully pipelined implementation.

The experimental specifications are covered with five individual flow identifications (maximum possible number of flows in the microfluidic chips for the experimental protocol). The design however is modular, thus allowing the increase of flow number identification to be a trade-off with area and memory usage. Currently the experimental protocol doesn't detect bubbles in the fluids. If bubbles are to be included in the detection protocol as separate entities then the only adjustment necessary is that the algorithm should anticipate more flow "heads" at the same coordinates after the "tail" has entered the chip, which can be easily implemented with an increase in the internal RAM size. Bubble formation is common in microfluidic microsystems and their detection may be used as a qualitative criterion for following microsystem design optimizations. The same principle could be applied to droplet based microfluidic experiments.

3.9.1. Center of Mass Calculation Module

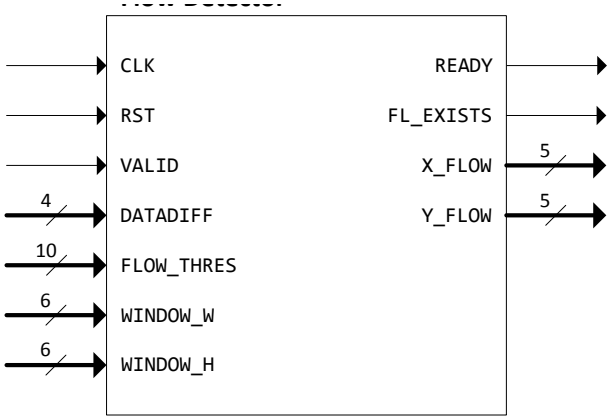


Figure 3.15. Center of Mass Module Interface

The center of mass calculation module is a generic module used to calculate the barycenter of the detected menisci of the flows. The external interface of the module can be seen in Figure 3.15. The module accepts the "active pixels" of the detection window as an input. The detection window size is generic and it is also an input parameter for the module. Detection window size can change on runtime.

The output of the module is the set of coordinates for the canter of mass of the active pixels in the window. However, the center of mass is only calculated if the

number of “active pixels” in the window is above a threshold value (“FLOW_THRES”). This value can also change on runtime.

The active pixels are read sequentially from the module, from the upper left corner of the detection window to the bottom left corner. In order to recover the coordinates of each active pixel a generic counter is used. In the machine vision implementation the pixels are read in groups of four, keeping up with the 4x parallelism implemented in the system. However, the degree of parallelism in the module is generic. There are three accumulators in the module:

- Active Pixels Accumulator: The active pixels accumulator counts the total number of active pixels in the detection window. The value from this accumulator will be compared to the threshold value to decide whether there is a flow or not in the detection window.
- X/Y Coordinates Accumulators: Each coordinate accumulator contains the value of the accumulated coordinates in the X and Y axis respectively, These values will be used for the center of mass calculation.

When all the pixels from the window are read the active pixel total value from the Active Pixel Accumulator is compared to the threshold value. If the value is greater than the threshold then the center of mass must be calculated.

The center of mass is calculated using Equation 3.3. A Xilinx Fixed Point Divider IP Core is used [95] to execute the division process. A Radix-2 implementation was used with a fractional remainder. The implementation requires one DSP slice from the FPGA. Two divisions are required for the coordinate calculation, but they are executed in a pipeline from the same divider. The first result requires 19 clock cycles for calculation. The second result requires only one additional clock cycle. Both results are stored in two output registers until next iteration. A block diagram of the Center of Mass Calculation module is presented in Figure 3.16.

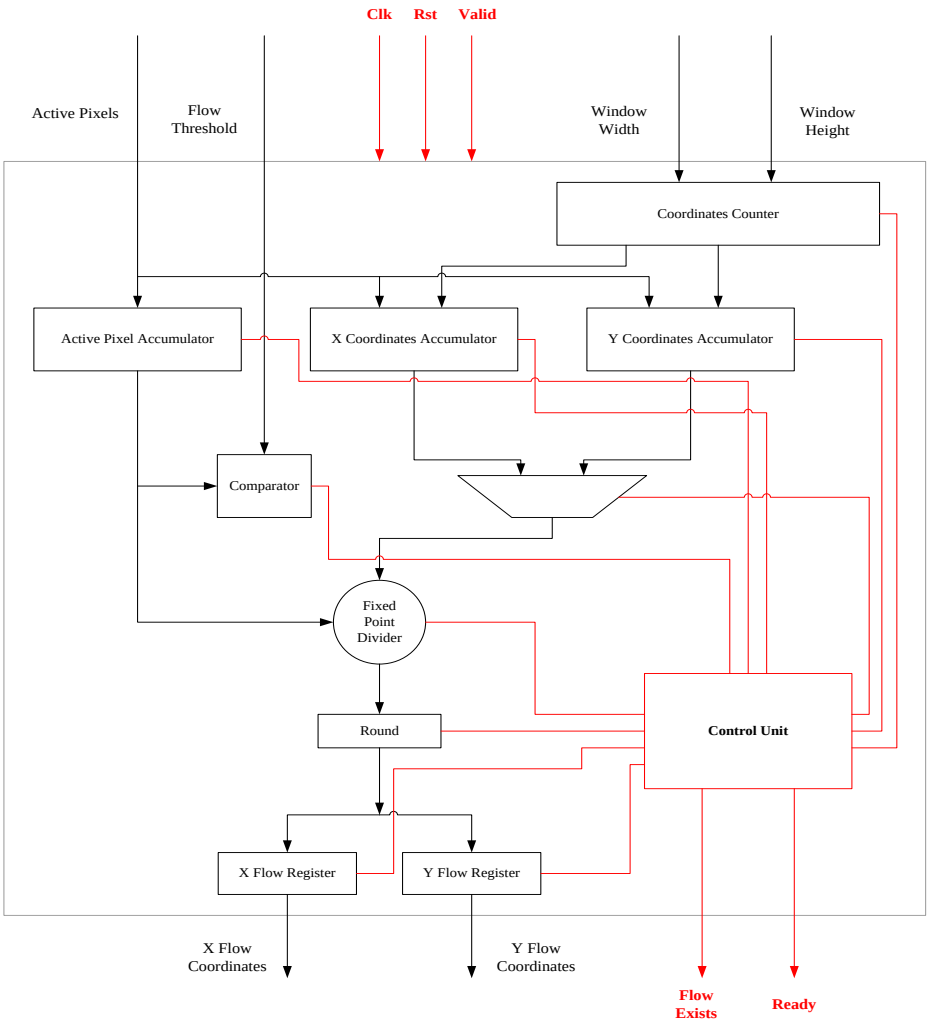


Figure 3.16. Center of Mass Module Block Diagram

The total number of clock cycles required for the center of mass calculation is a function of the window size and the active pixel parallelism:

$$N_{cycles} = \frac{(W \bullet H)}{N_{par}} + 22$$

Equation 3.4. Centroid Calculation Module Execution cycles

were W and H are the pixel width and height of the detection window respectively and N_{par} the number of parallelism in the active pixel input (defined by the parallelism of the whole flow detection module). The 22 extra clock cycles are the

total number of cycles required for the divisions, the comparisons and register buffering. For a 32 x 32 window with 4 pixel parallelism, 278 clock cycles are required to calculate the center of mass coordinates.

3.9.2. Median Calculation

The second approach used to calculate the flow front position in the machine vision system is a median calculation module. The most common use of median calculations is in the form of an image processing filter. Another use of the median in image processing is to find location coordinates from a set of pixels. In this case the operation is relatively different than when used as a filter. In a normal median filter the sequence of the input data is irrelevant to the result and only the values are important. These values need to be shorted and the median value from the set is the filter output. In a median localization module the input pixels have binary values, discriminating from “active” (‘1’) to background (‘0’). The sequence of the incoming data is of outmost importance as the median of the coordinates (the input data index) is the actual module output. In the case of 2D median calculation the pixels arrive in the form of a 2D detection window. In Figure 3.17 such a window of 8 x 8 pixel size is displayed. In this example the active pixels are the ones with the orange background color. In order to calculate the median coordinates the total number of active pixels needs to be known (11 in the example) and also the number of pixels in each x and y coordinate. The median coordinates are the coordinates where the total number of active pixels that exists up to (including) this coordinate is greater than or equal to half of the total active pixels. In the example the median coordinates are (3,2) for pixel 19.

In the implementation used for the machine vision systems the input pixels arrive in batches of four pixels as streaming data for a median detection window with generic size and a maximum size of 32 x 32 pixels (as show in Figure 3.14. The median coordinates are calculated by reference to the upper left corner of each detection window.

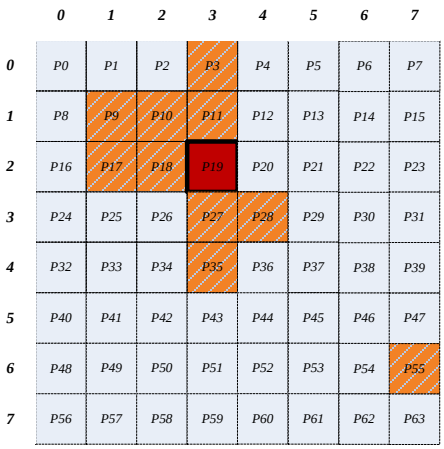


Figure 3.17. 8 x 8 Median Detection Window

3.9.2.1. **Implementation**

As described in Section 3.9.2 the median module has three processing steps: a) identification of the incoming data index (coordinates), b) accumulation of the active pixels of each coordinate, and c) calculation of the 2D median value.

A. ***Coordinates Counter***

The incoming pixel data come as a binary stream. In order to get the coordinates information a counter is needed which will provide the index of each pixel. A special double counter with user defined upper limits was designed. The two upper limits of the counter are the sizes of the detection window. The four pixel parallelism of the design dictates that the x coordinate upper limit is a multiple of number four. The counter for the x coordinates counts the number of four pixel batches on each line and when a window line is completed the y counter is increased by one. The counter outputs are used as inputs for the two following computational steps.

B. ***Pixel Accumulators***

The most complicated part of the design is the pixel accumulators. In order to achieve a more generic and versatile design the accumulated values are stored in memories. Each memory address is assigned the accumulated value of one

coordinate. The memories used are dual port memories with a write enable and they are initialized by being written with ‘0’ to all addresses.

In the case of the pixel accumulator for x coordinates (presented in Figure 3.18) four separate dual port memories are used. Each memory has an address space equal to the x dimension of the detection window divided by four and a word size of $\log_2$(size_x) bits. The first memory is assigned the CoorX coordinates, the second the CoorX+1, the third the CoorX+2 and the fourth the CoorX+3. CoorX is the x output of the coordinates counter. This means that the next address of each memory is only incremented by 1 as an address, but corresponds to an incrementation by 4 in the actual coordinates. Each active pixel acts as a write enable signal for the corresponding memory. The CoorX signal arrives as a read address for the memories. In the next clock cycle the read address is read out from the memory. If the input pixel is active then the output value from the memory is incremented by one and in the next clock cycle is written in the same memory address CoorX, which now serves as a write address. The write enable signals are carefully synchronized with the write address by using flip flops. In this way the accumulator is fully pipelined.

The operation of the pixel accumulator for y coordinates is similar (presented in Figure 3.19). In this case only one memory is used. The memory has an address space equal to the y dimension of the detection window and a word size of $\log_2$(size_y) bits. In this case each bunch of pixels belongs to the same y coordinate. Therefore the write enable signal is the OR value of all four pixels. The memory is read at the read address (CoorY) and the output is incremented by the sum of the four input pixels. In addition there is an extra register which is used to store the accumulated value of all active pixels. The value is necessary for the median calculation in the next step.

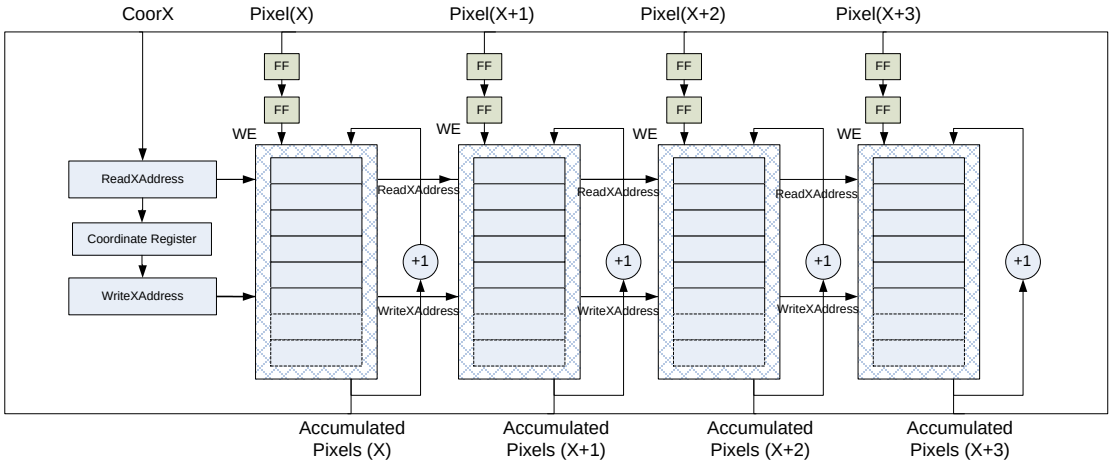


Figure 3.18. Pixel Accumulator for X Coordinates

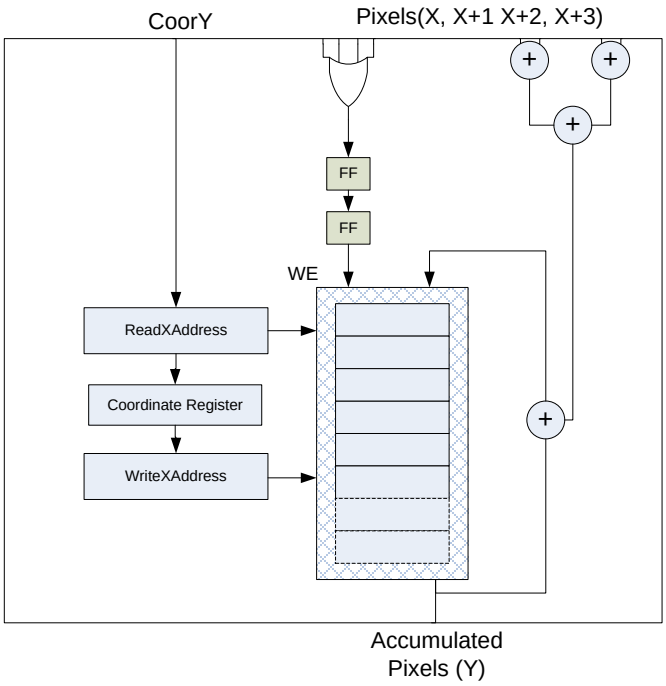


Figure 3.19. Pixel Accumulator for Y Coordinates

C. *Median Calculation*

When all the input pixels form the detection window are read, the median calculation can begin. The total number of active pixels is calculated in the previous step. The median coordinates are those where the accumulated number of active pixels up to (including) this coordinate is equal to half the total active pixels number.

The accumulators are read from the top left corner to the right and from top to bottom (starting from coordinates (0,0) and incrementing step by step). The coordinates counter is reset and the pixel accumulators for x and y coordinates are read out word by word. The outputs of each accumulator are added and stored into two separate registers, one for x coordinates and one for y coordinates. Each time a word is read out and added to the register, the register value is compared to the median threshold (total_active_pixels/2). If the value is greater than or equal to the median threshold then the current coordinate is the median coordinate. If not the process continues. It must be noted that the x and y median coordinates calculation are completely independent, therefore one coordinate is usually identified before the other and the two processes run without interfering.

When the process is completed the median coordinates are sent out of the module with a valid signal. It is necessary to initialize the memories before another median calculation can begin. For this reason the coordinates counter is reset once more to be used for addressing so as to fill the memories with '0'.

The total number of cycles necessary to complete a median calculation is data dependent, but the maximum number of cycles for a defined detection window size can be calculated. For the memory initialization the coordinates counter counts the x and y coordinates independently, therefore the total number of cycles required is the maximum value of (size_x / 4, size_y). The total number of cycles required to read the input data is equal to the number of four pixel bunches that arrive in the module (size_x * size_y) / 4. The maximum number of cycles required for the median calculation would be necessary if all the pixel accumulators had to be read out to identify the median coordinates. In this worst case scenario the maximum pixel cycles required are equal to the maximum dimension of the detection window. A total of 8 cycles must be added for reset and register synchronization purposes. Therefore if N_{Cycles} are the total execution cycles of the median module, the maximum execution cycles required are:

$$\max(N_{Cycles}) = 8 + \max(size\_x/4, size\_y) + \\ +(size\_x \cdot size\_y)/4 + \max(size\_x, size\_y)$$

Equation 3.5. Median Calculation Maximum Execution Cycles

It must be noted that the maximum size of the median detection window which can be used is defined by the maximum size of the accumulation memories. The implementation is fully generic for the window size as long as the x dimension is a multiple of four.

D. ***Exploration***

The 4x parallelism of the implementation was defined by the machine vision system. The module is easily adjustable to a bigger parallelism in the calculations. In this case the maximum number of execution cycles will be:

$$\max(N_{Cycles,Par}) = 8 + \max(size\_x / N_{par}, size\_y) + \\ +(size\_x \cdot size\_y) / N_{par} + \max(size\_x, size\_y)$$

Equation 3.6. Median Calculation Maximum Execution Cycles for Generic Parallelism

Where N_{par} is the parallelism at the input. The used memory size will not change, what will change is the number of memory banks the memory will be divided for the X coordinate calculation. Instead of having four memories, N_{par} memories will be implemented and the last one will be assigned to coordinate CoorX+ N_{par} -1.

3.9.2.2. **Results**

The median module was implemented on the Xilinx Spartan 6 LX150T FPGA device used for the machine vision system. As an independent module the design achieved a clock frequency of 204MHz. For the implementation used in our machine vision application and a maximum detection window of size 32 x 32, a maximum of 328 clock cycles is necessary to calculate the median coordinates of each flow front. In the machine vision implementation a clock of 170 MHz was used, which led to an execution time of 1.9 μsec for each detection window.

In TABLE 3.1 the implementation results of the median module are presented. In the current implementation the accumulation memories were

implemented in distributed memory because of their small size (slices used as memory in the table below). In case a large detection window is used (e.g. 128 x 128 pixels) the use of block RAMs is suggested. As can be seen the median module occupies a very small area. With such a small area and small execution time the median calculation modules can be implemented and used in parallel.

In terms of performance improvement in the flow identification process, an example by using Figure 3.17 can be demonstrated. In Figure 3.17 as can be seen the active pixels are gathered in the upper left corner of the detection window, apart from a single one on the bottom right corner. In the current case of 8 x 8 detection window the calculation of the center of mass and median coordinates does not differ significantly. But if a much larger window is used (e.g. 32 x 32) then the median calculation is more accurate as the isolated pixels are very likely to be noise. The impact of such a change is especially significant in cases of fragmented flow fronts. In Figure 3.14 d) an example of a slightly fragmented flow front can be seen as captured by the implementations camera and processed by the machine vision system.

TABLE 3.1. Median Module Implementation Results

Slice Logic Utilization			
Slice	***Number***	***Total***	***Used(%)***
Registers	215	184304	~0.1%
LUTs	307	92152	~0.3%
Used as memory	20	21680	~0.1%

3.9.3. Alarm Point Detection Module

The alarm point detection module compares the detected flow coordinates with a set of predefined alarm point coordinates. If the flow is within a predefined distance of the alarm point an alarm signal is set and the alarm point index is sent out. The alarm point detection module interface is presented in Figure 3.20. The module accepts as an input the flow coordinates, the alarm point coordinates, the

predefined distance that will set the alarm (“ADM”) and the alarm point index number.

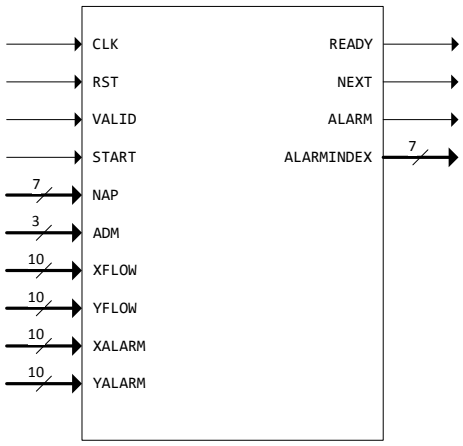


Figure 3.20. Alarm Point Detection Module Interface

The module reads the coordinates of the flow and then compares them to the list of alarm points that were set before run time in the module. If the flow is within the ADM defined distance of the current alarm point, operation is stopped and the alarm signal is set. Operation resumes when the next flow is identified. The alarm point detection module block diagram is show below (Figure 3.21).

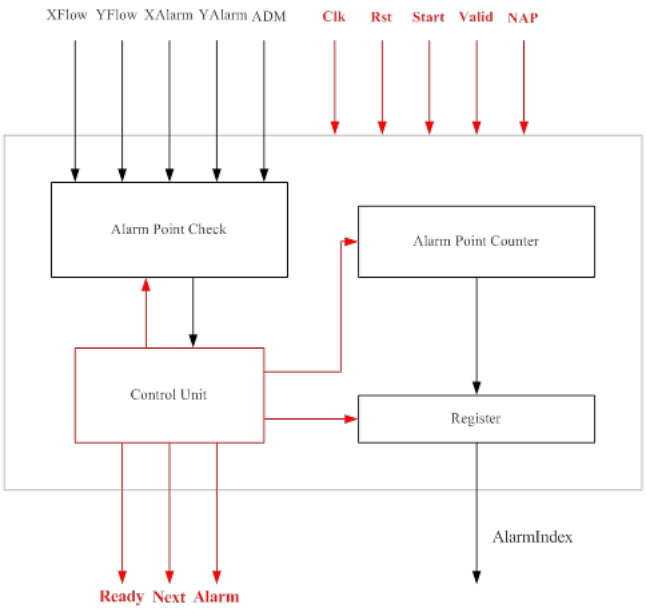


Figure 3.21. Alarm Point Detection Module Block Diagram

3.10. System Core

The system is implemented on a Xilinx Spartan 6 lx150t FPGA [97]. Special attention has been paid on exploiting the full potential of the FPGA resources available. The complexity of the complete experimental setup allows access to only a single port from the Multi-port Memory Controller. So as to take full advantage of the available port the machine vision system communicates with the external memory using a Video Frame Buffer Controller (VFBC) [98], which is the optimum setup for 2D streaming video data bursts. The memory word is set to 32 bits, which accommodates the 4 pixels parallelism requirements (8 bits x 4). A block diagram of the machine vision implementation is presented in Figure 3.22.

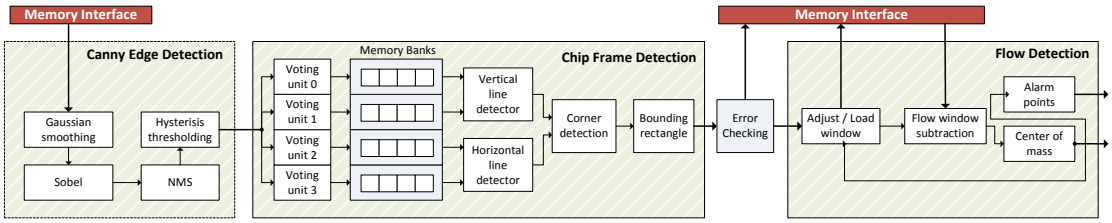


Figure 3.22.Machine Vision Implementation

The machine vision system is implemented in a Xilinx pcore, which is attached as user logic on a MicroBlaze processor via a PLB bus. The MicroBlaze acts as a central control module for the systems integrated on the same Spartan-6 device. All the user parameters (threshold values etc.) can be adjusted at runtime, thus making the design very versatile. The machine vision system is now integrated in the actual board with the complete system (system controls, bio-sensor controls, video compression, camera link and frame grabber) integrated as well. Therefore it the machine vision core as multicore implementation is also integrated into a complex multiprocessing system. The system achieves a maximum frequency of 173 Mhz after integration and place and route.

3.11. Results

The implementation results of the machine vision system (after place and route) on the Spartan-6 lx150t device are presented in TABLE 3.2. These results take into account the implementation of the system with the centroid calculation module for flow detection. However, the impact of changing the center of mass calculation module to the median calculation module is very small in the system area (less than ~0.3% of the total occupied area). Additionally, while center of mass calculation requires a fixed number of execution cycles, median calculation is data dependent, and on average around the same processing time is required for 1 Mpixel video resolution. The most fundamental features of the FPGA fabric (flip-flops, LUTs, DSP slices and BRAMs) are used as metrics for size. The results are shown for each separate stage of the machine vision system, for the complete machine vision system and for the integrated system on the device. The integrated system utilizes 10% of the FPGAs FFs, 17% of the LUTs and 64% of the BRAMs. The 18 Kbit BRAMs are used as 2 KB BRAMs, leading to a memory requirement of about 86 KB for the machine vision system and 356 KB for the complete prototype on the device. Apart from the design's internal memory requirements, an external memory space of 2 MB is needed to store two video frames from the frame buffer (1 MB per frame). Using the Xilinx XPower Analyzer [99] the power of the machine vision system was estimated at 0.33 W and for the integrated prototype at 0.9 W. These estimations were made using the worst case scenario for activity and signal toggle rates. The cost of the device used for development is about $240, but the design, after optimizations, can now be fitted to the Spartan-6 lx75 device which has enough BRAMs and a price of $90.

TABLE 3.2. Machine Vision Implementation Results

	FF	LUT	BRAM	DSP	Time (ms)
Canny Edge Detection	7003	3053	13	-	1.55
Chip Frame Detection	3772	7563	24	28	12.34
Flow Detection	1581	1447	6	1	$77 \bullet 10^{-3}$
Machine Vision	12356	12063	43	29	13.97
LoC System	16094	16396	178	32	-
FF: Flip Flops, **LUT**: Look-Up-Tables, **BRAM**: 18Kbit Block RAM blocks, **DSP**: DSP48A1 slices					

An experimental prototype was developed to test integrated system together with the machine vision system. The camera used was a JAI CV-M71 CL Camlink Camera [100]. The set specifications for the system require real-time response for a 1 Mpixel video frame at 60 fps. These specifications allow 16.7 ms computation time per frame. The prototype system is set to an operating frequency of 170 Mhz. With this operational frequency the system completes one frame computation of 1 Mpixel resolution in 13.97 ms (as presented in TABLE 3.2), which exceeds the given specifications. Time was calculated for five concurrent flows identification, maximum window detection dimensions and maximum number of alarm points. This performance leads to a 71.6 fps throughput for 1Mpixel resolution with the worst case scenario parameters chosen. Similar performance could also be achieved by using a GPU implementation of the algorithm. This solution was rejected, however, because it is not a system-on-chip solution for the complete system, it has great power consumption and it requires expensive hardware and a host pc to run. A DSP solution again does not offer SoC capabilities and devices that could possibly achieve acceptable performance are high end high cost. In TABLE 3.3 performances of different implementations can be seen in comparison to our system. As this is a unique machine vision implementation for continuous flow systems a straightforward comparison cannot be done. This is why image processing implementations of the algorithms used in the different stages of the machine vision system in different platforms have been added in order to make partial but valuable comparisons.

It must be noted that the proposed design is generic and therefore a simple change in parameters can double the design's parallelism thus reducing the execution time in half and doubling the effective frame rate of the system. Versatility, integration and performance were the key factors for the proposed system and as a result it can be safely assumed that the FPGA implementation is the optimum solution.

TABLE 3.3. Performance Comparison of Different Implementations

Implementation	Platform	Algorithm	Frame Resolution (pixels)	Exec. Time/Frame (ms)	
Our Continuous Flow Machine Vision System	Spartan 6 FPGA	Canny Edge Detection	1024x1024	1.55	Total:13.94
		Hough Transform		12.34	
		Flow Identification		$77 \bullet 10^{-3}$	
Sappupo et al. [69]	AnaFocus ACE16kv2 FPP /Altera Nios II	CNN based implementation	128x128	90μs	
Demiris, Blionas [76]	CPU	Flow Detection	640x480	more than 35ms	
Texas Inst. [101]	DSP	Canny Edge Detection	1024x1024	18.5	
Ogawa, Ito, Nakano [102]	GPU	Canny Edge Detection	1024x1024	444.29	
Li, Jiang, Fan [103]	Virtex-5 FPGA	Canny Edge Detection	512x512	5.24	
Khan et al. [104]	DSP	Hough Transform	320x240	12	
Van Der Braak et al. [105]	GPU	Hough Transform	1920x1080	10.6	

In terms of quality the Chip Frame Detection stage error was calculated at ± 2 pixels for X axis and ± 2 for Y axis. These error values are produced by the small quantization errors induced by the Chip Frame calculations. This small displacement of the reference point is compensated by the detection window size and the only influence to the performance is that for the worst case displacement the maximum flow speed for the first frame cannot exceed 15mm/sec, which is rarely the case. In the Flow Identification stage the inherent quantization error is just half a pixel due to the rounding used in the result of the center of mass division process. For the median calculation module there is no quantization error. Since the coordinates produced are those of the center of mass of the fluid's movement, these coordinates differ slightly from the actual flow front. The difference is smaller for the median calculation module. This difference is directly proportional to the fluid's speed and it is 8 pixels for the maximum speed (20mm/sec) for the center of mass module. The on-board MicroBlaze processor which controls the machine vision system compensates for this difference by adjusting the coordinates in both X and Y-axis in relation to the current flow speed. For the median calculation module the theoretical maximum difference of the flow front with the calculated coordinates is the same as with the center of mass, but the observed difference was by a factor of two smaller. The alarm point detection is not influenced by this process because the alarm point

detection distance is a user defined parameter and can be appropriately adjusted with regards to the fluidic speed in the experiment. This adjustment increases the error to ± 1 pixel per axis or $\pm\sqrt{2}$ pixels on the plane. The detection error has been confirmed by observational data even on experiments where vibration was induced on the surface where the experimental prototype was placed.

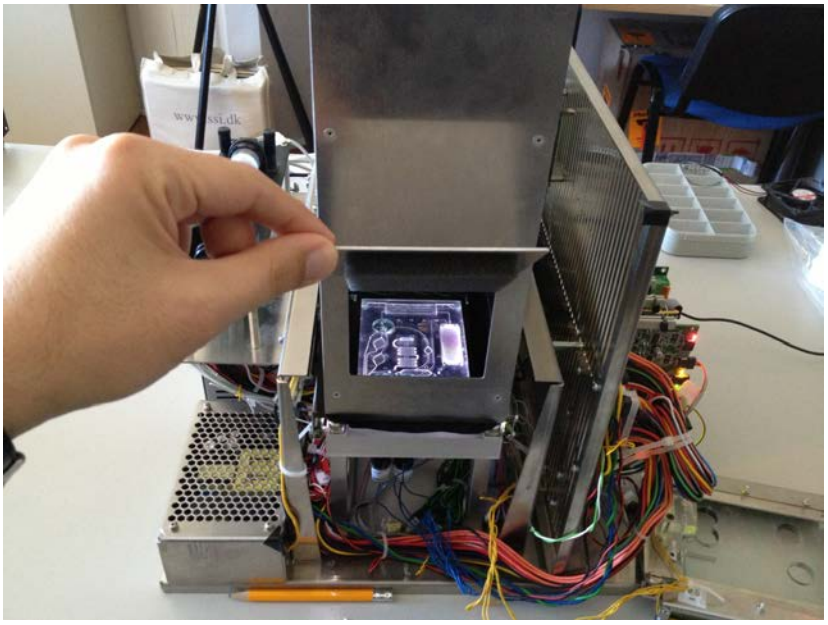


Figure 3.23. Machine Vision Prototype Setup

The camera used has a maximum resolution 767 x 576 pixels at 60fps. In the experimental prototype the camera is placed at a distance of 10-15cm directly above the LoC (Figure 3.23). The machine vision system's four pixel parallel computation design requires that the frame resolution has a number of pixels per line that is a multiple of four. Therefore the 767 x 576 pixel frame is transformed to a 768 x 576 pixel frame by adding an extra black pixel at the end of each line in a DMA engine that controls the data from the frame grabber. With this video resolution the system requires less than 6ms to complete one frame computation.

Prior to the machine vision initialization all the parameter values are loaded in a register file, which is accessible by the MicroBlaze processor and the Machine Vision system. These parameters are divided into two categories: a) those that are loaded only once and are stable during the experimental process (such as frame size,

detection window size, number of flows etc.) and b) those which are reloaded before every frame computation and therefore can be changed at runtime for extra flexibility (such as edge detection thresholds, frame detection thresholds, flow detection thresholds and alarm point detection error margin).

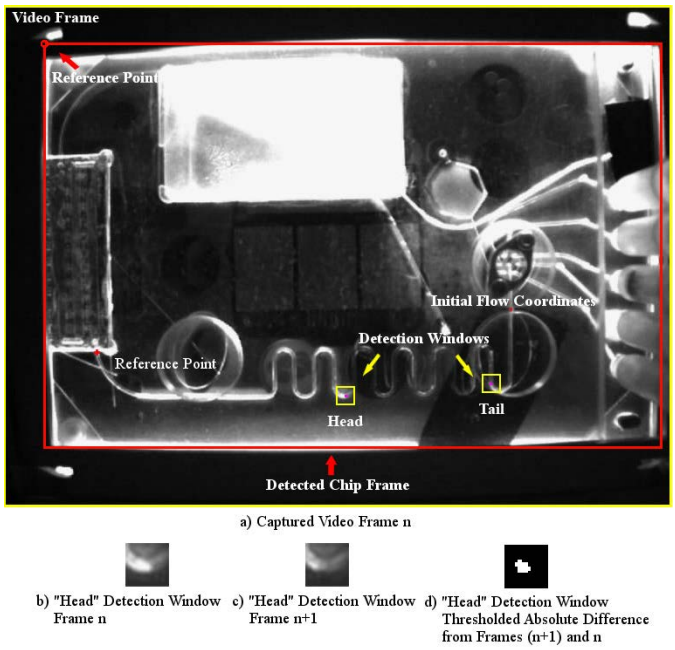


Figure 3.24. Machine Vision Operation

In Figure 3.24 a frame of the Machine Vision system operation is shown. Even though the camera lens imposes distortion on the chip frame shape, the Chip Frame Detection stage can identify the appropriate bounding box. The reference point is the upper left corner of the defined bounding box. The initial coordinates of the flow to be detected are defined by the user on the user interface. In this case the point of entry is defined as the exit from the input valve on the right. The initial coordinates are defined by reference to the upper left corner of the chip. The Machine Vision system defines the first flow detection window around the initial coordinates adjusted to the reference point calculated by the Chip Frame Detection stage. The snapshot chosen for demonstration is one where both the flow's head and tail are within the chip frame. Flows with and without coloring agents can be identified and a flow without coloring agent has been used in this demonstration.

In Figure 3.24 b) and c) the detection window data for two consecutive frames for the flow's "head" is shown. In Figure 3.24 d) the absolute difference of the two detection windows (with an absolute difference threshold value 40) is presented as a binary output. From this binary output the barycenter is calculated by using formula (1) (white pixels are the active pixels), thus producing the flow's "head" coordinates. The same procedure is followed for the flow's "tail". When the flow reaches a predefined area around the user set alarm point, then the alarm signal is raised. In Figure 3.25 two captured video frames have been isolated and subtracted. The grayscale image difference between the two captured frames is presented with a magnified view of the upper left corner. It can be seen that there has been a displacement of the chip during the video capture due to changes in lighting conditions and micromovements due to external pressure on the chip, or even vibrations on the experimental prototype's supporting surface. This change in position is compensated by recalculating the position of the chip frame in every single video frame. Thus, the detection window of every flow is correctly placed on the video frame adjusted to the reference point calculated on every frame. The Lab-On-Chip in Figure 3.24 requires single flow identification, however, there are Labs-On-Chip which require support for identification of multiple flows (up to five).

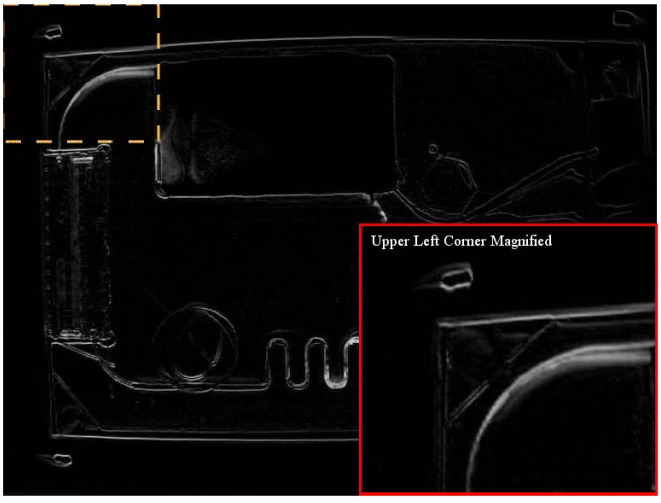


Figure 3.25. Chip Frame Movement Example During Video Capture

3.12. Conclusions

In this chapter the work on a high performance real-time machine vision implementation for an FPGA device is presented. The system uses a parallel architecture and is highly optimized. The implementation targets monitoring of microfluidic Lab-on-Chip experiments. This machine vision system is designed to follow up to five homogenous flows where the menisci of the fluids are always visible. The system is integrated with a video frame grabber, a video compression unit, bio-sensor control units and system control units on a Spartan-6 lx150t device. This device will be part of a complete Point-of-Care system, which will be portable and able to operate in non-ideal laboratory conditions. The given specifications require a real-time response for 1Mpixel input video resolution at 60 fps.

The machine vision system can compensate for changes in lighting conditions and even small LoC displacements by using an innovative Chip Frame Detection module that identifies the bounding box of the LoC on each video frame and subsequently calculating the flow coordinates with respect to the bounding box's upper left corner. The architecture exploits parallelism in every processing step.

For this machine vision system a design space exploration was executed to define the specifications and the characteristics of the whole architecture and in particular of the edge detection preprocessing step. Specific high performance modules were developed for center of mass calculation to define the flow front coordinates, as well an application specific alarm point detection module. Both these modules use generic design parameters to be easily adaptable to new specification and different application field. The exploit a 4x parallelism like the machine vision system, but the parallelism level can be adjusted.

In addition a median FPGA implementation is presented as an improvement for the center of mass module. The module is designed for localization purposes and it is used to identify the median coordinates of the flow fronts. It is a fully pipelined implementation, exploiting the parallelization capabilities offered by the FPGA devices. The module can use a detection window of generic size and in order to achieve the median calculation two unique coordinate accumulators were designed. The module is lightweight in both area and execution time, needing only a maximum

of 328 clock cycles to achieve a median calculation for a 32 x 32 pixel window. On a Xilinx Spartan 6 LX150T FPGA it achieved a clock frequency of 204MHz.

The system implementation performance is presented and compared with previous machine vision implementations on various platforms. Ours achieves significantly faster performance while the competition is slower and uses much lower resolutions for reference.

Future work will include the adaptation of the center of mass and median calculation modules for use in different image processing applications, such as the one presented in Chapter 4. Additionally, different approaches for microfluidic flow detection can be explored, such as the use of hardware implementations of segmentation algorithms or 2D clustering algorithms. Exploration can be executed for adjusting the machine vision system to different types of LoC devices and experimental protocols, such as droplet based experiments, microarray LoCs and others.

Chapter 4

High Performance MPSoC for High Data Throughput Applications

4.1. Introduction

Modern detector technology has advanced substantially in the recent years providing detectors with a very high resolution and flexibility. Thus they can be used in various highly demanding application domains. High resolution pixel detectors are used in a vast variety of applications from simple every day use to the demanding high energy physics, astrophysics, security and biomedical applications. The majority of these applications call for real-time data capture at high input rates.

For most of these applications complex image processing algorithms are used to process the captured data. Therefore, the demand for effective data reduction techniques is prominent. Data reduction techniques commonly used in this kind of applications are edge detection and data clustering. A design space exploration for the implementation of such algorithms is essential to achieve the optimum implementation for each different application case. The edge detection approach to data reduction was introduced in Chapter 3. Here, the clustering approach is examined as a commonly used algorithm to group two dimensional (2D) data (such as pixels) into clusters, thus minimizing the information of the data group to a single data word information for each cluster.

4.1.1. Thesis Contribution

To tackle the demanding data reduction problem for 2D images a novel pixel clustering implementation was developed. The presented implementation uses a moving window technique which is common to many image processing algorithms but unique to clustering implementations of this kind. The target was to design an efficient and robust high performance implementation that is generic enough to be used in different image processing applications. The 2D pixel clustering implementation was used as a platform to explore the design space for high data throughput streaming applications.

The generic characteristics of the design allowed for multiple parallelism were the number of working parallel engines can change by means of a simple variable number. The design exploits data and task level parallelism where possible and also pipelining where necessary.

The implementation was originally developed for the Fast TracKer Processor for the ATLAS detector in CERN but it can be easily adapted to be used in various different image processing applications.

The results of this thesis have been published in IEEE Transactions on Nuclear Science [106], and were presented in seven conferences ([107]-[113]) where they were included in the proceedings. Additionally, this work has been presented in the International Conference on Technology and Instrumentation in Particle Physics (TIPP 2014), where it received the award for best presentation (first prize) by a young researcher. More recently the work has been presented in the IEEE NSS/MIC conference (November 2014). Additionally, the research on this specific task led to the qualification of the researcher as an ATLAS author.

4.2. Related Work

The need for defining, determining and extracting meaningful information from large amounts of data is not restricted to image processing, but expands in several different and more general disciplines, such as science, engineering, economic studies etc. The principle behind clustering is to divide data into large

groups and to extract common features from each group. These common features are the useful extracted information.

Several clustering algorithms exist in the bibliography. A review of most clustering algorithms is presented in [114] by Casagrande et al. They are iterative algorithms where data are clustered with respect to an objective function. These algorithms are suitable for data mining and statistics use but they are too complicated to be implemented in hardware for high data throughput applications and thus unsuitable for hardware design space exploration. Hussain [121] presents a K-means clustering FPGA implementation that requires 0.723 ms to group a 2905 point data set to 8 clusters. Shanthi [122] demonstrates an FPGA implementation of a histogram based clustering algorithm. This implementation executes segmentation of a 512 x 512 pixel input in a few seconds. FPGAs have been also used as hardware accelerators for image segmentation process executed by a CPU, such as in [123], where the FPGA is only used to speed-up specific mathematic functions. In [124] an image segmentation approach using logarithmic arithmetic is again implemented on an FPGA device, but the performance results are inconclusive because of lack of available FPGA area for the complete system. Yamaoka et al. [125] present a pattern matching implementation architecture on an FPGA for input images of 80 x 60 pixels. The implementation uses an image segmentation cell-network for a region-growing algorithm. The principle behind this network has similar characteristics with our approach to use a cluster identification window (grid). However, the cell-network in [125] is used only to extract the size of a previously identified cluster, while in our implementation it is used to detect the cluster itself by identifying the directly neighboring pixels. This leads to a much more efficient use of the grid structure. All the above algorithmic implementations target unprocessed data with common properties and have a low throughput because of iterative data processing. In the High Energy Physics application domain, clustering is used to do a first level data filtering such as in [126]. Gregerson et al. use two 2 x 2 clusters on adjacent towers from the cylindrical CMS detector to identify energy patterns that exceed a desired energy threshold on the CMS calorimeter. The pixels are not zero-suppressed and the cluster size and shape is fixed. Pattern matching is executed on the deposited energy values.

The clustering algorithm and implementation in this work targets data with direct proximity (neighboring pixels) that are pre-processed and zero-suppressed. The data source is a pixel module with a size of 144 x 328 pixels on which in worst case conditions about 0.4 % of the pixels will pass the signal-to-noise threshold [132] (very low occupancy). After zero-suppression a single “pixel module image” from one pixel module is significantly compressed. Thanks to the reduced size, each input link will deliver “pixel module images” at rate of 1-2 MHz. Our implementation performs clustering at full speed processing all input data on the fly. These specific characteristics of the implementation do not allow direct comparison with the previously described FPGA image segmentation approaches. In [127] however, Wassatsch and Richter present a universal clustering engine with similar requirements for the Belle II experiment. In this approach the whole pixel frame has to be read before the clusters can be identified, which imposes a significant algorithmic delay. From the imposed delay derives a hard requirement for a 400 MHz clock that required the implementation of the algorithm on ASIC technology (TSMC 65 nm). A direct area comparison cannot be made between [127] and our proposed implementation, but an estimated 1000000 logic gates are needed for the DCE3 implementation, while around 15000 logic cells are used in our design on a Spartan-6 lx150t FPGA device. The small area occupation on the FPGA allows the flexibility to implement different algorithms on the same device, in addition to the reduced cost and development time for an FPGA implementation instead of an ASIC.

Our implementation uses an innovative moving window approach to reduce the FPGA resources required for the cluster identification process. It is an evolution of a sliding clustering algorithm [115]. This algorithm had a much higher cost in FPGA resources because it required a window as large as the module width, which is typically much larger than the maximum cluster size. The larger window size resulted in increased combinatorial logic that led to lower speed performance and significantly greater FPGA resources occupation. The proposed detection technique uses a smaller window, whose size is required to be comparable to the maximum expected cluster size. The size is adjustable to target specific maximum cluster size. Advanced control logic is required for the proper operation of the smaller window size, while the algorithm [115] uses simple control logic. The current

implementation introduces parallelization with a generic number of clustering engines instantiated in the design to increase overall performance. The result of the more sophisticated logic of the current design is that the incoming hits can be clustered at a much higher rate with the same use of FPGA resources.

4.3. Background Information

The clustering implementation presented here is part of the Fast TracKer processor (FTK) [128]. FTK is an approved ATLAS [129] upgrade. A brief introduction to ATLAS, FTK and the ATLAS Pixel Detector follows, to put the 2D Pixel Clustering implementation into context.

4.3.1. The ATLAS Detector

ATLAS (**A Toroidal LHC Apparatus**) is one of the seven particle detector experiments (ALICE, ATLAS, CMS, TOTEM, LHCb, LHCf and MoEDAL) constructed at the Large Hadron Collider (LHC), a particle accelerator at CERN. The experiment is designed to take advantage of the unprecedented energy available at the LHC and to observe phenomena that involve highly massive particles that were not observable using earlier lower-energy accelerators (such as LEP and Tevatron).

Inside the LHC and after the next two upgrades, bunches of up to 10^{11} protons (p) will collide 40 million times per second with the resulting collisions having energy up to 14 tera electron Volts (TeV). ATLAS is a detector designed to probe such collisions. The detector itself is 44 meters long, 25 meters in diameter, weighs approximately 7000 tones and contains nearly 3000 km of cable.

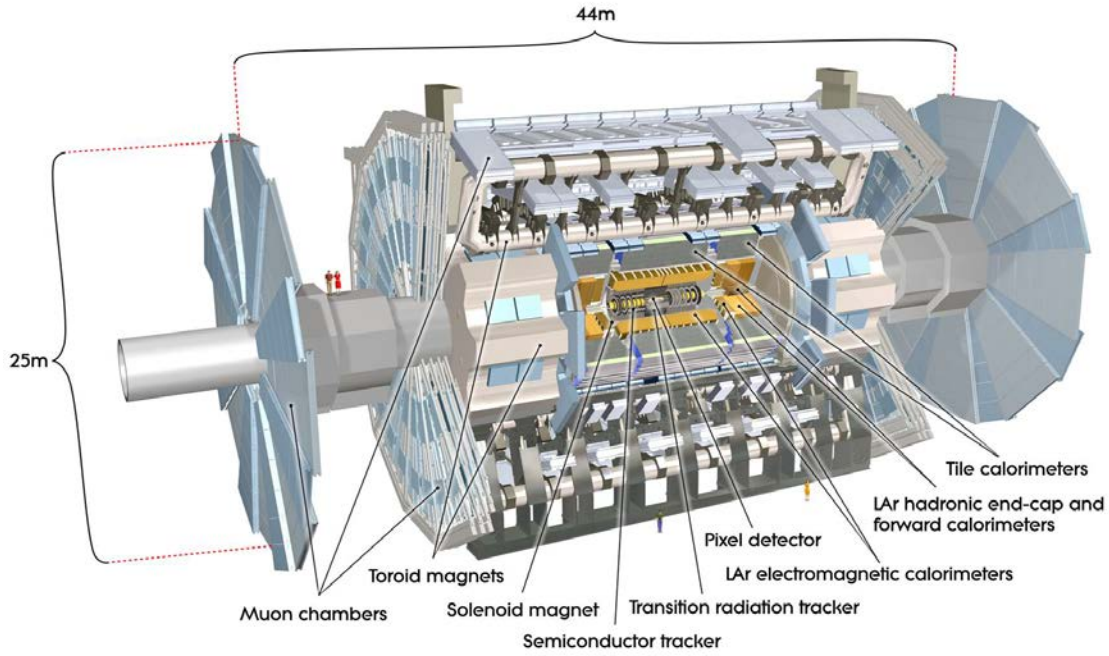


Figure 4.1. The ATLAS Detector

The ATLAS detector is barrel shaped and it is nominally forward-backward symmetric with respect to the interaction point (where the proton beams from the LHC collide). It consists of a series of concentric cylinders with increasing size and can be divided into four major parts (Figure 4.1): the Inner Detector, the calorimeters, the Muon Spectrometer and the magnet systems. Each of these parts is also made of multiple layers. These detectors are complementary: the Inner Detector tracks particles (the particle paths as they cross the detector's different layers), the calorimeters measure the deposited energy of particles on the detector, and the muon system makes measurements of highly penetrating muons. The two magnet systems bend the trajectory of charged particles in the Inner Detector and the Muon Spectrometer, allowing for momentum measurements. One event acquired by the ATLAS experiment consists of data from 100 million detector elements, which leads to about 1.6 MB of data after zero-suppression. There are 40 million beam crossings per second (40 MHz) and the amount of data produced reaches a total of 1 petabyte of data per second. It is therefore essential to have a fast and efficient data selection and reduction system to process fast events that are of real interest for new physics and physics analysis.

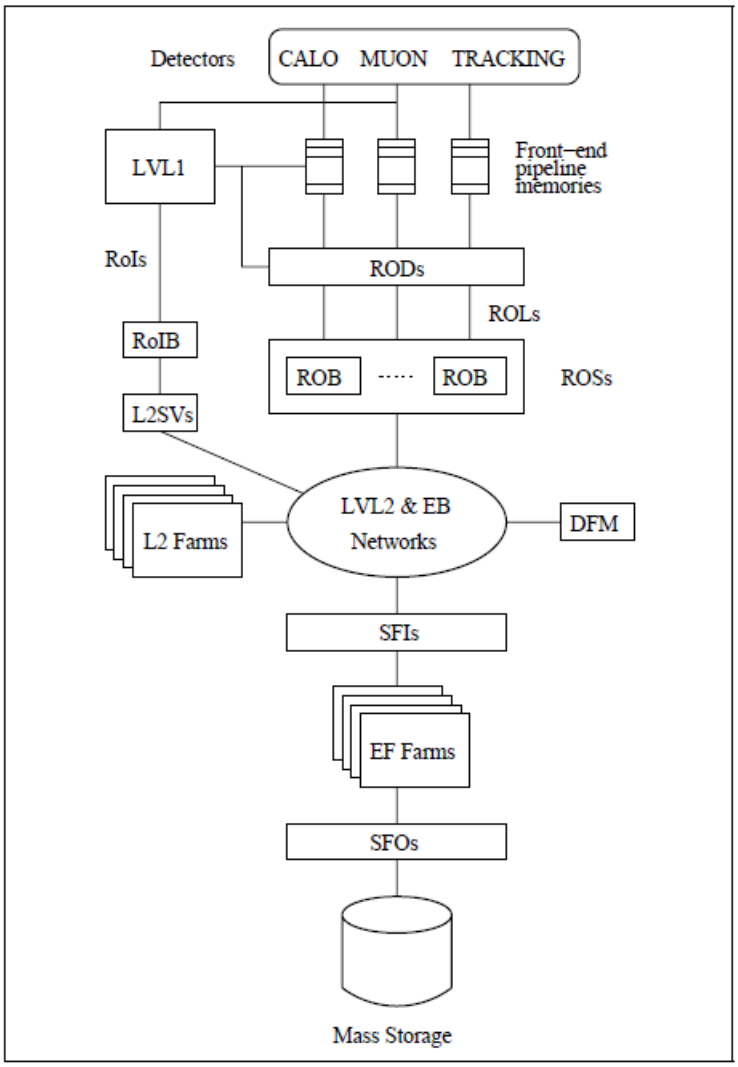


Figure 4.2. The ATLAS Detector Data Flow and HLT System

In order for the ATLAS experiment to reduce the event rate to the level at which only interesting events will be fully reconstructed, a three-level trigger system (Figure 4.2, [130]) has been deployed. The triggers use simple information to identify, in real time, the events that are most interesting and need to be retained for further analysis. The level 1 trigger (L1) reduces the rate of events to 100 kHz using custom, pipelined electronics and identifies Regions of Interest (RoI) worthy of further study in the trigger. The Region of Interest Builder (RoIB) delivers the RoI records to the level 2 trigger (L2) which runs selection algorithms on a farm of commodity processors to further reduce the rate to approximately 4.5 kHz. Finally the Event Filter (EF) reduces the rate to approximately 400 Hz for permanent

storage. The time budget for processing events at L2, and the EF is approximately 40 ms and approximately 1 s respectively. In L2 and EF track reconstruction is performed to execute the selection. Due to the time constraints (less time available) the L2 track reconstruction is much less detailed in comparison to the one executed by EF. For the next run starting in 2015 L2 and EF will be merged in a single High Lever Trigger (HLT).

The data are then stored in mass storage centers for offline processing and scientific analysis at a later stage.

4.3.2. The ATLAS Pixel Detector

The ATLAS Inner Detector is the detector part closest to the particle beam and the collision point (Figure 4.3). The Inner Detector consists of three different Silicon Detectors (SCT, Pixel and IBL) and a Xenon gas detector (TRT). The Pixel Detector is the first detector the particles cross in the radius of the ATLAS cross section (Figure 4.4).

The Pixel Detector [132] is again barrel shaped and consists of 3 cylindrical layers and the 3 endcaps (disks). The layers have average radii of 5 cm, 9 cm and 12 cm, and the endcaps 9 cm and 15 cm. The cylindrical layers of the detector consist of 1456 pixel modules while the 3 endcaps of 288. Each module (Figure 4.5) is 62.44 mm x 21.4 mm with 144 x 328 pixel elements each. There are 80 million pixels in total that cover an era of 1.7 m^2. The pixels of each pixel module are read out by 16 Front End chips (FEs). Each FE serves an array of 18 x 160 pixels. Most of the pixels have a size of 400 μm x 50 μm, but the pixels at the edges of the FEs (column 0 and column 17 of each FE) have a larger size of 600 μm x 50 μm. The innermost layer of the Pixel Detector is called B-Layer (layer 0), and the other two are Layer 1 and Layer 2. The Pixel Detector module data are read out by the Read Out Drivers (RODs) by using S-LINKs [131] running at 2.0 Gb/s.

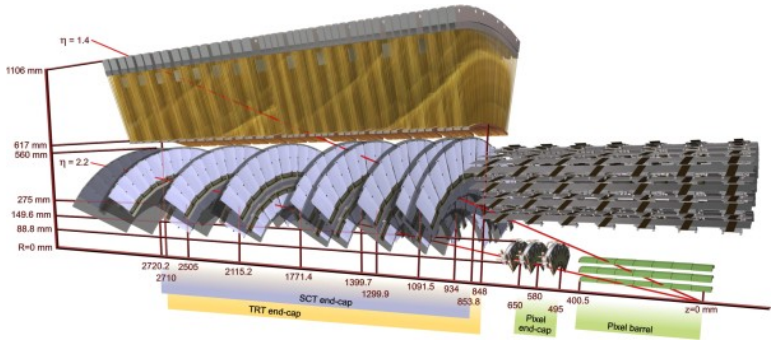


Figure 4.3. Crossing of the ATLAS Inner Detector by a High Energy Particle with Dimensions

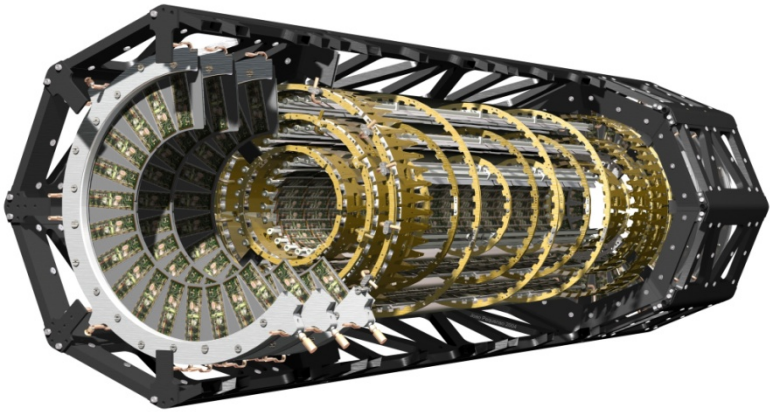


Figure 4.4. Tha ATLAS Pixel Detector 3D Model

In the current ATLAS upgrade a new layer is being added inside the pixel detector. The Insertable B-layer (IBL [133]) has a radius of about 3.2 cm and has just been inserted between the existing pixel detector and a new (smaller radius) beam pipe.

After each proton-proton collision a multitude of particles is generated that cross pixel detector layers. When a charged particle crosses a pixel it leaves a deposited charge on the pixel. If this charge is over a pre-defined threshold, the pixel is considered “hit” and the value is read out from the FEs. Taking into account the geometry of the detector and the nature of the collision events, it is obvious that the closer a layer is to the beam the greater hit occupancy per unit area it has. After the upgrade the IBL will be the pixel layer with the greatest hit occupancy.

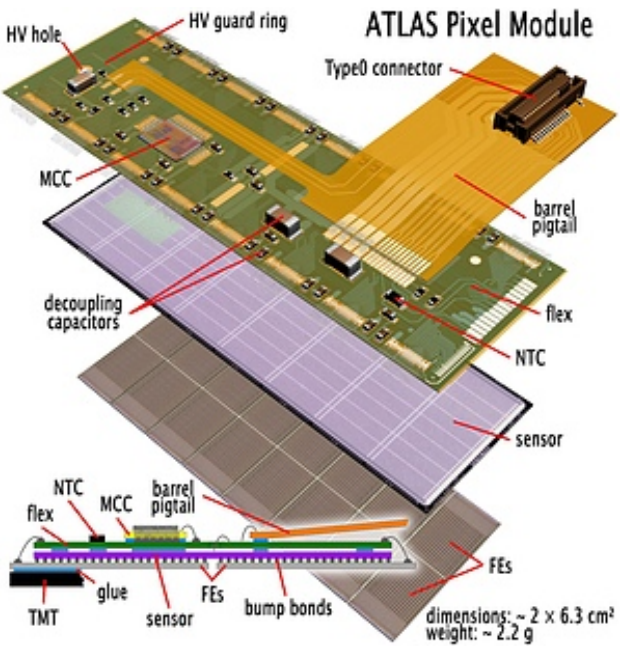


Figure 4.5. The ATLAS Pixel Module (© SISSA Medialab Srl. CC BY-NC-SA [132])

4.3.3. The FTK System

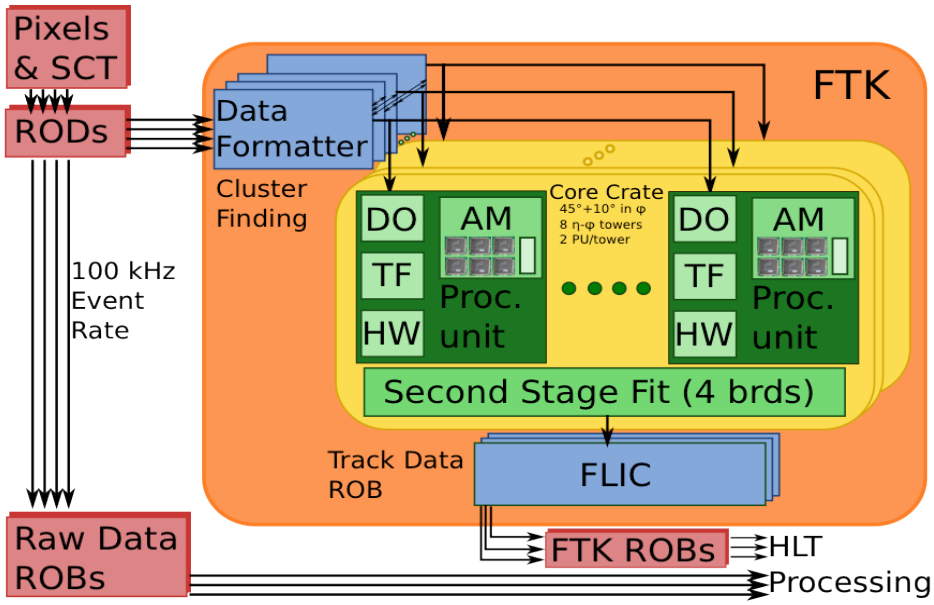


Figure 4.6. The Fast TracKer Processor

The FTK processor has the goal to provide a complete list of tracks to the ATLAS HLT at each Level-1 accept, up to 100 kHz, with very small latency, less than 100 ms. To achieve the goal of providing full tracking for the whole detector, FTK subdivides the tasks required to perform the full tracks in consecutive steps, executed in pipelined sequence by custom electronic boards.

The FTK will receive data from the pixel and microstrip detectors (the Inner Detector silicon layers) read out drivers (RODs) over 380 S-LINKs running at 2.0 Gb/s, and thus the total input rate will be 760 Gb/s. Since the hits from the silicon detectors need to be processed by subsequent algorithms, an early reduction of data optimizes the FTK processing. The processing steps of the FTK processor are:

Data formatting: The data (hits) from the silicon detector Read Out Drivers (RODs) are received and a data reduction is performed by implementing a clustering algorithm. The data are then split into 64 $\eta-\varphi$ towers that provide the parallelism needed to achieve the necessary system bandwidth. There is some overlap between adjacent towers to avoid efficiency loss at the boundaries. These functions are implemented in the Data Formatter (DF) boards and the Input Mezzanine boards (IM). Four IM boards are connected to one DF board.

Pattern matching: For each tower the clusters are compared with a pre-calculated list of about 15 million possible trajectories, using coarse hit resolution that (after proper analysis) has been deemed sufficient for pattern recognition. The comparison is performed in a few microseconds using “Associative Memory” (AM) chips. The AM chip is a custom designed ASIC that uses a CAM-like principle of operation. Pattern recognition is done using 8 out of the 12 silicon detector layers. Pattern matching is executed in the AM board (AMB) and the clusters are converted to coarse resolution data in the Auxiliary board (AUX).

Track fitting: When 7 out of the 8 layers in the pattern have a hit then a “road” is identified. For each road the clusters are retrieved at full precision and candidate tracks are built with all possible combinations of one hit per layer. The tracks are fitted using a linear approximation and the ones with good quality are extrapolated into the remaining 4 detector layers to be refit. In this way the track quality is improved and the number of fake tracks is greatly reduced. The first fit is executed

on the AUX board and the second fit with the extrapolation process is executed on the Second Stage Board (SSB).

Conversion to the HLT format: The FTK tracks are finally converted to the HLT format and sent to HLT by the FTK-to-Level-2 Interface Crate (FLIC).

4.3.4. The Clustering Problem for the FTK Processor

In Figure 4.7. a) an indicative collection of hits on an ATLAS pixel module area is demonstrated. The different colors represent how the hits are grouped in clusters. The numbers of the hits represent the sequence with which they will be read out from the pixel module. As mentioned before, the ATLAS pixel module is read out by 16 FE chips. The data from each chip are read out in column pairs. The data coming from the two columns are not sorted by position. In Figure 4.7. b) the sequence of the read out hits is shown. The colored arrows show the position of the hits that belong to the same cluster. It is therefore obvious that, as the hits are read serially to identify the different clusters it is necessary to loop over the hit list. As the pixel module hit occupancy increases so does the number of times the data must be looped in order to identify the different clusters. The target of the 2D Pixel Clustering implementation is to identify the 2D clusters as fast as possible, keeping up with the ATLAS L1 input rate.

The 2D-clustering is implemented on the FTK Input Mezzanine cards (FTK_IM) which are installed on the FTK Data Formatter boards [134]. On each FTK_IM card there are two Xilinx Spartan-6 lx150t FPGAs [138] that receive 2 S-LINKs each, one from a microstrip ROD [139] and one from a pixel ROD [132]. The latter transfers hits as 32 bit words at 40 MHz rate. Each ROD will transmit data of level-1 accepted events in sequence. The event data for one ROD contain the hits of the detector modules connected to that ROD. Clustering is executed independently on the data of each module. The natural data reduction replaces a cluster of contiguous hits with the position of the cluster centroid and the size of the cluster. The main challenge of the clustering implementation is to achieve the required 40 MHz hit processing rate for each S-LINK. The implementation performance is evaluated by using simulated Monte Carlo data for 80 overlapping proton-proton collisions that correspond to the maximum LHC luminosity foreseen until 2022

[140]. The hard real-time requirement is achieved by a multicore parallel architecture which can be implemented on the available FPGA products.

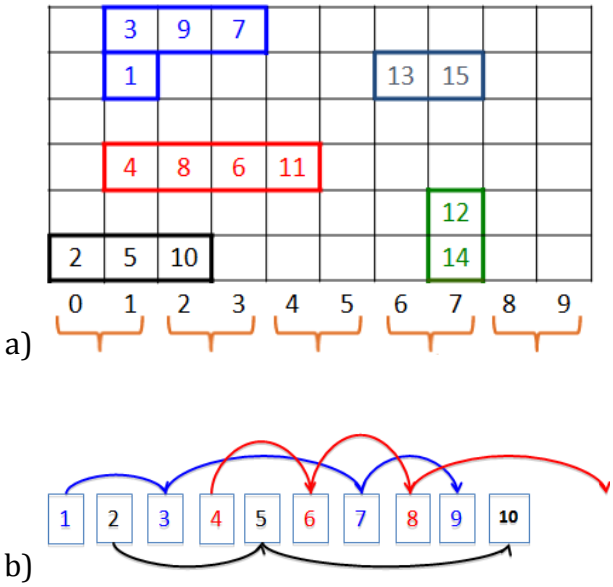


Figure 4.7. a) Indicative Collection of Hits on a Pixel Module Area, b) Indicative Read Out Sequence of the Hits

4.4. The 2D Pixel Clustering Implementation

The 2D-clustering implementation is designed to identify groups of contiguous pixels compatible with a cluster and then reduce the hit data to a single set of coordinates: the cluster centroid plus a few bits of cluster shape and size information. The data are received by an S-LINK decoder and are forwarded to a FIFO, which is the source of data for the clustering implementation. Each hit is a 32 bit word which carries the hit coordinates (column, row) and the Time-Over-Threshold (ToT) value that contains the deposited charge information (ATLAS Pixel Bytestream format, Figure 4.8).

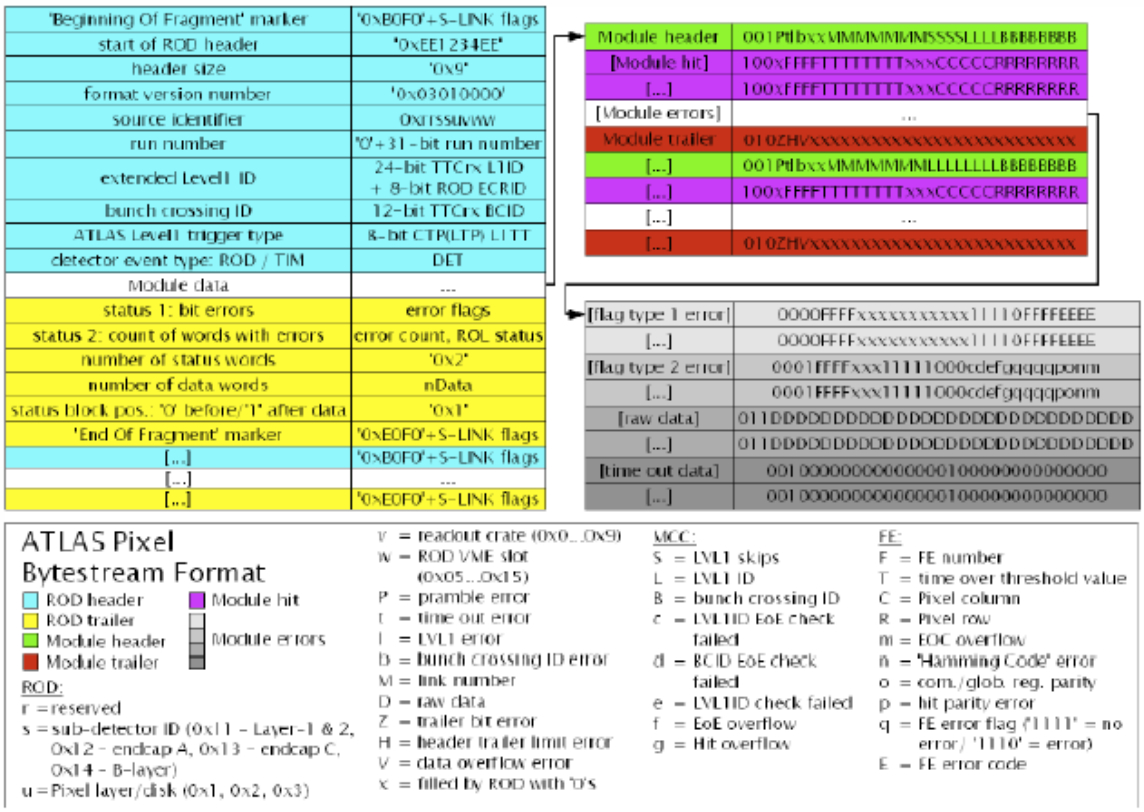


Figure 4.8. The ATLAS Pixel Bytestream Format [134], [135]

The clustering implementation is designed with a pipeline of three separate modules: a) the hit decoder module, b) the grid clustering module and c) the centroid calculation module. In Figure 4.9 the module sequence with the bus sizes is demonstrated. VHDL hardware description language was used for the design [20].

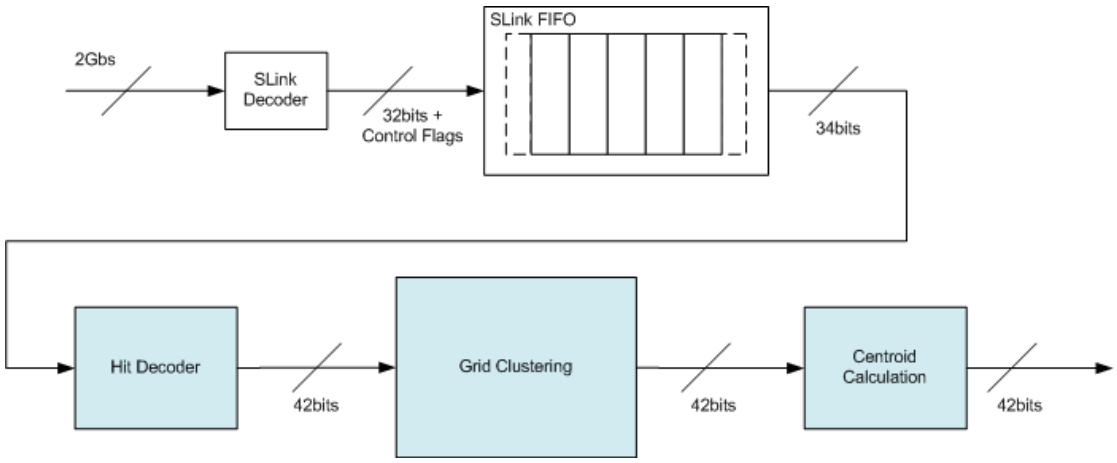


Figure 4.9. The 2D Pixel Clustering Single Flow

4.4.1. Hit Decoder Module

The input stage of the 2D-Pixel Clustering system is the hit decoder module. The hit decoder (Figure 4.10) transforms the incoming data from the ATLAS raw data format to a format useful to the following processing step. It is a pre-processing step that selects, formats and organizes the information that is used by the clustering algorithm. The ATLAS Bytestream (Figure 4.8) packages the data in two levels with flag words: start and end event words are the flag words that mark the beginning and the end of an event, module headers and module trailers are the flag words that mark the beginning and the end of hits from one pixel module. The module header word also contains the module number. The pixel hits are packaged between module header and module trailer. The hit decoder module is robust against bit errors in the input data and it is tolerant to errors in control words. In the rare case where an end event or a module trailer word is missing from the bytes stream, the hit decoder reinserts the missing word. If a more critical start event of module header word is missing, the hit decoder drops the hits that follow, because the source of these data cannot be identified. Every time such an error occurs it is flagged by a dedicated error bit.

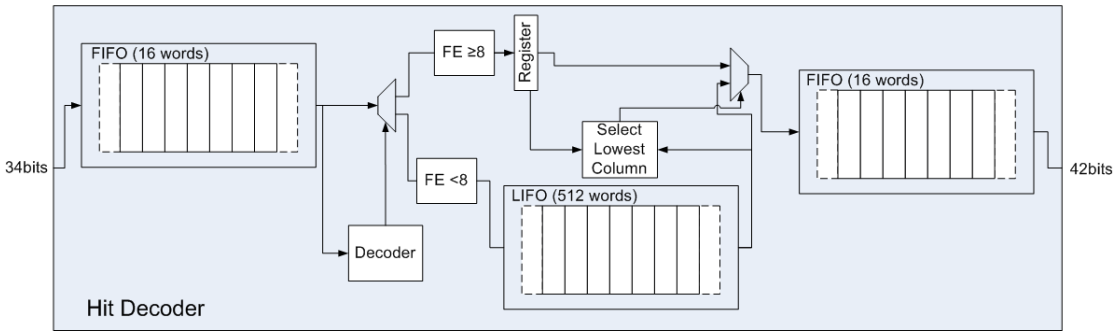


Figure 4.10. The Hit Decoder Module Block Diagram

The most important role of the hit decoder module is to properly align all the incoming data. The ATLAS pixel module FE chips are numbered from the bottom left corner to the upper left corner in a clockwise cycle and they are read out in the same sequence (Figure 4.11). This leads to half of the pixel module data arriving in reverse column order than the other half. The hit decoder module needs to restore

the column order of the hits since the clustering algorithm is based on the assumption that they are ordered by increasing column number.

To reverse the hit sequence a LIFO is used to store all the hits that arrive from FEs with numbers from 0 up to 7 (Figure 4.10). When a hit arrives from a FE chip with number from 8 up to 15 it is stored in a separate register. The value of the register is compared with the last value stored in the LIFO and the hit with the smallest column value is propagated to the next processing module. In this way increasing column sequence is restored. The LIFO size is 512 words which is largely sufficient for the expected hit occupancy in the pixel modules for up to 80 overlapping proton-proton collisions. In the rare case that LIFO size could be exceeded the only effect is to split some cluster for the corresponding pixel module. This condition is recorded in an end-event-word error flag. Two small FIFOs (16 words each) are added as input and output buffering stages for synchronization purposes.

For different image processing applications the hit decoder module can be appropriately adapted or, if no data preprocessing step is required, be completely removed.

8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Figure 4.11. The ATLAS Pixel Module FE Chip Sequence

4.4.2. Grid Clustering Module

The grid clustering module is the one that actually identifies the clusters and it is the most computationally intensive block of the implementation. The module uses a “moving window”, which is actually a rectangular grid of pixel cells of generic size. The grid takes advantage of the 2D structure of the FPGA fabric to avoid data

looping. The “cells” of the “window” are independent modules which change state from “empty”, to “hit”, to “selected” and back to “empty” depending on the current state of the “window” (Figure 4.12). The “window” size depends on the maximum expected cluster size per application and it must be big enough to fit this cluster size in both dimensions. The “window” is “moving” in the sense that during the several passes of the cluster identification process it is virtually placed on different coordinates on the pixel module and every time it is filled with data from different areas of the pixel module plane.

At the start of a module processing the detection “window” is filled with data around the first received hit. This hit is used as a reference and the grid is aligned to have this hit placed on the middle row of either column 0 or column 1 of the window. The alignment column depends on whether the hit belongs to an even or odd column of the pixel module. This is due to the double column scrambling of the data from the pixel modules, to allow for one column space for preceding hits arriving later. In Figure 4.13 the cluster window placement is presented by using the same pixel module example as in the clustering problem section. It must be noted that the 4 x 5 grid is a small version used for representation purposes only. In a) the window placement for a reference hit belonging to an odd column is demonstrated and in b) the placement for a reference hit belonging to an even column. When a hit is loaded to a cell, its state changes to “hit”.

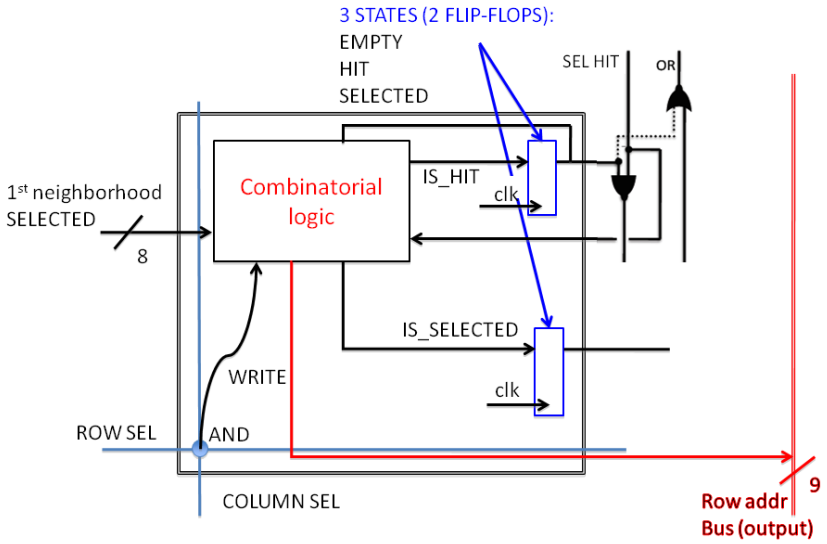


Figure 4.12. The Elementary Pixel Cell

The hits are read from the input until the first hit with a column beyond the column range spanned by the “window” arrives. This hit is kept in the input FIFO to be processed later. All the hits that belong to the “window” are loaded to the grid, while the hits that do not belong to the window but are within the window column span, either above or below it, are stored in a separate circular buffer (Figure 4.14: the hits in the darker colored boxes above and below the orange/light colored detection window are loaded in the circular buffer). The circular buffer is a custom design generic size memory using BRAM FPGA primitives and with extra added pointer functionality to accommodate the design’s needs.

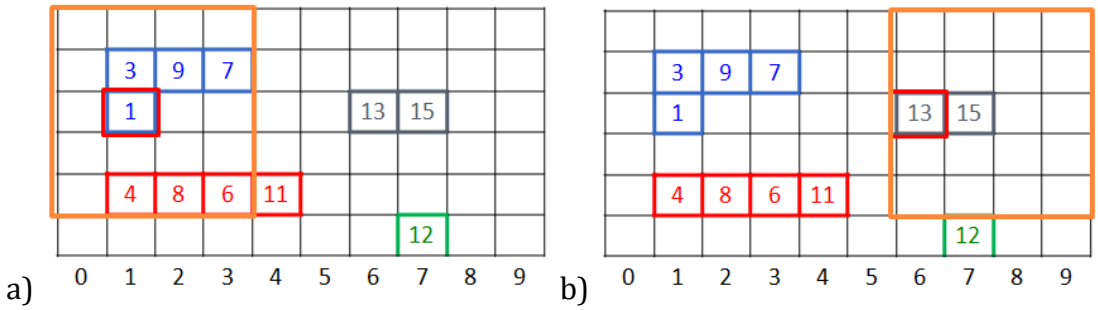


Figure 4.13. Cluster “Window” Placement and Reference Hit Choice

The cluster identification process begins by selecting two grid pixel cells as “seeds” on columns 0 and 1 on the middle row of the detection grid (Figure 4.15, a: cells marked with diagonal lines). These two coordinates are selected because either one of the two will definitely contain a hit (the “reference hit”). The “seed” cells that contain a hit when selected change their state to “selected”. The “selected” state is propagated on the next clock cycle to all neighboring hits (Figure 4.15, b, c: arrow demonstrating the propagation). On the same cycle one of the hits that were previously “selected” is read out (Figure 4.15, c, d: black dotted cells). When a hit is read out the cell returns to an “empty” state (Figure 4.15, d: cell marked with horizontal lines). Using the same process all the hits that form a cluster are read out and propagated to the next processing module in their relative coordinates with respect to the reference hit. In this way the coordinate bit width is reduced. After the cluster hits are all read out, a cluster flag word is sent to the next module which contains the absolute coordinates of the reference hit. The hits that remain in the

grid that do not belong to the identified cluster are then read out and they are saved in the circular buffer in their absolute coordinates. These recovered hits are not in column sequence with the previous hits of the circular buffer.

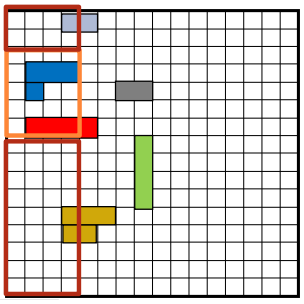


Figure 4.14. Discrimination between hits that belong to the "window" and hits that do not. The hits in the darker colored boxes are loaded in the circular buffer.

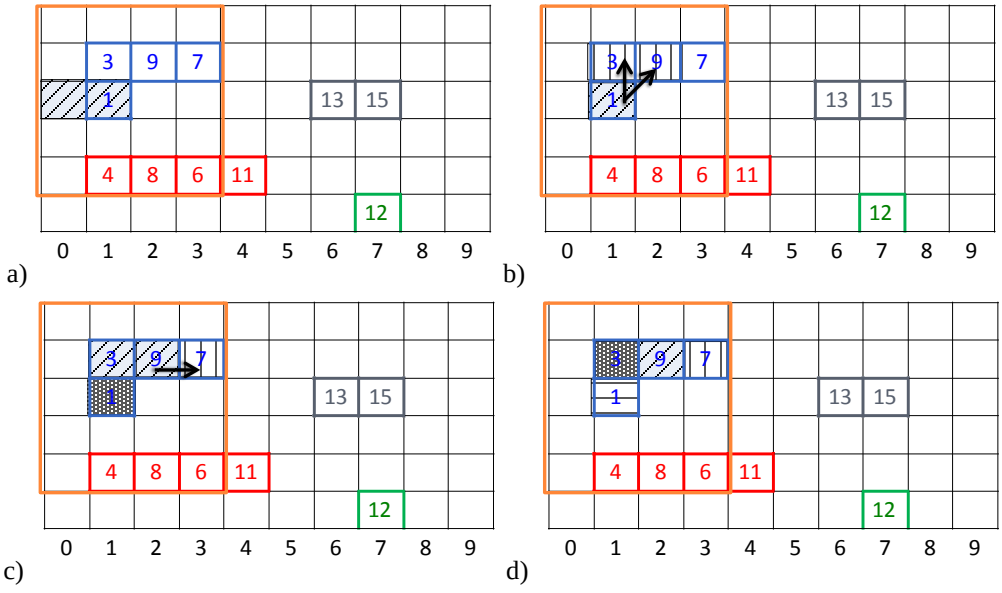


Figure 4.15. Cluster Read Out Process

On the next run of the clustering module the grid is loaded with hits from the circular buffer (Figure 4.16). The leftmost hit stored in the circular buffer is chosen as a new reference hit. This hit value is stored in a separate register, called the "leftmost register", as the circular buffer is being filled. While reading from the circular buffer to load the grid, hits that do not belong to the grid need to be saved

again in the circular buffer. Extra functionality was added to the circular buffer to control simultaneous reading and writing of hits without accessing the same data twice. If after reading the circular buffer there are hits in the input FIFO that belong to the column span of the current detection “window”, these hits will be read until a hit with a column number beyond the last grid column arrives at the input. The clusters are identified using the same process of the first iteration. A clustering module processes all the data that belong to one pixel detector module, so the cycle is repeated until a pixel detector module trailer word is received from the clustering input and the circular buffer is empty.

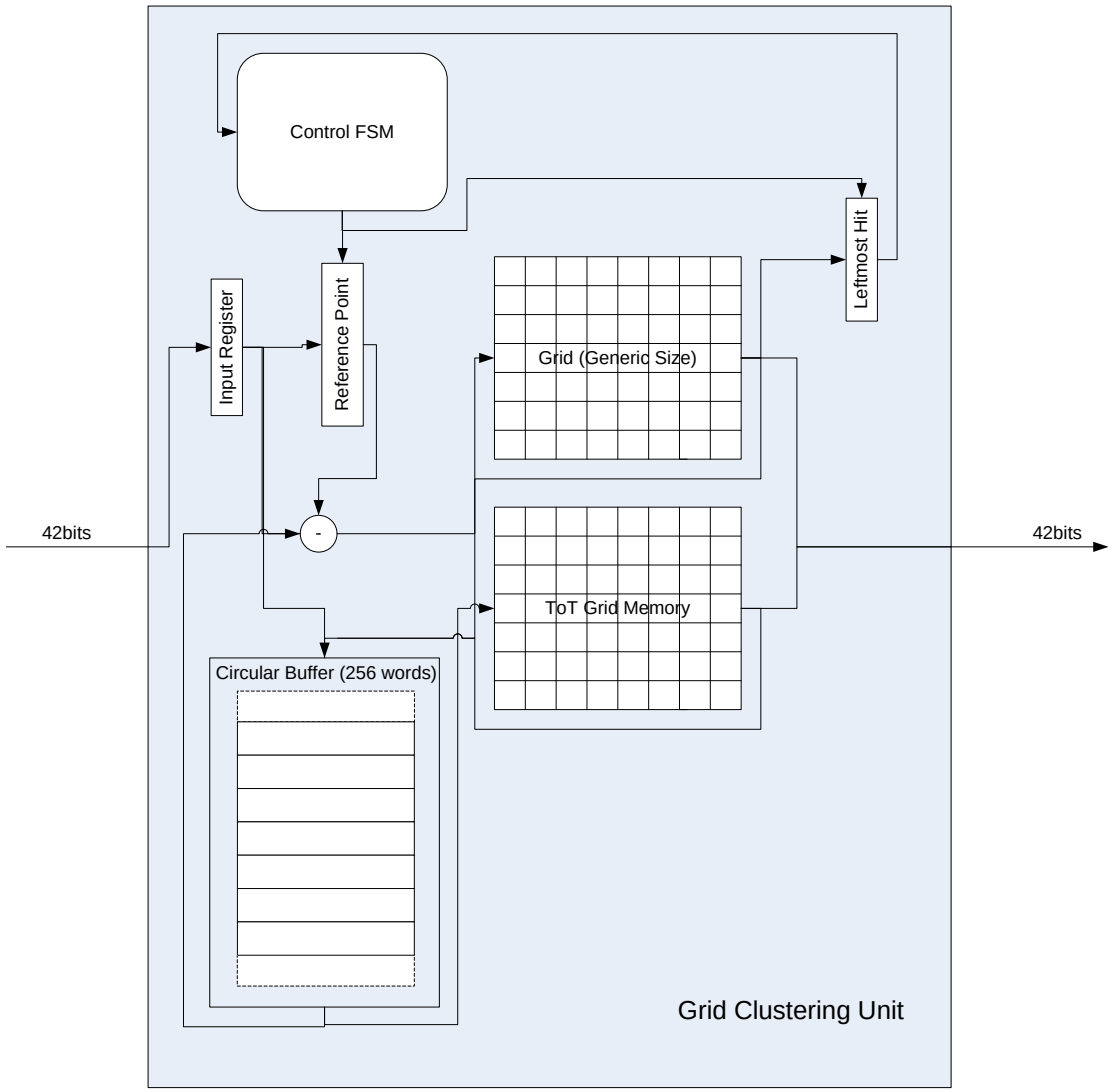


Figure 4.16. The Clustering Module Simplified Block Diagram

For the current 2D-clustering module implementation a “window” of 8 x 21 pixels is used, 8 for the *z* or *r* direction and 21 for the *r* - *φ* (ATLAS uses a right-handed coordinate system with its origin at the nominal interaction point (IP) in the centre of the detector and the *z*-axis along the beam pipe. The *x*-axis points from the IP to the centre of the LHC ring, and the *y*-axis points upwards. Cylindrical coordinates (*r*,*φ*) are used in the transverse plane, *φ* being the azimuthal angle around the beam pipe.). Most clusters recorded by the ATLAS pixel module (95%) fit within a box of 5 columns and 6 rows. The bigger grid is used to allow identification of the rare large clusters and clusters generated by merging hits from two or more clusters. Clusters bigger than the grid size, i.e. clusters extending from the reference hit beyond one of the grid edges, are split. Clusters that touch a grid edge are identified by a flag in the output, called the “split flag”. The “split flag” bit is a dedicated bit in the cluster flag word.

4.4.3. Centroid Calculation Module

The centroid calculation module is the post-processing step in the 2D-clustering implementation that performs the data reduction process. It is the module where the cluster data are replaced with one set of coordinates, the centroid coordinates. For each cluster a centroid value is calculated. The centroid value is calculated as the center of each cluster’s bounding box (Figure 4.17). The pixels of the ATLAS pixel module have two different sizes on the *x* axis: the pixels at the edges of the FEs are 600 μm long, while all others are 400 μm long. The centroid calculation module corrects this difference by calculating the centroid coordinates in normalized units of 25 μm for the *x* axis and 6.25 μm for the *y* axis. The normalization factors were chosen to agree with the FTK Simulation framework and to allow for normalization factors that require only bitwise shifts for their implementation. The divisions required for the normalization process are implemented in a Look-Up-Table.

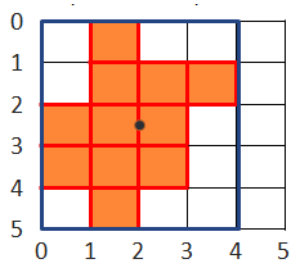


Figure 4.17. Cluster Bounding Box (with a "Universal" Pixel Size)

The centroid is then corrected taking into account the charge deposition in each hit measured by the Time-over-threshold (ToT) information. This is done to replicate the ATLAS offline code for centroid calculation. The ToT value for each hit arrives in the same word as the hit coordinates and while the hits are placed in the detection "window" of the grid clustering module these values are stored in a separate memory (ToT memory) and they are recovered while the cluster hits are read out.

The formulae used to calculate the centroid value for the column (*x* coordinate) value are demonstrated below:

$$x\_average = \frac{ColMin + ColMax}{2} + a \cdot (qRatio - 0.5)$$

$$qRatio = \frac{qColMax}{qColMin + qColMax}$$

Equation 4.1. Centroid Calculation with Charge Deposition Correction

ColMin and *ColMax* in equation (1) are the minimum and maximum columns of the cluster. Variable *a* is a function of pixel position and its value is always smaller than the size of a pixel ([141] for more information). In equation (2) the charge imbalance between the two sides of the cluster is calculated (left-right for *x* and top-bottom for *y*). *qColMax* is the sum of the Time-Over-Threshold values of *ColMax* and *qColMin* the sum of the same values for *ColMin*. *x_average* is the final centroid column value with the applied corrections. The same equations apply for *y_average*

by replacing columns with rows. Additional information such as cluster shape/size is also included in the output together with the centroid coordinates (see Figure 5.3 in Appendix A for details).

The post-processing step can be tailored on the application (e.g. center-of-mass [120], median calculation [142], etc.).

4.5. Parallel Implementation

The target of the 2D-Pixel Clustering design is to design a flexible system that is suitable for the FTK processor but can be adapted to various applications with different processing needs. One fundamental characteristic of this 2D-clustering implementation is that different clustering engines can work independently and in parallel on data from different modules (frames), therefore increasing performance while exploiting more FPGA resources. It has been noted that for the FTK application the pixel data arrive through an S-LINK. Additionally, the Data Formatter board also expects data through a single data stream [134]-[134]. Serial data propagation is the most common data propagation method in image processing applications, to carry data from the detector to the processing systems. Taking into account all the above, choosing the appropriate parallelization strategy is critical for the implementation's performance, not only for FTK but for all application fields.

The parallelization strategy chosen for the presented implementation is to instantiate multiple clustering engines (grid clustering modules) that work independently on data from separate pixel modules (Figure 4.18). The data acquired by each detector module can be considered an independent image because clusters are entirely contained in a single module. To achieve this, data parallelizing (demultiplexing) and data serializing (multiplexing) logic modules are necessary. There are two issues that require special attention. The execution cycles required for the clustering identification process are strongly data dependent. This leads to unequal processing time per pixel module per clustering engine. Additionally, the recovered single data flow after the parallel processing must be in the correct event order (a requirement from the subsequent processing steps, in the same way video frames in image processing need to be processed in time incremental order). To

tackle these issues a special parallel data distributor module and a data merger module were designed.

4.5.1. Parallel Data Distributor Module

The pixel data arrive in a single data stream, packed by event header and event trailer control words. The event header at the input of the 2D clustering code is a single word made of the ATLAS Level 1 ID number (event number) with a specific start event flag. Within each event the pixel module data are packed within module header and module trailer control words. The processing time of one engine is strongly data dependent and therefore it is impossible to predict which parallel clustering engine will finish processing before the others. In addition, the data merger module must be able to restore the data stream in the received event sequence (pixel module order is irrelevant). The parallel data distributor module must propagate the event and module control words in a way such that the data merger can retrieve the proper order.

Each parallel clustering engine is buffered by two FIFOs at input and output. This is done to facilitate the design routing and temporarily store input and output data while the clustering engine is busy or the data merger has put the clustering engine on hold. The input FIFO from each parallel clustering engine has a write data counter activated so that the parallel data distributor module can monitor which parallel engine has less data queued at the input. In the FTK version the FIFOs are implemented using the Xilinx LogiCORE IP FIFO Generator v9.3 [143]. As soon as the first event header appears at the input of the parallel data distributor it is written to a LVL1ID FIFO (Level 1 ID FIFO) as a reference for the received event header sequence (and for system monitoring purposes). The parallel data distributor chooses the clustering engine where the pixel hits will be propagated. On first run this is the engine with the smallest index number. On all subsequent runs it is the engine with the smallest write data counter value (clustering engine which currently has less data queued at input).

To identify the smallest write data counter value a generic binary tree comparator was implemented. In the binary tree comparator in case more than one write data counter have the same value, the one with the smallest index is chosen as

the minimum. The use of the generic binary tree comparator allows the flexibility to use an arbitrary number of engines restricted only by the available FPGA resources and the constraint that the number of parallel engines needs to be a power of two. If there is a need to implement a number of engines different than a power of two the generic binary tree comparator can be easily replaced by a non generic comparator to identify the minimum value of a specific number of write data counters.

The pixel module data are sent packed between the module header and the module trailer words. If there are more than one pixel module data in the event, another clustering engine is chosen by the binary tree comparator that will receive the next pixel module data. Multiple modules can be sent to each engine for each event. For each engine receiving at least one module, the event header is also sent before the first module in order to keep the event-module association. If during data send operation the input FIFO of the engine because almost full, backpressure is applied to the parallel distributor module and the data send operation is stalled until space if free in the input FIFO. The parallel distributor tracks all the engines that are active in one event by activating bit flags in a separate register. When the end event word arrives it is sent to all clustering engines that have processed data belonging to this event, to close the event for all engines. The event trailer is made by the LVL1ID and an event trailer specific flag bit.

4.5.2. Data Merger Module

The data merger module begins its operation as soon as the LVL1ID FIFO and one of the parallel clustering engines output FIFOs have a valid event header at the output. It then compares the event header with the one at the output of the LVL1ID FIFO and if it matches it starts reading the data. If more than one parallel engine has the same event header then the engine with the lowest index number is chosen (by using a priority encoder). After the data from one pixel module are read out, the data merger checks whether there are any more engines that have the same event header or whether a second pixel module is loaded in the same engine and if there is one it reads out the corresponding FIFO. While the data merger is busy reading the output FIFO of one parallel engine, the other parallel engines continue normal operation and write their outputs to the corresponding output FIFO, until this FIFO is almost

full. When the FIFO becomes almost full backpressure is applied to the grid clustering module (read out operation is stalled), until the corresponding FIFO has free space for data. As the backpressure is always propagated from the output of the system to the input (data merger to output FIFO, to grid clustering module, to input FIFO, to parallel distributor) it is critical that the data merger module works unhindered to propagate the data as fast as possible and release space in the system FIFOs. When all the parallel clustering engines output FIFOs that had data from the same event have the event trailer at the output the event trailer is read out and the LVL1ID FIFO is read to get the next event header and restart the event reading process. In the extremely rare case when no engines have the same event header as the LVL1ID FIFO this is declared a fatal error, it is flagged at the output error word and the clustering module is restarted.

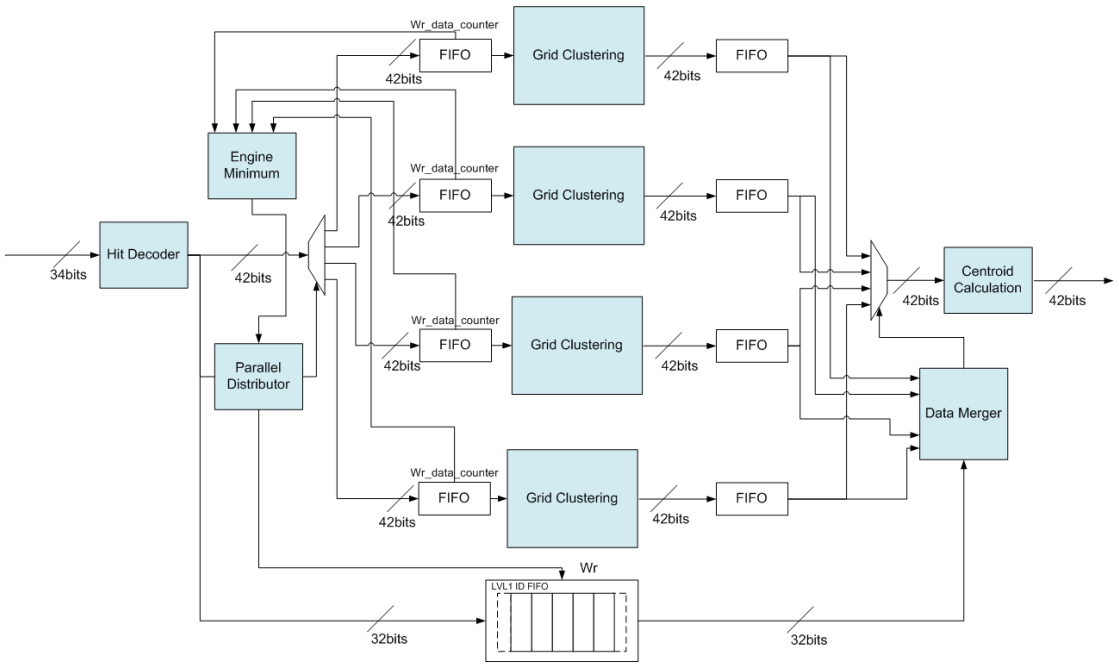


Figure 4.18. An Indicative 2D-Pixel Clustering Parallel Implementation with 4 Engines

4.6. Functional Verification and System Design Space Exploration

The 2D-Pixel Clustering system targets an application with critical data taking at a high speed. Unlike most image processing applications, when a bottleneck appears in the processing data stream, data cannot be dumped to release the flow (e.g. miss an image frame) and backpressure is applied from one processing module to the previous one until the bottleneck is resolved and all the data can be processed. Data can only be dumped in extreme cases of critical errors (such as synchronization errors in parallel data streams) when the data processing modules around the identified issue or the whole FTK System must be reset. In addition, the 2D-Pixel Clustering system must be able to process without errors data arriving at all possible input data speeds up to the maximum (40 MHz input data clock, 100 kHz total event data rate). Therefore it is highly critical that each module is properly verified to achieve bulletproof operation at all input data rates and backpressure signal combinations before integration.

The first step in achieving this is a functional verification of the VHDL code. As a functional verification environment ModelSim debug and analysis environment [144] was used. In parallel, a bit accurate software model of the 2D-Pixel Clustering system was developed. The bit accurate model was developed to be used for the verification of the FPGA firmware, as well as to be included in the FTK Emulation framework [134],[145]. The FTK Emulation framework is a software framework that described the hardware logic of the FTK Processor boards and modules to verify the computational load on the different components and to evaluate the quality of the output tracks.

As the design is incremental and generic so was the need for the verification. The first step was to properly debug and verify the single engine flow, first with simple input data files (few clusters approximating the maximum expected hit occupancy in a continuous loop) and then with input files from simulated Monte Carlo data for 80 overlapping proton-proton collisions that correspond to the maximum LHC luminosity foreseen until 2022.

After the verification process was completed, performance estimation was made for the single engine flow. The information was necessary to estimate the number of parallel engines that would be required to process the maximum input data rate (40 MHz continuous data input). The information was taken by using a VHDL testbench which captures the time each end event word is read into the pixel clustering system and the time the same end event word is read out from the system. In this way the processing time per event can be calculated. A Monte Carlo file from a $ZH \to \nu\bar{\nu}b\bar{b}$ event [146] was used as an input file. The results from this measurement can be seen in Figure 4.19. On the x-axis the number of events is demonstrated and on the y-axis is the event latency divided by the average number of event words, which produces an estimation of the system latency per input data word. It can be easily seen that the processing rate of the single flow pixel clustering engine quickly saturates to ~ 85 ns per data word.

The input rate of the pixel clustering system is 40 MHz. At maximum throughput the pixel clustering system will receive one data word every 25 ns. It can be calculated that the ratio of processing time per data word over the input data time (the maximum input data rate for the single flow implementation) is 85 ns / 25 ns = 3.4. The single flow 2D-clustering implementation is 3.4 times slower than the maximum input rate. Since pixel clustering is an iterating algorithm a deeper pipeline is not an option. Therefore a parallel implementation is essential to achieve the required performance. As the single flow implementation is 3.4 times slower than the maximum input data rate it can be anticipated that the parallel engine implementation must have at least 4 clustering engines working in parallel to achieve the required performance.

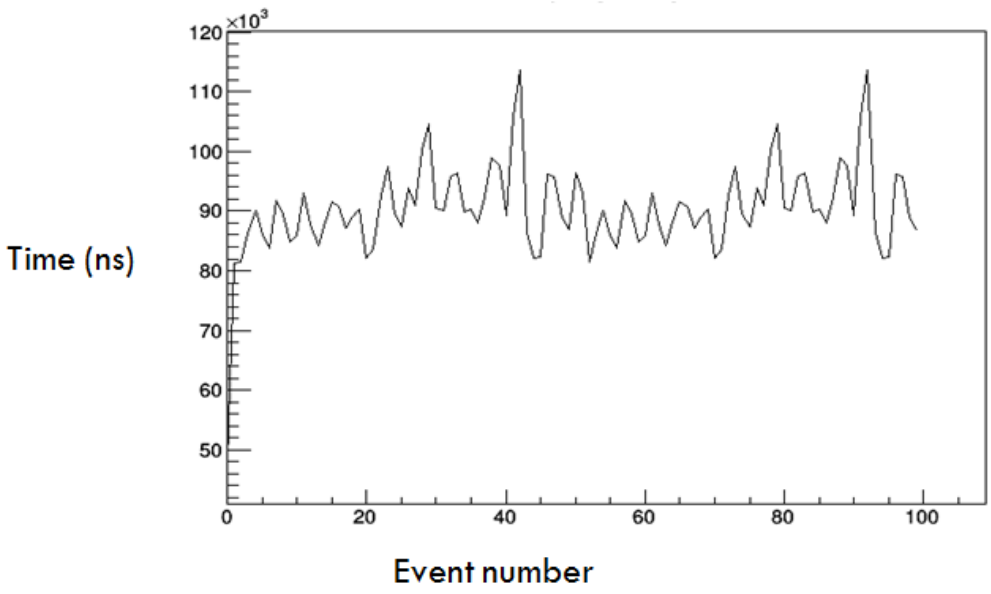


Figure 4.19. Single Flow Event Processing Latency

The results obtained by the single flow tests and performance estimation lead to a first parallel implementation with 4 parallel clustering engines. The parallel version was also verified by the bit accurate simulation. The same input data files were used for the data comparison. The output files from the two system versions were identical in content. Between the two outputs only one type of difference is possible: a change in the sequence the pixel modules are read out. This is due to the operation of the Data Merger module, which can read out the pixel modules in a sequence different to the one written by the Parallel Distributor. However, this is irrelevant to the operation of the following processing board (the DF) and the FTK in general. Therefore it is accepted as normal operation.

In the parallel engines enough buffering before/after each engine is needed to ensure that all engines can work at full speed without waiting for data from parallel distributor and without backpressure from data merger. In the first parallel version with 4 engines the minimum necessary buffering was used. FIFO size was chosen to be 256 words at both input and output of each pixel clustering engine. This size is sufficient to hold data from one pixel module at maximum occupancy (Figure 5.4, hits per pixel module). Using the same testbench event processing latency was captured with the same calculation principle. The obtained results are

shown in Figure 4.20. As can be seen the performance initially is better but the system quickly saturates to a similar event processing latency, demonstrating that backpressure is applied at the input of the pixel clustering system.

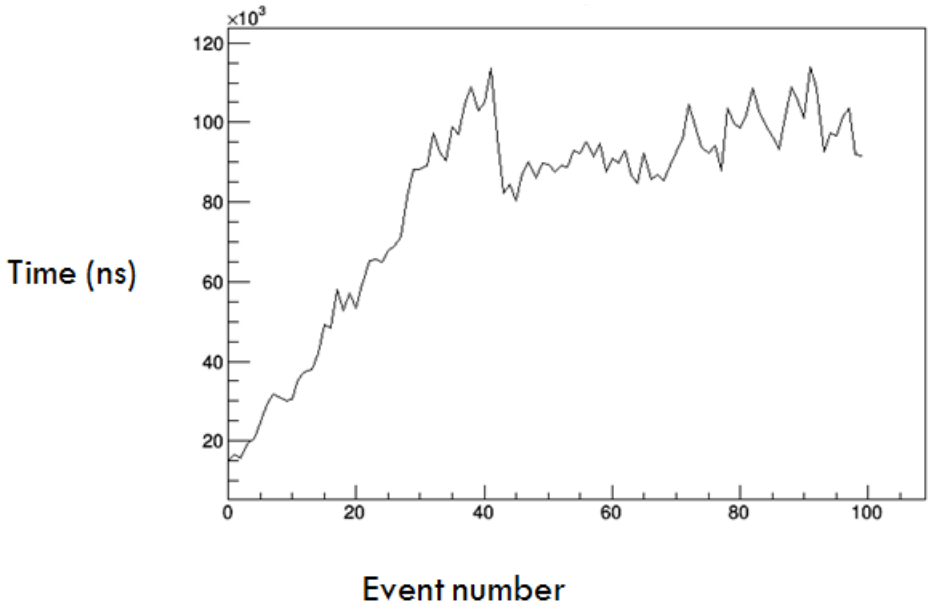


Figure 4.20. 4 Parallel Engines Flow Processing Latency with Small Buffering

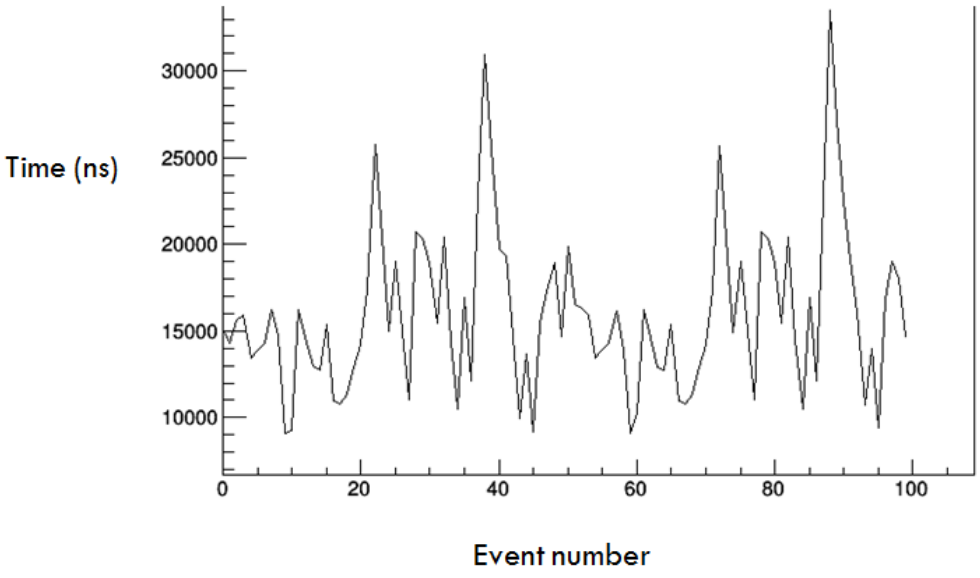


Figure 4.21. 8 Parallel Engines Flow Processing Latency with Small Buffering

A pixel clustering system with 8 parallel engines was also implemented using the same size FIFOs for buffering. Using the same testbench and the same input file the obtained results are shown in Figure 4.21. In this diagram it is obvious that the event processing rate remains stable. No saturation and no backpressure at the input appear.

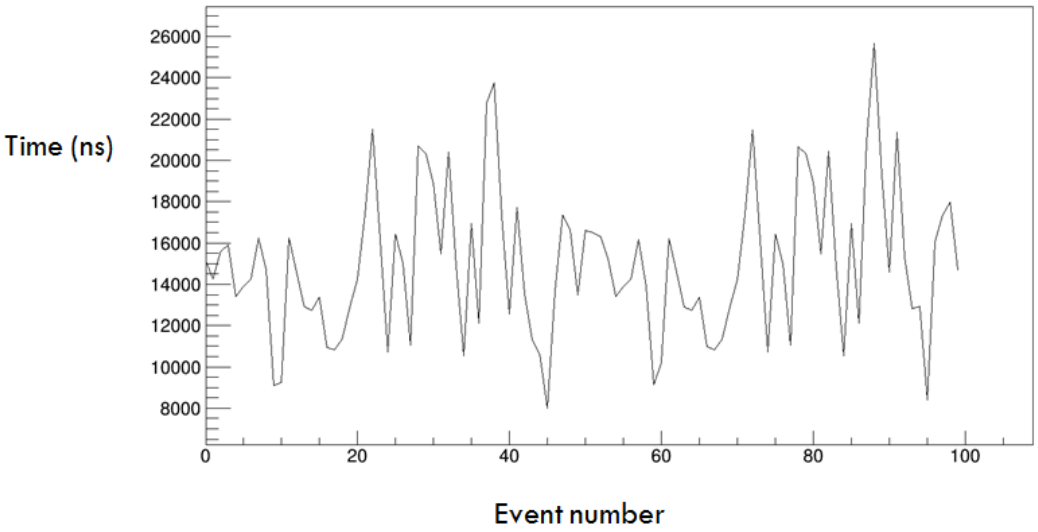


Figure 4.22. 4 Parallel Engines Flow Processing Latency with Big Buffering

The event processing latency estimations have demonstrated that 4 parallel engines should be sufficient to cover the 40 MHz hit input rate specifications. Therefore, the 4 parallel engine test war repeated with increased buffering size. The new buffer sizes used were 512 words for the input buffer and 1024 words for the output buffer of each parallel engine. This size is sufficient to store one full event of maximum occupancy in each parallel engine. The obtained results with the same testbench are shown in Figure 4.22. It can be seen that same with Figure 4.21 no processing rate saturation appears and no backpressure is applied at the input. The processing rate is stable and sufficient to cover the given specifications.

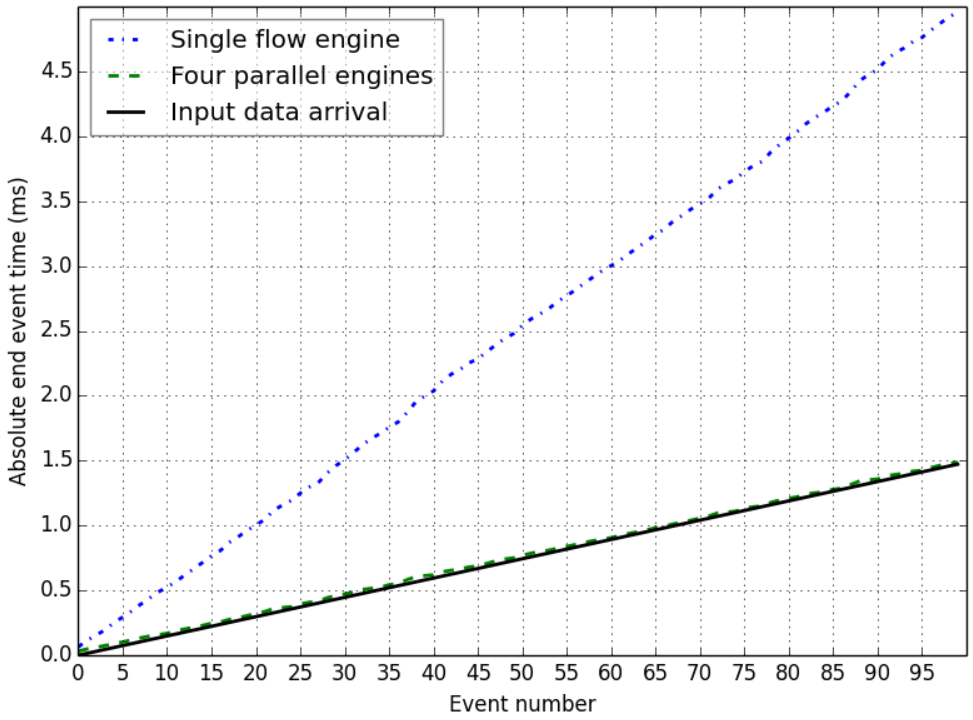


Figure 4.23. Performance Plot for the 2D-Pixel Clustering Implementation

To better demonstrate the performance of the pixel clustering various implementations a performance plot was made using the testbench captured data (Figure 4.23). On the *x*-axis the event number is presented, while on the *y*-axis the absolute time an event exits the 2D-clustering implementation (end_event output time). The continuous black line is the reference line that demonstrates the maximum input data rate, one word every 25 ns. As it can be seen the single flow (dash-dot blue line) cannot follow the input rate with a much higher slope and a processing time of ~85 ns per data word. The 4 parallel engine flow (dashed green line) is almost identical to the reference line, demonstrating that it can fully respond to the maximum input data rate. The results for the 8 parallel engine system are almost identical to the 4 parallel engines with the larger FIFOs. The above performance results demonstrate that 4 parallel engines with big buffering are sufficient for the ATLAS detector pixel modules. From these 100 events a total of 21050 clusters were identified out of 58890 data words (hits and control words).

4.7. Results

The 2D-clustering implementation has been developed, verified by simulation and tested on multiple FTK_IM boards in both single and parallel versions. Implementation results are presented for both the single and parallel flow versions. The most fundamental features of the FPGA fabric (flip-flops, LUTs and BRAMs) are used as metrics for the implementation size.

4.8. Single Flow Results

The 2D-clustering single flow implementation (results presented in TABLE 4.1) occupies 0.9 % of the device's FFs, 3.4 % of the LUTs, 0.4 % of the 18 kbit BRAMs and 2.9 % of the 9 kbit BRAMs. The small differences between the sum of resources of the separate modules and the complete system are due to different routing choices applied by the Xilinx PAR (place and route) tool when the complete system is implemented. The extra 9 kbit BRAM belongs to the output FIFO implemented after the grid clustering module. The operational frequency is defined by the grid clustering module's critical path. The centroid calculation module calculates the center of the cluster bounding box in normalized coordinates without taking into account the charge deposition. In addition to the centroid calculation module, an indicative implementation of a center-of-mass calculation is presented in [120] and of a median calculation in [142]. Both implementations are for the same FPGA device and have an operating frequency much higher than 81.5 MHz. The center-of-mass implementation uses 237 FFs and 409 LUTs while the median implementation uses 215 FFs and 307 LUTs for a 32 x 32 pixel detection window. The center-of-mass, the median calculation and the centroid calculation module have comparable area occupation and shorter critical path than the grid clustering module, therefore it can be assumed that all three can be used as a final processing step for the clustering implementation.

TABLE 4.1. 2D-Clustering Implementation Results for Single Flow

	FF	LUT	BRAM (18 kbit)	BRAM (9 kbit)	Maximum Frequency (MHz)
Hit Decoder	306	486	1	3	120.0
Grid Clustering	700	2257	-	2	81.5
Centroid Calculation	539	492	-	2	185.0
Total System	1580	3128	1	8	81.5
FF: Flip Flops, **LUT**: Look-Up-Tables, **BRAM**: 18 kbit Block RAM and 9 kbit Block RAM					

4.9. Parallel Flow Results

In 4.6 a design space exploration was described that was used to define the proper number of parallel engines for the specified performance requirements. Each version was designed and verified through the Xilinx validation chain and its performance was measured with post place and route simulation by using worst case ATLAS simulated data. The design space exploration demonstrated a strong correlation between the size of the output FIFOs on each clustering engine and system performance. This correlation was due to the backpressure applied by the data merger module to the clustering engines. A parallel flow version of four parallel clustering engines with a small output FIFO (256 words) failed to meet the performance specifications, while a version with eight parallel clustering engines and the same output FIFO size met the specifications, since the larger parallelization completely removed the backpressure from the data merger. However, a parallel version of eight engines increases significantly the amount of used resources. To avoid this risk a parallel flow version with four parallel engines and bigger buffering was implemented to completely remove the backpressure with less parallelization and smaller use of resources. Sufficient FIFO size was determined by testing measurements to be 1024 words for each clustering engine.

This parallel flow version of four parallel clustering engines was implemented and measured on the FTK_IM board. In TABLE 4.2 the implementation results for the 4-engine parallel flow with big buffering are presented. The new modules that were introduced for the parallel design version, the parallel data

distributor, the binary tree comparator and the data merger, occupy a very small percentage of FPGA resources with respect to the total system. The total system occupies 3.1 % of the device's FFs, 11.5 % of the LUTs, 5.6 % of the 18 kbit BRAMs and 3.5 % of the 9 kbit BRAMs. It can be seen that the number of BRAMs that are required for the implementation with the four parallel engines is greater than the number required for the single flow when multiplied by four because of the extra required buffering at the input and at the output of each parallel clustering engine (FIFOs before and big FIFOs after Grid Clustering modules in Figure 4.23).

TABLE 4.2. 2D-Clustering Implementation Results for 4-Engine Parallel Flow

	FF	LUT	BRAM (18 kbit)	BRAM (9 kbit)	Maximum Frequency (MHz)
Parallel Data Distributor	51	79	-	-	270.0
Binary Tree Comparator	84	51	-	-	334.8
Data Merger	70	113	-	-	247.5
Total System	5739	10583	15	21	80.5
FF: Flip Flops, **LUT**: Look-Up-Tables, **BRAM**: Block RAMs					

The clock performance is again defined by the grid clustering module and the maximum frequency drops a little due to routing to 80.5 MHz. The 4-engine parallel clustering flow has been tested on the FTK_IM board. The board test was executed with the same clock configuration as with the single flow, 40 MHz clock for the input FIFO and 80 MHz for the rest of the implementation. By using the same input data as with the single flow clustering implementation a worst case of ~ 25 ns processing time per data word was estimated. This performance covers the given specifications of maximum input data rate of 40 MHz.

A parallel version with 16 engines was also integrated for potential more computationally demanding applications (such as the Insertable B-Layer). The implementation results for the parallel 2D-clustering version with 16 clustering engines are presented in TABLE 4.3. This system version is implemented with output FIFOs of 128 words as the buffering requirements will be recalculated after extensive testing with IBL data. The implementation uses 9.5 % of the device's FFs, 38.8 % of the device's LUTs, 6.3 % of the 18 kbit BRAMs and 16.4 % of the 9 kbit BRAMs. As this implementation uses a significant percentage of the FPGA device

resources and has modules with a 16 times bus fan out, it is important to keep the critical path buffered in order to avoid performance degradation. The initial implementation achieved a clock frequency of 79 MHz for the grid clustering modules. Additional constraints and optimizations in the place and route options allowed for the use of the same 80 MHz clock in the 16 engine implementation.

TABLE 4.3. 2D-Clustering Implementation Results for 16-Engine Parallel Flow

Slice Type	Used Slices	Total Slices	Used Slices(%)
FFs	17558	184304	9.5
LUTs	35724	92152	38.8
BRAMs (18 kbit)	17	268	6.3
BRAMs (9 kbit)	88	536	16.4
FF: Flip Flops, **LUT**: Look-Up-Tables, **BRAM**: Block RAMs			

4.10. Comparison with previous clustering approach

The current implementation is an evolution of a sliding clustering algorithm that had a much higher cost in terms of FPGA resources [115]. In the sliding clustering algorithm grids of 4 x 168 and 8 x 328 pixels were used. The grid clustering module of the old sliding algorithm version has been re-implemented on a Spartan-6 LX150T device for direct comparison with the current implementation. The extrapolated FPGA resources and clock frequency results are presented in TABLE 4.4. It must be noted that the maximum frequency is directly related to the number of cells of the detection window. As it can be seen the sliding version of the algorithm occupies a significantly larger number of FPGA resources (9.8 % LUTs for the 4 x 168 and 38.2 % for the 8 x 328 versions). For the 4 x 168 version, which is a compromise between the window size and the fraction of split clusters leading to more than 10% of clusters flagged as split, an average of 3 clocks per incoming hit were required. In order to achieve 40 MHz hit processing rate about 9 parallel cores would have been required, reaching a very crowded design with almost 90 % LUTs usage on the Spartan-6 LX150T device. For comparison the current implementation occupies a factor of 8 less LUTs for even slightly better processing rate. The option that would reduce the fraction of split clusters for the algorithm in [115] is the use of the ideal 8 x 328 window size. This version would match the pixel module width as

well as the number of columns (8) in our implementation, but it would require a factor of 64 more LUTs than the presented implementation, as it can be derived from Table IV.

TABLE 4.4. Extrapolated Results For Sliding Clustering Algorithm

Grid Size	FFs	LUTs	Maximum Frequency (MHz)
8 x 21 (current)	700	2257	81.5
4 x 168	2800	9028	14.4
8 x 328	10933	35252	7.4
FF: Flip Flops, **LUT**: Look-Up-Tables			

4.11. Statistics

The outputs of the 2D-clustering implementations have been verified by a bit-accurate simulation model. Using the same model as well as the outputs of the FPGA firmware statistics for the 2D-clustering performance were gathered. The most important aspect of these statistics is the percentage of the identified clusters that have an active "split flag" with the current 8 x 21 pixel detection window specifications. The "split flag" is active whenever a cluster touches the grid (detection window) edge. This is an indication, but not a confirmation, that the cluster might be split. It was estimated that at most ~1.5 % of all clusters will be flagged as split (Figure 5.5 in the Appendix shows the cluster bounding box sizes that appear in events with 80 overlapping proton-proton collisions). Not all clusters with the "split flag" are actually split. It is possible to reduce the fraction of split clusters using a larger pixel clustering grid. For example a 12 x 31 grid would reduce the fraction of potentially split clusters to ~0.4 %. The proposed design is totally generic, therefore adjusting the size is easily achievable. An implementation with this larger grid was realized. The FPGA maximum clock speed is reduced to 65 MHz. (along with the fact that the number of cycles required to process a hit increases to 4.7 – by reference to the 40 MHz input clock). This means that 8 engines can fit on the Spatan-6 and process pixel data at full input speed, by occupying 43 % FPGA resources (LUTs). We have chosen the 8 x 21 grid size, which doesn't split any cluster of size up to 7x11, since larger clusters are not of interest. They originate mostly from tracks that are not from beam collision or from low transverse

momentum particles [141] that are not reconstructed by the FTK. Furthermore, what matters more is the fraction of split clusters for tracks with transverse momentum above 1 GeV/s. The effect on tracking performance is under study.

4.12. Hardware Tests

The firmware has been tested extensively on FTK_IM cards (Figure 4.24) at Lab4-CERN. The testing configuration is as follows:

A QUEST computer is used as a data source. A QUEST is an ATLAS ROBIN card configured as a data source [147]. Gigabit fibers are connected from the QUEST to the FTK_IM cards. The FTK_IM cards are mounted on the DF board which is placed in an ATCA crate [148]. The Data Formatter board is acting as a pass through. The same QUEST computer is also used as a data sink with another ATLAS ROBIN card being configured as a FILAR [149] (data receiver).

For the board test a 40 MHz clock was used for the input FIFO of the hit decoder (input interface) and a 80 MHz clock for the rest of the implementation. In all the tests the output results have been confirmed by comparison with the bit-accurate simulation.

Input files from Monte Carlo simulations of 80 overlapping proton-proton collisions were used as test data. A number of monitoring registers were added to the VHDL wrapper of the design (Appendix B, 5.2) for continuously monitoring the status of the hardware. These registers include the status of all Finite State Machines of the design, the status of all FIFOs, data validation signals, and data words from critical points in the hardware data path (such as data parallelizing and data merging points).

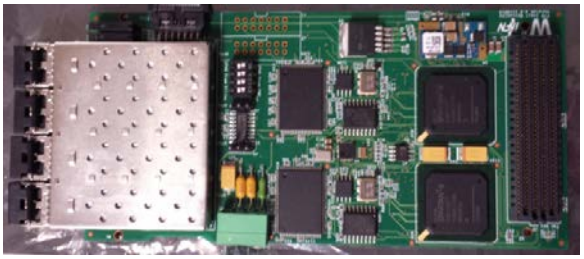


Figure 4.24. The FTK_IM card

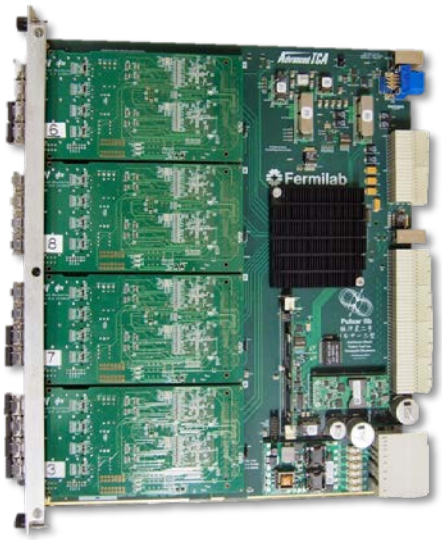


Figure 4.25. The Data Formatter Board with Four Mounted FTK_IM Cards

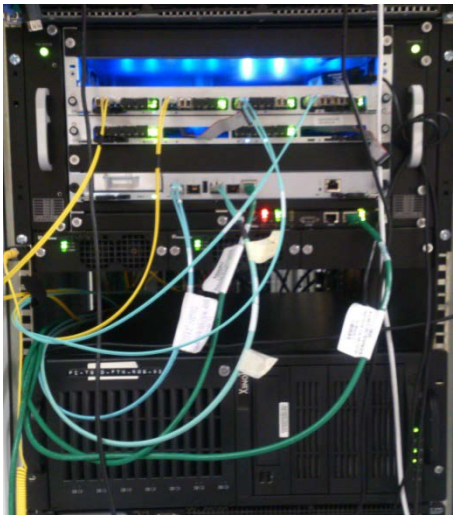


Figure 4.26. Testing Configuration at Lab4-CERN

In addition a special LVL1ID monitoring module was designed. The LVL1ID monitoring module continuously monitors the LVL1IDs processed by the Pixel Clustering system. LVL1IDs from ATLAS come in sequence increasing by a value of 1, apart from special counter reset cases. The module checks whether the current LVL1ID has a valid value in comparison to the previous one. If the value is invalid a special error flag is raised. The LVL1ID monitoring module will be used to identify possible problems in the data flow of the system.

All monitoring registers for the status of the FTK_IM are accessed externally through I^2C protocol and monitoring software (called SpyMon). A screenshot of the SpyMon software is presented in Figure 5.8. FTK_IM was tested with data of 80 overlapping proton-proton collisions in continuous loop and input data rate of 40 MHz (100 kHz event rate) without errors for more than 14 hours (overnight tests) and no backpressure applied from the hardware (no processing speed bottlenecks were initiated by the firmware).

4.13. Future Developments

The FTK_IM board and firmware has been reviewed by CERN electronics department and is currently in mass production. The system is being integrated in the Fast TracKer Processor with more complicated data flow tests being run on continuous basis as more boards from the complete system are being integrated in the flow.

A faster processing version of the 2D-Pixel Clustering system will be designed for the IBL. The IBL Modules consist of one or two front end chips (FEs), each one with 80 columns and 336 rows of pixels. Data from each FE arrive half in increasing column order and half in reverse column order. Additionally hits arrive packed in two different levels of compression. One hit word can contain data from two pixel hits, and 5 hit words can be packed in 4 data words. This results in a maximum input rate of 100 MHz (with an actual average of 70-80 MHz). To cope with these new specifications the hit decoder must be redesigned, the number of parallel engines must be increased and the centroid calculation module must be adapted to the geometry of the new layer.

A preliminary (pass through) version of the new hit decoder has been used in behavioral simulations to estimate the required number of parallel clustering engines for the 2D-Pixel Clustering implementation by using event files of 80 overlapping proton-proton collisions on the IBL. Current estimation demonstrate that 16 engines are sufficient to cover the new specifications, with area occupation demonstrated in TABLE 4.3.

Additionally, an FTK_IM prototype board with two Artix-7 XC7A200T FPGA devices [150] is currently being constructed. In the future the current firmware will be ported in the new devices to allow for better performance while occupying less FPGA resources. The available resourced can be used for more complicated post-clustering processing modules to better approximate the ATLAS offline processing software calculations. One option for such a processing module is the use of a neural network [151].

4.14. Conclusions

A multi-core FPGA-based 2D-pixel clustering implementation was developed for the Fast TracKer Processor in the ATLAS experiment. The implementation is designed to be fully generic in bus sizes, memory sizes and number of parallel engines used, therefore it is easily adjustable to various image processing application that require 2D pixel clustering. Each parallel engine can work independently on different pixel modules. The implementation uses a moving window technique to reduce the FPGA resources required for the cluster identification process.

A design space exploration was executed to identify the bottlenecks in the parallel data processing implementations. The number of clock cycles required per cluster identification is strongly data related and therefore use of Monte Carlo data was required to assess the performance of the design. A strong correlation between the size of buffering used before and after each clustering engine was identified and the buffering sizes were fine-tuned to fit the FTK specifications.

Additionally monitoring modules were developed and monitoring tools were used at the critical data path points to remove all possible processing bottlenecks and ensure unhindered data flow.

The specifications for the Fast TracKer were to process data at an input rate of 40 MHz (100 kHz event rate). Previous clustering implementations would require more than eight times the resources to achieve similar computational performance. These specifications were covered by a 4 parallel engine version of the design. The 2D-Pixel Clustering is integrated in the Fast TracKer Processor and the FTK_IM board as well as its firmware have been reviewed by CERN electronics department.

The versatility of the design allows for future exploration of its adaptation to different application field, by changing the number of clustering engines, the various parameters of the design as well as the post-clustering processing step.

Chapter 5

Conclusions

5.1. Review

This dissertation describes the work of the researcher on multiprocessing system development for implementation of applications. This work is divided into two major parts: the model formulation part and the implementation part.

The model formulation derives from the need to optimize heterogeneous MPSoC systems according to various applications' needs and based on different hardware platforms. With the new available technologies especially on reconfigurable platforms (FPGAs) the number and complexity of parameters has increased so significantly that the impact on design development time is reducing the time to market advantage of reconfigurable platforms over ASIC technology. In this thesis a hands on example of the design flexibility offered by MPSoCs implemented on reconfigurable platforms is presented with a complex design space exploration to study the impact of data/task level parallelism and different memory architectures on the sytem performance and resource usage. A new parameter called *Hardware Efficiency* is introduced to correlate performance with increase in area (hardware resources). With this parameter we quantify how "efficiently" the hardware resources are used.

An Integer Linear Programming model is formulated to solve the above problem. This model can be used as an optimization tool by the designers early in the design development process. The model includes parameters not only for performance and resource usage, but for memory usage as well, which to the best of our knowledge was a first for ILP formulations of this kind. The model can be used for optimizing the system based on a variety of objective functions for area, performance and memory, as well as combination of the above with varying importance weights.

The model was also extended to include early power and temperature estimation for each processing unit of a hybrid FPGA MPSoC, as well as the mean device temperature at regular time intervals. With this extension a complete formulation is offered where the optimization parameters can be all the major MPSoC characteristics: performance, area, memory usage, power/temperature.

In the implementation part two major applications are used as working examples of high performance MPSoC. The first one is a complex machine vision system for real-time flow detection on microfluidic LoC devices. The machine vision system is integrated in a Point-of-Care system with sensors, actuators and microcontrollers. The machine vision architecture is parallel and optimized to achieve the required performance of 60 fps following a camera with 1 Mpixel resolution.

The required parallelism to achieve the performance specifications was explored and a bit accurate simulation for the edge detection preprocessing step was developed. The simulation was used to define the precision of the implemented algorithm as well as hardware verification. Advanced generic modules for center of mass, median calculation and an application specific alarm point module were developed for the machine vision system. These modules follow the parallelism of the system but are generic and therefore the parallelism and performance of each module can be appropriately adapted. The machine vision system was integrated in a working prototype to demonstrate proof of concept. Comparisons with equivalent machine vision systems on various platforms prove that the presented implementation is significantly faster for equivalent or bigger video resolutions.

The second application was a 2D pixel clustering implementation for streaming data. The implementation was originally developed for the ATLAS Fast TracKer processor as an addition to the ATLAS Trigger system but it is generic and configurable and therefore easily adapted to various image processing applications. The characteristics of the application require that no data bottlenecks must appear for the maximum input data rate of 40 MHz. Therefore extensive exploration was executed to define the appropriate combination of buffering and parallelism to remove all possibilities of backpressure generation from the pixel clustering system to preceding processing modules. The implementation is modular and uses a moving window technique that exploits the 2D FPGA fabric to reduce data looping for clustering identification. Design parameters such as number of parallel engines, detection window size, etc. can change by simply changing a variable value in the design libraries. Therefore the implementation can be easily adapted to the performance requirements of the denser IBL detector with 100 MHz input data rate. Comparison with older FPGA clustering implementations demonstrates that 64 times more hardware resources would be required to achieve equivalent performance.

5.2. Future Work

The work presented in this dissertation offers great opportunities for extended research in the future.

The ILP model formulation presented in Chapter 2 is a tool for design optimization on a heterogeneous MPSoC early in the design development process. A target for the future is the transformation of this model to an automated tool using a parser and compiler. For this tool the description of the application and the specifications of the architecture would be required as input. The application would be described as a task graph, while the architecture as preset constant values or as variables with preset maximum and minimum values to describe system characteristics such as number of processing elements, maximum available memory, total available area etc. These variables will set the boundaries for the design space exploration execution.

The current power and temperature model formulation can produce estimation at an early design development time. For improving accuracy more details such as thermal diffusion properties of the device and possible neighboring of the processing elements can be added, as the additive effect of temperature is influenced by the processing elements neighborhood temperatures apart from the one generated by the processing element itself.

The work on image processing implementations in Chapter 3 and Chapter 4 is highly correlated. The moving window approach proposed for 2D pixel clustering in Chapter 4 can be adapted to be used for flow detection in machine vision systems like the one presented in Chapter 3. Flow detection in the Machine Vision system is already executed by using detection windows. As the moving windows in the Pixel Clustering implementation are also generic in size, the clustering algorithm can be used to identify the menisci of the flows by using only the active pixels that are within direct neighborhood of each other, thus increasing even more the precision of the implementation. The center of mass and median modules can be used as a post processing step for the clustering implementation, as it is already suggested in Chapter 4.

In addition, a faster 2D pixel clustering implementation for the IBL detector is planned, which will use a bigger number of parallel engines and appropriately adjusted hit decoder and centroid calculation modules (based on the different dimensions of the IBL pixel module). Using a more advanced Artix 7 device will also increase performance and leave more resources available on the reconfigurable device to exploit more complicated post processing implementations, such as neural networks (as is currently the case in the ATLAS offline code).

Another research approach could be the use of the Associative Memory boards of the FTK system to execute fast pattern matching for edge detection purposes and then adjust appropriately the 2D pixel clustering implementation as a post processing step for data reduction. The principle behind this implementation is the idea presented by M.M. Del Viva in [152].

The experience gained by the researcher on the operation of the Fast TracKer system and its parallel data flow led to the assignment of the responsibility of the

hardware monitoring modules development for the complete system. System modules that identify the loss of data synchronization, monitor the data flow and flag or possible execution critical errors will be developed.

References

[1] EETimes, “Power Management in mobile devices: an energy conservation view,” Technology News, Available Online: http://www.power-eetimes.com/en/power-management-in-mobile-devices-an-energy-conservation-view.html?cmp_id=7&news_id=207801765&page=2

[2] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE* , vol.86, no.1, pp.82,85, Jan. 1998

[3] C. E. Shannon, "A mathematical theory of communication." *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1 (2001): 3-55.

[4] Sriram, Sundararajan, and Shuvra S. Bhattacharyya. *Embedded multiprocessors: Scheduling and synchronization*. CRC press, 2012.

[5] Paulo Manuel Baltarejo de Sousa, "Real-Time Scheduling on Multi-core: Theory and Practice," PhD Thesis, Universidad Porto (2013).

[6] Intel Inc, “Intel Core Processors in the LGA2011 and LGA1150 Sockets,” Product Brief, available online: http://www.intel.co.uk/content/www/uk/en/processors/core/4th-gen-core-desktops-brief.html

[7] Qualcomm, “Snapdragon 805 Processors,” Product web page, available online: https://www.qualcomm.com/products/snapdragon/processors/805

[8] Aeroflex Gaisler,” Quad Core LEON4 SPARC V8 Processor,” User Manual, Available online: http://www.gaisler.com/doc/LEON4-N2X-DS.pdf

[9] Kuhn, K.J., "Considerations for Ultimate CMOS Scaling," *Electron Devices, IEEE Transactions on* , vol.59, no.7, pp.1813,1828, July 2012.

[10] A. Cassidy and A. Andreou, “Beyond Amdahl’s Law: An Objective Function That Links Multiprocessor Performance Gains To Delay and Energy,” *IEEE Transactions on Computers*, no. 99. IEEE, pp. 1–1, 2011.

[11] N. Myhrvold, "The next fifty years of software." *ACM97 Conference*. 1997.

[12] Frank, D. J., Dennard, R. H., Nowak, E., Solomon, P. M., Taur, Y., & Wong, H. S. P. (2001). Device scaling limits of Si MOSFETs and their application dependencies. *Proceedings of the IEEE*, *89*(3), 259-288.

[13] Intel Inc., “Intel 22nm technology,” Process technology description, available online: http://www.intel.com/content/www/us/en/silicon-innovations/intel-22nm-technology.html

[14] Xilinx Inc, “Xilinx UltraScale™ MPSoC Architecture,” white paper, available online: http://www.xilinx.com/publications/prod_mktg/ultrascale-mpsoc-architecture-bacgrounder.pdf

[15] Altera Inc., “The Breakthrough Advantage for FPGAs with Tri-Gate Technology,” white paper, available online: http://www.altera.com/literature/wp/wp-01201-fpga-tri-gate-technology.pdf

[16] E. Wu, K. Abugharbieh, B. Banijamali, S. Ramalingam, P. Wu, and C. Wyland, “Interconnect and package design of a heterogeneous stacked-silicon FPGA,” in *Custom Integrated Circuits Conference (CICC), 2013 IEEE*, 2013, pp. 1–8.

[17] A. Rahman, J. Schulz, R. Grenier, K. Chanda, M. Lee, D. Ratakonda, H. Shi, Z. Li, K. Chandrasekar, J. Xie, and others, “Interconnection requirements and multi-die integration for FPGAs,” in *Interconnect Technology Conference (IITC), 2013 IEEE International*, 2013, pp. 1–3.

[18] C. Erdmann, D. Lowney, A. Lynam, A. Keady, J. McGrath, E. Cullen, D. Breathnach, D. Keane, P. Lynch, M. De La Torre, and others, “6.3 A Heterogeneous 3D-IC consisting of two 28nm FPGA die and 32 reconfigurable high-performance data converters,” in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, 2014, pp. 120–121.

[19] M. Duranton, K. De Bosschere, A. Cohen, J. Maebe and H. Munk, “On the road for the HiPEAC Vision 2015,” available online: http://www.hipeac.net/system/files/hp-roadmap-2015-draft.pdf

[20] “IEEE Standard VHDL Language Reference Manual," *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pp.c1,626, Jan. 26 2009

[21] Calliope-Louisa Sotiropoulou, Spiridon Nikolaidis, "Design space exploration for FPGA-based multiprocessing systems," *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on* , pp.1164-1167, 12-15 Dec. 2010

[22] Calliope-Louisa Sotiropoulou, Spiridon Nikolaidis, "ILP formulation for hybrid FPGA MPSoCs optimizing performance, area and memory usage," *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on* , vol., no., pp.748-751, 11-14 Dec. 2011

[23] H. Nikolov, T. Stefanov and E. Deprettere, “Systematic and Automated Multiprocessor System Design, Programming and Implementation,” IEEE Transactions on Cumputer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 3, pp. 542-555, Mar. 2008.

[24] M. J. Rutten et al., “A heterogeneous multiprocessor architecture for flexible media processing,” IEEE Des. Test. Comput., Vik, 19, no 4, pp. 39-50, Jul./Aug. 2002.
[25] T. Kodaka, K. Kimura and H. Kasahara, “Multigrain Parallel Processing for JPEG Encoding on a Single Chip Multiprocessor,” in Proc. International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, 2002, pp. 57-63.
[26] J. Wu, J. Williams, N. Bergmann and P. Sutton, “Design Exploration for FPGA-based Multiprocessor Architecture: JPEG Encoding Case Study,” in Proc. 17th IEEE Symposium on Field Programmable Custom Computing Machines, FCCM ’09, pp. 299-302.
[27] Y. Cho, G. Lee, S. Yoo, K. Choi, N.-E. Zergainoh, "Scheduling and timing analysis of HW/SW on-chip communication in MP SoC design," Design, Automation and Test in Europe Conference and Exhibition, 2003 , pp. 132- 137 suppl., 2003
[28] H. Yang, S. Ha, "ILP based data parallel multi-task mapping/scheduling technique for MPSoC," SoC Design Conference, 2008. ISOCC '08. International , vol.01, pp.I-134-I-137, 24-25 Nov. 2008
[29] S. L. Shee, S. Parameswaran, "Design Methodology for Pipelined Heterogeneous Multiprocessor System," Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE , pp.811-816, 4-8 June 2007
[30] H. Javaid, A. Ignjatovic, S. Parameswaran, "Rapid Design Space Exploration of Application Specific Heterogeneous Pipelined Multiprocessor Systems," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.29, no.11, pp.1777-1789, Nov. 2010
[31] I. Kadayif, M. Kandemir, U. Sezer, "An integer linear programming based approach for parallelizing applications in on-chip multiprocessors," Design Automation Conference, 2002. Proceedings. 39th , pp. 703- 708, 2002
[32] S.-R. Kuang, C.-Y. Chen, R.-Z. Liao, "Partitioning and Pipelined Scheduling of Embedded System Using Integer Linear Programming," Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on , vol.2, pp.37-41, 22-22 July 2005
[33] G. Theodoridis, N. Vassiliadis, S. Nikolaidis, "An integer linear programming model for mapping applications on hybrid systems," Computers & Digital Techniques, IET , vol.3, no.1, pp.33-42, January 2009
[34] J. Wu, J. Williams, N. Bergmann, "An ILP formulation for architectural synthesis and application mapping on FPGA-based hybrid multi-processor SOC," Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on , pp.451-454, 8-10 Sept. 2008
[35] Xilinx MicroBlaze, Xilinx Inc.,http://www.xilinx.com, 2011
[36] Xilinx Inc., “MicroBlaze Processor Reference Guide,” available online: http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf
[37] Xilinx Inc., “LogiCORE IP Fast Simplex Link (FSL) V20 Bus,” product specification, available online: http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf
[38] Xilinx Inc., “IP Processor Block RAM”, http://www.xilinx.com/support/documentation/ip_documentation/bram_block.pdf
[39] Lp_solve reference guide [Online], http://lpsolve.sourceforge.net/5.5
[40] Kumar, A., Li Shang, Li-Shiuan Peh, Jha, N.K.: HybDTM: a coordinated hardware-software approach for dynamic thermal management. In: 43rd ACM/IEEE Design Automation Conference, pp.548-553 (2006)
[41] Stavrou, K., Trancoso, P.: Thermal-aware scheduling for future chip multiprocessors. In: *EURASIP Journal of Embedded Systems., Vol. 2007, Article ID 48926, 15 pages, Hindawi* (2007)
[42] Sassolas, T., Ventroux, N., Boudouani, N, Blanc G.: A power-aware online scheduling algorithm for streaming applications in embedded MPSoC. In: *20th International Conference on Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation* (PATMOS'10), van Leuken and Gilles Sicard (Eds.). Springer-Verlag, pp.1-10 (2010).
[43] Bhoj, S., Bhatia, D.: Thermal Modeling and Temperature Driven Placement for FPGAs. In: IEEE International Symposium on Circuits and Systems (ISCAS), pp.1053-1056, IEEE press (2007)
[44] Liu, Z., Bian, J., Zhou, Q.: A thermal-aware ILP-based algorithm in behavioral synthesis. In: 7th International Conference on ASIC (ASICON), pp.1178-1181 (2007)
[45] Coskun, A.K., Rosing, T.T., Whisnant, K.A., Gross, K.C.: Static and Dynamic Temperature-Aware Scheduling for Multiprocesor SoCs. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 55(7), pp.1127 – 1140, IEEE Press, New York (2008)
[46] Mohanty, S., Ranganathan, N., Chappidi, S.: ILP models for simultaneous energy and transient power minimization during behavioral synthesis. In: *ACM Transactions on Design Automation of Electronic Systems, Vol. V, No. N, pp.111-135, ACM (2005).*

[47] D. S. Chen, R. G. Batson, and Y. Dang, *Applied integer programming*. Wiley Online Library, 2010.
[48] Papadimitriou, Christos H. "The Euclidean travelling salesman problem is NP-complete." *Theoretical Computer Science* 4.3 (1977): 237-244.
[49] S. Bayliss, C. S. Bouganis, G. A. Constantinides, and W. Luk, "An FPGA implementation of the simplex algorithm," in *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*, 2006, pp. 49–56.
[50] S. Mittal, L. Wang, A. Pande, and P. Kumar, "Design Exploration and Implementation of Simplex Algorithm over Reconfigurable Computing Platforms."
[51] S. Varsamopoulos, "Implementation of FPGA Accelerator for the Simplex Algorithm," available online: http://invenio.lib.auth.gr/record/135071?ln=en
[52] Xilinx Device Package User Guide, http://www.xilinx.com/support/documentation/user_guides/ug112.pdf
[53] Xilinx Power Solutions, http://www.xilinx.com/products/technology/power/index.htm
[54] Altera PowerPlay Early Power Estimators (EPE) and Power Analyzer, http://www.altera.com/support/devices/estimator/pow-powerplay.jsp
[55] S. J. Lee and S. Y. Lee., "Micro total analysis system (μ-TAS) in biotechnology," *Applied Microbiology and Biotechnology*, vol. 64, no. 3, pp. 289–299, Apr. 2004.
[56] C. Zhang, J. Xu, W. Ma, and W. Zheng, "PCR microfluidic devices for DNA amplification," *Biotechnology Advances*, vol. 24, no. 3, pp. 243-284, 2006.
[57] F. Sapuppo, F. Schembri, L. Fortuna, and M. Bucolo, "Microfluidic circuits and systems," *IEEE Circuits Syst. Mag.*, vol.9, no.3, pp.6-19, Third Quarter 2009.
[58] Lab-on-chip gene quantification (use of LoC figure), http://lab-on-chip.gene-quantification.info/
[59] Lab-on-chip, (use of LoC photo) http://en.wikipedia.org/wiki/Lab-on-a-chip
[60] N. T. Nguyen and T. Q. Truong, "Flow Rate Measurement in Microfluidics Using Optical Sensors," *1st International Conf. on Sensing Technology*, Palmerston North, 2005.
[61] P. Gravesen, J. Branebjer and O. Jensen, "Microfluidics - A review", J. Micromech. Microeng., Vol. 3, pp. 168-182, 1993
[62] S. Yong-Jun and L. Jeong-Bong, "Digital microfluidics-based high-throughput imaging for systems biology," *IEEE Sensors*, pp. 1202-1205, 2008.
[63] Y.-J. Shin, and J.-B. Lee, "Machine vision for digital microfluidics," *Review of Scientific Instruments*, vol. 81, no. 1, Jan. 2010.
[64] E. T. Dimalanta, A. Lim, R. Runnheim, C. Lamers, C. Churas, D. K. Forrest, J. J. de Pablo, M. D. Graham, S. N. Coppersmith, S. Goldstein, and D. C. Schwartz, "A Microfluidic System for Large DNA Molecule Arrays," *Analytical Chemistry*, vol. 76, no. 18, pp. 5293-5301, 2004.
[65] F. Merola, L. Miccio, P. Memmolo, M. Paturzo, S. Grilli, and P. Ferraro, "Simultaneous Optical Manipulation, 3-D Tracking, and Imaging of Micro-Objects by Digital Holography in Microfluidics," *Photonics Journal, IEEE* , vol.4, no. 2, pp.451-454, Apr. 2012.
[66] S. Batabyal, S. Rakshit, S. Kar, and S. K. Pal, "An improved microfluidics approach for monitoring real-time interaction profiles of ultrafast molecular recognition," *Review of Scientific Instruments*, vol. 83, no. 4, p. 43113, 2012.
[67] H. Uvet, T. Arai, Y. Mae, T. Takubo, and M. Yamada, "Miniaturized vision system for microfluidic devices," *Advanced Robotics*, vol. 22, no. 11, pp. 1207–1223, 2008.
[68] G. Kornaros, "A soft multi-core architecture for edge detection and data analysis of microarray images," *Journal of Systems Architecture*, vol. 56, no. 1, pp. 48-62, 2010.
[69] F. Sapuppo, M. Intaglietta, and M. Bucolo, "Bio-microfluidics real-time monitoring using CNN technology," *IEEE Trans. Biomed. Circuits Syst.*, vol. 2, no. 2, pp. 78-87, 2008.
[70] J.F. Canny, "A computation approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no 6, pp. 769-798, November 1986
[71] W. He and K. Yuan, "An improved Canny Edge Detector and its Realization on FPGA," Proc. of 7th World Conference on Intelligent Control and Automation, 2008
[72] H. Zeljko, V. Suzana and H. Verica, "Improved Canny Edge Detector in Ceramic Tiles Defect Detection," IEEE Industrial Electronics, IECON 2006 – 32nd Annual Conference, pp. 3328-3331, November 2006
[73] Y. Luo and R. Duraiswami, "Canny Edge Detection on NVIDIA CUDA," Proc. Of IEEE Computer Vision and Pattern Recognition Workshops, 2008 , pp. 1-8
[74] H.S. Neoh and A. Hazanchuk, "Adaptive Edge Detection for Real-Time Video Processing using FPGAs," Altera Corp.

[75] D. V. Rao and M. Venkatesan, “An Efficient Reconfigurable Architecture and Implementation of Edge Detection Algorithm Using Handle-C,” Proc of International Conference on Information Technology: Coding and Computing, ITCC 2004, Vol. 2, pp. 843-847
[76] Shi, Yu, and Timothy Tsui. "An FPGA-based smart camera for gesture recognition in HCI applications." *Computer Vision–ACCV 2007*. Springer Berlin Heidelberg, 2007. 718-727.
[77] Lu, Xiaofeng, Diqi Ren, and Songyu Yu. "FPGA-based real-time object tracking for mobile robot." *Audio Language and Image Processing (ICALIP), 2010 International Conference on*. IEEE, 2010.
[78] Grull, Frederik, et al. "Accelerating image analysis for localization microscopy with FPGAs." *Field Programmable Logic and Applications (FPL), 2011 International Conference on*. IEEE, 2011.
[79] McCurry, Peter, Fearghal Morgan, and Liam Kilmartin. "Xilinx FPGA implementation of an image classifier for object detection applications." *Image Processing, 2001. Proceedings. 2001 International Conference on*. Vol. 3. IEEE, 2001.
[80] R. Swenson and K. Dimond, “A hardware FPGA implementation of a 2D median filter using a novel rank adjustment technique,” in *Image Processing And Its Applications, 1999. Seventh International Conference on (Conf. Publ. No. 465)*, vol. 1, 1999, pp. 103–106.
[81] P. Wei, L. Zhang, C. Ma, and T. S. Yeo, “Fast median filtering algorithm based on FPGA,” in *Signal Processing (ICSP), 2010 IEEE 10th International Conference on*, 2010, pp. 426–429.
[82] S. A. Fahmy, P. Y. Cheung, and W. Luk, “Novel FPGA-based implementation of median and weighted median filters for image processing,” in *Field Programmable Logic and Applications, 2005. International Conference on*, 2005, pp. 142–147.
[83] S. A. Fahmy, P. Y. Cheung, and W. Luk, “High-throughput one-dimensional median and weighted median filters on FPGA,” *IET computers & digital techniques*, vol. 3, no. 4. IET, pp. 384–394, 2009.
[84] A. Demiris, S. Blionas, "Integrated System for the Visual Control, Quantitative and Qualitative Flow Measurement in Microfluidics," Hellenic Industrial Property Organisation Patent 20110100390, July 7, 2011
[85] Code::Blocks IDE: http://www.codeblocks.org/
[86] Minimalist GNU for Windows: http://www.mingw.org/
[87] E. Strataki, “Διερεύνηση αρχιτεκτονικών για την υλοποίηση του αλγορίθμου Canny,” Master Thesis, available online: http://electronics.physics.auth.gr/tomeas/thesis/msc/Strataki_Thesis.pdf
[88] C. L. Sotiropoulou, L. Voudouris, C. Gentsos, S. Nikolaidis, N. Vassiliadis, A. Demiris, and S. Blionas, “FPGA-based machine vision implementation for Lab-on-Chip flow detection,” in *Proc. 2012 IEEE International Symp. Circuits and Systems*, pp. 2047–2050, 2012.
[89] J. Canny, "A Computational Approach to Edge Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no.6, pp.679-698, Nov. 1986
[90] C. Gentsos, C. L. Sotiropoulou, S. Nikolaidis, and N. Vassiliadis, “Real-time canny edge detection parallel implementation for FPGAs,” in *Proc. 17th IEEE International Conf. Electronics, Circuits, and Systems*, pp. 499-502, 2010.
[91] Calliope-Louisa Sotiropoulou, Christos Gentsos, Spiridon Nikolaidis, “FPGA-based Canny Edge Detection for Real-Time Applications,” published at the 26th Conference on Design of Circuits and Integrated Systems (DCIS), Albufeira, Portugal, 2011.
[92] R. O. Duda, and P. E. Hart, “Use of the Hough Transform to detect lines and curves in pictures,” *Communications of the ACM*, Vol. 15, pp.11-15, Jan. 1972
[93] Z.-H. Chen, A. W. Y. Su, M.-T. Sun, "Resource-Efficient FPGA Architecture and Implementation of Hough Transform," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no.8, pp.1419-1428, Aug. 2012
[94] L. Voudouris, S. Nikolaidis, and A. Rjoub, "High speed FPGA implementation of hough transform for real-time applications," in *Proc IEEE 15th International Symp. Design and Diagnostics of Electronic Circuits & Systems*, pp.213-218, 2012.
[95] Xilinx Inc., “LogiCore IP Divider Generator ver. 4.0,” Product Specification, available online: http://www.xilinx.com/support/documentation/ip_documentation/div_gen/v4_0/ds819_div_gen.pdf
[96] L. Voudouris, C.-L. Sotiropoulou, N. Vassiliadis, A. Demiris, and S. Nikolaidis, "High-speed FPGA-based flow detection for microfluidic Lab-on-Chip," in *Proc. IEEE 20th Mediterranean Conf. Control & Automation (MED)*, pp.1434-1439, Barcelona, 2012.
[97] Xilinx Inc., “Spartan 6 FPGA Family overview,” Product Overview, DS160 (v2.0), 2011, Available: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf
[98] Xilinx Inc., “Integrating a Video Frame Buffer Controller (VFBC) in System Generator,” Application Note, XAPP1136 (v1.0), 2009, Available: http://www.xilinx.com/support/documentation/application_notes/xapp1136.pdf
[99] Xilinx Inc., “XPower Analyzer Overview,” Help File, Available:

http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/isehelp_start.htm#xpa_c_overview.htm

[100] Jai Ltd., "CV-M71 CL Progressive Scan RGB Colour Camera," Available: http://www.1stvision.com/cameras/JAI/dataman/Final_CV-M71CL_screen.pdf

[101] Y. S. Kumar, "Canny Edge Detection Implementation On TMS320C64x/64x+ Using VLIB," Texas Instruments Inc., Application Report SPRAB78, Nov. 2009.

[102] K. Ogawa, Y. Ito, and K. Nakano, "Efficient canny edge detection using a gpu," in *Proc. Networking and Computing (ICNC)*, pp. 279–280, 2010.

[103] X. Li, J. Jiang, and Q. Fan, "An improved real-time hardware architecture for Canny edge detection based on FPGA," in *Proc. Intelligent Control and Information Processing (ICICIP)*, pp. 445–449, 2012.

[104] M. Khan, A. Bais, K. Yahya, G. Hassan, and R. Arshad, "A swift and memory efficient Hough transform for systems with limited fast memory," *Image Analysis and Recognition*, pp. 297–306, 2009.

[105] G.-J. van den Braak, C. Nugteren, B. Mesman, and H. Corporaal, "Fast hough transform on GPUs: exploration of algorithm trade-offs," in *Proc. Of Advanced Concepts for Intelligent Vision Systems*, pp. 611–622, 2011.

[106] C.-L. Sotiropoulou, S. Gkaitatzis, A. Annovi, M. Beretta, P. Giannetti, K. Kordas, P. Luciano, S. Nikolaidis, C. Petridou and G. Volpi "A Multi-Core FPGA-based 2D-Clustering Implementation for Real-Time Image Processing", in IEEE Trans. on *Nuclear Science, vol. 61, no. 6, pp. 1-8, December 2014, doi:* 10.1109/TNS.2014.2364183.

[107] Y. Okumura, C.-L. Sotiropoulou et. al., "ATCA-based ATLAS FTK Input Interface System", *Journal of Instrumentation*, IOP Publishing, TWEPP 2014

[108] C.-L. Sotiropoulou, S. Gkaitatzis, A. Annovi, M. Beretta, K. Kordas, S. Nikolaidis, C. Petridou and G. Volpi, "A Parallel FPGA Implementation for Real-Time 2D Pixel Clustering for the ATLAS Fast TracKer Processor", *Journal of Instrumentation*, IOP Publishing, JINST 9 C10018 doi:10.1088/1748-0221/9/10/C10018.

[109] A. Annovi, C.-L. Sotiropoulou et al., "Design of a hardware track finder (Fast TracKer) for the ATLAS trigger," *Journal of Instrumentation*, vol. 9, no. 01. IOP Publishing, p. C01045, 2014.

[110] N. Kimura, A. Annovi, C.-L. Sotiropoulou et al., "A Highly Parallel FPGA Implementation of a 2D-Clustering Algorithm for the ATLAS Fast TracKer (FTK) Processor", *Proceedings of IEEE Real Time Conference 2014, Nara, Japan.*

[111] The FTK Group, "The FTK: A Hardware Track Finder for the ATLAS Trigger", *Proceedings of IEEE Real Time Conference 2014, Nara, Japan.*

[112] C.-L. Sotiropoulou, A. Annovi, M. Beretta, P. Luciano, S. Nikolaidis, G. Volpi, "A Multi-Core FPGA-based Clustering Algorithm for Real-Time Image Processing," *Proceedings of IEEE NSS/MIC 2013, Seoul, Korea, pp.1-5*, Oct. 27 2013-Nov. 2 2013.

[113] C.-L. Sotiropoulou, S. Nikolaidis, A. Annovi, M. Beretta, G. Volpi, P. Giannetti and P. Luciano, "A Multi-Core FPGA-based 2D-Clustering Algorithm for High-Throughput Data Intensive Applications," *Proceedings of ICATPP 2013, Como, Italy*.

[114] D. Casagrande, M. Sassano, and A. Astolfi, "Hamiltonian-based clustering: Algorithms for static and dynamic clustering in data mining and image processing," *Control Systems, IEEE*, vol. 32, no. 4., pp. 74–91, Aug. 2012.

[115] A. Annovi et al., "A fast FPGA-based clustering algorithm for real time image processing," Nuclear Science Symposium Conference Record (NSS/MIC), IEEE, pp.4138-4141, Oct. 24 2009-Nov. 1 2009.

[116] N. N. Gopal and M. Karnan, "Diagnose brain tumor through MRI using image processing clustering algorithms such as Fuzzy C Means along with intelligent optimization techniques," in *Proc. IEEE International Conf. on Computational Intelligence and Computing Research (ICCIC)*, pp. 1–4, 2010.

[117] S. N. Sulaiman and N. A. M. Isa, "Adaptive fuzzy-K-means clustering algorithm for image segmentation," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 4, pp. 2661–2668, Nov. 2010.

[118] X. Fu, H. You, and K. Fu, "A statistical approach to detect edges in SAR images based on square successive difference of averages," *IEEE Geoscience and Remote Sensing Letters*, vol. 9, no. 6, pp. 1094–1098, Nov. 2012.

[119] M. Diwakar, P. K. Patel, and K. Gupta, "Cellular automata based edge-detection for brain tumor," in *Proc. International Conf. on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 53–59, 2013.

[120] C.-L. Sotiropoulou, L. Voudouris, C. Gentsos, A. M. Demiris, N. Vassiliadis, and S. Nikolaidis, "Real-Time Machine Vision FPGA Implementation for Microfluidic Monitoring on Lab-on-Chips." *IEEE Transactions on Biomedical Circuits and Systems*, *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 2, pp. 268-277, April 2014.

[121] H. M. Hussain, K. Benkrid, A. T. Erdogan, and H. Seker, "Highly parameterized K-means clustering on FPGAs: Comparative results with GPPs and GPUs," in *Proceedings of 2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 475–480, 2011.

[122] K. Shanthi, L. Ashok, A. Anandu, and B. Gokul Das, "FPGA Implementation of Image Segmentation Processor," in *Proceedings of 2nd International Conference on Emerging Trends in Engineering and Technology (ICETET)*, pp. 364–367, 2009.

[123] S. Klupsch, M. Ernst, S. A. Huss, I. S. und Systeme, M. Rumpf, and R. Strzodka, "Real time image processing based on reconfigurable hardware acceleration," in *Workshop Heterogeneous Reconfigurable Systems on Chip (SoC)*, 2002.

[124] R. Bannister, D. Gregg, and A. Simon Wilson, "Fpga implementation of an image segmentation algorithm using logarithmic arithmetic," in *48th Midwest Symposium Circuits and Systems*, vol. 1, pp. 810–813, 2005.

[125] K. Yamaoka, T. Morimoto, H. Adachi, T. Koide, and H. J. Mattausch, "Image segmentation and pattern matching based FPGA/ASIC implementation architecture of real-time object tracking," in *Proceedings of Asia and South Pacific Conference on Design Automation*, p. 6, 2006.

[126] A. Gregerson et al., "FPGA design analysis of the Clustering Algorithm for the CERN Large Hadron Collider," in *Proceedings of 17th IEEE Symposium on Field Programmable Custom Computing Machines, FCCM'09*, pp. 19–26, 2009.

[127] A. Wassatsch and R. Richter, "DCE3-An universal real-time clustering engine," in *Proceeding of 2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 3242–3245, 2012.

[128] Andreani, A. et al., "The FastTracker Real Time Processor and Its Impact on Muon Isolation, Tau and b-Jet Online Selections at ATLAS," *IEEE Transactions on Nuclear Science*, vol.59, no.2, pp. 348,357, April 2012.

[129] The ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider," *Journal of Instrumentation* 3 S08003, 2008.

[130] The ATLAS Collaboration, "ATLAS High-Level Trigger Data Acquisition and Controls Technical Design Report", CERN/LHCC/2003 - 022 (2003).

[131] E. Van der Bij, R. McLaren, and Z. Meggyesi. "S-LINK: A Prototype of the ATLAS Read-out Link," available online: https://cds.cern.ch/record/405075

[132] G. Aad et al., "ATLAS pixel detector electronics and sensors", Journal of Instrumentation, Vol. 3, P07007, July 2008.

[133] The ATLAS Collaboration, "ATLAS Insertable B-Layer Technical Design Report,", ATLAS-TDR-19, CERN-LHCC-2010-013, available online: https://cds.cern.ch/record/1552953

[134] D. Dobos, "Commissioning perspectives for the ATLAS pixel detector.", PhD Thesis, Fachbereich Physik, Universitat Dortmund (2007).

[135] The ATLAS Collaboration, "Fast TracKer (FTK) Technical Design Report,", ATLAS-TDR-021, CERN-LHCC-2013-007, available online: https://cds.cern.ch/record/1552953

[136] The FTK Collaboration, "Fast Track data formats, numbering schemes and internal memories," Internal Note

[137] J. Olsen, T. Liu, and Y. Okumura, "A full mesh ATCA-based general purpose data processing board," *Journal of Instrumentation*, vol. 9, no. 01. IOP Publishing, p. C01041, 2014.

[138] Xilinx Inc, "Spartan-6 Family Overview", available online: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[139] P. Haefner, "The ATLAS silicon microstrip tracker. Operation and performance," *Journal of Instrumentation*, vol. 5, no. 12. IOP Publishing, p. C12050, 2010.

[140] The ATLAS Collaboration, "Technical Design Report for the Phase-I Upgrade of the ATLAS TDAQ System," ATLAS-TDR-023, available online: https://cds.cern.ch/record/1602235

[141] The ATLAS collaboration and others, "A neural network clustering algorithm for the ATLAS silicon pixel detector," arXiv preprint arXiv:1406.7690, 2014.

[142] C.-L. Sotiropoulou, C. Gentsos and S. Nikolaidis, "High Performance Median FPGA Implementation For Machine Vision Applications," to be published in *Proc. IEEE International Conf. on Electronics, Circuits, and Systems (ICECS)*, 2013.

[143] Xilinx Inc., "LogiCORE IP FIFO Generator v9.3," Product Guide, available online: http://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v9_3/pg057-fifo-generator.pdf

[144] Mentor Graphics Inc., “ModelSim User’s Manual,” available online: http://www.microsemi.com/index.php?option=com_docman&task=doc_download&gid=13161

[145] Ancu, L. S., et al. *Associative Memory computing power and its simulation*. No. ATL-DAQ-PROC-2014-009. ATL-COM-DAQ-2014-049, 2014.

[146] Bertram, Iain, et al. (D0 Collaboration) "Search for the Standard Model Higgs Boson in the ZH-> vvbb Channel in 5.2 fb-1 of p-pbar Collisions at sqrt (s)= 1.96 TeV." *Physical Review Letters* 104.7 (2010).

[147] R. Cranfield, G. Crone, D. Francis, B. Gorini, B. Green, M. Joos, G. Kieft, A. Kugel, A. Misiejuk, M. Müller, and others, “The Atlas Robin,” *Journal of Instrumentation*, vol. 3, no. 01, p. T01002, 2008.

[148] S. Ballestero, et al., “ATCA in ATLAS”, Specification document, EDMS ATU-GE-ES-0001, available online: https://edms.cern.ch/document/1304001/1

[149] “FILAR: Quad HOLA S-LINK to 64-bit/66 MHz PCI Interface”, Specifications Description, available online: http://hsi.web.cern.ch/HSI/s-link/devices/filar/

[150] Xilinx Inc, “Artix-7 FPGA Product Brief”, available online: http://www.xilinx.com/publications/prod_mktg/Artix-7-Product-Brief.pdf

[151] A. Cassidy, A. G. Andreou, and J. Georgiou, “Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis,” in *Information Sciences and Systems (CISS), 2011 45th Annual Conference on*, 2011, pp. 1–6.

[152] Del Viva, Maria M., Giovanni Punzi, and Daniele Benedetti. "Information and Perception of Meaningful Patterns." *PloS one* 8.7 (2013): e69154.

Appendix A

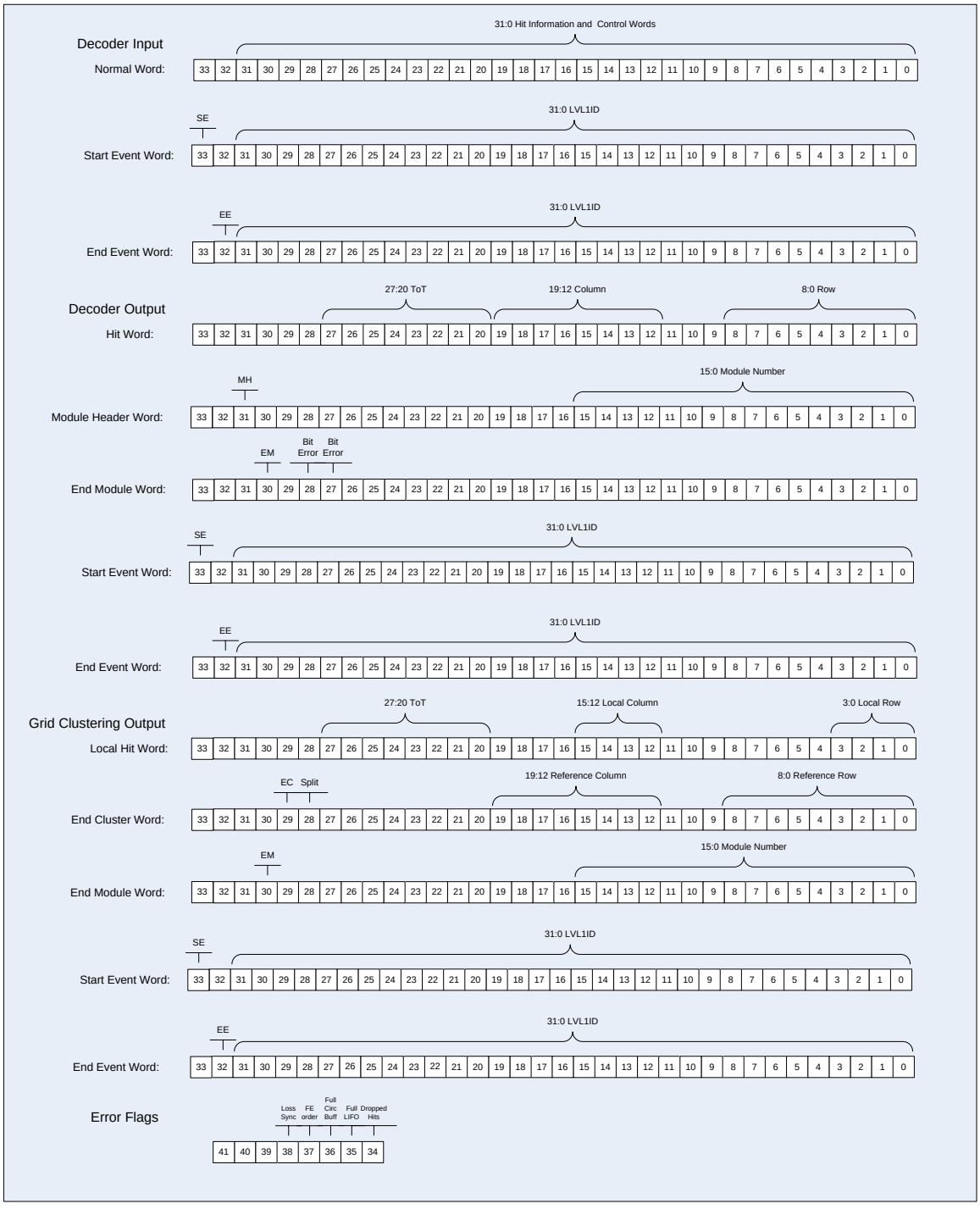


Figure 5.1. The 2D-Pixel Clustering Internal Data Format

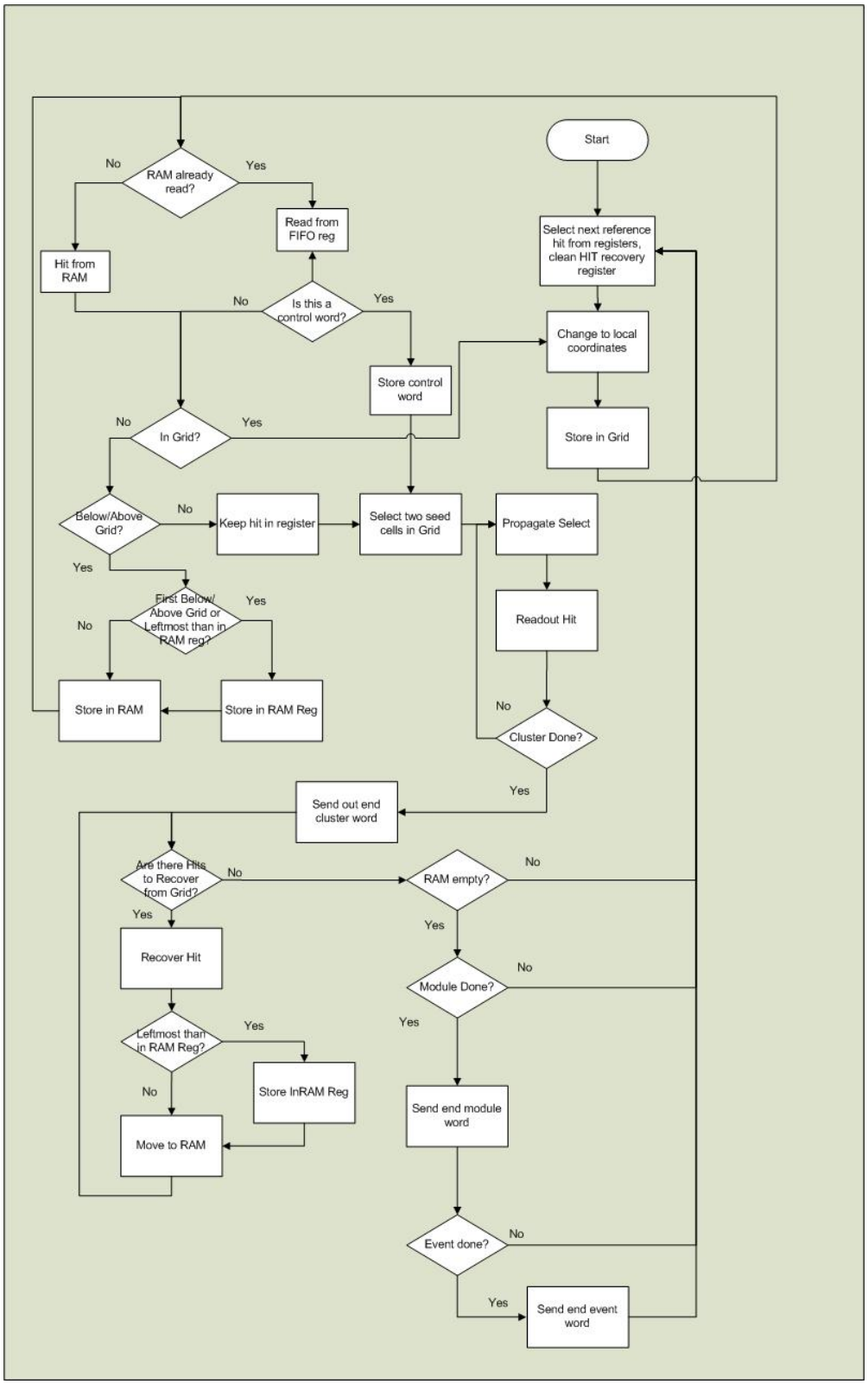


Figure 5.2. Grid Clustering Module Flow Chart

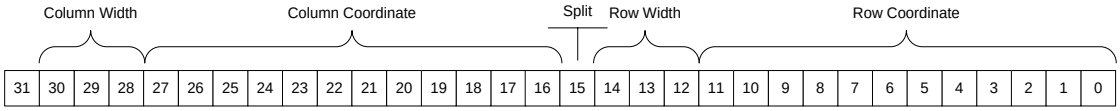


Figure 5.3. Pixel Centroid Data Format

TABLE 5.1. Bit by Bit Description of the 2D-Pixel Clustering Centroid Word Format

Bits	Content
0-11	Row Coordinate (Integer – Normalized Units of 6.25 um for Rows)
12-14	Row Width: 3 bits give us a range of 0 to 7. We consider minimum Row Width to be 1 and maximum Row width to saturate to 8. Explanation: Actual Row Width 1 → Representation "000" (0) Actual Row Width 2 → Representation "001" (1) Actual Row Width 3 → Representation "010" (2) ….. Actual Row Width 8 → Representation "111" (7) Actual Row Width ANY NUMBER OVER 8 → Representation "111" (7)
15	Potentially Split Cluster Bit (As defined in Grid Clustering Module)
16-27	Column Coordinate (Integer – Normalized Units of 25 um for Cols)
28-30	Col Width: 3 bits give us a range of 0 to 7. We consider minimum Col Width to be 1 and maximum Col width to saturate to 8. Explanation: Actual Col Width 1 → Representation "000" (0) Actual Col Width 2 → Representation "001" (1) Actual Col Width 3 → Representation "010" (2) ….. Actual Col Width 8 → Representation "111" (7) Actual Col Width ANY NUMBER OVER 8 → Representation "111" (7)

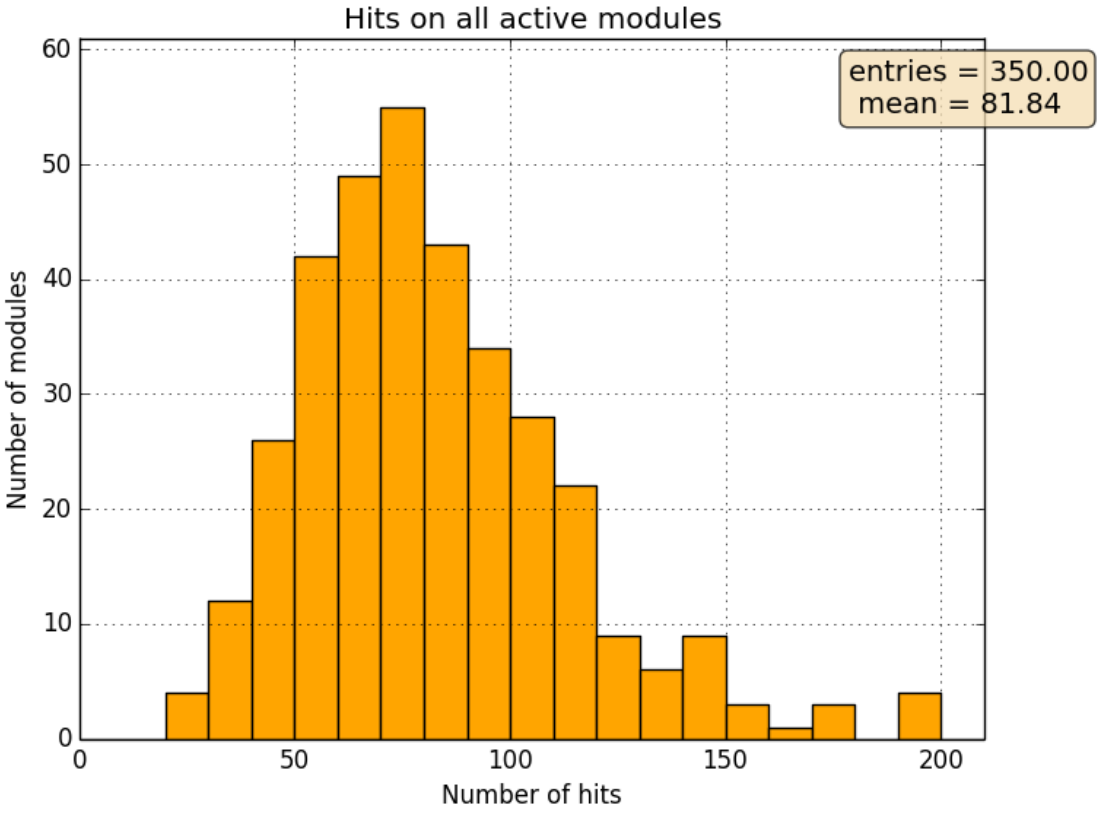


Figure 5.4. Hits per Pixel Module

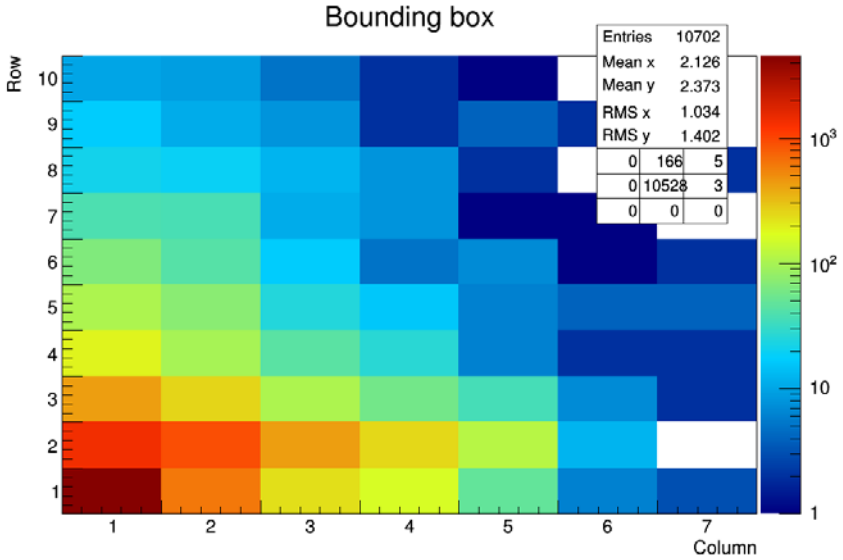


Figure 5.5. Bounding Box Size for Pixel Clusters

Appendix B

5.3. The FTK_IM Mezzanine Top Level

The FTK_IM Board has two Spartan-6 LX150T FPGA devices mounted on each board. The two FPGA devices have identical firmware implementations.

The top level block diagram of the FTK_IM FPGAs is presented in Figure 5.6. There are two parallel data flows implemented on each device, one for the SCT data processing and one for Pixel Data processing. Each data flow receives data from the Inner Decoders RODs through a GTP connection. The GTP data stream is handled by a SerDes module and following the SerDes module and S-LINK Receiver module (LDC) is implemented to decode the S-LINK data format. The output of the S-LINK Receiver module is a stream of data words with a clock cycle of 40 MHz.

The output of the S-LINK Decoder is captured by a SpyBuffer (the Input SpyBuffer of the IM Board) and is forwarder to the two ID_Cluster modules. A SpyBuffer is a circular buffer that continuously stores data processed from the system in strategically chosen points. SpyBuffers are used to monitor data processing and debug the system. The ID_Cluster modules are the wrappers that handle the interface to the two clustering modules (for SCT and Pixel). The output of the ID_Cluster modules is again captured by a SpyBuffer (the Output SpyBuffer of the IM Board) and also forwarded to the SenderDF module.

The SenderDF modules use a DDR logic working at 200 MHz to propagate data to the Data Formatter FPGA through an LVDS bus. The data are propagated at 50 MHz of 6 bits x 8 - 50 MHz x (32 bits data + control bits).

In addition there are control modules to control the available digital clocks (DCM, BCOmanager etc.) as well as modules to handle the communication interfaced between the IM board and the DF board. Two protocols are available: I^{2}C and SerDes. The I^2C is used to program the onboard memories, access the SpyBuffers and the monitoring registers. The SerDes protocol is a faster alternative to I^2C.

The FTK_IM JTAG chain (for FPGA programming etc.) is controlled by a DIP switch and can be either used through an external connector directly on the IM or through the FMC connector and the DF board.

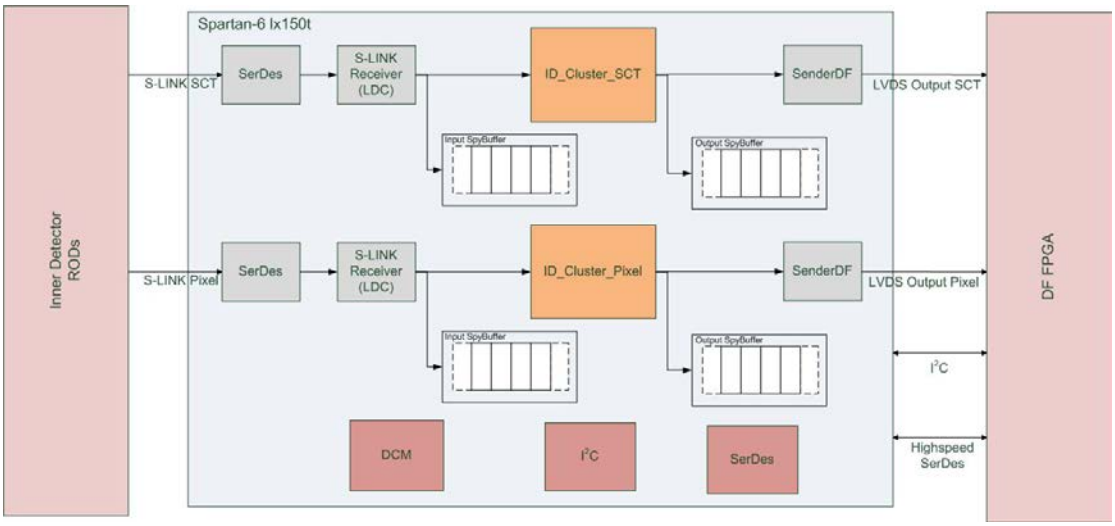


Figure 5.6. FTK_IM Firmware Top Level Block Diagram

5.4. The ID_Cluster Modules

The ID_Cluster Modules are the wrappers that act as an interface between the two clustering modules (SCT and Pixel) and the rest of the FTK_IM firmware (Figure 5.7).

The ID_Cluster wrapper for the SCT Clustering system consists of one Input FIFO, one Output FIFO and the interface that decodes the ATLAS Bytestream format for the start_event and end_event word packages to provide to the SCT Clustering module a clean stream of hits.

The ID_Cluster wrapper for the Pixel Clustering system is more complex. The arriving data stream goes through an input FIFO before it arrives to a demultiplexing module. This module divides the incoming bytes stream to two data paths: a hit data path and a header/trailer (control word) path. The hit data path's contents are the hit data, module header/trailer words and start/end event words in the data format expected by the Pixel Clustering system (see Figure 5.1). The data are first

propagated to the SYS FIFO from where they are read by the Pixel System. The control word data path propagates the begin-of-fragment (B0F) and end-of-fragment (E0F) word packets to the ROD FIFO.

The output of the Pixel Clustering System is stored to the DATA FIFO. A data multiplexer merges the contents of the ROD FIFO and the DATA FIFO back to a single stream. The LVL1 ids on both streams need to match at all times. The merged stream is stored to the OUT FIFO and from there is propagated to the SenderDF module to be sent to the DF board.

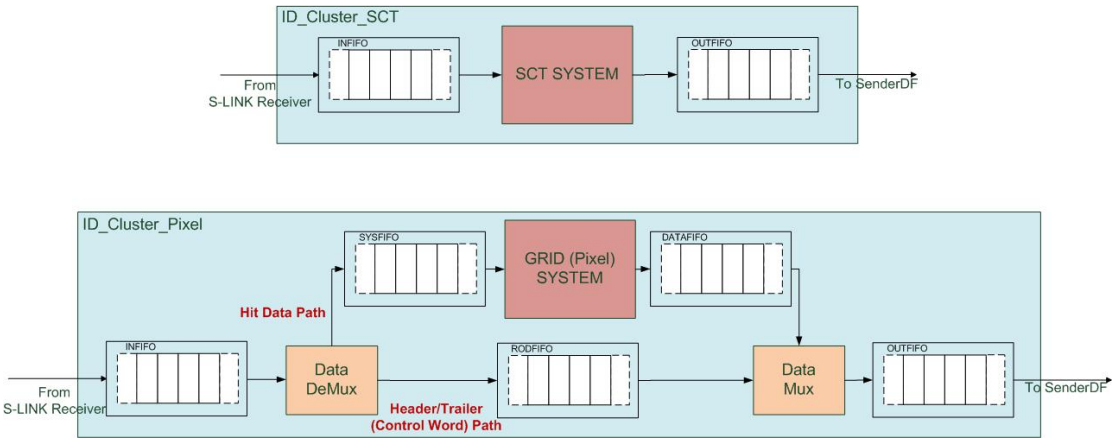


Figure 5.7. The ID_Cluster Modules Block Diagram

5.5. FTK_IM Monitoring Software (SpyMon)

The FTK_IM Monitoring Software (SpyMon) is a set of software functions developed by researchers from the University of Waseda (Japan) to extract monitoring information for the operation of the FTK_IM. The functions read the monitoring registers installed in the hardware and provide the information to the designer. Information includes status of all Finite State Machines, FIFO empty, full and almost full signals, current event number, previous event number, FIFO status and others.

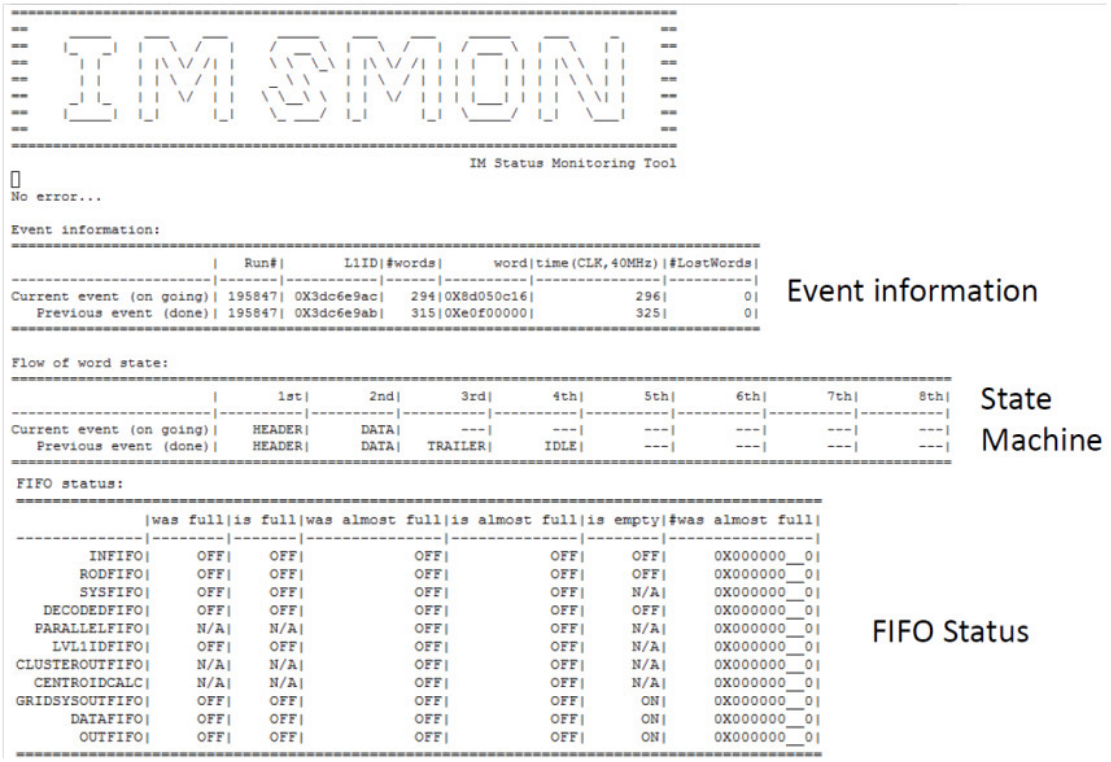


Figure 5.8. SpyMon Screenshot – No Backpressure and Errors Identified

Publications

International Peer Reviewed Journal Papers

1. **C.-L. Sotiropoulou**, S. Gkaitatzis, A. Annovi, M. Beretta, P. Giannetti, K. Kordas, P. Luciano, S. Nikolaidis, C. Petridou and G. Volpi "A Multi-Core FPGA-based 2D-Clustering Implementation for Real-Time Image Processing", in IEEE Trans. on *Nuclear Science, vol. 61, no. 6, pp. 1-8, December 2014, doi:* 10.1109/TNS.2014.2364183.
2. **C.-L. Sotiropoulou**, L. Voudouris, C. Gentsos, A. Demiris, N. Vassiliadis and S. Nikolaidis, "Real-Time Machine Vision FPGA Implementation for Microfluidic Monitoring on Lab-on-Chips," in IEEE Trans. on *Biomedical Circuits and Systems,* vol.8, no.2, pp.268,277, April 2014.

International Peer Reviewed Journal Papers from Conference Proceedings

3. Y. Okumura, **C.-L. Sotiropoulou** et. al., "ATCA-based ATLAS FTK Input Interface System", *Journal of Instrumentation,* IOP Publishing, TWEPP 2014
4. **C.-L. Sotiropoulou**, S. Gkaitatzis, A. Annovi, M. Beretta, K. Kordas, S. Nikolaidis, C. Petridou and G. Volpi, "A Parallel FPGA Implementation for Real-Time 2D Pixel Clustering for the ATLAS Fast TracKer Processor", *Journal of Instrumentation,* IOP Publishing, JINST 9 C10018 doi:10.1088/1748-0221/9/10/C10018.

   (cited 1 time - Scholar)
5. A. Annovi, A. Andreani, V. Libarali, **C.-L. Sotiropoulou** et al., "The Associative Memory Serial Link Processor for the Fast TracKer (FTK) at ATLAS", *Journal of Instrumentation*, vol. 9, no. 01. IOP Publishing, p. C11006, 2014.
6. P. Luciano, **C.-L. Sotiropoulou** et al., "The Serial Link Processor for the Fast TracKer (FTK) at ATLAS", *Proceedings of Science,* TIPP 2014 Conference, June 2014, ATL-DAQ-PROC-2014-019.
7. A. Annovi, **C.-L. Sotiropoulou** et al., "Design of a hardware track finder (Fast TracKer) for the ATLAS trigger," *Journal of Instrumentation*, vol. 9, no. 01. IOP Publishing, p. C01045, 2014.

Proceedings from International Peer Reviewed Conferences

8. N. Kimura, A. Annovi, **C.-L. Sotiropoulou** et al., "A Highly Parallel FPGA Implementation of a 2D-Clustering Algorithm for the ATLAS Fast TracKer (FTK) Processor", *Proceedings of IEEE Real Time Conference 2014, Nara, Japan.*
9. The FTK Group, "The FTK: A Hardware Track Finder for the ATLAS Trigger", *Proceedings of IEEE Real Time Conference 2014, Nara, Japan.*
10. **C.-L. Sotiropoulou**, C. Gentsos and S. Nikolaidis, "High Performance Median FPGA Implementation For Machine Vision Applications," *Proceedings of IEEE ICECS 2013,* pp. 173-176, 8-11 Dec. 2013, Abu Dhabi, UAE.
11. **C.-L. Sotiropoulou**, A. Annovi, M. Beretta, P. Luciano, S. Nikolaidis, G. Volpi, "A Multi-Core FPGA-based Clustering Algorithm for Real-Time Image Processing," *Proceedings of IEEE NSS/MIC 2013, Seoul, Korea, pp.1-5,* Oct. 27 2013-Nov. 2 2013.

    (cited 1 time - Scholar)
12. **C.-L. Sotiropoulou**, S. Nikolaidis, A. Annovi, M. Beretta, G. Volpi, P. Giannetti and P. Luciano, "A Multi-Core FPGA-based 2D-Clustering Algorithm for High-Throughput Data Intensive Applications," *Proceedings of ICATPP 2013, Como, Italy.*
13. L. Voudouris, **C.-L. Sotiropoulou**, N. Vassiliadis, A. Demiris, and S. Nikolaidis, "High-speed FPGA-based flow detection for microfluidic Lab-on-Chip," in *Proc. IEEE 20th Mediterranean Conf. Control & Automation (MED)*, pp. 1434-1439, Barcelona, 2012.
14. **C.-L. Sotiropoulou**, L. Voudouris, C. Gentsos, S. Nikolaidis, N. Vassiliadis, A. Demiris, and S. Blionas, "FPGA-based machine vision implementation for Lab-on-Chip flow detection," in *Proc. 2012 IEEE International Symp. Circuits and Systems*, pp. 2047–2050, 2012.

    (cited 1 time - Scholar)

15. **Calliope-Louisa Sotiropoulou**, Spiridon Nikolaidis, "ILP formulation for hybrid FPGA MPSoCs optimizing performance, area and memory usage," *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on* , vol., no., pp.748-751, 11-14 Dec. 2011

    (cited 3 times - Scholar)

16. **Calliope-Louisa Sotiropoulou,** Christos Gentsos, Spiridon Nikolaidis, "FPGA-based Canny Edge Detection for Real-Time Applications," published at the 26th Conference on Design of Circuits and Integrated Systems (DCIS), Albufeira, Portugal, 2011.

17. **Calliope-Louisa Sotiropoulou**, Spiridon Nikolaidis, "Design space exploration for FPGA-based multiprocessing systems," *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on* , pp.1164-1167, 12-15 Dec. 2010

    (cited 2 times – Scopus, 3 times - Scholar)

18. C. Gentsos, **C.-L. Sotiropoulou**, S. Nikolaidis, and N. Vassiliadis, "Real-time canny edge detection parallel implementation for FPGAs," in *Proc. 17th IEEE International Conf. Electronics, Circuits, and Systems*, pp. 499-502, 2010.

    (cited 11 times – Scopus, 12 times - Scholar)

Proceedings from National Peer Reviewed Conferences

19. **Calliope-Louisa Sotiropoulou**, Nikolaos Vassiliadis, George Theodoridis, Spiridon Nikolaidis, "JPEG Implementation Through Programming of an ARISE Machine," 1st Pan-Hellenic Conference in Electronics and Telecommunications (PACET 2009), Patras, March 2009.

Conference-Workshop Presentations

(1) HiPEAC Computing Systems Week, Athens, Greece, October 8-10, 2014
Co-organizer of thematic session “High Performance Embedded Systems for High Energy Physics Applications”, one presentation

(2) International Conference on Technology and Instrumentation in Particle Physics – TIPP 2014, Amsterdam, The Netherlands, June 2-6, 2014
One presentation

(3) Scientific Workshop, “3rd Workshop on Modern Circuits and Systems Technologies,” Thessaloniki, Greece, Aristotle University of Thessaloniki, March 14th-15th , 2014
One presentation

(4) 2013 IEEE Nuclear Science Symposium and Medical Imaging Conference, Seoul, Korea, October 27-November 2, 2013
One poster presentation

(5) 14th ICATPP Conference on Astroparticle, Particle, Space Physics and Detectors for Physics Applications, Como Italy, September 23-27, 2013
One presentation

(6) Scientific Workshop, “2nd Workshop on Modern Circuits and Systems Technologies,” Thessaloniki, Greece, Aristotle University of Thessaloniki, February 1st, 2013
One presentation

(7) 2012 IEEE International Symposium on Circuits and Systems (ISCAS), Seoul, Korea, May 20-23
Poster presentation

(8) Scientific Workshop, “Modern Circuits and Systems Technologies,” Thessaloniki, Greece, Aristotle University of Thessaloniki, March 16 2012
One presentation

(9) 18th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2011, Beirut, Lebanon, December 11-14 2011
Poster presentation

(10) Scientific Workshop, “Current Trends in Nanoscale VLSI Design,” Irbid, Jordan, Jordan University of Science and Technology, October 24-27 2011

One paper presentation

(11) 17th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2010, Athens, Greece, December 12-15, 2010
Two presentations

(12) 1st Pan-Hellenic Conference in Electronics and Telecommunications 2009, PACET 2009, Patras, March 20-22 2009
One presentation

Awards and Distinctions

- Best presentation award (1st place) by a young researcher in the International Conference on Technology and Instrumentation in Particle Physics – TIPP 2014 (sponsored by Elsevier - NIM) **June 2014**
- Scholarship for academic achievement in post-graduate level from the "Nikolaos H. Kastanidis" Foundation **2008-2009**

End of an era….

Start of a new one….