



University
of Glasgow

Sims, Oliver (2007) *Efficient implementation of video processing algorithms on FPGA.*

EngD thesis

<http://theses.gla.ac.uk/4119/>

Copyright and moral rights for this thesis are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the Author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the Author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Efficient Implementation of Video Processing Algorithms on FPGA

Volume 1 (of 2)

Oliver Sims

A themed portfolio submitted to
The Universities of

Edinburgh,
Glasgow,
Heriot-Watt,
and Strathclyde

for the degree of
Doctor of Engineering in System Level Integration

© Oliver Sims, 2007



Abstract

The work contained in this portfolio thesis was carried out as part of an Engineering Doctorate (EngD) programme from the Institute for System Level Integration. The work was sponsored by Thales Optronics, and focuses on issues surrounding the implementation of video processing algorithms on field programmable gate arrays (FPGA).

A description is given of FPGA technology and the currently dominant methods of designing and verifying firmware. The problems of translating a description of behaviour into one of structure are discussed, and some of the latest methodologies for tackling this problem are introduced.

A number of algorithms are then looked at, including methods of contrast enhancement, deconvolution, and image fusion. Algorithms are characterised according to the nature of their execution flow, and this is used as justification for some of the design choices that are made. An efficient method of performing large two-dimensional convolutions is also described.

The portfolio also contains a discussion of an FPGA implementation of a PID control algorithm, an overview of FPGA dynamic reconfigurability, and the development of a demonstration platform for rapid deployment of video processing algorithms in FPGA hardware.

Declaration

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a higher degree.

Signature: 

Date: 15/06/2007

Acknowledgements

I would like to acknowledge my academic supervisor Dr James Irvine, always supportive and enthusiastic, who never failed to make the EngD sound a lot easier.

I'm also very grateful to my sponsoring company Thales Optronics and everyone who made my time there so enjoyable. In particular I thank my supervisor Andrew Parmley, and also Stephen McGeoch and Douglas Gibson - both instrumental in establishing the EngD programme within the company. Special mention should also go to the ECAD group, who put up with my questions and were generous with cakes and sweets.

Finally, thanks to family and friends who offered encouragement and patience at all the right moments.

Contents

Abstract	ii
Declaration	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	ix
List of Abbreviations	x
Executive Summary	xiii
1 Introduction	1
1.1 Problem statement	2
1.2 Research goals	3
1.3 Contributions	4
1.4 Portfolio organisation	5
2 Taught Modules	7
2.1 Technical credits	7
2.2 Business credits	10
3 External Events and Training	12
3.1 Training	12
3.2 Conference publications	13

3.3	Industrial events	14
4	Commercial Relevance	16
4.1	Author's contribution	19
5	Technical Background	21
5.1	Field-programmable gate arrays	21
5.1.1	Floating-point performance	25
5.1.2	Dynamic reconfigurability	25
5.2	Video processing systems	26
5.2.1	Nature of video data	26
5.2.2	Nature of image processing algorithms	27
5.3	Algorithm implementation process	33
5.3.1	Manual tasks for high-level synthesis	34
5.3.2	Design methodologies	39
5.4	Related work	43
5.5	Summary	47
6	Results & Discussion	49
6.1	Implementation of LRM contrast enhance	49
6.1.1	Background	50
6.1.2	Hardware constraints	52
6.1.3	Transforming behaviour to structure in C	53
6.1.4	The control/datapath model	55
6.1.5	Translation to VHDL and synthesis	57
6.1.6	System Generator implementation of LRM algorithm	58
6.2	Implementation of RL deconvolution	59
6.2.1	Background	59
6.2.2	Efficient implementation of large convolutions	61
6.2.3	Hardware implementation	66
6.3	Implementation of pyramidal image fusion	68
6.3.1	Background	69

Contents	vii
6.3.2 Pyramid generation	69
6.3.3 Gradient pyramid fusion	73
6.3.4 Hardware implementation	74
6.4 Other algorithms	76
6.4.1 PID servo control algorithm	77
6.4.2 Image registration by polynomial warping	78
7 Design Guidelines	81
7.1 Convolution	81
7.2 Control- versus datapath-dominant code	82
7.3 Optimisations	83
7.4 General computation	84
8 Conclusion	85
8.1 Future direction	89
References	91
Appendices	98

List of Figures

1	Basic FPGA structure.	22
2	Sliding window method of convolution	29
3	Example image filters	29
4	Multiscale decomposition of well-known image “Lena”.	31
5	Two-dimensional interpolation	32
6	Transformation from specification to hardware	35
7	System Generator design flow.	41
8	Demonstration of contrast enhancement through a linear stretch . . .	51
9	Partitioning of input image into blocks and nodes for LRM contrast enhancement.	52
10	Block diagram of a system with separate control and datapath.	55
11	Surface diagram of an 11×11 point spread function.	60
12	Summation of k separable filters to produce a non-separable response.	63
13	Increase in accuracy with successive stages of a multistage filter . . .	65
14	Example image before and after RL deconvolution	67
15	Difference image between Matlab and hardware implementations of Richardson-Lucy deconvolution	68
16	Pipelined image pyramid generation.	72
17	Fusion of gradient pyramids.	73

List of Tables

1	Breakdown of technical credits	8
2	Breakdown of business credits	10
3	Training courses attended during the EngD programme	13
4	Conferences attended during the EngD programme	14
5	Industrial events attended during the EngD programme	15
6	Resource usage of LRM algorithm.	58
7	Resource usage of RL deconvolution algorithm.	66
8	Resource usage of pyramidal image fusion.	75

List of Abbreviations

ANSI - American National Standards Institute

API - Application Programming Interface

ASIC - Application Specific Integrated Circuit

CAD - Computer Aided Design

CLB - Configurable Logic Block

CORDIC - COordinate Rotation DIgital Computer

COTS - Commercial Off-The-Shelf

CPLD - Complex Programmable Logic Devices

DCM - Digital Clock Manager

DCT - Direct Cosine Transform

DFT - Discrete Fourier Transform

DSP - Digital Signal Processing

EDA - Electronic Design Automation

EDIF - Electronic Design Interchange Format

EEPROM - Electronically Erasable Programmable Read-Only Memory

EngD - Engineering Doctorate

ESL - Electronic System Level

FFT - Fast Fourier Transform

FIFO - First-in, First-out

FIR - Finite Impulse Response

FPGA - Field Programmable Gate Array

fps - frames per second

FSM - Finite State Machine

FSMD - Finite State Machine with Datapath

GCD - Greatest Common Divisor
HDL - Hardware Design Language
HLSM - High-Level State Machine
ICAP - Internal Configuration Access Port
IOB - Input/Output Block
IP - Intellectual Property
JPEG - Joint Photographic Expert Group
JTAG - Joint Test Action Group
LRM - Local Range Modification
LSB - Least Significant Bit
LUT - Look-Up Table
MAC - Multiply ACcumulate
MBA - Master of Business Administration
MOSFET - Metal Oxide Silicon Field Effect Transistor
MPEG - Moving Pictures Expert Group
MSB - Most Significant Bit
MSE - Mean Squared Error
NRE - Non-Recurring Engineering
PAL - Phase Alternating Line (TV standard) or
 Programmable Array Logic (electronic device)
PAR - Place And Route
PCI - Peripheral Component Interconnect
PID - Proportional, Integral, Derivative
PROM - Programmable Read-Only Memory
PSF - Point Spread Function
RAM - Random Access Memory
RE - Research Engineer
ROM - Read-Only Memory
RTL - Register Transfer Level
RTOS - Real-Time Operating System
SAD - Sum of Absolute Difference

SCOTCAT - SCOTtish Credit Accumulation and Transfer

SDRAM - Synchronous Dynamic Random Access Memory

SIMD - Single Instruction Multiple Data

SoC - System-on-Chip

SVD - Singular Value Decomposition

SXGA - Super eXtended Graphics Array

VGA - Video Graphics Array

VHDL - VHSIC Hardware Design Language

VHSIC - Very High Speed Integrated Circuit

VLSI - Very Large Scale Integration

Executive Summary

The objective of this engineering doctorate (EngD) research project was to investigate some of the issues surrounding implementation of algorithms for real-time processing of video data on field programmable gate arrays (FPGA). This was initially quite a broad assignment covering similar related topics such as dynamic reconfigurability, but after an initial phase of exploratory research the brief was narrowed to look solely at the problems faced in translating a description of an algorithm written in a high-level language into an efficient hardware design. The adopted research methodology looks at the implementations of several algorithms of varying types, and attempts to draw conclusions that could be applied to the class of video processing algorithms in general.

The practical element of the project started with an investigation into the local range modification (LRM) method of contrast enhancement, an algorithm that had recently been implemented in hardware within Thales Optronics but had involved a lengthy and difficult design process. One of the immediate tasks facing the RE was to attempt to explain why this was the case. The same algorithm was implemented by the research engineer (RE) as part of the research effort, but rather than following a traditional design flow the C language was used as an intermediate step between algorithm and hardware; the C was written in a specific style that incorporated structural information and allowed the notion of concurrent processing to be demonstrated. This provided a means of implementing algorithms that contain complex execution flows, of which the LRM algorithm was an example, in reasonable timeframes, and an implementation was produced that was of comparable performance with hand-coded methods but completed in a fraction of the development time.

The same algorithm was also implemented using the Xilinx System Generator

tool, a methodology that at the time was gaining in popularity within the industry and appealed to Thales Optronics as a means of not only smoothing the design flow, but also allowing the algorithm engineers to experiment with hardware implementations. The aim of the work was to assess System Generator's suitability to the type of algorithms Thales Optronics are likely to implement in hardware. The results indicated that System Generator could potentially be a useful tool in this area, but is not universally applicable to image processing algorithms due to its somewhat limited library of in-built functions and propensity for algorithms that readily conform to a datapath style.

The next algorithm looked at was the Richardson-Lucy method of deconvolution, a method used to remove certain types of blur from an image. This algorithm could, with some minor manipulation, be classified as datapath dominant, and so was more susceptible to standard implementation methods. However, in order to reduce the high hardware resource requirements it was necessary to find an efficient means of implementing the several large two-dimensional convolutions that are required. The method that was employed, known as multistage separable filters, is shown to be extremely efficient when implementing a large range of image processing filters due to the common characteristic of a linear phase response.

The third major algorithm that was looked at is a method of performing image fusion using a multiscale decomposition known as Gaussian pyramids. Previously reported methods of producing image pyramids had been based on sequential processes, but using the insight gained from the need to remove the complex control flow from the design an alternative structure was developed. The design was based on the use of multiple clock rates to enable the various levels of the image pyramid to be generated and processed concurrently, and exhibited over a hundredfold speedup when compared to a PC-based implementation.

Other algorithms that were investigated include a PID servo control algorithm and a method of performing image warping that requires system-level functionality (a memory controller) to operate. Additional work was carried out to design a demonstration platform for video processing algorithms with an automated implementation flow, which provides a fast means of demonstrating System Generator

implementations of video processing algorithms in hardware.

Chapter 1

Introduction

Field programmable gate arrays (FPGA) have developed from niche electronic devices into a ubiquitous component of a wide variety of systems. As device geometries have shrunk the capabilities of programmable logic have increased, to the point where FPGAs have flexibility approaching that of general-purpose processors and performance nearly comparable to application specific integrated circuits (ASIC). Now FPGAs are not only filling the gap between these device categories, but are encroaching on the markets where they have traditionally been dominant.

The sponsor of this engineering doctorate (EngD), Thales Optronics, are active in the defence industry. The systems that are commonly developed within the industry must provide high levels of performance whilst accommodating the recurring constraints of low-power consumption, small physical size, robustness and reliability. It is not uncommon for systems to have a life span of 25 years, with upgrades taking place at fairly infrequent intervals over that period. Compared to the commercial sector, defence market volumes are extremely small, with the highest volume applications numbering in the thousands and the smallest possibly less than a hundred systems. This situation is forcing defence contractors to move away from bespoke manufacturing towards systems integration roles using commercial off-the-shelf (COTS) parts.

The products developed by Thales Optronics, historically based on purely optical systems, are gradually incorporating an ever increasing amount of electronics in order to provide intelligent automatic processing of data. A large amount of research effort within the company is now focused purely on development of cutting-edge

algorithms for making greater use of the data captured by imaging sensors. These algorithms form commercially sensitive intellectual property (IP) that drives the future product portfolio of the company. As all defence contractors reduce the amount of bespoke manufacturing they are undertaking it is the algorithms that can provide a competitive advantage between them.

With the continuing development of programmable technology platforms such as FPGAs there is a growing potential to commercialise increasingly advanced algorithms. Unfortunately most algorithms, whatever the purpose, are written as a sequential list of operations that does not translate easily into a parallelised design suitable for hardware implementation. The onus is on the company to reduce the expense and difficulty involved in implementing complex algorithmic IP so that the commercial opportunities it offers can be exploited. Adoption of new methodologies that improve the system-level design process offers a range of advantages including the ability to perform analysis of an algorithm's suitability for the application at an early stage, before significant engineering resources are committed, and leaving decisions on target platforms until as late a time in the development process as possible.

1.1 Problem statement

This EngD research programme was commissioned as a opportunity to investigate these issues surrounding translation of algorithmic IP to FPGA hardware. The process of translating an algorithm, usually expressed as a set of mathematical formulae and implemented in a computing environment such as Matlab or high-level programming languages, is typically done by hand. There are a number of stages involved in the transformation if the benefits of a hardware implementation are to be realised. Each stage of the process may depend on unique data representations and language constructs, and thus errors and inaccuracies may be introduced at any point. Hardware implementations must make use of parallelism if they are to achieve maximum performance, and this must be manually incorporated into the design. Verification is also clearly of equal importance, and so any translation process should emphasise

the opportunities for ensuring design correctness.

Alongside this main problem lies the opportunities presented by reprogrammable platforms to utilise dynamic reconfiguration, potentially enabling greater levels of functionality to be implemented in a smaller device. A benefit of this is increased device efficiency, since the utilisation levels can be kept high by keeping the whole device active rather than having chip functionality going unused. If there are functions that are not needed concurrently then they can be loaded onto the chip as and when required and thus the temporal and spatial utilisation of the device is maximised.

Although a fully automated route between algorithm behaviour and hardware structure would be the ideal outcome, it is perhaps an unrealistic aim. The so-called behavioural synthesis problem has been a topic of research in both industry and academia for many years, yet a single automated process is unlikely to become commonplace in the foreseeable future. Some of the reasons for this will be outlined in this document, but an overriding issue is that the size of the design space makes design rules impossible to create with any degree of success. For this reason the CAD software developers are increasingly taking a domain-specific approach to the synthesis problem, or focusing on one particular aspect of the translation process. This inevitably leads to a combination of tools being used, which can introduce further difficulties.

1.2 Research goals

The main goal of this work is to investigate some of the issues previously highlighted and to find a workable, efficient route between a high-level description of an algorithm's behaviour and a description of structure that can be used by the vendor implementation tools to produce an FPGA bitstream. In this context we define an efficient design process as one that can be completed using a minimum of engineering resources, whilst maintaining or exceeding existing standards of quality.

More specifically, the research programme aims to determine what makes some algorithms difficult to implement, where problems in the implementation process can occur, and what can be done to mitigate these problems in future work. The

approach taken to achieving these goals involves investigation of a series of algorithms that are expected to form integral capabilities of future Thales Optronics products, covering a range of functions including contrast enhancement, deconvolution, and image fusion. Investigation of these algorithms provides an opportunity to attempt to identify what underlying operations cause difficulties in their implementation. The individual mathematical operations that make up an algorithm may each be implemented in hardware in a number of ways, and without an understanding of the demands they place on the hardware it is difficult to assess an algorithm's suitability for a particular platform, and for instance whether it is more suited to a hardware or software implementation. There are also features that appear repeatedly in many algorithms of this nature, for instance two-dimensional convolution, and if a company is to operate efficiently it is important that where these features are encountered there is a clear understanding of the options available. If the lessons learnt from these implementations are documented and made available for designers of future systems, the efficiency of the design process may be improved and the goals of this research will have been achieved.

Additionally, more information is required into how the new design tools being released may or may not improve certain aspects of this situation. Only by understanding the nature of video processing algorithms and the challenges they present, can a design tool's suitability to the task be assessed.

1.3 Contributions

The work presented in this portfolio makes a number of contributions that aim to increase the level of understanding of the issues outlined above and facilitate future work carried out in this area.

A period of exploratory research has helped to identify some of the issues surrounding dynamic reconfiguration of Xilinx FPGAs and some tools and techniques that are available to address this area.

A theoretical approach to the algorithm implementation process has been used to understand why some algorithms pose such difficulties compared to others. The

model that is used in this context, which looks at algorithms according to the nature of their execution flow, has been successfully used to simplify the implementation of complex video processing algorithms.

Several major algorithms have been implemented. The Richardson-Lucy deconvolution and pyramidal image fusion implementations were the first reported real-time implementations of these algorithms on an FPGA device, and resulted in publications at major conferences.

During the implementation of these algorithms some features that are common to the application domain have been identified. Efficient solutions have been found to implement two such recurring features: Gaussian pyramid transforms and two-dimensional convolutions.

A PID servo-control algorithm has been implemented and the resulting design used in a commercial product. Additional work was also carried out to provide a fast means of demonstrating System Generator implementations of video processing algorithms in hardware.

The experience gained during the practical aspects of the research has enabled a set of design guidelines to be produced, to improve the efficiency of future design efforts. The design guidelines will be presented in this portfolio.

1.4 Portfolio organisation

This portfolio thesis is split into two volumes. Volume 1 contains an overview of the research project and highlights the novelty and commercial relevance of the completed work. Volume 2 contains the reports that were written during the research project, providing more detail on each unit of work than is present in volume 1.

The chapters in volume 1 are as follows. Chapter 2 details the taught component of the EngD and highlights the value of both the technical and business elements to the research. Chapter 3 lists the external training courses and conferences attended by the research engineer (RE) during the EngD programme. Chapter 4 discusses in more detail the commercial relevance of the research work completed during the EngD, with a discussion of why Thales Optronics have identified this as being a topic

of research that could be beneficial to the company and the contributions that this thesis has made.

Chapter 5 introduces the technical aspect of the research by providing background information on the areas surrounding implementation of video processing algorithms on FPGA. FPGAs will be briefly described, and some of the challenges faced when working with video data will be highlighted. This will be followed by an introduction to the more general problem of implementing sequential algorithms in hardware, a brief overview of some of the tools and methodologies that have been developed to tackle this problem, and look at some of the research on the subject that has been published by other authors.

Chapter 6 will then go on to discuss the results obtained during the research period. The chapter will describe each algorithm that has been looked at and examine the solutions that have been found and that can be applied in general to other algorithms of this type. Even from looking at a relatively small number of video processing algorithms it is possible to see some similarities between them that allow some general solutions to be defined. The ability to characterise algorithms according to the nature of their execution flow will be discussed and this will be used as a justification for some of the design choices made.

Chapter 7 will attempt to document some of the design guidelines and recommendations that have resulted from the completed research. It is hoped that these will improve the efficiency of the design process when implementing image and video processing algorithms in the future.

Finally, chapter 8 will summarise the conclusions that can be drawn from the work and suggest ways in which the research could be continued.

Chapter 2

Taught Modules

Taught modules worth 180 SCOTCAT credits constitute 25% of the total EngD requirement. These are split into two parts, with two thirds coming from technical subjects and one third from business and management subjects. The technical modules were taken from the MSc in System Level Integration offered by the Institute for System Level Integration (ISLI); the business classes were taken from the MBA programme of the University of Strathclyde Graduate School of Business (USGSB).

2.1 Technical credits

The 120 credits of technical modules were obtained through completion of the subjects listed in table 1. The subjects were chosen to give a solid theoretical foundation to the research work, with many being directly relevant to the research project. The 12 subjects covered a variety of issues, as follows:

- **Analogue and Mixed Signal Design:** Involved sections of work looking at amplifier design using both MOSFETs and bipolar transistors, Op-Amp design and analysis, and analysis of various methods of performing A/D and D/A conversion (including a significant project on sigma-delta A/D converters). Much of the work used SPICE simulations.
- **Communications Algorithms:** This subject was an overview of many of the mathematical methods and techniques involved in communications applica-

Subject	Credits
Analogue & Mixed Signal Design	12
Communications Algorithms	12
Embedded Software I - System on Chip	6
Embedded Software II - Operating Systems	12
Embedded Software III - Applications	6
Introduction to Hardware Design Automation	6
IP Block Authoring	12
IP Block Integration	12
Microcontrollers & Microprocessors	12
Multimedia & Video	6
System Partitioning	12
VLSI Design	12

Table 1: Breakdown of technical credits

tions, including Fourier analysis, design of FIR and IIR filters, Z-transforms and Laplace transforms, convolution, etc.

- Embedded Software I - System on Chip: Looked at the software implementation process and tool-chain for embedded targets, including compilation and linking using *make* files. Also involved a significant project looking at fixed-point design for FIR filters, something that would later prove invaluable.
- Embedded Software II - Operating Systems: Concentrated on development of applications using real-time operating systems (RTOS), involving a theoretical study of multi-tasking concepts such as pre-emption and priority inversion, and a practical task to develop a multi-tasking application for the commercial VxWorks RTOS.
- Embedded Software III - Applications: Focused on networking concepts including the seven layer network model and TCP/IP stacks. Also looked at the common internet protocols HTTP and SMTP, and markup languages HTML

and XML (including XML schema).

- Introduction to Hardware Design Automation: An look at hardware design languages, concentrating on Verilog but also a brief look at VHDL, and implementation tool-chains for simulation and synthesis.
- IP Block Authoring: IP issues, including the differences between soft, firm, and hard IP. Also design of IP for resale, according to guidelines from the popular Reuse-Methodology Manual and the OpenMORE ratings. Some back-end design issues such as chip layout (e.g. interconnect, clock distribution issues), capacitance and delay calculation, power and speed issues.
- IP Block Integration: IP-based design methods, for instance using commercial integration platforms. Modern verification methods such as formal equivalence checking, static timing analysis, and dedicated verification languages. Also some design-for-testability issues, including boundary-scan (JTAG) and built-in-self-test (BIST).
- Microcontrollers & Microprocessors: microcontroller topics covered included microcontroller architectures, register models, instruction sets, communication methods (e.g. I²C, SPI, CANBUS). Microprocessor topics were instruction sets and addressing modes, cache architectures, virtual memory, pipelining and superscalar architectures.
- Multimedia & Video: Included several topics that would later form a major part of the research work, notably JPEG and MPEG compression methods for images and video, and constituent elements such as basic image filtering techniques, the direct cosine transform, run-length encoding etc.
- System Partitioning: This subject included a look at some modern system-level design issues, including hardware/software partitioning and area/timing/power estimation. Also involved a study of modelling languages (in particular UML) and a project based on the system-level design language SpecC. This theoretical background to the system-level design problem and design tools and techniques would prove useful in the later research work.

Subject	Credits
Finance & Financial Management	12
Financial & Management Accounting	9
International Business	6
Managing People in Organisations	12
Marketing Management	12
Effective Project Management	6
The Learning Manager	3

Table 2: Breakdown of business credits

- VLSI Design: The subject looked at digital design from first-principles, including design of logic gates from individual transistors and CMOS circuit design. Also looked at some higher-level concepts such as multiplier architectures (i.e. Booth multipliers), speed/power/area trade-offs, logic synthesis.

2.2 Business credits

The 60 credits of business modules were obtained through completion of the subjects listed in table 2. The business modules provided an opportunity to look at the EngD from a commercial perspective, and to understand some of the business implications of the research work. The subjects were chosen to cover a variety of aspects of business.

- Finance & Financial Management: This subject looked at capital expenditure, investment criteria, company financing, and share pricing amongst other things. These topics were useful in providing an understanding of how projects (for instance engineering projects) are financed and what factors make such an investment opportunity worthwhile.
- Financial & Management Accounting: Covered costing, balance sheets, auditing and accounting practices, and other financial topics which affect all engineering projects but are rarely tackled by engineers.

- **International Business:** This subject looked at the ways in which business is conducted across international boundaries, including modes of entering foreign markets such as direct exporting, franchising, “greenfield” investments and joint-ventures. These topics are increasingly relevant to the modern electronics industry which is carried out on a global scale.
- **Managing People in Organisations:** This subject showed the human aspects of a business, and highlighted the fact that in any company there may be a variety of forces at work that lead to a particular organisational culture. Some topics covered included models of leadership, change management, performance management, motivation, and management structures.
- **Marketing Management:** Marketing Management highlighted the difference between a market-oriented and a production-oriented business: a distinction that demonstrates how it is important for a technology-based company to be led by the market rather than just producing technologically advanced products because the technology is available. The popular ‘4-Ps’ model (product, price, place, promotion) was also covered, along with topics on market research, segmentation and positioning.
- **Effective Project Management:** The project management subject was very relevant to the EngD, looking at several aspects of project management including critical path analysis, risk management, and established project management methodologies such as the industry standard ‘PRINCE2’.
- **The Learning Manager:** This was an introductory course to the MBA programme, which covered topics such as group-working, communication and presentational skills. Although the course was primarily intended to improve effectiveness of people undertaking the MBA programme, these are transferable skills which were equally useful for the remainder of the EngD and other aspects of professional life.

Chapter 3

External Events and Training

Over the course of the EngD several events were attended and participated in, both for training purposes and for presentation and dissemination of research results.

3.1 Training

Two training courses on FPGA design were attended at the Xilinx UK office. These are listed in table 3, and were used as an opportunity early in the EngD programme to become familiar with the product offerings and design techniques from the FPGA manufacturer favoured by Thales.

The Fundamentals of FPGA Design course covered the basic architecture of Xilinx FPGAs, such as the structure of the configurable logic resources and I/O elements that make up the programmable fabric of the device. It then went on to cover the Xilinx design flow and an in-depth look at the software tools used in the implementation process. After implementing an example design the reports generated by the tools were examined, and the information they provide was used to create timing constraints that could be used to improve the implementation results. This process is an important part of designing for FPGA systems.

Designing for Performance was a more advanced course over two days that went into greater detail than the fundamentals course described above. The course looked at some architectural features of Xilinx FPGAs in more detail, including clock management and pipelining techniques, and a look at the IP customisation tool Core

Course Title	Location	Date
Fundamentals of FPGA Design	Xilinx, Weybridge, Surrey, UK	14 Jul. 2003
Designing for Performance	Xilinx, Weybridge, Surrey, UK	15–16 Jul. 2003

Table 3: Training courses attended by the RE during the EngD programme.

Generator. A section was also dedicated to HDL coding techniques, and specifically the best way to write HDL code to infer efficient hardware designs. There was also a further study of methods available for achieving timing closure of FPGA designs, and a focus on the use of advanced timing constraints.

3.2 Conference publications

Two poster presentations were given at major academic FPGA conferences, as listed in table 4. The poster at FPL also involved publication of an accompanying short paper in the conference proceedings.

The poster presented at the FPGA 2006 conference was based on the work contained in Appendix E of this portfolio. The work looks at the hardware implementation of an algorithm for performing deconvolution of two-dimensional data. The algorithm, known as Richardson-Lucy deconvolution, is an important technique in the recovery of images that have been subjected to a blurring process. The implementation uses multistage separable filters as an efficient means of performing the several large 2D convolutions that are required. The results showed that real-time full scene deconvolution is viable with today's FPGA technology.

The poster and paper presented at the FPL 2006 conference were based on the work contained in Appendix F of this portfolio, describing an implementation of an algorithm for performing image fusion. The aim of image fusion is to combine multiple images (from one or more sensors) into a single composite image that retains all useful data without introducing artefacts. Pattern-selective techniques attempt

Poster Title	Conference	Date
A Real-Time Implementation of Richardson-Lucy Deconvolution	ACM/SIGDA 14th International Symposium on Field-Programmable Gate Arrays (FPGA 2006). Monterey, California, USA.	22-24 Feb. 2006
An FPGA Implementation of Pattern-Selective Pyramidal Image Fusion	2006 International Conference on Field Programmable Logic and Applications (FPL 2006). Madrid, Spain.	28-30 Aug. 2006

Table 4: Conferences participated in by the RE during the EngD programme.

to identify and extract whole features in the source images to use in the composite, and usually rely on multiresolution image representations such as Gaussian pyramids since they enable identification of features at many scales simultaneously. The description given of an FPGA implementation of the pyramidal decomposition and fusion process for dual video streams was the first reported instance of a hardware implementation of pattern-selective pyramidal image fusion.

3.3 Industrial events

The RE was also invited to give a 30 minute presentation at an internal Thales event held at the division headquarters in Paris. The details are given in table 5.

The Techno-Day conference is an annual event which is well attended by representatives from Thales offices in several countries. The format of the conference is a mixture of new product demonstrations, poster sessions, and multiple concurrent streams of technical talks on a variety of research and development subjects relevant to the Thales business, with the aim of disseminating innovative achievements to the wider business group. The presentation given by the RE described the implementation of Richardson-Lucy deconvolution as outlined above (and in more detail in Appendix E), and was one of only two full-length talks given at the event by

Presentation Title	Event	Date
A Real-Time Implementation of Richardson-Lucy Deconvolution	Techno Day, Thales Colombes, Paris, France	25 Jan. 2006

Table 5: Industrial events participated in by the RE during the EngD programme.

representatives from the UK Optronics side of the business.

Chapter 4

Commercial Relevance

As the computational power of integrated electronic devices continues to progress at exponential rates, advanced data processing techniques become available to a wider target market and in a wider variety of situations. Whereas real-time processing of video data was infeasible even in a lab situation only a short time ago, it is now possible to find complex real-time processing of video embedded in everyday consumer level items such as mobile phones and video cameras. As FPGAs become smaller and more efficient, they are challenging these markets that have previously been dominated by ASICs.

Video compression and decompression is a pertinent example of a capability that is now expected in a wide variety of situations. Evolving compression standards and techniques mean that dedicated hardware can quickly become obsolete, and for this reason it is often the case that compression and decompression are still performed in software. Inevitably this is a power drain for embedded devices with finite power supplies. FPGAs are set to exploit this gap. Although currently seen as too large and inefficient to be used in mobile phones and laptops, these platforms will almost certainly see embedded programmable logic in the near future. Offloading to programmable logic those algorithms that are complex but suitable for concurrent processing, for instance MPEG decoding for DVD playback, will provide gains in terms of both performance and power consumption.

These benefits apply equally to the defence market. One future objective of the defence industry is network-enabled capability, which implies large numbers of

intelligent sensing devices spread throughout the operating environment. In order to avoid information overload, devices must be capable of automatically processing data to extract the relevant detail from the mass of background information and make it more effective for its intended purpose, at the point at which it is collected. They should also comply with the recurring constraints of the industry: small physical size, low weight, and low power consumption. Embedded systems design and integration is consequently a critical factor in this environment.

Within the defence industry the major contracts are increasingly becoming system-level design projects covering a number of complex functions. In order to remain profitable, contractors are moving away from custom-built devices towards commercial off-the-shelf (COTS) parts in a bid to reduce design times and non-recurring engineering (NRE) costs. This situation is inevitably leading to adoption of FPGAs, which offer nearly equivalent processing capabilities of ASICs but with drastically reduced NRE. Besides the benefits of performance gains and reprogrammability that have already been discussed, FPGAs typically have an abundance of I/O resources and provide an ideal opportunity to integrate several system functions into a single device. The associated reduction in the number of devices makes design, manufacture, and test of circuit boards easier, reduces the number of potential points of failure, and makes future upgrades more straightforward since the functionality of an FPGA-based system can be modified without costly modifications to hardware designs.

With FPGA-based systems providing fairly generic hardware platforms, it is now a company's algorithmic IP that determines what capabilities it is able to bring to market. However, it will probably always be the case that the algorithms being developed by specialist algorithm engineers lead the current capabilities of embedded processing. The reason for this discrepancy is not necessary due to lack of transistors on the target device, but rather the huge complexity of the resulting electronic systems resulting in a hardware design process that is too lengthy and/or expensive. At present FPGAs do not provide the same degree of programmability as purely software-driven solutions, and so there is evidently a significant additional NRE cost associated with producing and verifying FPGA firmware. There are therefore clear

financial incentives to reduce design times. The most obvious way to do this is through use of more efficient design methodologies. In addition to this, the gap between what can be done offline in the lab and what designable embedded systems are currently capable of is an identifiable source of unexploited revenue. It is improved design methodologies that can help to leverage the algorithmic IP that exists within the company and close this design gap.

It has also traditionally been the case that those designing the algorithms work separately to the engineers designing hardware implementations. Both groups will have their own methodologies. Making FPGAs more accessible to those without formal hardware design experience, by abstracting away as much low-level detail as possible, can ensure that the algorithm design process is not completely isolated from the capabilities of the target platform, and should have a positive effect on the way in which the two disciplines interact.

There are also other reasons for looking to improve the design process. At present, the decisions on the precise behaviour of a system and potential implementation platforms are performed at an early stage in the system's development. Hardware/software partitioning is often carried out using fairly arbitrary guidelines, and significant amounts of engineering resources may be committed to a project before realising that the system is not optimal. Modern design techniques are attempting to abstract away the underlying implementation platforms in order to allow multiple designs to be analysed for their suitability at an early stage. Known as model-based design, these techniques rely on executable specifications to allow a high proportion of the necessary analysis and verification to be performed at an early stage in the development cycle. This also reduces reliance on written specifications and the inherent scope for misinterpretation. These factors all lead to reduced costs, and increase the chance of getting a design correct at the first attempt.

Alongside model-based design methodologies, IP-based design methods have become standard within both the defence and wider electronic design communities. IP cores covering a wide range of functions are available from FPGA manufacturers as well as specialist IP vendors. For some system-level functions, such as Ethernet or PCI interfaces, or high-level DSP functions such as FFTs, the instantiation of

IP is vital to reduce design time. However, commercially available IP cores do not cover the whole gamut of operations required for video processing, and developing and maintaining an internal library of IP can be expensive and demanding. For this reason it is often the case that a formal knowledge repository and clear design guidelines are of greater use to a company wishing to make their design processes more efficient. This requires a clear understanding of the types of algorithms the company wishes to implement and the technical challenges they are likely to face.

Whilst a fully automated route from behaviour to structure is still some time away, the work here has aimed to be general enough such that the design process is smoothed for all complex video processing algorithms.

4.1 Author's contribution

This work has involved an in-depth study of current design methodologies, and their suitability for the types of systems Thales Optronics produce. This has provided background information on why existing design flows must evolve to keep up with the capabilities of modern devices. An appraisal of some of the new methodologies that are appearing on the market has generated insights into which new tools and methodologies may prove relevant to the application domain.

The work carried out during this EngD project on individual algorithms has resulted in suggested solutions to some of the challenges that are faced when implementing algorithms of this type. These algorithms are all likely to appear in some form in future Thales products.

The work describing the use of manual scheduling and allocation of an algorithm written in a high level language provides a means of implementing model-based design methodologies within existing design flows. This also provides a method of making implementations of control-dominant algorithms more tractable.

The work on two-dimensional convolutions examines this recurring feature of image and video processing algorithms and explains how they may be efficiently implemented. The findings in this report show that in many cases it is beneficial to implement them using a structure known as multistage separable filters. This

technique enabled a design capable of performing real-time deconvolution of a VGA video stream on a single FPGA, and was published at a major FPGA conference.

The work on generating image pyramids has produced a hardware design capable of outperforming any previously reported methods. Since image pyramids are a feature of a number of algorithms this is useful IP for the company. Use of this pyramid generation technique enabled a design that could fuse dual VGA video streams in real-time on a single FPGA, and was also published at a major conference in the field.

Alongside the main algorithms featured here, the RE has also assisted in the implementation of other less demanding algorithms that feature in Thales products currently being marketed. A set of design guidelines have been produced, which aim to improve the efficiency of future design efforts within the company, based on the findings of this research.

An additional period of work involved development of an automated route from an algorithm design environment to a custom Thales hardware platform, for the purposes of testing algorithms in hardware. This can be used without any manual HDL coding or knowledge of the design tools, and so could be used to enable algorithm engineers to experiment with different hardware structures without requiring detailed knowledge of the implementation process.

Chapter 5

Technical Background

An overview of some of the technical issues of implementing image and video processing algorithms on FPGAs will now be presented. The following section will briefly discuss FPGA technology, the nature of video data and video processing algorithms, and some of the tasks involved in the implementation process. There will also be an overview of some of the new methods of implementing algorithms on FPGA, followed by a summary of these issues and the areas this research aims to address.

5.1 Field-programmable gate arrays

Field programmable gate arrays (FPGA) are integrated circuits containing programmable logic, which have evolved from the Complex Programmable Logic Devices (CPLD) and Programmable Array Logic (PAL) of the 1980s. The basic format of an FPGA is shown in figure 1. Modern FPGAs are vastly more technologically advanced than their predecessors, acting as true system-on-chip (SoC) devices with integrated memory, microprocessors, digital signal processing (DSP) elements, high-speed transceivers, clock management, and numerous other features. In addition to these hardwired capabilities the FPGA vendors also provide libraries of optimised “soft” IP cores, often at no cost to the user, which cover a large range of functions and application domains.

The basic elements of FPGAs are configurable logic blocks (CLB) connected together via a hierarchy of routing resources and programmable switch matrices.

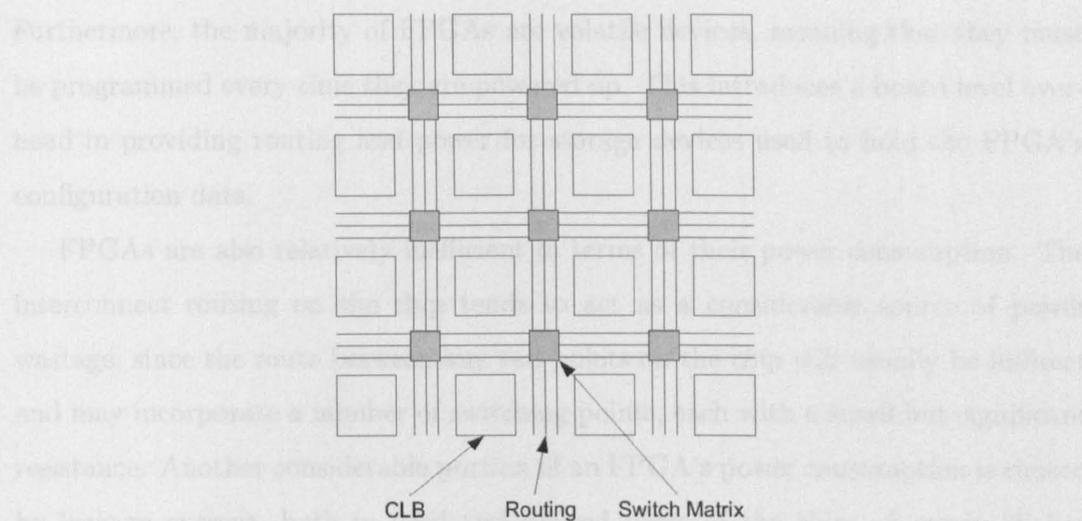


Figure 1: Basic FPGA structure.

Each CLB contains a relatively small amount of memory and some logic resources that may be programmed to implement the desired function, with the memory acting as a look-up table (LUT), RAM, or a shift register. When configured as a LUT it may be used to replicate combinatorial logic, and CLBs may be chained together to implement logic functions of any size. Four-input LUTs have historically been the predominant architecture, but the major FPGA manufacturers have recently begun producing FPGAs with six-input LUTs. The size of the LUT elements is important: when LUTs must be cascaded together it introduces delay that may limit the performance of the design, which would suggest that larger LUTs are preferable; conversely four-input LUTs had previously been shown to be the most efficient in terms of area due to the reduced routing requirements [1].

The routing on an FPGA is an important feature that ultimately determines device performance. It is essentially a network of interconnecting wires with switching matrices at crossover points comprised of pass-transistors and multiplexers [2]. Programmable routing resources take up a large proportion of available die area, and constitute a considerable overhead compared to ASICs. In addition to the programmable routing there is a separate clock distribution network that must cover the entire chip. Alongside the transistors that are needed for configuration and programmability this forms another significant overhead when compared to an ASIC.

Furthermore, the majority of FPGAs are volatile devices, meaning that they must be programmed every time they are powered up. This introduces a board level overhead in providing routing and power for storage devices used to hold the FPGA's configuration data.

FPGAs are also relatively inefficient in terms of their power consumption. The interconnect routing on the chip tends to act as a considerable source of power wastage, since the route between any two points on the chip will usually be indirect and may incorporate a number of switching points, each with a small but significant resistance. Another considerable portion of an FPGA's power consumption is caused by leakage current, both in used and unused parts of the chip. A study [3] has calculated that on an FPGA with 75% utilisation, 45% of the power consumption caused by leakage current will occur in unused areas of the chip. Modern design tools and techniques go some way towards correcting this, but it is still important that correctly sized FPGAs are chosen to minimise unused capacity.

It is therefore evident that in terms of performance and power consumption FPGAs are relatively inefficient compared to their ASIC counterparts. The real benefit of FPGAs lies in their flexibility, the low non-recurring engineering (NRE) costs associated with designing FPGA systems, and the complete elimination of the need to perform any "back-end" design. There are thus two main reasons to implement a design in FPGA rather than ASIC. Firstly, whenever there is a possibility that the system will be updated or changed at some point in its lifetime. Secondly, when the application is sufficiently low volume as to make the NRE costs of ASICs prohibitive. Both of these reasons can be applied to the defence sector, which highlights the importance of FPGA technology to a company such as Thales Optronics.

Many of the benefits of FPGAs in terms of cost and flexibility are also available in software implementations based on general-purpose microprocessors, but FPGAs have clear performance advantages over such methods. A von Neumann architecture may have to execute several instructions in order to achieve something that an FPGA can do in a single clock cycle, where custom instructions are coded into the logic structure. This instructional efficiency, along with parallel processing, gives FPGAs a computational advantage over processing devices based on von Neumann

architectures, and although general-purpose microprocessors may run at clock speeds that are 10–20 times greater than typical FPGA clock speeds, it is the inefficiency inherent in fixed instruction sets that leads to inferior performance in benchmarks [4].

The von Neumann architecture is now a limitation on the processing power of microprocessors, despite the advances enabled by Moore's law. In fact it is feasible to say that the von Neumann style architecture has only endured because of Moore's law: the sustained increase in the number of transistors on a chip has mostly been used to counter the problems of von Neumann (for instance in development of sophisticated branch prediction routines and cache structures). In contrast to this, FPGAs are able to make better use of the developments modelled by Moore's law since their dataflow nature means that they are not restrained by the need to fetch data from memory (and face the so-called "memory wall") [5]. DeHon [6] has further shown that the instruction efficiency of FPGAs enables them to have an order of magnitude more computational capacity per unit area of silicon die when compared to a RISC processor. This is a remarkable finding, especially when the logic overhead for configuration of the FPGA is taken into account.

One of the areas in which FPGAs have a clear efficiency advantage over microprocessors is in the ability to size instructions according to the demands of the application. For instance a 16-bit microprocessor operating on 8-bit data words will only be able to realise 50% of its peak processing power [6]. This is equally true of general-purpose microprocessors or specialised DSP devices, where static computational elements (for instance multiply blocks) promote inefficiency unless the algorithm can be designed in such a way as to fully utilise their capacity. In the programmable logic domain the firmware is designed to suit the algorithm, not vice versa. Modern FPGAs sacrifice some of this flexibility through integration of hard-wired IP cores (such as multipliers) in an effort to achieve higher speeds and densities, with an implicit reliance on the software tools to minimise inefficiency.

Image processing algorithms are ideal candidates for implementation on FPGAs, due to the requirement to perform often complex operations on very large data sets, usually at speeds high enough to meet the constraints imposed by real-time video sources. Since image processing algorithms often involve performing the same action

on each pixel or region of an image, there are usually clear opportunities to exploit parallel processing within the FPGA.

5.1.1 Floating-point performance

One area in which microprocessors have traditionally been seen as superior to FPGAs is in the area of floating-point performance. Floating-point maths is used intensively in the software domain, particularly in the area of DSP. Calculations that are carried out in floating-point are equally possible in fixed-point, but the main advantage that floating-point representations offer is in dynamic range, that is the ability to accurately represent very large and very small numbers with a reasonably small word length. Furthermore, fixed-point design can often be a time consuming and difficult process. Recently, there has been a renewed interest in floating-point maths from the main FPGA manufacturers, with floating-point operations becoming more feasible since the inclusion of hardwired multipliers in FPGAs. Studies have shown that FPGA floating-point performance is improving at a greater rate than that of microprocessors, some estimates are that FPGAs will have an order of magnitude higher floating-point performance by 2009 [5].

5.1.2 Dynamic reconfigurability

Dynamic reconfigurability is a potential facet of FPGA technology that has generated huge amounts of research since it was first conceived, particularly from the academic community. The main feature that dynamic reconfigurability offers is the ability to include more functionality into a smaller device, by swapping in and out parts of the design as they are required; in effect a virtual hardware system analogous to the virtual memory present in modern computer operating systems [7]. At present the major verification and synthesis toolsets do not provide explicit support for dynamically changing systems, and such systems must be designed using ad-hoc methods; this typically takes the form of incremental synthesis techniques, and manipulation of configuration bitstreams. The FPGA vendors are starting to take dynamic reconfigurability more seriously, with the latest generation of Xilinx

Virtex devices offering increased support. Thales Optronics are interested in the potential of dynamic reconfigurability, in particular the ability to load algorithms into hardware as required by the current situation. To this end the EngD research involved a substantial study of the methods and techniques surrounding dynamic reconfigurability (see the portfolio document in Appendix A). On conclusion of the study it was however felt that dynamic reconfigurability was not commercially viable at that time, mostly due to difficulties involved in verification and extended design times.

5.2 Video processing systems

The implementation of image and video processing systems on FPGAs presents some unique challenges. These may be attributed to the format of the data being operated on and the complexities of multidimensional algorithms. Some of these challenges will now be discussed.

5.2.1 Nature of video data

The common format for digital representation of analogue colour video signals is in the ITU656 format [8], which consists of one brightness signal and two colour components (known as the YUV colour space) sampled using a 4:2:2 scheme with 8-bits per sample, but for the purposes of algorithm development (and the work herein) it is usually assumed that the source data is 8-bit monochrome. Extension of an algorithm for colour operation is usually straightforward, but can require three times the hardware and possible colour space conversion, a topic that is not covered here. The size of the image depends on the source: a common size for video derived from analogue sources is 720×576 (based on the PAL standard), whereas digitally sourced video is typically a minimum of 640×480 in most applications and can be much larger. At 25fps a PAL video signal would present over 10 million pixels per second, so it is clear that data rates can be considerable. Data arrives in raster scan format, so the sequence of pixels goes from left to right, and top to bottom. Due to limited memory bandwidth data must normally be processed in a similar

sequence, and for real-time operation at the rate it arrives in the system, so without pipelining this would imply a processing time per pixel of analogue video data of sub 90ns, which is non-trivial for complex algorithms. This usually forces pipelining to be employed, whereby one pixel per clock cycle can be produced as output, at the expense of end-to-end latency. To a certain extent some latency can be tolerated for video signals as anything under a few milliseconds is not usually noticeable by human observers; there are however some applications where minimal latency is critical. The need to implement algorithms in this way can often necessitate a change in how an algorithm is approached conceptually [9]. A common problem is how video data can be processed temporally (for instance processing data from multiple frames simultaneously) when data is arriving in raster scan format and there is insufficient memory on the device to buffer more than a few lines.

Video data from analogue sources is often in an interlaced format, which can potentially cause additional problems. Deinterlacing is an inexact process that unavoidably introduces errors or artefacts. The simplest methods of deinterlacing: weaving (combining consecutive fields by overlaying on alternate lines), and line doubling (writing each line twice, to double the lines per field), reduce temporal and vertical resolution respectively. More sophisticated methods may employ motion estimation techniques to detect and compensate for movement between fields, but introduce a significant computational burden. Any artefacts or errors introduced during the deinterlacing process will have an impact on subsequent processing stages, so this is necessarily an important part of any video processing system with analogue sources. For an overview of deinterlacing techniques see [10].

5.2.2 Nature of image processing algorithms

Awcock & Thomas [11] identify five categories for image pre-processing algorithms, which are applied to alter pixel values to make an image more suitable for subsequent operations:

- Point operations
- Global operations

- Neighbourhood operations
- Geometric operations
- Temporal operations

These categories cover simple algorithm operations. Many higher-level algorithms may be formed from combinations of operations from these five categories. Four of the categories are based on processing of single static images; temporal operations are specific to video. Hardware implementation of each of these categories may have some commonalities that can be generalised. For instance, an example temporal algorithm would be to detect motion by subtracting a frame from the previous one. This would require frame buffering, and it is clear that whenever processing is required that utilises the temporal dimension of video data the storage requirements increase rapidly. Alternatively, global operations involve high speed processing, as multiple passes through the image data will usually be required; this will increase the memory requirement and pose implementation challenges in timely processing of the data.

There are some features of image processing algorithms that occur repeatedly. A brief overview of these will be given below before their impact on hardware implementations is discussed in a later section.

Two-dimensional convolution

A large proportion of digital image processing algorithms involve spatial filtering of two-dimensional data. This is usually implemented in the time domain as a simple convolution between the two-dimensional filter response and the image data. The discrete convolution integral for two-dimensional data is:

$$g(x, y) = f(x, y) * h(x, y) = \sum_i \sum_j f(i, j)h(x - i, y - j) \quad (1)$$

where f is the input image, h is the filter kernel, g is the resulting filtered output, and $*$ denotes the convolution operation.

The convolution may be thought of conceptually as a sliding window that moves over the image data and performs a weighted summation, as shown diagrammatically in figure 2. This class of algorithm can be used to effect a wide range of outcomes,

			137	59	61	154	170	52	24	79
			17	123	242	105	138	232	205	56
			22	119	52	98	2	96	203	196
			118	188	4	86	183	61	123	221
			234	60	215	198	129	214	31	9
			175	120	89	107	116	95	50	96
		79	95	110	83	234	113	159	135	38
			43	152	221	100	73	8	9	105

Figure 2: Conceptual sliding window highlights a group of pixels used to form single element of the output.

$$h_{\text{prewitt}} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad h_{\text{gaussian}} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \frac{1}{16} \quad h_{\text{unsharp}} = \begin{bmatrix} -1 & -4 & -1 \\ -4 & 26 & -4 \\ -1 & -4 & -1 \end{bmatrix} \frac{1}{6}$$

Figure 3: ‘Prewitt’ vertical edge-detection filter, Gaussian smoothing filter, and ‘unsharp mask’ sharpening filter respectively.

including smoothing, sharpening, edge-detection, correlation (e.g. for simple object recognition), and others. Typical kernel sizes for these operations are 3×3 or 5×5 , although much larger kernels are sometimes required. Some typical filter kernels are shown in figure 3; note that some kernels include scaling factors to prevent altering the overall image brightness level.

Since the calculations are all simple add and multiply operations they are straightforward to implement in hardware and spatial parallelism easily exploited. For instance, to calculate a convolution with a 3×3 window the nine multiply operations could be performed simultaneously, and the subsequent additions pipelined such that one output pixel is produced per clock cycle (albeit with some latency). The main consideration lies in being able to buffer data temporally such that pixels in different image rows can be collated and processed simultaneously. This usually requires sufficient temporary storage to hold several lines of the image. For example, a simple implementation of a 5×5 filter kernel will require that four lines of image data be buffered on chip. For a VGA image this would require 4 first-in, first-out (FIFO) buffers of 640×8 bits, a total of 20Kb, so it is clear that on-chip memory is an im-

portant resource that can quickly become scarce. This kind of temporal processing, where data sets are comprised of elements that do not arrive in sequence but are distributed in time, is common in many video applications but because of high data rates can often involve a significant requirement for memory resources.

Another factor that must always be considered when performing convolutions is how to handle the image edges. When the filter kernel is placed such that the central element falls on the edge of the image, some of the filter weights will fall outside the image boundaries. If the value of zero is chosen to represent these non-existent pixels the resulting output image will have darkened edges. Common solutions to this problem involve copying or mirroring the pixels that are closest to the image edge to the imaginary points outside the bounds of the image, but the only way to guarantee that the output data is correct is to disregard the data at the edges. This would mean that the output image is smaller than the input by $\frac{k-1}{2}$ pixels, where k is the side length of the filter kernel.

Some two-dimensional filter kernels are separable, which means that they can be formed from two orthogonal one-dimensional filters. This usually results in a reduction in computational complexity in the cases where it can be applied. This will be looked at further in a later section.

Transforms

Transforms are used throughout signal processing whenever a computational advantage may be achieved by working with a different data representation. The most well known of all such transforms is the Fourier transform, and its discrete form (DFT) and optimised “fast” form (FFT), which find multiple uses throughout the field of image processing. Application of the DFT to images mirrors the application to one-dimensional data: the DFT is a separable algorithm and so can simply be applied twice to the input data, once in each direction [12].

Working with images in the resulting spatial-frequency domain is less common than in one-dimensional signal processing, as Fourier transforming an image removes spatial locality between the pixels and makes an image unrecognisable. This makes it impossible to identify all but the most rudimentary image features. One area where



Figure 4: Multiscale decomposition of well-known image “Lena”.

the Fourier transform is often used in spatial filtering: as with the one-dimensional case, time domain convolution with a filter’s impulse response is equivalent to multiplication in the frequency domain, and so if the filtering operation is complex enough the advantages of performing a standard multiplication rather than a two-dimensional convolution may amortise the cost of performing the transform and its inverse. Many of the other common spatial-frequency domain applications are concerned with modification of frequency components for compression or enhancement based on the frequency perception of the human eye [13].

Methods that retain locality in both spatial and spatial-frequency domains, such as multiscale decompositions and the wavelet transform, are increasingly being used because the resulting data can be processed with respect to its position and spatial-frequency simultaneously. These transforms are usually implemented using two-channel filter banks, where the lowpass branch is applied iteratively [14]. The study of wavelets has now become a large and active field of research with applications in many varied areas. The resulting data appears as multiple copies of the original, with each copy band-limited to only contain image features at a particular scale. These data sets are commonly called image pyramids; an example is shown in figure 4.

Another transform that has found widespread use is the direct cosine transform (DCT), which is a fundamental component of JPEG and MPEG compression techniques. One of the features of the DCT over the FFT is that it does not produce complex coefficients.

Implementation of the DFT and DCT for two-dimensional data has been cov-

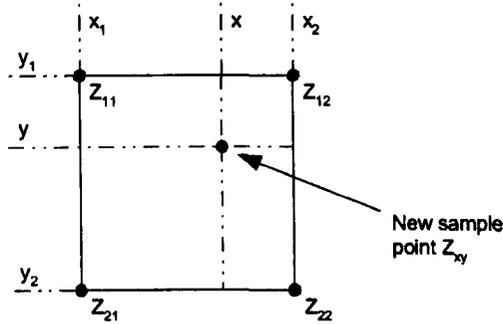


Figure 5: Interpolation from four neighbouring samples.

ered thoroughly in the literature and there are freely available IP cores to perform these tasks. However, the implementation of an algorithm based on a multiscale decomposition merits investigation, and will be covered in a later section.

Interpolation

Interpolation is a common operation in many image processing algorithms, needed whenever a single output pixel value must be approximated from multiple input pixels. A diagram of a situation where interpolation is required is shown in figure 5. The most basic method of performing interpolation is by a nearest-neighbour approach, which simply uses the value of the nearest pixel in the input image, so in figure 5 the pixel z_{12} would be used. This method introduces errors and blockiness in the output image, but may often be deemed acceptable due to the simplicity of its implementation. A more common method is bilinear interpolation, which, as the name suggests, is a linear interpolation carried out in two directions consecutively. The bilinear equation for the interpolation in figure 5 is

$$\begin{aligned}
 Z_{xy} = & \left(\frac{y_2 - y}{y_2 - y_1} \right) \left[\left(\frac{x - x_1}{x_2 - x_1} \right) Z_{12} + \left(\frac{x_2 - x}{x_2 - x_1} \right) Z_{11} \right] \\
 & + \left(\frac{y - y_1}{y_2 - y_1} \right) \left[\left(\frac{x - x_1}{x_2 - x_1} \right) Z_{22} + \left(\frac{x_2 - x}{x_2 - x_1} \right) Z_{21} \right]
 \end{aligned} \quad (2)$$

If the interpolation is between adjacent pixels, the distance between data elements is equal to one, and this reduces to the equivalent form:

$$Z_{xy} = (1 - y) [xZ_{12} + (1 - x) Z_{11}] + y [xZ_{22} + (1 - x) Z_{21}] \quad (3)$$

Obviously the reduced form is much simpler to implement in hardware because of the absence of divide operations. However the more complex form will be needed whenever interpolation is required over blocks or regions of the image that contain multiple pixels.

More complex methods of interpolation exist, notably bicubic (which averages over a 16 pixel area using derivatives) and Lanczos (which utilises *sinc* functions to produce a best-fit curve). These methods are significantly more computationally expensive than either nearest neighbour or bilinear methods, but offer improved accuracy. A study of the relative performance of interpolation algorithms is given in [15]. The tradeoff between computational latency and area versus quality of results must be constantly balanced when implementing algorithms and is part of the difficulty of mapping such algorithms to hardware.

5.3 Algorithm implementation process

Development of signal and image processing algorithms is usually performed in software at a high level of abstraction, using languages such as C/C++, or in particular Matlab. There are also graphical development environments such as the Matlab-based Simulink and the specialist video processing environment WiT [16]. By working at a high abstraction level the designer need only be concerned with an algorithm's behaviour (implemented as a series of computational steps), and does not need to understand the underlying operations that are being performed by hardware. These high-level tools will often be able to operate with the image as a single entity, or in the case of Matlab the image is treated as a matrix. The design of the algorithm will often employ floating-point arithmetic, and include complex high-level commands that map to a long sequence of processor instructions, such as the single command in Matlab to perform a two-dimensional convolution. Abstract concepts such as pointers and recursion are occasionally employed, particularly by algorithm designers with a software background or used to programming DSP chips, but these have no direct analogy in hardware. Memory structures in high-level languages are also much more abstract, for instance multidimensional arrays are often used in C,

as are *structs* and *unions*. The object-oriented paradigm present in C++ and other languages introduces a further level of abstraction. There are also differences in the way data is obtained and displayed, for instance when designing an algorithm in software it is usually assumed that the whole image is stored in memory, rather than streaming into the system in real-time.

The design of hardware is usually carried out in a hardware design language (HDL), in register transfer level (RTL) code that describes structure rather than behaviour, and contains timing information and synchronisation at a clock cycle level [17]. The code infers, or explicitly instantiates, low-level hardware elements such as registers and multiplexers. Although the major HDLs have the ability to describe some higher-level behavioural constructs these are not commonly used. One of the reasons for this is the small subset of HDL that is synthesisable by modern logic synthesis tools.

It is the transformation from the high abstraction level of the software domain to the low abstraction level of the hardware domain that introduces difficulties and inefficiencies into the implementation process. The hardware design process through the various levels of abstraction is shown in figure 6. The eventual aim will be a description of hardware written purely in the synthesisable subset of HDL.

5.3.1 Manual tasks for high-level synthesis

The transformation from behaviour to structure, also known as high-level synthesis, is usually a manual process that involves several specialised and complex activities. Some of these activities will now be examined.

Iterative processes

Iterative program loops are an integral part of many image-processing algorithms. Matlab code may often hide its iterative constructs from the user through use of matrix and vector data types and so-called *vectorised* instructions, but at a low level the flow of execution will involve loops (in some instances SIMD processor directives may be used in the place of explicit loop instructions). The ability of Matlab to natively support data in matrix and vector format is a key feature, which lends itself

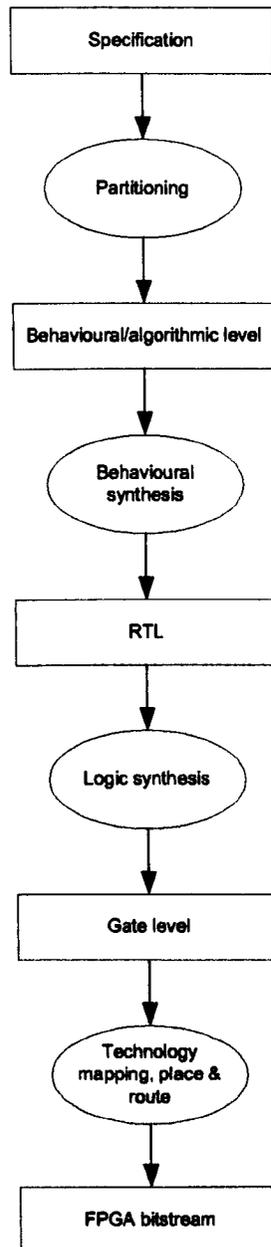


Figure 6: Transformation from specification to hardware. Rectangles denote levels of abstraction; ellipses are transformations. Only the lower two ellipses are processes that are commonly performed automatically.

ideally to DSP algorithm development, but can often be one of the main difficulties faced when implementing the same algorithm in the hardware domain, typically requiring manual translation to an iterative scheme as an initial step. Care must obviously be taken to ensure this does not alter the algorithm's behaviour. The loops in C code algorithms will be more explicit, but could be numerous and often nested to create complex program control flows.

Implementation of loops in hardware presents several design choices to be made. Modern synthesis tools have some limited support for looping HDL constructs, such as *for* and *while* loops, but using loops in a software style is inefficient in the hardware domain. An algorithm implemented using iterative loops is fundamentally a serial process, and direct implementation of this will only result in hardware acceleration of a design intended for a completely different (von Neumann) architecture, and fail to take advantage of features such as concurrency. Removal of loops via code transformations is well documented, but requires a considered approach to identify opportunities to exploit concurrency whilst balancing area/speed trade-offs.

Scheduling and insertion of timing information

Scheduling is the process of introducing timing information into the design and deciding which elements of a system may run in parallel and which must execute sequentially; the scheduling task is essentially to balance the cost/speed trade-offs of the design [18]. Synchronous design techniques, the de-facto design style for various reasons, mean that groups of instructions are activated by a transition on the clock signal; the designer must identify where data dependencies exist that prevent instructions executing concurrently, and then schedule or synchronise hardware events to the clock as appropriate. In order to do this the designer must be aware of the latencies of individual processing elements, and be able to map program operations to the available hardware while managing these latencies. To increase design speed, the designer may instantiate more parallel hardware elements; to decrease area, the designer may schedule computational steps serially. Scheduling also includes the important process of inserting pipeline stages, which, if done incorrectly, will result in poor performance and possibly unmet timing constraints.

Fixed-point design

The majority of algorithms designed in a high-level language will make use of floating-point data types for increased accuracy and convenience of design. As previously discussed, although the floating-point performance of FPGAs is improving rapidly most current algorithm implementations involve a significant process of fixed-point design. One of the key benefits of FPGAs over fixed hardware is the freedom to select appropriate word lengths rather than being constrained by the sizes of fixed registers. This freedom allows the range and resolution to be individually tailored to the algorithm being implemented, but despite (or maybe because of) this, translation of floating-point to fixed-point arithmetic (which involves calculation of word lengths, scaling factors, and handling bit-growth where appropriate) can be a time consuming process, often necessitating systematic analysis in order to determine appropriate representations [19]. Optimal representations must be found not only for instantaneous data values but also for any constants (such as filter coefficients) used in the system: using word lengths that are too long will result in hardware inefficiency that ripples through the system; too short and accuracy will be sacrificed. Matlab algorithms present additional difficulties when translating to hardware given that variables in Matlab do not need to be declared before being used and are sized dynamically. This adds an extra stage of analysis that must be performed in order to ascertain appropriate hardware representations.

Mathematical operations

There are numerous mathematical operations that are often taken for granted in the software domain but can cause difficulty in translating to hardware. Divide operations are now fairly tractable through use of free IP cores from the FPGA vendors, as are trigonometric, hyperbolic, and square root functions by way of the CORDIC algorithm [20], but these all consume a substantial amount of logic resources. Cores such as this are parameterised from the vendor tools and then integrated into the design as pre-synthesised netlists, often removing flexibility for synthesis optimisations and requiring design of an appropriate interface. Integration of IP cores at a HDL level is not a straightforward task for people without digital design experience, and

so the inaccessibility of some fairly common mathematical operators may be viewed as a limitation on the programmability of FPGAs compared to software methods.

Verification

Offen [21] discusses the need for hierarchical verification that demonstrates the behavioural equivalence between design and specification at each level of abstraction. Although more formal methods exist, the simplest approach is through use of the same verification stimulus at each stage, from executable specification right through to register transfer level HDL, and subsequently any post-synthesis (gate-level) simulations.

Verification of logic designs is at present carried out predominantly through RTL simulation, using event-driven HDL simulators. It is the responsibility of the designer to produce testbenches that correctly drive the simulation software and cover a sufficient range of test cases to ensure the original specification is being met. Due to the fact that testbenches are not synthesised the full extent of VHDL or Verilog instructions may be used, which provides a considerable number of additional capabilities over HDL that is to be synthesised, but designing the testbench and performing the simulation is still a lengthy and complicated process. Within traditional HDL design flows reuse of testbench modules is not common, and test strategies tend to be applied on an *ad-hoc* basis. Some recent developments in this area have focused on enabling testbench reuse across design languages and environments, although adoption of such methods is not yet widespread.

HDL for FPGAs is usually simulated in a modular style, with the simulation tool essentially acting like a software debugger for HDL modules. Despite this, simulation still takes substantially more time to complete than execution of a software model. This is an inherent drawback of event-based simulation, which models a design as a collection of independent events happening at various times. The move to cycle-based simulation may improve this situation: cycle based simulation is much simpler and quicker to execute and may complete in a fraction of the time, but currently only supports a limited subset of designs, specifically fully synchronous designs that are already compiled to gate-level (i.e. post-synthesis), or in some cases RTL code [22].

There is also no ability to model timing information and separate timing analysis must therefore be performed; FPGA vendor software tool chains usually incorporate static timing analysis capabilities which can be used for this task.

5.3.2 Design methodologies

The problem of generating hardware designs from behavioural descriptions, known as behavioural synthesis, has been the subject of dozens of research papers and initiatives over many years. One of the reasons for this is its potential to tackle the so-called design-gap in the electronics industry, caused by the fact that designer productivity is increasing at a lesser rate than the advances in hardware prescribed by Moore's law [23]. To achieve increased productivity the designer should be abstracted from designing structure as much as possible and able to concentrate on designing behaviour. This mirrors the shift to high-level languages in the software domain, where the implementation details (such as processor instructions and registers) are hidden from the designer unless direct control over them is needed. In this respect the aim is for HDL code to become analogous to assembly language, useful on occasions where tight control or very high performance is needed, but otherwise too close to the physical hardware to be considered a viable platform for efficient development of complex systems.

Due to the complex tasks involved in this process, some of which were outlined above, automatic behavioural synthesis remains unviable in the general case, though there have been some successes in domain-specific applications. Recently there has been a renewed emphasis on tackling automatic behavioural synthesis, with the problem now commonly given names such as high-level synthesis, algorithmic synthesis, and electronic system-level (ESL) design.

The FPGA manufacturer Xilinx recently launched an ESL initiative [24], which attempts to promote ESL and drive its adoption. The benefits to the FPGA manufacturers of easier device programmability are enormous: they will begin to challenge the markets currently dominated by DSP devices and will become attractive to scientists, software engineers, and others who could benefit from the high computational power of FPGAs but are unable or unwilling to spend the time learning and devel-

oping HDL code. Xilinx describe ESL as being design tools or methodologies that start above RTL level, so this is an inclusive umbrella term that covers many diverse approaches to the problem. Some of these methodologies are specifically targeted to a particular application domain, while others aim to be general-purpose solutions. Some of these tools and methodologies will now be described.

Xilinx System Generator/Altera DSP Builder

The two largest FPGA manufacturers have launched competing products in their efforts to ease the process of implementing algorithms from Matlab/Simulink: System Generator [25] from Xilinx, and DSP Builder [26] from Altera. Both tools have similar capabilities, and are essentially comprised of Simulink libraries of parameterisable blocks that map directly to FPGA IP cores. System Generator was used extensively during the EngD research, for a detailed description of its capabilities see the portfolio document (Appendix C). An overview of the System Generator design flow is shown in figure 7. Given that Matlab and Simulink are a common design environment for algorithm engineers, System Generator (and similar tools) offers a valuable method of describing hardware in a format that essentially hides the underlying HDL code. An attractive feature of the tool is the ability to surround the hardware design with blocks from other Simulink libraries, including stimuli such as waveform generators as well as the matrix types native to Matlab, and which would otherwise be difficult and time-consuming to implement using existing HDL testbench methods. When the algorithm simulates in Simulink as desired, Verilog or VHDL RTL code can be generated along with a HDL testbench that essentially performs equivalence checking with the original model. This type of design flow is commonly called *model-based design*, which means that the original specification of the design is written in an executable or self-verifying format, and subsequent developments are compared with the original design. This allows verification and debugging to occur earlier in the design process, when it is much easier to diagnose errors than later on using HDL simulators [27]. A further benefit of model-based design is the facilitation of testbench reuse at each level of abstraction, something which can reduce the necessary verification effort considerably.

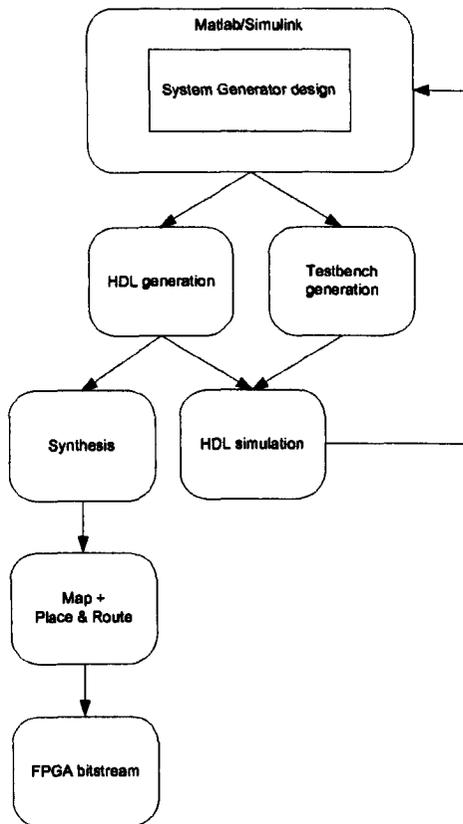


Figure 7: System Generator design flow.

One of the problems of System Generator and similar environments is that the design process can be at a very low level of abstraction, for instance manually connecting together elements such as individual registers and logic blocks. This is particularly the case if the higher-level blocks in the System Generator library are not suitable for the design being implemented. This type of low-level design can be time-consuming, and can feel like a step backwards from the automatic synthesis of hardware possible with HDL design methods. The strengths of methodologies such as that offered by System Generator is in the ability to visualise data flowing through the system, its close integration with Simulink and Matlab, and inherent support for fixed-point design and pipelining.

A recent release of System Generator allows generation of behavioural HDL for some blocks instead of just mapping to IP cores on the device, which gives more power to the synthesis tool to make optimisations and lessens some of the inflexibility of

IP based design. Mathworks, the makers of Matlab and Simulink, have recently introduced their own product into this area, known as HDL Coder [28]. Whilst offering similar capabilities to the tools from Xilinx and Altera, the Mathworks product has the advantage of not being vendor specific.

Synthesis of high-level languages

Several companies are looking to synthesise hardware directly from high-level languages; this includes synthesis from Matlab M-code, a capability offered by Accelchip (recently acquired by Xilinx) and their product based on a commercialisation of the MATCH compiler [29]. The other main language that is attractive to algorithm designers is C/C++. Impulse Accelerated Technologies [30] offer a product that takes ANSI C, alongside their proprietary libraries, and produces HDL. The tool offers the ability to design a system in software and then map computationally complex sections to custom hardware. The portions of the design that remain in software can be implemented on soft or hard processor cores in the FPGA. The C code that is used as a starting point must conform to a certain programming model, known as stream processing, which describes how the hardware and software processes communicate.

Mentor Graphics' Catapult C [31] also synthesises hardware from C-based algorithms. The tool presents multiple implementation choices to the designer, who may then investigate the area/speed trade-offs and make decisions accordingly. This means that the designer is still making the implementation decisions, but design can be performed at an algorithmic level and exploration of the design space is guided by the tools and consequently made easier. Celoxica are another company offering high-level synthesis, their products are largely based on a variant of ANSI C known as Handel-C, a language that has in-built support for representation of parallelism [32].

System-level languages

Other electronic design automation (EDA) tool manufacturers are looking to replace Verilog and VHDL with more sophisticated languages. The most prominent candidates in this regard are SystemC and SystemVerilog.

SystemC is a collection of C++ classes that is specially formed to describe system-

level concepts, such as interfaces and transactions, and hardware concepts such as concurrency, and allows for a unified hardware/software co-design. Because it is based on C++ it inherits the object-oriented programming paradigm and other characteristics of that language. SystemC can handle modelling concepts at a range of abstraction levels, from system level down to RTL. The benefit of this approach is that it provides the ability to design an entire system, starting from an executable specification and then passing off to possibly separate hardware and software teams, without the need for a change of language that may potentially introduce errors. Scheduling is handled explicitly through use of *wait* statements, which demarcate groups of instructions that are considered to execute concurrently. Several vendors now offer synthesis of behavioural-level SystemC; the synthesisable subset is considered comparable to that of traditional HDLs [33].

SystemVerilog is an attempt to approach system-level design by augmentation of the existing Verilog HDL. The additional features available in SystemVerilog are extensive support for verification constructs, support for complex interfaces, new data types, ability to work with arrays as a single entity (potentially useful for image handling) and some high-level language constructs such as *structs* and *unions*. All of these new features are synthesisable, albeit with some constraints [34].

5.4 Related work

This chapter has included a look at new tools and languages that are gaining traction within the electronics community, but the primary aim of this work is to achieve improved efficiency within existing design flows, based on intimate knowledge of the application domain and its requirements. This approach naturally draws on areas of both academic and industrial research. This section will briefly review some of the most relevant published research.

A group from Massey University, New Zealand have published a number of papers concerning implementation of image processing algorithms on FPGAs. Reference [9] gives an overview of the topic, in terms of the difficulty of mapping sequential algorithms to parallelised hardware, and discusses the constraints that are faced

when implementing image processing algorithms. These constraints are classified as timing constraints (for real-time processing), bandwidth constraints (applicable when off-chip memory is required), and resource constraints on the device itself. The impact these constraints have on implementation of different classes of image processing algorithms (for instance window operations) is briefly discussed.

Reference [35] from the same group provides more practical advice in the form of “design patterns”. These are defined as generalised, reusable solutions, that act to assist in the implementation of image processing algorithms on FPGA by conveying previous design experience. The paper goes on to discuss how such design patterns should be documented and categorised. Although conceptually similar to the work that will be presented here, the design patterns that are suggested refer to issues such as how data is stored in memory and when pipelining should be applied, and are generally concerned with quite low-level technical details rather than the high-level operations that are the focus of this portfolio. Some relevant high-level operations have separately been investigated by the same group, with publications on FPGA implementations of division [36] and bilinear interpolation [37] operations, both of which are prevalent in image and video processing algorithms.

A group from Queens University, Belfast is also doing work in this area. Reference [38] describes the concept of using “hardware skeletons”: a parameterisable architecture designed for a specific task, in this case image and video processing. Hardware skeletons may be thought of as being similar in nature to SoC platforms, where specific functionality is implemented by inserting functional units (usually pre-designed IP cores) into a domain-specific framework. In order to design a hardware skeleton the authors model image processing algorithms using directed acyclic graphs, which are similar in nature to dataflow graphs, and describe how high-level image operations (represented by nodes on the graph) are connected together. In a similar way to that described in section 5.2.2, operations are classified according to the locality of their data access requirements, for instance point operations, neighbourhood operations, global operations etc. This methodology becomes useful when a range of possible implementations, characterised by their cost in terms of speed and area, are available for each node. This allows the designer to perform

cost-based analysis of the different implementation options for each operation. For instance there are a number of ways of performing a convolution, and if cost-based information is available about each method the designer may choose the appropriate technique according to the constraints that are faced. The designer can then populate the hardware skeleton using the necessary operations.

The stated advantage of the approach is that system builders do not themselves require detailed hardware description skills, which is one of the aims of this portfolio. Unfortunately, although hardware skeletons would speed the implementation process when a full set of cores are available, it requires a library of optimised cores to be available for each hardware platform (or type of FPGA) that may be used, which could be difficult and costly to maintain. The method is also obviously less flexible than the more general idea of using design guidelines, and is heavily oriented to those algorithms that can be easily represented using data-flow graphs.

Researchers at the same institution have also looked at some specific high-level algorithm operations, including a method of automatically generating designs for direct implementations of 2D convolutions [39] and an efficient FPGA implementation of a wavelet transform [40].

An MSc thesis from Vanderbilt University, USA and completed in 2000, [41] focuses on the implementation on FPGA hardware of image processing algorithms described originally in Matlab. The algorithms are all windowing operations, including basic convolution and some more complicated methods such as rank-order filters and erosion/dilation, and are all based on small kernel sizes, typically 3×3 elements. The designs are all hand-coded in VHDL and are not heavily optimised. The findings are that although the advantages of FPGAs are clear, the benefits can be outweighed by the difficulty of implementing complex mathematics on FPGA devices. A suggested solution to this problem is use of FPGAs and DSP devices in tandem, to take advantage of the positive benefits of each platform. This viewpoint is less popular today, given the large gains in the mathematical capabilities of FPGAs that have been made in the intervening years since the work was completed.

The same topic of implementing Matlab algorithms is covered by work from Northwestern University, USA which resulted in the MATCH compiler [42]. (The

MATCH compiler was later commercialised as AccelDSP, a product that is briefly discussed in section 5.3.2). Although now a proprietary technology, early papers on the subject describe a process of high-level synthesis that can also be performed manually. It works by unrolling loops and the vectorised Matlab instructions used to operate on matrices, and then scheduling and binding the resulting simplified code to hardware, in a similar way to that presented for C code in Appendix A of this portfolio. The description of the MATCH compiler given in [43] gives a good overview of the steps that are involved in translating behavioural-level code into an efficient hardware design.

Athanas and Abbott [44] describe an attempt to classify image processing algorithms in order to expedite their implementation on programmable logic (in this case the multi-FPGA platform Splash-2). Their classifications are image combination, transformation, measurement, conversion, and generation. Of particular interest are transformation operations (which include convolutions) and combination operations (where Gaussian and Laplacian pyramids are discussed). Comparison of the different classes is performed in terms of the demands they place on the underlying hardware, for instance the resource requirements of an 8×8 2D convolution are compared with a 2D floating-point FFT. Some discussion is also given of the difficulties caused by raster scan data formats. Although the nature of the Splash-2 platform differs from the single FPGA approach taken in this work, the attempt to form general conclusions regarding image processing algorithms in terms of their hardware implementations is obviously related.

In addition to those mentioned above, there are many publications that investigate efficient implementations of individual mathematical operations that are pertinent to the field of image processing. One of note is reference [45], which looks at implementation on FPGAs of fast Fourier transforms for signal and image processing. Their approach looks at the different methods of performing an FFT (radix-2, radix-4, finite Hartley transform etc.) and considers the demands they place on the hardware. The findings are that the radix-2 approach performs well in terms of speed and area, but the finite Hartley transform has the lowest memory requirement. Comparisons are also made with some commercially available FFT cores. This type

of approach, where an operation common to many algorithms in a particular application domain is assessed for its hardware impact, is one approach that is advocated by this EngD.

5.5 Summary

The future of high-level synthesis will necessarily have to include new methodologies if the complexity faced by hardware designers is to be manageable within reasonable timeframes. Rather than a single solution emerging, it is increasingly evident that domain-specificity is going to be integral to developing methodologies that can produce designs of comparable performance to handcrafted methods. Each of the available tools and methodologies impose their own constraints, some are more suitable to certain design styles than others, and there is a general lack of knowledge about how image and video processing algorithms fit into this environment.

Whilst most current algorithm implementations are still carried out by hand, and despite the fact that the majority of image and video processing algorithms face similar constraints and difficulties in their implementation, there are no clear guidelines as to how their implementation should be tackled. It is clear that the process of decomposing algorithms into their basic constituent operations should be performed with an understanding of those operations' suitability to the target hardware. Some operations may be more difficult to implement than others, may require more hardware resources, or have reduced performance on a particular hardware platform. High-level transformations may result in drastic improvements in resource use, power consumption, and performance, but at present this can only be performed on a trial and error basis with no guidelines available. There are many classes of mathematical operations that may result in a variety of hardware implementations, some more efficient than others.

Investigation of these issues will produce data that allows assessments to be made of algorithms at an early stage in their development, relating to how effectively they may be implemented with the software tools at hand, and which hardware architectures are most suitable. By looking closely at a range of algorithms, and their

constituent parts, the aim of this work is to demonstrate some of the problems and potential solutions involved in the implementation of video processing algorithms.

Chapter 6

Results & Discussion

The main research effort focused on the implementation of several algorithms. These algorithms cover a range of applications and are likely to form integral capabilities of future Thales Optronics products. The main objective in looking at these algorithms is to identify those parts that cause problems in defining appropriate hardware structures and attempt to explain why this is the case. Looking at a range of algorithms in this way also helps to identify common features that could provide benefit if they were implemented as reusable IP.

This chapter will discuss these algorithms and their implementation. The starting point for each algorithm was a Matlab implementation along with associated test vectors, with the aim of producing verifiable hardware implementations for Xilinx FPGAs. Each algorithm will be briefly described, followed by some discussion of the implementation process and the general conclusions that can be drawn.

6.1 Implementation of LRM contrast enhance

The first algorithm implemented as part of the research was the local range modification (LRM) contrast enhancement algorithm by Fahnestock & Schowengert [46]. A full description of the implementation process can be found in the portfolio document (Appendix B).

6.1.1 Background

The contrast of an image is a way of referring to its dynamic range, i.e. the range between the brightest and darkest pixel. An image with poor contrast will suffer from a reduced amount of visible detail, with it usually being difficult to distinguish foreground objects from the background. Poor contrast is a result of under- or over-exposure of the imaging sensor, or use of a badly chosen digital representation that does not provide sufficient range for the signal being captured. A simple method of improving the contrast of an image works by applying a linear stretch to the dynamic range to increase it to the maximum range available, which is usually determined in the digital domain by the binary word length of the data values representing pixels. For example, the contrast of an n -bit greyscale image could be enhanced by applying the following calculation to each pixel:

$$out = \frac{in - min}{max - min} \times (2^n - 1) \quad (4)$$

where min and max are the minimum and maximum pixel values in the entire image. Application of this process will ensure that the brightest pixels in the input image will become peak white, and the darkest will become black. Another name for this process is histogram stretching, named for the effect the operation has on the image's histogram. Example images and their histograms can be seen in figure 8.

The linear stretching operation acts globally on the image, using parameters derived from the image as a whole. This form of contrast enhancement is widely used as it provides a simple and effective means of improving contrast when the source image has a poor dynamic range. There are, however, some situations in which the method will give less satisfactory results: notably when the contrast varies over different regions of the image (where regions of good contrast will limit the effectiveness of the algorithm in regions of poor contrast), or when the image is affected by noise (a single pixel affected by additive noise will significantly reduce the amount of contrast stretching that will be applied). In these cases a global contrast stretch will provide less than optimal results. The purpose of the LRM algorithm is to provide a spatially-variant means of improving contrast, that is, one that alters a pixel's value dependent on the contrast of its surrounding region rather than the

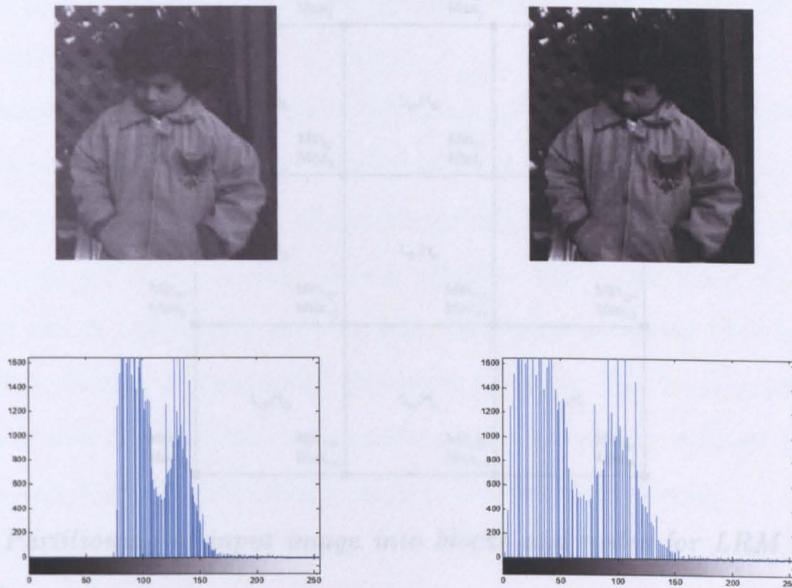


Figure 8: Demonstration of contrast enhancement through a linear stretch. The effect on the image histogram is also shown.

image as a whole. In this way the algorithm is adaptive to regional variations in contrast levels, and although the LRM method is not completely noise tolerant the effects of rogue pixel values will be limited to small confined areas.

The LRM algorithm works by dividing the input image into blocks; these are typically 8×8 or greater since blocks that are too small will unnecessarily highlight insignificant details. The original algorithm allows for non-square blocks, and also allows the block size to vary within the image (a necessity at the edges when the image does not contain an integer number of whole blocks). As shown in figure 9, at the corners of each block is a node, and each node may have one, two, or four neighbouring blocks depending on whether it is at the corner, edge, or centre of the image respectively. The algorithm operates by finding minimum and maximum pixel values within each block, then sorting these to find the minima and maxima at each node. The minimum and maximum values for each pixel are then found using two separate bilinear interpolations that operate with the four closest corresponding node values. Finally a standard stretch operation (using equation 4) is applied using the interpolated values for *min* and *max*.

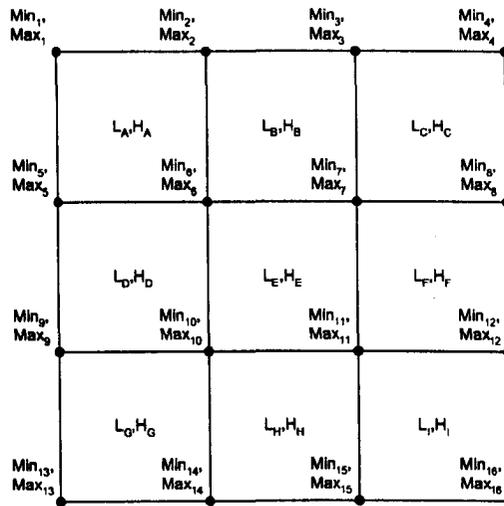


Figure 9: Partitioning of input image into blocks and nodes for LRM contrast enhancement.

A typical software implementation will require multiple passes through the input data to achieve this: once to find block minima and maxima; once to find node minima and maxima; twice to perform the two bilinear interpolations; and once to apply the stretch to each pixel. There is also a large amount of intermediate data generated by the algorithm that would typically be stored in multidimensional data structures. Evidently this does not lend itself to a direct hardware implementation when real-time operation is required.

6.1.2 Hardware constraints

In order to translate the algorithm into hardware it is necessary to make some design choices regarding the implementation of the computational stages. The main computational effort of the algorithm is in calculating bilinear interpolation equations similar to that shown in equation 2. These interpolations take place over an area the size of a block in the input image. In the software version of the algorithm the block size is freely definable, but since the block size forms the denominator of the interpolation equations one of the early design decisions was to limit the block size to be a power of two, and force all blocks to be square and of uniform size. This immediately simplifies the mathematics considerably: the six divide operations that

make up the interpolation calculation are now replaced by a single binary shift, a trivial operation in hardware.

A side effect of forcing all blocks in the image to be an equal size is that the image must therefore contain an integer number of blocks. This then forces the choice of block size to be a common factor of the image height and width, with the maximum block size their greatest common divisor (GCD). The dimensions of most digital images (e.g. VGA 640×480) have at least one common factor that is a power of two, so this constraint is reasonable. However, for video data from analogue sources the constraints on the block size and/or shape may have to be relaxed, which would consequently increase the complexity of the calculations involved.

Having simplified the algorithm in this way work could begin on developing a hardware implementation. It was decided that the transformation from software to hardware would be performed manually, using a clearly defined process of applying well-known transformations from the field of high-level synthesis, and carrying out verification after each stage. The language chosen for this transformation was ANSI C, since it is flexible enough to describe both high- and low-level behaviour. It does not natively handle concurrency, but by using a special coding style the impression of concurrency can be achieved.

6.1.3 Transforming behaviour to structure in C

The conversion of a Matlab algorithm into an equivalent C implementation may usually be undertaken without significant difficulty, as many of the formal semantics of the languages are similar. High-level Matlab instructions with no direct equivalent in C, for instance instructions that perform a two-dimensional convolution or FFT, will obviously require a much more involved and time-consuming approach. Fortunately the LRM algorithm uses fairly simple operators throughout, and the complexity of the algorithm lies in its control flow rather than its computational aspects. Another difficulty at this stage lies in determining the shape and size of arrays and vectors used in the Matlab code, a process that Halder et al. term *scalarization* [43]. This process also includes the act of translating Matlab's vectorised instructions into C-compatible loops.

The equivalence of the C code algorithm to the Matlab original may be verified by testing both versions of the algorithm with the same stimulus, chosen to cover a range of test cases. Once the algorithm is correctly described in C, a sequence of transformations can be applied that gradually incorporates more structure and simplifies higher-level behavioural constructs. Some of these transformations may involve replacing floating-point arithmetic operations with fixed-point ones, unrolling program loops, or simplifying statements to reduce the number of operators in a single line of code (known as *levelization* [43]).

C is useful for this purpose because it is flexible enough to contain high-level and low-level code. Although a language such as VHDL also contains behavioural-level constructs, C remains executable, which means it can be verified quickly and without the use of HDL simulators. This is a further example of model-based design, as described for the System Generator design flow in chapter 5, here with the C code forming an executable specification of the algorithm. After each of the transformations described above have been applied the current version of the algorithm can be verified against the original Matlab model using the same test-cases as were used in the original algorithm specification. This evidently aids verification greatly. Having an executable model of the system is equivalent to performing a cycle-based simulation, and avoids the difficulties of event-driven simulation as described in chapter 5.

The next stage in incorporating structure into the design is to introduce scheduling, such that the design incorporates the notion of clock cycles and timing. The process involves manual identification of instructions that can be executed on separate time steps, and those that must be executed sequentially due to data dependencies. The execution flow may be modelled in C using a large *switch* statement, with each branch containing a single instruction and representing a time-step. Each branch of the *switch* statement must be able to complete in one clock cycle. The designer is essentially assigning computational steps to states of a controller, such that the design becomes a large finite state machine (FSM), with as many states as there are instructions in the algorithm. The states of the controller are stepped through in sequence according to the program flow of the original algorithm, for instance where

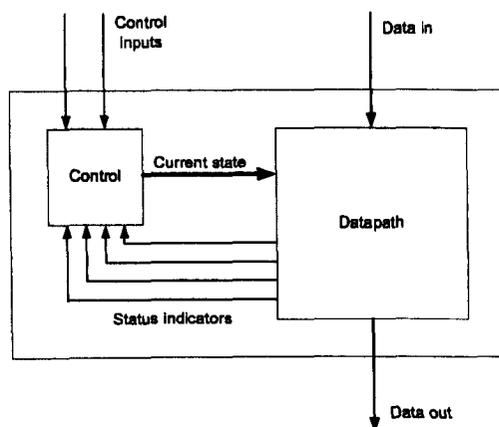


Figure 10: Block diagram of a system with separate control and datapath.

previously there existed program loops the FSM can be designed to jump back to a previous state. Subsequently, states that contain instructions that are data independent may be combined, so that multiple instructions are executed on a single branch of the switch statement; these may then be thought to execute concurrently. See Appendix B, section 3.1 for a more detailed explanation of this process.

6.1.4 The control/datapath model

A theoretical model that uses the technique of scheduling computational steps using a state machine is the finite state machine with datapath (FSMD) first proposed by Gajski and Ramachandran [47]. Construction of an FSMD requires separation of the control operations from the rest of the algorithm. Instructions that perform control are looping constructs such as *for* and *while* statements, and branches such as *if* and *case*. The control elements and data processing elements are implemented as separate FSMs, and the data-processing FSM may then be considered to be a pipelined datapath. An overview of a system of this format is shown in figure 10. Note that the datapath is monitored by the controller using status flags, which may be used to affect the control flow of the system. The current state, which is used to control the datapath, may be thought to represent the instruction to be executed. This kind of system is known elsewhere as a high-level state machine (HLSM) [48].

If the number of states has been minimised as far as possible the datapath may

be considered optimal in terms of performance, as the maximum amount of spatial parallelism has been utilised. If resource constraints are present, for instance due to a limited number of hardware multipliers, then the number of states can be increased to reduce concurrency and hence use less hardware at a given point in time.

Implementing the control and datapath aspects in separate modules allows comparisons to be made between the relative sizes of the two. A design that has a large and complex controller but relatively small datapath is said to be control-dominant; conversely, a design with a simple controller but large datapath is datapath-dominant. Characterisation of algorithms in this way provides important insights into their suitability for a particular target hardware.

Complex control-dominant flows are much more suited to a software implementation on a programmable architecture such as a microprocessor, microcontroller, or DSP chip. The fundamental elements of software development (nested loops, branches) are characteristics of complex control and irregular execution flows. Controllers are characterised by their ability to make decisions according to the values of their inputs. Control flows are essentially sequential processes that may have irregular structure but usually only need to do a single thing at a given time. Although it is possible to implement complex control in hardware using FSMs, it is tedious to design and inflexible.

Datapaths are evidently more suited to a pure hardware implementation, where spatial and temporal parallelism can be exploited, i.e. when the system must perform the same operation(s) repeatedly, and does not need to radically change its function at any time.

It is clear that algorithms intended for implementation on FPGAs should conform to the dataflow paradigm if the potential performance benefits of the platform are to be realised. The LRM algorithm is control-dominated: the bulk of its execution is concerned with looping through image blocks and selecting (i.e. making decisions on) local minima and maxima. This implies that the LRM algorithm is not an algorithm that is simple to implement in hardware, and the resulting implementation is not certain to provide huge performance benefits over a software one.

The manual process of scheduling and allocation that has been described here

(and in the portfolio document) provides a mechanism for implementing control-dominant algorithms that do not readily conform to the dataflow paradigm. Having a clearly defined framework for implementing such algorithms can reduce the design time significantly. Using a C code version of the algorithm that can be quickly verified after each modification allows the designer to gradually insert structure into the behavioural-level specification without introducing large-scale errors, and thus implement model-based design within the existing design flows.

6.1.5 Translation to VHDL and synthesis

The aim of the manual scheduling and allocation methodology presented here is to have the algorithm written in a subset of C where each instruction has a direct equivalent in the synthesisable subset of VHDL. This is perfectly feasible: the C *switch* statement can be translated to the VHDL *case* statement with minor modifications, conditionals such as *if* have direct equivalents, and any looping operators should have been removed or decomposed as part of the sequencing of the FSM. Although the LRM algorithm is computationally quite simple, had there been any complex mathematical operators left at this stage that could not be decomposed any further within the FSM model integration of standalone modules or IP cores would be necessary.

Once translated to VHDL synthesis can be undertaken as normal. Any subsequent verification effort must be applied using HDL simulators and custom testbenches; verification may be required to ensure the equivalence of the VHDL code with the final C model. More details on the implementation and verification strategies are available in Appendix B, sections 4-5.

The design was implemented on a commercial development board featuring a Xilinx Virtex-2 XC2VP20 FPGA and ADC/DAC devices. Table 6 shows the resource usage for the LRM algorithm after implementation on the Virtex-2 FPGA. The development time for implementing the design, from the Matlab specification to a gate-level (post-synthesis) netlist was approximately six weeks. A simultaneous design effort within Thales for the same algorithm, but using a traditional implementation flow that had not utilised the FSMD model, had resulted in a considerably

Resource	Used	Available	% of XC2VP7
Occupied slices	470	4,928	9
4-input LUTs	765	9,856	7
Block RAM	13	44	29
Multipliers	14	44	31

Table 6: Resource usage of LRM algorithm.

longer development time. The two implementations had comparable resource usage and performance. The resource figures shown here highlight the relatively simple computational aspect (i.e. small datapath) of the algorithm. The control parts of the algorithm do not use significant logic resources. The performance of the final system was sufficient to process VGA data in real-time.

6.1.6 System Generator implementation of LRM algorithm

The LRM algorithm was also implemented in the Xilinx System Generator environment. LRM was chosen as an ideal opportunity to become familiar with System Generator using a familiar algorithm that was well understood. The implementation (described in Appendix C) faced many of the same difficulties as a hand-crafted HDL implementation. The System Generator environment is designed with DSP applications as the primary target domain, and so it is heavily oriented towards dataflow style algorithms. For this reason the LRM algorithm is not an ideal candidate for implementation using the System Generator block set. Because of the System Generator library's emphasis on high-level mathematical operators many of the LRM algorithm's constructs had to be formed from low-level blocks, such as individual registers and logic functions.

Some of the more complex control methods posed difficulties in their System Generator implementation that were easier to solve with hand-coded methods. For instance, the overall control of the design was still implemented using an FSM, albeit this time in Matlab M-code imported into System Generator with the 'M-Code Block'. The nested looping constructs that appear in the algorithm were

implemented using a complicated system of selectively enabled counters. More details on some of the challenges faced in the implementation are available in Appendix C, section 3.

Comparison of the results from the hand-crafted and System Generator implementations of the LRM algorithm gave very similar resource usage and performance, however the design time was longer for the System Generator work (see Appendix C, section 4). The overall findings of this preliminary work with System Generator was that although strong in certain areas, notably fixed-point design and some verification tasks, it was ill-suited to complex control algorithms such as LRM. System Generator would be used later in the research to more positive effect when implementing a datapath-dominant design.

6.2 Implementation of RL deconvolution

The Richardson-Lucy deconvolution algorithm was the second major algorithm studied as part of the research. The full report describing the work and implementation may be found in the portfolio documents (Appendix E).

6.2.1 Background

Richardson-Lucy deconvolution is an established method for the recovery of images that have become blurred. The blurring process may be caused by various factors, including the inherent characteristics and possible defects of the optical equipment. The blur may be modelled mathematically as a two-dimensional convolution operation, as described in chapter 5, with the filtering kernel replaced by a point spread function (PSF). The PSF represents the spreading or smearing of a single point source, and may be thought of as a probability distribution describing the potential destinations of a photon originating from a given point in the input. A PSF is specific to a particular imaging apparatus, and may be derived mathematically using models of the optical path, or more commonly deduced empirically through measurement of the spreading of a point source against a black background. An example PSF, obtained empirically by engineers at Thales for a specific imaging apparatus,

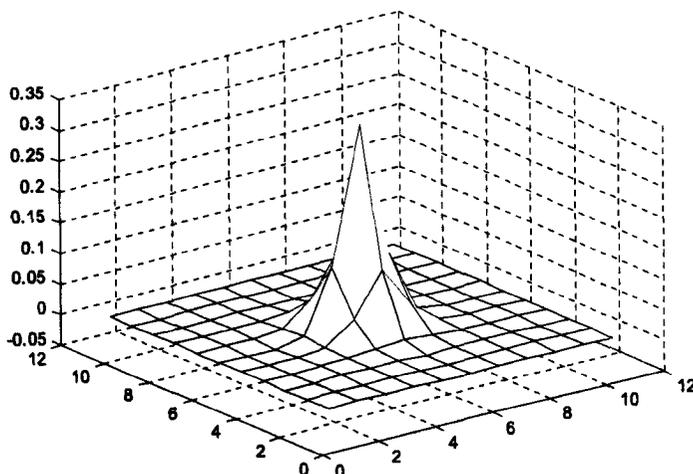


Figure 11: Surface diagram of an 11×11 point spread function.

is shown in figure 11 as a discrete function, 11×11 elements in size. The objective of deconvolution is to remove the detrimental effects of the PSF and obtain the same data as would be observed by a hypothetical, perfectly resolving instrument. Such an instrument would have a PSF equivalent to a two-dimensional Dirac function [49], with no spreading occurring.

Several algorithms have been reported that can be used to deconvolve the image and the PSF. A direct inverse to the blurring process is not usually possible due to the available data being mathematically incomplete (i.e. the data in the observation is not sufficient to describe a unique solution), and the problem therefore being ill-posed. For this reason statistical estimation techniques are usually required. These techniques attempt to estimate the data of maximum likelihood given the observation and the PSF.

The Richardson-Lucy algorithm [50] [51] uses Bayesian probability theory to achieve this. The algorithm hinges on the insight that the input image, the output image, and the PSF may all be considered probability distributions. The mathematical derivation is available in the portfolio document (Appendix E, section 2) and will not be reproduced here, but the algorithm may be described by the following

iterative operation:

$$I^{r+1} = I^r \times \left(\text{PSF} * \frac{\text{observation}}{\text{PSF} * I^r} \right)^\beta \quad (5)$$

where I^r is the current estimate of the true image and I^{r+1} is the next estimate, the $*$ symbol denotes the two-dimensional convolution operation, and the divide and multiply operations and exponent β act in a pixelwise fashion. β is an additional component that is not present in the original papers by Richardson and Lucy, which acts to accelerate the rate at which the algorithm converges. The Richardson-Lucy algorithm is semi-converging [52], meaning it will approach a solution but after a certain number of iterations diverge again rather than converging, which makes it important to stop iterating at the right time. Considering the hardware implications it is clear that the fewer iterations that are required the better. The exponent β is typically chosen as a value between one and three (any higher and the algorithm becomes unstable), and is reported to reduce the number of iterations required by a factor of β [53]. Analysis was necessary to determine the minimum number of iterations that could realistically be implemented in hardware while still producing a sufficient level of deconvolution. It was found that two iterations could be used with a varying acceleration factor (β is equal to two for the first iteration, then reduced to one for the second) to good effect, with a fairly low hardware cost (for more details see Appendix E, section 3).

Due to the low number of iterations required it was decided to unroll the iterative process and implement the algorithm as one continuous datapath. The alternative to unrolling the algorithm is to implement one instance of the loop but then run it at a clock speed that is a multiple of the data rate. This may not always be possible with high bandwidth video signals, and introduces a control aspect to the design. Unrolling the loop removes any control aspects to the algorithm and thus facilitates a high performance hardware design, overcoming some of the difficulties encountered with the LRM algorithm described in the previous section.

6.2.2 Efficient implementation of large convolutions

Each iteration of the deconvolution algorithm contains two two-dimensional convolutions with the PSF. The PSF in figure 11 is a fairly typical real-world example, and

measures 11×11 elements. Performing an 11×11 two-dimensional convolution is a relatively expensive task in hardware: a direct implementation requires 121 multiply operations and 120 additions, and a significant memory requirement to buffer and align data elements that are arriving in a raster scan format. To implement four of these operations (two per iteration) would incur an unmanageable resource overhead.

Some two-dimensional filter kernels may be decomposed into two orthogonal one-dimensional filters acting separately in the horizontal and vertical directions. A filter that can be decomposed in this way is said to be *separable*, meaning that the coefficient matrix that represents the filter can be expressed as the product of a column vector multiplied with a row vector. Separable filters are advantageous because they allow a two-dimensional convolution to be implemented using simple and well-understood structures such as one-dimensional finite impulse response (FIR) filters. A separable 11×11 kernel could then be implemented using 22 multiplies and 21 additions. Unfortunately, while some well-known functions are separable (for instance the Gaussian function), the majority of real-world two-dimensional functions are not separable (they cannot be expressed as the product of a single column vector multiplied with a single row vector).

One method to overcome this is based on the fact that a parallel connection of separable filters results in a non-separable response [54]. This allows several pairs of orthogonal one-dimensional filters to be combined as necessary to reproduce the desired filter response. A diagram of this kind of system is shown in figure 12. Each separable filter is formed from a horizontal and vertical 1D filter that acts on the image rows and columns respectively. A varying number of separable filters may be required depending on the filter response being approximated. The decomposition will be more efficient than a direct implementation whenever

$$k(m+n) < mn \quad (6)$$

where m and n are the dimensions of the filter kernel being decomposed, and k is the number of separable filter pairs required for an exact reproduction.

The implementation of non-separable responses using simple one-dimensional filters was first proposed by Treitel and Shanks [55], who used the term *multistage separable filters*. They show that a matrix can always be decomposed into a finite

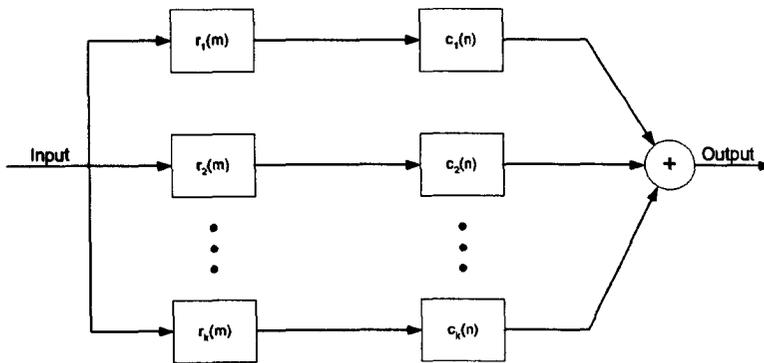


Figure 12: Summation of k separable filters to produce a non-separable response.

number of separable filters, using a mathematical technique known elsewhere as the singular value decomposition (SVD). The SVD of a matrix X is given as (noting that superscript T denotes the matrix transpose operation):

$$X_{m \times n} = U_{m \times n} \Sigma_{m \times n} V_{m \times n}^T \quad (7)$$

where U and V are orthogonal matrices whose columns contain the eigenvectors of the matrices $Q = XX^T$ and $S = X^T X$, and Σ is a diagonal matrix containing values (known as singular values) equal to the square roots of the associated eigenvalues (the eigenvalues of Q and S are identical). The columns of U and V are used as the coefficients of the one-dimensional filters used in the parallel configuration of figure 12. The rank of Σ , which for a diagonal matrix is equivalent to the number of non-zero values, determines k , the number of stages that are required for an exact reproduction.

Efficient use of this method to achieve a perfect reproduction of the original filter kernel is dependent on the kernel's linear dependence. Linear dependence may be thought of as redundancy in the kernel. If the filter kernel, X , being decomposed is linearly independent (i.e. if none of the vectors contained in X can be expressed as a linear combination of the other vectors in X [56]), the SVD will not provide efficiency savings over a direct implementation (equation 6 will not be satisfied). However, it is a feature of the majority of image processing kernels that they should have a linear phase response, since it has previously been shown that the intelligibility of an image relies on its phase characteristic [57], and distortion of an image's phase would

therefore make it unrecognisable. The phase linearity of an image processing filter is manifest in the symmetry of its impulse response, which also implies linear dependence. It is therefore evident that image filtering applications will usually provide opportunities to make significant reductions in hardware costs through application of the multistage separable filter technique. A square, linear phase filter kernel, with m^2 elements, will require a maximum of $\frac{m+1}{2}$ separable stages. Substituting this value for k into equation 6 does not satisfy the inequality (the number of multiplies required would be $m^2 + m$). However, whenever the multistage expansion of the kernel results in less than $\frac{m+1}{2}$ separable stages, equation 6 will be satisfied and the savings will be made.

The resulting one-dimensional kernels will also be symmetrical, which provides an opportunity to employ a well-known optimisation in 1D filter design that can reduce the number of multiplies required by up to a half. The method works by pre-adding data elements that would otherwise be separately multiplied by a common coefficient. It can however cause some problems as it introduces extra latency before the multiply stages, which for the vertical filters especially interferes with the buffering required. It can also result in long, non-pipelined adder chains on the output side, which may cause timing difficulties when the design is placed and routed on the device.

Additional savings are possible if an associated loss of accuracy is acceptable. The singular values are normally listed in the matrix Σ in descending order, with the values of the greatest magnitude contributing more to the final response than the others. Using a reduced number of singular values (and consequently a reduced number of filter stages) will result in an approximation to the original matrix X , with an error that is the equal to the ratio of the sum of the discarded singular values to the sum of all singular values, or

$$\varepsilon_k = 1 - \frac{\sigma_1 + \sigma_2 + \dots + \sigma_k}{\sigma_1 + \sigma_2 + \dots + \sigma_n} \quad (8)$$

which gives the error ε_k from using k singular values from a maximum of n , where $k \leq n$, and σ denotes the individual singular values in the matrix Σ . Studies showed (Appendix E, table 1) that the PSF of figure 11 could be represented to 99.4% accuracy using two stages (44 multiplies, 43 additions). The rate at which the accuracy improves with successive stages is illustrated in figure 13. Similar results

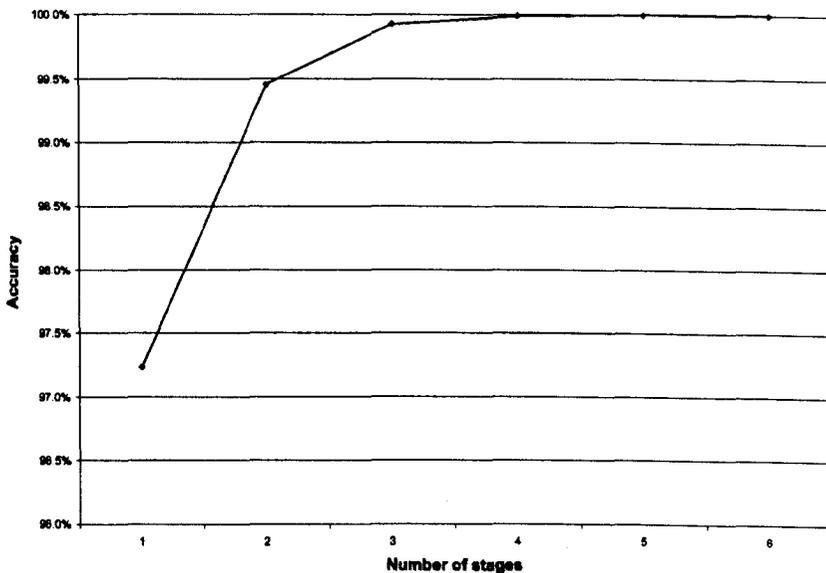


Figure 13: Increase in accuracy with successive stages for the multistage decomposition of the PSF in figure 11.

are reflected in other image processing filters; Andrews [58] notes that a large subclass of 2D filters can be represented with a multistage separable filter with two stages.

The multistage separable filter method provides a simple and efficient means of implementing large two-dimensional convolutions using one-dimensional FIR structures, such as the transpose-form FIR filter, which is ideal for FPGA implementation as it is a systolic, inherently pipelined architecture. The filter coefficients used are usually constant, which means that the multiplications can be implemented using either LUTs or hardware multipliers, providing freedom to utilise the available resources in the best possible manner.

Implementation of multiple stages means that there will be several vertical one-dimensional filters that act on the image columns. As described in chapter 5, vertical alignment of data arriving in a raster scan format is performed using FIFO buffers equal in length to the width of the image. Although this could become resource expensive with a multiple stage filter, the delay lines can be shared between stages to reduce the memory requirement.

Resource	Used	Available	% of XC2VP20
Occupied slices	9,278	9,280	100
4-input LUTs	9,722	18,560	52
Block RAM	83	88	94
Multipliers	88	88	100

Table 7: Resource usage of RL deconvolution algorithm.

6.2.3 Hardware implementation

System Generator was used to map the system into hardware since the main loop body of the algorithm could easily be unrolled and implemented as a datapath. Apart from the convolutions the other operations performed by the algorithm are divides (implemented using a vendor IP core) and multiplies/raising to an exponent, which are all straightforward to implement with existing library blocks. The final design resembled a fully pipelined datapath, and as such could produce one output pixel per clock cycle. The synthesis tools reported a maximum clock speed of 63MHz, which could comfortably process 30fps SXGA video at real-time rates (in a device with sufficient memory).

The resource usage for a VGA implementation is shown in table 7. The smallest device that could accommodate the algorithm in its initial form was a XC2VP50, however subsequent optimisation reduced the size of the design sufficiently to fit on a XC2VP20, so that it could be implemented on a custom Thales FPGA circuit board available for video processing demonstrations (more details on the hardware platform are available in Appendix G). The key areas where optimisations were possible were in reduction of fixed-point word-lengths, and manually mapping multipliers to logic (i.e. LUTs) and delay lines to the block RAM on the device that was previously going unused. The resulting design, including the overhead for the video I/O interfaces, utilised 100% of the slices on the device for either logic functions or routing. Some example images were produced and are shown in figure 14. The deconvolved image shows a general increase in spatial resolution and dynamic range, and some of the finer detail is clearer.

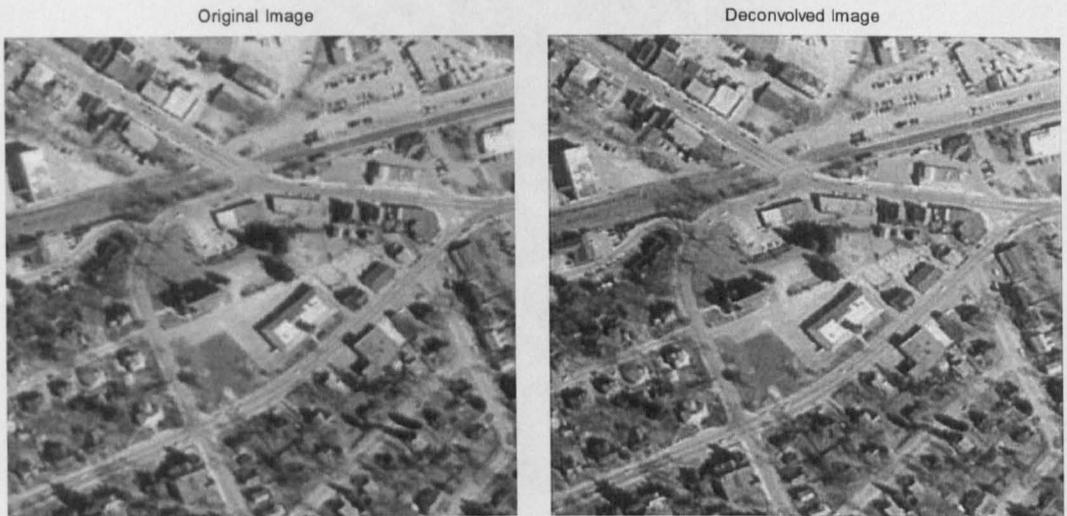


Figure 14: Original image, and same image after two iterations of accelerated RL deconvolution.

Figure 15 shows a difference image between the FPGA implementation and the original Matlab algorithm after two iterations. For presentation purposes five lines have been removed from each edge (to remove the edge-effects of the convolutions), and the image has been inverted and scaled to use the whole greyscale range. Over the region shown in the difference image the maximum error between corresponding pixels in the two implementations is equal to nine greyscale values, and the mean squared error (MSE) between the two implementations is approximately 1.56. Comparison with the original image in figure 14 shows that errors occur along the edge detail within the image, as would be expected for a sharpening algorithm. The main sources of error are the multistage separable filter approximation of the true PSF, and the fixed-point hardware divide operation which has limited precision compared to the Matlab equivalent.

The improvement in image quality that is possible with hardware deconvolution comes at a considerable price: using a whole FPGA for deconvolution may be deemed an unnecessary overhead when there are more basic methods of image sharpening available. One such method is unsharp masking, which is a simple filter that emphasises high-frequency detail and can be implemented with minimal resources, but may also introduce noise into the output image. Introduction of noise may not always be



Figure 15: Inverted and scaled difference image between Matlab and hardware implementations of Richardson-Lucy deconvolution. Five lines have been cropped from each edge to remove the edge-effects of the convolutions.

detrimental, particularly if the image is to be used by humans who can extract and recognise detail in the presence of often quite severe noise and distortion.

Deconvolution remains the most accurate method of removing blur caused by processes that can be modelled by a convolution with a PSF. This covers a wide range of image degradations, including focus blur, motion blur, optical aberrations, and others. That it does so at a high computational cost presents choices to the system designer who must make judgements on the desired accuracy of the output.

6.3 Implementation of pyramidal image fusion

The third major algorithm that was investigated was a method of performing image fusion using Gaussian image pyramids. The implementation that is presented here is based on Matlab source code that is flawed in its interpretation of the published fusion algorithm; however many of the design details remain valid. The algorithm and its implementation will now be summarised along with a description of the problems with it; the portfolio document contains the full details (Appendix F).

6.3.1 Background

Imaging sensors are responsive to a particular region of the electromagnetic spectrum, and it is often beneficial for a scene to be captured with multiple sensors in order to collect more information about a scene than would be captured with a single device. A resulting problem is then how to display this information effectively. Displaying multiple observations of a scene on multiple monitors is wasteful and makes the images difficult to compare. This has led to a requirement for methods of fusing images such that the useful information from each source image is displayed in a single composite. This process may also be used to display observations of a scene that have been taken with a single sensor with multiple points of focus; in this way the depth-of-field of the image can be increased.

Simple methods of combining images take no account of their content, and perform a linear operation such as averaging. A more sophisticated approach uses edge-detection and other operators to identify whole features in the source images that are then extracted and used in the composite. In order to extract features at different scales a multiresolution approach may be used, as described in chapter 5, whereby the input image is decomposed into a set of frequency band-limited images that are localised in both space and spatial-frequency. The multiresolution decomposition used here is the Gaussian pyramid [59], which is a special case of the more generalised area of wavelets.

It should be noted that the images to be fused are assumed to be perfectly aligned. Frequently when multiple sensors are used there is a boresight discrepancy that would introduce errors into the fusion process. Alignment of images that are captured from differing viewpoints is a process known as image registration; a registration algorithm was also investigated during the research and will be discussed in section 6.4.2.

6.3.2 Pyramid generation

The construction of a Gaussian pyramid involves repeated low-pass filtering and subsampling of the input image. The subsampling is by a factor of two both horizontally and vertically, which results in an image with one quarter the number of pixels. The

filter kernel used is a Gaussian, usually of fairly low order, and is chosen such that the resulting frequency content of the image is reduced by one octave at each level of the pyramid, which is necessary in order to prevent the subsampling from causing aliasing. (The Gaussian kernel does not have an ideal brick-wall response, which means that some aliasing can still occur; these slight effects are however usually disregarded for these purposes.) One of the benefits of using a Gaussian kernel is its separability which, as described in section 6.2.2, significantly reduces the computational burden involved in convolving it with the image. An image pyramid is typically generated with three levels above the base image, any more than this and the images become too small to be useful and the edge-effects of the convolutions too detrimental.

In the original paper on the generation of Gaussian pyramids, Burt [60] uses the following notation:

$$G_k(i, j) = \sum_{m=-2}^2 \sum_{n=-2}^2 w(m, n) G_{k-1}(2i + m, 2j + n) \quad (9)$$

which describes the generation of pyramid level G_k from level G_{k-1} using filter kernel $w(m, n)$, in this case a 5×5 Gaussian function. The reverse operation, which is needed in the inverse transform as well as to construct another related image pyramid type known as the Laplacian, involves two-dimensional upsampling and then a further application of the generating filter kernel to smooth the image. This may be defined as

$$G_k(i, j) = 4 \sum_{m=-2}^2 \sum_{n=-2}^2 w(m, n) G_{k+1}\left(\frac{i+m}{2}, \frac{j+n}{2}\right) \quad (10)$$

where only terms for which $\frac{i+m}{2}$ and $\frac{j+n}{2}$ are integers contribute to the output. The multiplication by four is due to the fact that the upsampling and filtering process involves spreading the energy of one pixel over the area of four pixels. More details on this process are provided in Appendix F, section 2.

The Matlab implementation that forms the starting point for the hardware design presented here is based on an incorrect interpretation of this equation where the subsampling is performed before the filtering. This situation bypasses the anti-aliasing function of the filtering stages and therefore the resulting images may exhibit severe aliasing artefacts. The pyramids produced using this method have been used

here for image fusion with passable results, but the method should not be used in general due to the aliasing problem. The implementation that follows does not therefore constitute an exact implementation of image fusion as published by Burt in his original paper.

A further consequence of this method is that by performing the subsampling first the amount of data to be processed is reduced, hence reducing the computational expense of the pyramid generation portion of the algorithm. A brief discussion of the effects of performing the pyramid generation process in this way will be discussed shortly; a more detailed analysis is available in Appendix F, section 6.

The most obvious method of generating Gaussian pyramids in this way is a sequential process, repeated for each level of the pyramid, whereby the image is subsampled in the way it is read out of memory, filtered, then stored back into a separate set of memory locations. This process is inefficient for a number of reasons. Firstly, the Gaussian pyramid with three levels above the base constitutes nearly $\frac{4}{3}$ times the amount of data as the original image, so a significant amount of memory is required. Since storing this amount of data on-chip is infeasible, this would have to be stored off-chip in RAM. However, limitations on memory bandwidth would make getting $\frac{4}{3}$ times the video data rate into and out of memory in real-time extremely problematic. Another detriment to generating the pyramid levels sequentially is that it would be a control-dominant process, and therefore time-consuming and unwieldy to implement in hardware for the reasons previously described.

As has been discussed, to utilise the performance benefits of the FPGA it is important to implement the pyramid generation process as a pipelined datapath. However, due to the subsampling process the later stages of the pipeline operate at lower data rates than the earlier stages, and therefore the only way the process can be implemented as a single pipeline is by reducing the clock rate of the pipeline after each subsampling stage, otherwise the later stages will stall while they wait for data. This mechanism allows generation of a pyramid level to commence before the level beneath it is complete. A block diagram of the pyramid generating pipeline is shown in figure 16. The clock rate of each section of the pipeline is reduced by a quarter, to match the fact that it is receiving one quarter the amount of data. Asynchronous

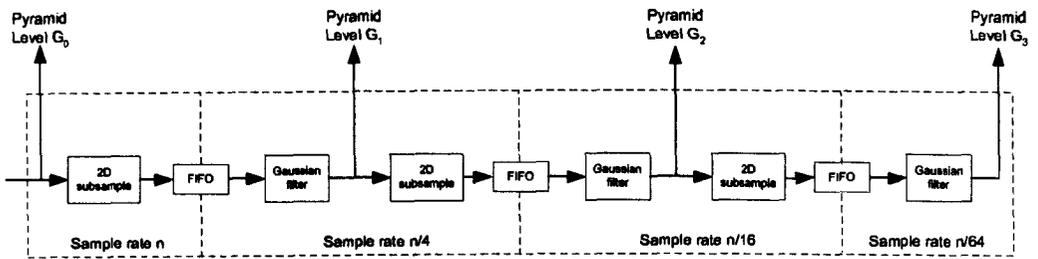


Figure 16: Pipelined image pyramid generation.

FIFO buffers act as the borders between clock domains. Because the clock speeds are integer divisions of each other they remain held in lockstep and metastability is not an issue. Two-dimensional subsampling is achieved through selectively enabling data writes to the FIFO; horizontal subsampling involves discarding every alternate image pixel; vertical subsampling requires discarding every other image row. Note that the lack of a filtering stage before the first subsampling block means that there is nothing to prevent aliasing from occurring, and therefore the design as presented in figure 16 is not a method that should be used in general to produce image pyramids. A correct version of this pipeline is shown in Appendix F, figure 8.

Generation of pyramids in this manner ensured a design that could produce full VGA image pyramids at rates of over 100 frames per second. Other implementations of this process have been reported and a comparison is made in Appendix F, table 1. (It should be noted that the published methods generate image pyramids in the correct manner by filtering and then subsampling. However, as will be discussed below, in performance terms this affects the latency only and has no effect on throughput.) Transformation of an essentially control-based process into a data-path through manipulation of clock rates had made the algorithm more suitable for FPGA implementation, and enabled later stages of the fusion process to be similarly parallelised. In addition, by generating pyramid levels concurrently an overall lower clock speed can be used, with the accompanying reduction in power consumption.

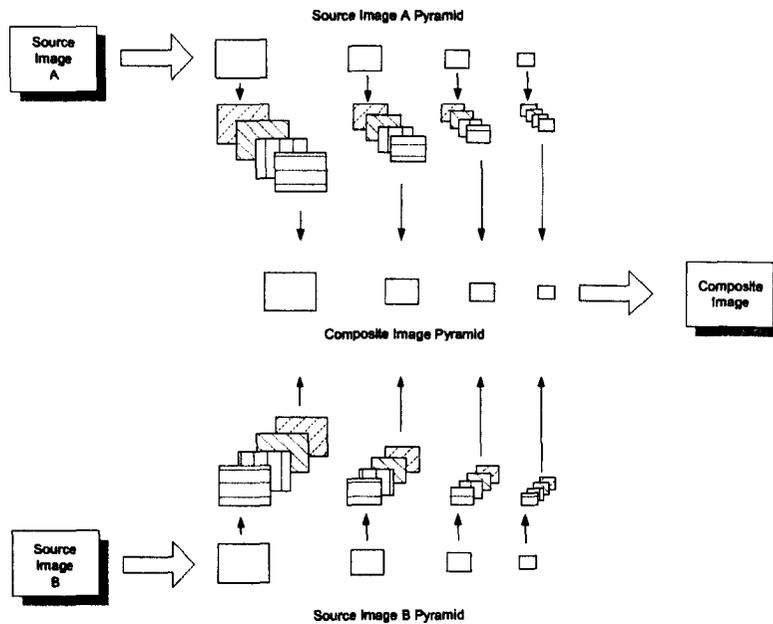


Figure 17: Fusion of gradient pyramids.

6.3.3 Gradient pyramid fusion

Successful fusion of image pyramids relies on selecting the important detail at each scale that should be carried forward into the composite image. The method suggested by Burt [61] uses gradient operators in four orientations: horizontally, vertically, and in the two diagonal directions. All four operators are applied at each scale of the two source pyramids, which results in each level of each pyramid being decomposed into four gradient maps. An overview of this process is shown in figure 17. The resulting data are known as gradient pyramids, and there are now eight of them (four gradient pyramids per source image) to be handled and processed concurrently. This evidently constitutes a substantial amount of data, and reinforces the need to implement the algorithm as a datapath structure. In contrast to the original software version of the algorithm, in hardware the gradient filters are applied to the source pyramids in parallel, splitting the data at each scale into four parallel streams, an example of loop-unrolling. These separate data streams are then combined again using comparators to select from the four orientations the pixel with the greatest absolute value, which is assumed to offer the most salient detail.

The inverse pyramid transform involves further application of two-dimensional filters followed by a summation of the four orientations, and then, starting from the top of the pyramid, each level is expanded using equation 10, and then added to the level beneath it. At the end of this process the resulting image will represent the composite of the two source images. The detail extraction and fusion process, and inverse pyramid transform, are exact implementations of Burt's original method. For more information on these aspects see Appendix F, section 3.

6.3.4 Hardware implementation

System Generator was again used to map the behaviour onto Xilinx library cores. All the elements of the software version of the algorithm had direct hardware equivalents. As with other System Generator models, it was possible to verify the design using the same test stimulus as was used for testing the original version of the algorithm.

The fully pipelined nature of the design ensured that the system could produce output pixels at a rate of one per clock cycle. The design could comfortably run at clock rates that would allow fusion of dual VGA video streams at 30fps, with a latency of <50ms. Experiments showed a hundredfold speedup over a PC-based version of the same algorithm (example images and more details are given in Appendix F, section 5).

The resulting resource usage after synthesis and mapping to the Virtex-2 family of devices is shown in table 8. Given the amount of intermediate data generated by the algorithm it is unsurprising that the most utilised resource is that of on-chip RAM. The RAM is mostly being used to implement the long buffers that are required for the vertical filters. One interesting feature of the algorithm is that it contains no multiply operations, due to the coefficients of the filters all being inverse powers of two and thus easily implemented using shifting and slicing operations. When hardwired resources, such as the dedicated multipliers, are unused they effectively become a source of inefficiency both in terms of area and power, which is one argument against incorporating hardwired elements in FPGAs.

By bearing in mind the need to implement algorithms as datapaths it becomes possible to implement the complex fusion algorithm quickly and efficiently using

Resource	Used	Available	% of XC2VP100
Occupied slices	13,287	44,096	30
4-input LUTs	24,533	88,192	27
Block RAM	430	444	96
Multipliers	0	444	0

Table 8: Resource usage of pyramidal image fusion.

model-based methods. Sequential generation of pyramid stages would not only be time-consuming to implement but would also prevent use of tools such as System Generator due to its lack of support for bidirectional buses and consequent inability to interface to external memory. Conceptualising an algorithm's operation with respect to the raster scan format in which image data usually enters a system is critical to ensure performance in the face of limited memory bandwidths and the difficulties of implementing control-dominant applications.

Alternative method of pyramid generation

As described in section 6.3.2, the method of generating image pyramids that has been presented here is not a true implementation of the image fusion algorithm as published by Burt, because the subsampling is performed before filtering at each stage of the pyramid generation process. There is thus no mechanism to prevent the subsampling from causing aliasing and therefore, depending on the input data, the image pyramid that is produced is liable to distortions. A further consequence is that there is less data to be filtered at each stage, meaning that the effects of the filtering are more pronounced and the resulting image pyramid exhibits an increased level of blur. A comparison of the results from the two methods is available in Appendix F, section 6.

The design presented here could be modified to generate true Gaussian pyramids with minor modification. At each level of the pyramid the blocks for performing the subsampling and filtering could simply be swapped; the multiple clock rate design would still provide an efficient method of implementing the algorithm. When the

image is filtered as part of the pyramid generation process it is twice the size in each dimension than it would be if filtering was done before subsampling. Therefore to generate true Gaussian pyramids the line-buffers for the vertical filters would all need to be twice as large, doubling the memory requirement of this aspect of the algorithm. For a 640×480 image this requires another 2240 memory locations, which is 35kb for 16-bit data (for precision the implementation uses an internal filter representation of 16-bits). This is applied to both images, so the total memory increase is 70kb. Table 8 shows that there are 14 Block RAMs available on the target device; each Block RAM has 18kb capacity so the modified implementation would require an extra four Block RAMs and could therefore be accommodated on the same device. The filters are the same size regardless of the way the image pyramids are generated and so the computational demands are the same; the latency of the process would however be increased by a factor of two. Because the latency of the pyramid generation process constitutes approximately a third of the total latency of the design (along with one third for detail extraction and one third for the inverse pyramid transform), it is estimated that the total latency of the alternative design would be in the region of 60-70ms. The design would remain fully pipelined so, providing the same clock rate was used, throughput would not be affected. The four levels of the pyramid are the same size whether the filtering is done before filtering or after, and so the implementation details for the rest of the algorithm remain the same.

6.4 Other algorithms

Several other smaller algorithms were investigated during the period of research. Two of note were an implementation of a polynomial image registration algorithm, and a PID control algorithm, as well as a number of more simple image processing algorithms. The implementations of the image registration and PID algorithm will now be briefly reviewed, as they provide an opportunity to look at algorithms different in operation to those previously discussed. The full reports written at the time the work was carried out may be found in the portfolio documents (Appendices D

& H).

6.4.1 PID servo control algorithm

Implementing a PID control algorithm offered an opportunity to look at other types of algorithms that could benefit from implementation on FPGA hardware. The PID algorithm is computationally quite simple, and from a systems integration perspective it makes sense to implement the control loop in digital hardware if possible, particularly as it usually takes up relatively few resources and may utilise spare capacity on devices that exist in the system for other tasks.

The PID algorithm is used in a multitude of situations and is widely understood. It acts to produce an input to a system based on an error signal derived from the difference between the system's current and desired output. To do so it uses a computational process with three terms: proportional, integral, and derivative. A closer look at the mathematics and the mapping of this process to System Generator blocks is presented in the portfolio document. However it is worth discussing some aspects of the design here, due to the interesting problems that were faced that are not usually present in implementation of image and video processing algorithms.

The actual calculation aspects of the algorithm are fairly trivial to implement, being made up of multiplies and additions. However, it is usually desirable to have minimal end to end latency in the controller, which can sometimes be problematic. Latency in the controller can introduce instability into a closed-loop system, because the accompanying phase shift can lead to positive feedback. In order to counter forward latency it is usual to increase the sample rate. However, large sample rates are not desirable in digital controllers: the integral and derivative coefficients are scaled by the sample time and in high sample rate systems this results in coefficients that are fractional and very small. Consequently their fixed-point representation can require large word lengths that impact the size of the design when it is mapped to the resources in the FPGA. In addition, any inaccuracies in the fixed-point representations are compounded by the fact that a high sample rate means the controller is performing more calculations, and so rounding errors can build up. For these reasons it is usual to work at a sample rate that is the bare minimum to correctly handle

the bandwidth of the system under control. There is thus a tradeoff, between the need for low sample rates but also low latency, that the control system designer must balance.

As with signal processing, Matlab and Simulink are the environments of choice for control algorithm designers, and designing control systems in System Generator is a natural progression. Enabling the control engineers to utilise model-based design techniques to implement algorithms in hardware has obvious benefits. The more generic advantages of FPGAs are also fully exploitable in control system design: the ability to exploit spatial parallelism allows all three terms of the controller to be calculated concurrently, thus reducing forward latency and controller efficiency, and the added flexibility of the FPGA design allows the controller to be modified or tuned after integration into the rest of the system. Since this work was carried out, other reports have been published that come to similar conclusions [62].

The resulting design used in the region of 650 slices, which is less than 10% of a small Virtex-2 device. The work was successful in that the controller designed during this phase of research became an integral part of the focusing mechanism on a commercial Thales product, and implementation of controllers in hardware looks set to continue in this manner for the foreseeable future.

6.4.2 Image registration by polynomial warping

Image registration is a technique to apply affine transformations to an image, such that it is given the appearance of being captured from a different viewpoint. This is necessary in many fields where images taken at different times or with different sensors must be compared. It is also a critical step that must be performed before images may be fused. The kind of transformations that are necessary are usually global (i.e. they do not change over the image), and are composed of rotations, scaling, and translations. In addition, the image to be registered may be a different resolution to the image to which it is being compared.

The method of polynomial image warping depends on manual identification of control points in each image, i.e. points in each image that correspond to one another. These control points are used to form a system of simultaneous equations which, when

solved, provide coefficients for polynomial functions that translate input coordinates to output coordinates (for more details on this process see Appendix H, section 2). This is equivalent to a spatial mapping between the two images. Interpolation is usually required to determine pixel values where the desired location falls between pixel boundaries.

The control points are determined once and, providing the two image capturing locations do not move relative to one another, the transformation remains fixed. For this reason the calculations to determine the spatial mapping can be performed off-line and do not require real-time processing. The resulting polynomials are fairly simple to implement in hardware, consisting of nothing more complex than multiplies and additions, and are ideal candidates for System Generator design. The main difficulty presented by an algorithm like this is caused by the need to access elements of the input image in a non-linear order when it is not possible to store a whole frame of data on-chip. For this reason it is necessary to buffer image data in external SDRAM before being read into the device. The image registration then occurs by modifying the SDRAM controller such that the addresses it accesses were determined by the polynomial generators.

Limited memory bandwidth is an issue when data must be buffered off-chip and multiple input pixels are needed to generate a single output pixel. This would be the case with some of the more complex interpolation methods that could be used in the registration algorithm. Here we can alleviate this concern through use of nearest-neighbour style interpolation, which is trivial to implement, just requiring a rounding of the desired pixel location to the nearest integer value.

The system was implemented as an augmentation to the video processing demonstration platform that had previously been developed (as described in Appendix G), and used a small amount of additional resources (see Appendix H, table 1). The algorithm is notable for being an example where model-based techniques are applicable for the mathematical operations, but hand-crafted VHDL and manual design techniques are required for system-level aspects such as the memory controller and interfaces. A system such as this cannot be wholly implemented in System Generator, since the tool does not include support for bidirectional buses, an integral part

of a memory controller. The memory controller also includes some fairly complex control mechanisms, since it must detect whenever a memory row change is required and issue the necessary activate and precharge commands to the memory.

The complexity of the memory controller and the necessity for bidirectional buses meant that a VHDL implementation of an FSM was necessary to perform this task. However the remaining parts of the design, the polynomial functions, are ideally suited to a System Generator implementation. This demonstrates how it can be beneficial to implement different parts of a design using the most appropriate tools; here the datapath elements are quickly and easily implemented in System Generator, but the control-dominant parts require different design methods. The memory controller state machine could alternatively have been constructed in one of the many commercial products that allow graphical design entry and automatic code generation for state machines. This mixture of automated tools and hand-coded HDL will continue to be common when designing at a system level.

Chapter 7

Design Guidelines

This chapter will attempt to crystallise some of the general design guidelines that came out of the research. Although these are not concrete design rules, they are intended to convey some of the experiences and findings of the research that may assist in future efforts to implement image and video processing algorithms on FPGA.

7.1 Convolution

Two-dimensional convolutions formed a substantial part of the outcomes of the research. The work on deconvolution required efficient implementations of large (11×11) convolutions to be found, and discussed how the separability of a kernel can be found using the singular value decomposition.

When the kernel is fairly small (say 3×3 or 5×5) and non-separable a direct implementation is most efficient, using n^2 multiply and $n^2 - 1$ add operations, where n is the order of the filter.

Small, separable kernels, such as Gaussian filters, can be separated into a single pair of orthogonal 1D filters. This reduces the number of multiply operations from n^2 to $2n$, and the number of add operations from $n^2 - 1$ to $2(n - 1)$.

Larger kernels that are also linear phase can use multistage separable filters to achieve a resource efficient implementation. Phase linearity, characterised by symmetrical filter kernels, represents redundancy that can be avoided in hardware. The number of multiply operations required for a multistage separable filter is $2kn$, and

the number of additions is $2k(n - 1)$, where k is the number of filter stages used. To keep the memory requirements down it is important that when several filter stages are used a single set of FIFO line-buffers is shared between all the vertical filters, as shown in Appendix H, figure 6. Further savings are also possible when accuracy can be sacrificed, as demonstrated in Appendix H, section 4.1.

Linear phase 1D kernels can pre-add data that is to be multiplied by common coefficients (and thus reduce the number of multiply operations that are required), but care must be taken when inserting register delays to avoid long adder chains while still maintaining correct operation.

Large filter kernels that are not linearly dependent are more suitable for implementation using FFT methods. Although the memory requirements are significantly greater, the ease of performing a multiplication in the transform domain rather than a 2D convolution can result in overall savings in terms of resources and delay.

7.2 Control- versus datapath-dominant code

As has been described in section 6.1.4, whether a segment of code is primarily concerned with control or dataflow operations directly determines how effective and efficient a hardware implementation can be. Datapath-dominant algorithms map to hardware in an intuitive way, with spatial and temporal parallelism easy to identify and exploit. Control-dominant code does not map easily to the pipelined datapath structure favoured by high-speed FPGA designs.

But how can control-dominant code be identified? Simple loops and branches are control-oriented elements that are common to the majority of high-level program code, but with careful manipulation based on techniques such as loop-unrolling these simple loops, even when nested quite deeply, can be removed.

More problematic however are loops with non-static limits, which execute in a non-deterministic way, and cannot be easily represented by a regular hardware structure. Code that includes loops with non-static limits will cause problems with standard HDL design methods, and are especially unsuitable for tools such as System Generator. Use of finite-state machines, or possibly redesign of the algorithm, may be

necessary. The manual scheduling and allocation process described in this portfolio is one such method of proceeding with the implementation of an algorithm with complex control flow. Unfortunately FSM structures can become quite unwieldy and inflexible, and do not greatly benefit from the computational advantages offered by FPGAs.

It is therefore suggested that during the algorithm design phase, loops with non-static limits are avoided if possible when a hardware implementation is to be attempted. Techniques such as recursion should also be avoided for similar reasons.

7.3 Optimisations

Several well-known optimisations used in software compilers are valuable tools for a hardware designer and should be applied where possible. Sometimes these optimisations may be applied by experienced hardware designers intuitively, but it may be useful to formally discuss them here.

Loop unrolling has been mentioned above. Loops can either be unrolled temporally or spatially, depending on the data-dependencies between the iterations: if an iteration relies on the previous one for data they can be implemented sequentially and pipelined (parallelised in time); if they are completely independent they can be implemented to run concurrently (parallelised in space). If a loop cannot be unrolled then it becomes necessary to run the hardware at a multiple of the data rate (for real-time operation). Trade-offs are often possible, for instance unroll half the loop iterations and run the design at twice the data rate if constraints prevent completely unrolling the loop.

Common sub-expression elimination is another useful optimisation used in high-level compilers, where multiple expressions that produce an identical result are replaced by a single variable. This can be combined with constant propagation, and usually results in simplification of computational steps in the algorithm. Situations where these techniques can be useful often occur when loops are unrolled.

7.4 General computation

Complex mathematical operators must be handled effectively. Divide or square-root operations for instance use many resources and cause delay. In some circumstances it may be more effective to use look-up tables, as was the case with the implementation of the LRM algorithm described in detail in Appendix B. When using LUTs in this way there is a trade-off between the level of accuracy and the amount of memory that is consumed, but it is a useful technique when execution speed is paramount.

Multiply operations are fairly well catered for on modern day FPGAs, but multiplications by a constant can often be implemented with LUTs or alternatively using just shifts and adds, as used in Appendix B, section 4.3. This method was also used to avoid use of multipliers in the implementation of image fusion (Appendix F). As an example $y = 7.5x$ can be replaced by $y = 8x - 0.5x$, i.e. a single multiplication with a constant (7.5) is replaced by two binary shifts (with zero cost in hardware) and a subtraction. If dedicated multipliers are not available on the device this is likely to be a more efficient implementation. Formally, this is a use of canonical signed digit (CSD) representation.

More obvious examples are issues such as using powers of two wherever possible in an algorithm's design (for instance in the block sizes of the LRM algorithm, Appendix B), which tends to simplify hardware implementations. If efficient hardware implementations of algorithms are to be facilitated it is important that these considerations are made at an early stage in the design process.

Chapter 8

Conclusion

Efficient implementation of behavioural algorithms in FPGA hardware is a topic of research and debate in both the academic and industrial communities. The EngD programme offers an excellent opportunity to approach the problem from both vantage points simultaneously. The algorithms central to the business of Thales Optonics, primarily (but not limited to) image and video processing, present multiple challenges to the engineer wishing to utilise FPGAs to their fullest potential.

This portfolio thesis began by introducing the state of modern FPGA technology and how recent developments are making FPGAs the platform of choice for many DSP applications, and in particular those of the defence market. The nature of video processing algorithms was then discussed, highlighting some of the operations common to the application domain such as two-dimensional convolutions, multiscale transforms, and interpolation. The need for video processing algorithms to process data temporally when there is limited storage capacity presents difficulties, and it is often the case that algorithms must be approached conceptually with this limitation in mind.

The currently predominant method of designing firmware for FPGAs, hand-coded HDL at a structural level, was then looked at. Some of the problems of this approach, such as its requirement for the designer to have an intimate knowledge of the underlying hardware structures, prevent widespread adoption by many of the people who could benefit most from the capabilities FPGAs offer. The problems facing firmware designers in mapping behaviour onto FPGA resources was discussed, and

brief descriptions were given of some of the latest developments in EDA tools that have appeared to tackle this challenge.

One of the problems facing Thales and other defence contractors in adopting new methodologies is that they must be able to support their products for a nominal lifespan of 25 years, a very substantial length of time given the rate of progress in the electronics industry. It is envisaged that languages such as VHDL and Verilog will continue to be supported (in some form) for this amount of time, but since this is less probable with many of the more esoteric languages that have been proposed in recent years the reaction to them in the defence industry has been muted. An approach that may lead to integration between disciplines is that demonstrated through languages such as SystemC. The ability to use this language for the whole design flow, from system-level specification through to hardware/software partitioning, and subsequent implementation of both hardware and software, should not be overlooked. The foundation of SystemC, C++, is a proven language that already has a multitude of tools and development environments available.

The practical phase of the research began with the implementation of the LRM contrast enhancement algorithm. This work demonstrated how algorithms may be characterised according to the nature of their execution flow, and the associated control/datapath model is a useful theoretical framework with which to approach an algorithm and understand the influences that will shape any hardware implementation. While datapath-dominant algorithms are well suited to current hardware design tools and techniques, control-dominant algorithms such as the LRM algorithm often cause problems. The research showed that in these circumstances use of the control/datapath model can reduce the open-endedness of the problem whilst still providing an efficient implementation. Use of an intermediary language, in this case ANSI C, allowed a formal progression to take place from the behavioural-level Matlab code to a structural style representable in the synthesisable subset of VHDL. This progression includes tasks that are well known from the field of synthesis, including loop unrolling, scheduling, and allocation. The resulting design is implemented as manually scheduled FSMs which, although time-consuming to design, are structures that should be familiar to most hardware engineers. The benefit of using C over

a HDL is that the model remains executable throughout, and therefore verification is much simpler than methods based on HDL simulators. The methodology is an example of model-based design within existing implementation flows.

One of the key findings of this section of work was how there is a requirement for the designer to be aware of how code transformations may improve the suitability of an algorithm for hardware, including maximising opportunities for both spatially parallel and temporally parallel (pipelined) processing. This is particularly the case for any looping or vectorised instructions in the original description. FPGAs rely on exploitation of both types of parallelism to achieve performance gains, and so when undertaking manual behavioural synthesis in this way it is important that the appropriate transformations are employed where appropriate.

This concept was illustrated by the implementation of Richardson-Lucy deconvolution, an algorithm that could naturally exploit both types of parallelism: spatially in the multistage FIR filters; and temporally in implementing successive iterations of the algorithm as a single pipeline. In doing so the algorithm also became a design that conformed to a datapath-dominant style, and as such was an example of a complex system that could be readily implemented using System Generator. In these circumstances, the ability of System Generator to abstract away some low-level details and leave the behavioural design as a simple connection of blocks greatly speeds the work that would otherwise be done in hand-coded HDL. Tasks such as register retiming and fixed-point design are much simpler to perform in a graphical environment.

Another feature of the Richardson-Lucy algorithm was the demonstration of multistage separable filters as a very efficient means of implementing the large linear-phase two-dimensional convolutions that are common in the application domain. This is an example of the use of mathematical techniques to alter the format of the algorithm into something more suitable for hardware. Techniques such as this can provide huge benefits, but are something that often falls in-between the usual remit of both algorithm and hardware engineers. Usually, the algorithm engineers do not know enough about the underlying hardware to know whether a particular method is preferable, and the hardware engineers are wary of modifying algorithms that are presented to them for implementation. The division of responsibility be-

tween those who design the behaviour (typically systems or algorithm engineers) and those designing the corresponding structure (the hardware/firmware engineers) is a continuing problem. Given that each discipline will be using their own software tools and techniques, data representations, and verification strategies, it is clear that productivity will be less than optimal while this division between them exists. Software tools alone cannot solve this problem, and companies that wish to ease this process should look at closer integration between the two camps, or dedicated engineers that have knowledge of both fields.

The implementation presented here of pyramidal image fusion provides an example of how an algorithm can be reconceptualised given an understanding of the underlying hardware. The implementation challenges the established method of performing multiscale image decompositions by utilising the insight gained from the control/datapath model. By making a conscious effort to remove complex control from the algorithm through manipulation of clock rates the resulting system is more effective for processing raster scan data and is consequently a more efficient design, which allows subsequent processing to be parallelised and therefore take advantage of the benefits FPGAs offer.

The PID algorithm discussed here shows that FPGAs can be put to good use on other types of algorithms besides high-speed data processing. The work highlighted the need for careful understanding of issues such as fixed-point design and coefficient representation, but also demonstrates the flexibility of FPGA technology. When designing embedded systems integration of multiple functions onto a single device is desirable for a number of reasons, not least power savings and cost reduction.

It is clear that the problem of implementing these algorithms in FPGAs is not going to be solved in the near future with a single solution. The tools available at the present time may assist in the process, but a fully-automated route from behaviour to structure is currently unfeasible. It is necessary to have sufficient understanding of the algorithms to be able to understand how combinations of the currently available tools may be employed to the greatest effect. The approach that is advocated here is the formulation of an informal knowledge base of techniques and solutions that can assist in future design efforts. The fact that many image processing algorithms

contain similar features, such as convolutions and interpolations, allows us to do this. In addition, the control/datapath model provides a theoretical justification as to why some algorithms can cause so much difficulty when forming a hardware realisation, and that, coupled with a full understanding of the available techniques, can successfully smooth the implementation process.

8.1 Future direction

The work covered in this EngD forms part of an ever changing environment, as new methodologies and design tools are released to tackle aspects of the implementation problem. As such there are always new tools that can be trialled for their suitability to some of the issues that have been highlighted. Work using System Generator has formed a significant part of this research, but there are competing products that have not been trialled in detail. The Accelchip environment is one such tool that shows much promise for the problems described here, as do the developments in design languages such as SystemC and SystemVerilog, and future work could take a closer look at the capabilities offered by these methodologies. It may also be beneficial to develop an IP library for environments such as System Generator that supplements its existing functionality and makes it more suitable for image and video processing.

The practical work that has been carried out has enabled a number of design guidelines to be suggested that may improve the efficiency of future efforts to implement image and video processing algorithms. Expansion of this work could either broaden the current guidelines to cover more aspects of the application domain, or use quantitative data to formalise the guidelines into more strict design rules. Both tactics would be likely to prove beneficial.

One way in which the guidelines could be broadened would be to investigate an recurring feature of video processing algorithms that was only briefly covered here: interpolation. A good research topic would be to find efficient implementations of several interpolation algorithms and produce results comparing their effectiveness versus hardware costs. It would also be particularly beneficial to have a range of interpolation algorithms implemented as IP cores that could be inserted into a design

where needed.

The manual process of scheduling and allocation presented here could be automated in several ways. In particular the conversion of Matlab code to C is possible with in-built Matlab functions, and would take one of the manual stages out of the process.

As a means of extending the work on Richardson-Lucy deconvolution it could be useful, given the computational expense of the algorithm, to trial other methods of removing blur from images. Benefit could be gained from producing definitive results quantifying the improvement offered by deconvolution methods compared to other methods of image sharpening. There are also algorithms for performing blind deconvolution, i.e. when the PSF is unknown, that would make good topics of research concerning their hardware implementation.

The work on multiscale decompositions that formed part of the implementation of the pyramidal image fusion algorithm could also be extended in several ways. In particular, an expansion of the work on Gaussian pyramids to include the more general field of wavelet transforms could provide a source of commercially valuable research.

References

- [1] J. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1217–1225, Oct. 1990.
- [2] V. Betz and J. Rose, "Circuit design, transistor sizing and wire layout of FPGA interconnect," in *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999*, San Diego, CA, USA, 1999, pp. 171–174.
- [3] T. Tuan and B. Lai, "Leakage power analysis of a 90nm FPGA," in *Custom Integrated Circuits Conference, 2003. Proceedings of the IEEE 2003*, Sep. 21–24, 2003, pp. 57–60.
- [4] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A quantitative analysis of the speedup factors of FPGAs over processors," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on field programmable gate arrays*. New York, NY, USA: ACM Press, 2004, pp. 162–170.
- [5] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. New York, NY, USA: ACM Press, 2004, pp. 171–180.
- [6] A. DeHon, "The density advantage of configurable computing," *IEEE Computer*, vol. 33, no. 4, pp. 41–49, Apr. 2000.
- [7] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, Apr. 1998.

- [8] *ITU-R BT.656-4*, International Telecommunication Union Std., Feb. 1998.
- [9] C. T. Johnston, K. T. Gribbon, and D. G. Bailey, "Implementing image processing algorithms on FPGAs," in *ENZCON '04: Proceedings of the 11th Electronics New Zealand conference*, 2004, pp. 118–123.
- [10] G. D. Haan and E. B. Bellers, "Deinterlacing - an overview," *Proceedings of the IEEE*, vol. 86, no. 9, pp. 1839–1857, Sep. 1998.
- [11] G. J. Awcock and R. Thomas, *Applied image processing*. Macmillan Press, 1995.
- [12] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, 2nd ed. Academic Press, 1982, vol. 1.
- [13] J. Watkinson, *MPEG-2*. Butterworth-Heinemann, 1999.
- [14] T. Chen, "The past, present, and future of image and multidimensional signal processing," *IEEE Signal Processing Magazine*, vol. 15, no. 2, pp. 21–58, Mar. 1998.
- [15] Z. Ye, J. Suri, Y. Sun, and R. Janer, "Four image interpolation techniques for ultrasound breast phantom data acquired using Fischer's full field digital mammography and ultrasound system (FFDMUS): a comparative approach," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 2, Sep. 11–14, 2005, pp. 1238–41.
- [16] WiT. DALSA Digital Imaging. Accessed: 03/01/07. [Online]. Available: <http://www.logicalvision.com/default.htm>
- [17] I. Moussa, Z. Sugar, R. Suescun, M. Diaz-Nava, M. Pavesi, S. Crudo, L. Gazi, and A. Jerraya, "Comparing RTL and behavioral design methodologies in the case of a 2M-transistor ATM shaper," in *Design Automation Conference, 1999. Proceedings. 36th*, New Orleans, LA, USA, Jun. 21–25, 1999, pp. 598–603.

- [18] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 4, pp. 464–475, Apr. 1991.
- [19] J. Carletta, R. Veillette, F. Krach, and Z. Fang, "Determining appropriate precisions for signals in fixed-point IIR filters," in *Design Automation Conference, 2003. Proceedings*, Jun. 2–6, 2003, pp. 656–661.
- [20] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *FPGA '98: Proceedings of the 1998 ACM/SIGDA sixth international symposium on field programmable gate arrays*. New York, NY, USA: ACM Press, 1998, pp. 191–200.
- [21] R. J. Offen, *VLSI image processing*. Collins, 1985.
- [22] L. Ghasemzadeh and Z. Navabi, "A fast cycle-based approach for synthesizable RT level VHDL simulation," in *Microelectronics, 2000. ICM 2000. Proceedings of the 12th International Conference on*, Tehran, Oct. 31–Nov. 2, 2000, pp. 281–284.
- [23] Towards quicker high level chip design. Information Society Technologies. Accessed: 03/01/07. [Online]. Available: <http://istresults.cordis.lu/index.cfm?section=news&tpl=article&BrowsingType=Features&ID=63144>
- [24] Xilinx launches ESL initiative to accelerate adoption of system level design for FPGAs. Accessed: 18/10/06. [Online]. Available: http://www.fpgajournal.com/news_2006/03/20060313_01.htm
- [25] System Generator user guide. Xilinx Inc. Accessed: 03/01/2007. [Online]. Available: http://www.xilinx.com/support/sw_manuals/sysgen_ug.pdf
- [26] DSP Builder user guide. Altera Inc. Accessed: 03/01/2007. [Online]. Available: http://www.altera.com/literature/ug/ug_dsp_builder.pdf
- [27] J. Wilber. (2006, Sep.) BAE Systems proves the advantages of model-based design. MATLAB Digest. Accessed: 8/11/06. [Online]. Available: <http://www.mathworks.com/company/newsletters/digest/2006/sept/bae.html>

- [28] Simulink HDL Coder datasheet. The Mathworks Inc. Accessed: 05/01/2007. [Online]. Available: <http://www.mathworks.com/mason/tag/proxy.html?dataid=7214&fileid=35737>
- [29] P. Banerjee, "An overview of a compiler for mapping MATLAB programs onto FPGAs," in *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, Jan. 21–24, 2003, pp. 477–482.
- [30] Impulse Accelerated Technologies. Accessed: 03/01/07. [Online]. Available: <http://www.impulsec.com/>
- [31] Catapult Synthesis datasheet. Accessed: 03/01/07. [Online]. Available: http://www.mentor.com/products/c-based_design/catapult_c_synthesis/upload/Catapult_DS_0308.pdf
- [32] DK Design Suite product brief. Accessed: 03/01/07. [Online]. Available: <http://www.celoxica.com/techlib/files/CEL-W0602151E9V-45.pdf>
- [33] A. Donlin, A. Braun, and A. Rose, "SystemC for the design and modeling of programmable systems," in *Field Programmable Logic and Application 2004: 14th International Conference, FPL 2004, Antwerp, Belgium, August 30-September 1, 2004, Proceedings (Lecture Notes in Computer Science)*. Springer, 2004, pp. 811–821.
- [34] S. Sutherland, "A proposal for a standard synthesizable subset for SystemVerilog-2005: What the IEEE failed to define," in *Proceedings of Design and Verification Conference, DVCon*, Feb. 2006.
- [35] K. T. Gribbon, D. G. Bailey, and C. T. Johnston, "Design patterns for image processing algorithm development on FPGAs," in *TENCON 2005 2005 IEEE Region 10*, Melbourne, Australia, Nov. 2005, pp. 1–6.
- [36] D. Bailey, "Space efficient division on FPGAs," in *Electronics New Zealand Conference (EnzCon'06)*, Christchurch, New Zealand, 2006.

- [37] K. T. Gribbon and D. G. Bailey, "A novel approach to real-time bilinear interpolation," in *Electronic Design, Test and Applications, 2004. DELTA 2004. Second IEEE International Workshop on*, Jan. 28–30, 2004, pp. 126–131.
- [38] K. Benkrid, D. Crookes, J. Smith, and A. Benkrid, "High level programming for FPGA based image and video processing using hardware skeletons," in *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, 2001, pp. 219–226.
- [39] K. Benkrid and S. Belkacemi, "Design and implementation of a 2D convolution core for video applications on FPGAs," in *Digital and Computational Video, 2002. DCV 2002. Proceedings. Third International Workshop on*, Nov. 14–15, 2002, pp. 85–92.
- [40] A. Benkrid, K. Benkrid, and D. Crookes, "Design and implementation of a generic 2D orthogonal discrete wavelet transform on FPGA," *2003. FCCM 2003. 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 162–172, Apr. 9–11, 2003.
- [41] A. E. Nelson, "Implementation of image processing algorithms on FPGA hardware," Master's thesis, Vanderbilt University, Nashville, TN, USA, May 2000.
- [42] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, "A system for synthesizing optimized FPGA hardware from Matlab," in *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, San Jose, CA, USA, 2001, pp. 314–319.
- [43] M. Haldar, A. Nayak, N. Shenoy, A. Choudhary, and P. Banerjee, "FPGA hardware synthesis from MATLAB," in *VLSI Design, 2001. Fourteenth International Conference on*, Bangalore, Jan. 3–7, 2001, pp. 299–304.
- [44] P. M. Athanas and A. L. Abbott, "Real-time image processing on a custom computing platform," *Computer*, vol. 28, no. 2, pp. 16–25, Feb. 1995.

- [45] I. S. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of fast Fourier transforms for real-time signal and image processing," *Vision, Image and Signal Processing, IEE Proceedings-*, vol. 152, pp. 283–296, Jun. 3, 2005.
- [46] J. D. Fahnstock and R. A. Schowengerdt, "Spatially variant contrast enhancement using local range modification," *Optical Engineering*, vol. 22, no. 3, pp. 378–381, 1983.
- [47] D. D. Gajski and L. Ramachandran, "Introduction to high-level synthesis," *IEEE Design and Test of Computers*, vol. 11, no. 4, pp. 44–54, 1994.
- [48] R. A. Bergamaschi and A. Kuehlmann, "A system for production use of high-level synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 3, pp. 233–243, Sep. 1993.
- [49] P. A. Jansson, *Deconvolution of Images and Spectra*. Academic Press, 1996.
- [50] W. H. Richardson, "Bayesian-based iterative method of image restoration," *Journal of the Optical Society of America*, vol. 62, no. 1, pp. 55–59, Jan. 1972.
- [51] L. B. Lucy, "An iterative technique for the rectification of observed distributions," *Astronomical Journal*, vol. 79, no. 6, pp. 745–754, Jun. 1974.
- [52] M. Bertero and P. Boccacci, "Image deconvolution," in *From cells to proteins: imaging nature across dimensions. Proceedings of the NATO Advanced Study Institute*, V. Evangelista, L. Barsanti, V. Passarell, and P. Gualtieri, Eds., Pisa, Italy, Sep. 2004.
- [53] H. Lanteri, M. Roche, O. Cuevas, and C. Aime, "A general method to devise maximum-likelihood signal restoration multiplicative algorithms with non-negativity constraints," *Signal Processing*, vol. 81, no. 5, pp. 945–974, 2001.
- [54] D. Dudgeon and R. M. Mersereau, *Multidimensional digital signal processing*. Prentice-Hall, 1984.

- [55] S. Treitel and J. L. Shanks, "The design of multistage separable planar filters," *IEEE Transactions on Geoscience Electronics*, vol. 9, no. 1, pp. 10–27, Jan. 1971.
- [56] H. Anton, *Elementary Linear Algebra: with applications*, 9th ed. John Wiley & Sons, 2005.
- [57] A. V. Oppenheim and J. S. Lim, "The importance of phase in signals," *Proceedings of the IEEE*, vol. 69, no. 5, pp. 529–541, May 1981.
- [58] M. Andrews, "Architectures for generalized 2D FIR filtering using separable filter structures," in *Acoustics, Speech, and Signal Processing, 1999. ICASSP '99. Proceedings., 1999 IEEE International Conference on*, vol. 4, Phoenix, AZ, USA, Mar. 15–19, 1999, pp. 2215–2218.
- [59] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA Engineer*, vol. 29, no. 6, pp. 33–41, Nov. 1984.
- [60] P. Burt and E. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 532–540, Apr. 1983.
- [61] P. J. Burt and R. J. Kolczynski, "Enhanced image capture through fusion," in *Computer Vision, 1993. Proceedings., Fourth International Conference on*, Berlin, May 11–14, 1993, pp. 173–182.
- [62] S. Gifford, "Some experiences with FPGA controllers," in *International Conference Control (ICC2006)*, Glasgow, UK, Aug./Sep. 2006.

Efficient Implementation of Video Processing Algorithms on FPGA

Volume 2 (of 2)

Oliver Sims

A themed portfolio submitted to
The Universities of

Edinburgh,
Glasgow,
Heriot-Watt,
and Strathclyde

for the degree of
Doctor of Engineering in System Level Integration

© Oliver Sims, 2007

Contents

Appendix A: Dynamic Reconfigurability for Thales Optronics	99
Appendix B: Implementation of LRM Algorithm using Scheduling & Allocation	133
Appendix C: Implementation of LRM Algorithm using System Generator	166
Appendix D: Implementation of PID Servo Control Algorithm	184
Appendix E: Implementation of Richardson-Lucy Deconvolution	193
Appendix F: Implementation of Pyramidal Image Fusion	213
Appendix G: Video Processing Demonstration Platform	233
Appendix H: Implementation of Image Registration by Polynomial Warping	245

**Appendix A: Dynamic
Reconfigurability for Thales
Optronics**

Dynamic Reconfigurability for Thales Optronics

A Review of FPGA Applications and Techniques

September 2003

Summary: An initial report was produced that acted as an opportunity to review the literature and identify some of the issues surrounding FPGAs and in particular dynamic reconfigurability.

Oliver Sims

EngD 1st Year

Industrial Sponsor: Thales Optronics

1 Introduction

The drivers behind reconfigurable computing in the context of Thales Optronics' systems are lower spares inventories through hardware reuse, increased functionality for a given amount of hardware meaning improvements in power and space consumption, and improved performance through the use of hardware tailored to the application. From a functional viewpoint, three levels of reconfiguration can be defined as follows:

Theatre dependent

The value of whole groups of algorithms may be largely dependent on the theatre of operation. For instance, contrast enhancement techniques may give significant benefits in a featureless landscape, but be less effective in an urban environment. Reprogramming time here is not an issue (within limits), and data storage is unlimited. Implementing reconfigurability in this context may simply be a case of exchanging a Flash memory module with the system powered down.

Mission dependent

Different missions within the same theatre will require different functionality, and consequently a range of algorithms will be selected for their suitability to a particular mission. Reprogramming will be performed before commencement of mission and will have low time and data constraints.

Mode dependent

Reconfiguration performed during operation, to select the required functionality as required. The speed of the reconfiguration should be as high as possible, in order to minimise downtime. Because of the time constraints and the need for the system to remain as operational as possible whilst reconfiguring, it is envisaged that dynamic partial reconfiguration will be the most suitable technique.

FPGAs can be configured in a variety of ways, and since each of these timeframes for reconfiguration has different requirements in terms of speed and data, different styles of reconfiguration may be applicable depending on the context.

There exists a requirement for research to be carried out into the feasibility of developing a reconfigurable system that satisfies the requirements listed above. As a precursor to commencement of the research, a survey was carried out of the available literature for related applications and relevant technologies.

This report details the findings of the survey, and is split into the following sections: Section 2 looks at similar applications of dynamically reconfigurable logic in the available literature. Section 3 gives an overview of programming methods of Xilinx FPGAs, and the configuration management solution, SystemACE. Section 4 focuses on partial reconfiguration, including a section on the specific case of self-reconfiguration. Section 5 describes two of the CAD tools that support dynamic reconfiguration, and section 6 concludes the report.

2 Background research

A comprehensive survey of the available literature was carried out to cover previous research in the areas of reprogrammable logic, reconfigurable image processors, and dynamic partial reconfiguration. Because of the inter-disciplinary nature of the subject, relevant articles have been found in a number of different sources ranging from computer science to electronic engineering and specialist reprogrammable logic publications.

2.1 Fundamentals

Excellent introductions to the architectures of FPGAs can be found, and although the technology has evolved rapidly in recent times, the descriptions of such matters as routing architectures and SRAM switches given in [1], [2], [3] are still relevant today. These papers also cover CPLDs and (E)EPROM type technologies.

2.2 Applications

The applicability of FPGAs to certain tasks is examined in several articles, and it is often concluded that FPGAs are most effective when applied to data-parallel applications, where a similar function must be carried out on large volumes of data,

leaving irregular calculations and complex control flow to regular microprocessors [1], [2], [4]. In this way the overhead for reprogramming the FPGA can be minimised and thus more easily amortised by the speedup due to hardware execution. This use of FPGAs as coprocessors to speed up simple, repetitive tasks is an area of active research, and explains the move by FPGA vendors to offer System on a Chip (SoC) devices comprising microprocessors embedded within reconfigurable logic arrays.

Within the more specific field of video/image processing, early work presented in [5] demonstrates the advantages of a reconfigurable architecture over traditional DSP methods, with savings in hardware, power and board space. The system consists of those tasks that are required throughout being implemented in dedicated hardware, with a single reconfigurable device performing the processing of the image data. The use of a reconfigurable device allows for the creation of a more generalised platform for video processing, with application overlays implementing different algorithms as required. Although this was a board-level solution, many of the ideas presented remain relevant, particularly the discussion of the potential pitfalls arising from dynamic reconfiguration, with similar conclusions to those drawn in [6]. Here, the pitfalls are deemed problems of consistency, with lack of hardware consistency arising from dynamic reconfiguration defined within three categories. Port Contention is a single port being driven by two drivers due to an incorrect reconfiguration, Token Loss/Duplication is the breakdown in communication between static and dynamic portions of a device, and Device State Maintenance is the name given to the faults that occur when external hardware fails to recognise the change in the reconfigured device. The authors' solution to avoiding these problems is the creation of a design approach and execution environment that guards against such faults. An overview of the requirements for such a run-time environment is also given in [7], where the comparison is made to a real-time operating system (RTOS), with hardware objects analogous to software threads. This concept shows promise for the future effective management of reconfiguring systems, but is still a research area and not a feasible proposition without more specialised design tools. It should be noted that Xilinx recommend techniques for use with their partially reconfiguring devices that go some way towards preventing the problems described here [8].

The application domain for the previous paper is automatic target recognition. This is a topic that has generated a degree of interest, and is pertinent to the application intended here. Early work in [9] implements an algorithm that compares regions of interest to target templates, and reports advantages of a single FPGA over several ASICs. This is largely due to the parallel execution of multiple templates on a single FPGA, and consequently the possibility of combining those with commonalities. This characteristic, when coupled with the ability to reconfigure the templates as required, make for a powerful system, although the authors also warn of the need to keep the reconfiguration overhead down to maximise performance. This is achieved either through maximising the use of each configuration (and thus minimising the occurrence of reconfiguration), or minimising the time the device is idle during reconfiguration. (At the time the paper was written, dynamic partial reconfiguration was not a viable proposition.) A related application is the subject of [10], this time for reconfigurable radar DSP. The system consists of a reconfigurable FIR filter made up of Multiply and Add Cells (MAC) with two banks of coefficients but only one being active at any one time. The other can be updated with new coefficients while the device remains active, and thus the switchover incurs no downtime. (The single bank alternative would leave the filter non-operational whilst the coefficients were being updated.) Valid points are raised on the design choices required to implement a distributed arithmetic structure such as a FIR filter on an FPGA; a more mathematical account of distributed arithmetic is given in [11]. Another paper that has a particularly relevant application area describes a system comprising a FPGA coupled with a DSP that can go from a power efficient monitoring mode to an image capture and tracking mode upon detection of a moving object, through reconfiguration of the hardware [12]. Although this paper discusses the partitioning aspects of a FPGA and DSP operating together, there is little discussion of the details of the reconfiguration process. In order to find such details, one needs to turn to papers concentrating on these aspects.

A review of available (mostly coarse-grained) devices that support dynamic reconfigurability and the methods of programming them is available in [13]. This covers devices from Xilinx, Lattice Semiconductors, Atmel, and Altera; although brief it

Device	Total no. of configuration bits	Approx SelectMAP download time (50MHz) ms	Approx. serial download time (50 MHz) ms	Approx. JTAG download time (33 MHz) ms
XC2VP2	1,305,440	3.3	26.1	39.6
XC2VP4	3,006,560	7.5	60.1	91.1
XC2VP7	4,485,472	11.2	89.7	135.9
XC2VP20	8,214,624	20.5	164.3	248.9
XC2VP30	11,589,984	29.0	231.8	351.2
XC2VP40	15,868,256	39.7	317.4	480.9
XC2VP50	19,021,408	47.6	380.4	576.4
XC2VP70	26,099,040	65.3	522.0	790.9
XC2VP100	34,292,832	85.7	685.9	1039.2
XC2VP125	43,602,784	109.0	872.1	1321.3

Table 1: Virtex-II Pro full bitstream lengths and programming times

offers a good overview of the options available. Altera in particular offer the Stratix device with embedded DSP cores, which directly supports algorithm implementation from Matlab through Altera's tools. Other device architectures, such as Quicksilver's ACM [14] and XPP from PACT [15], that are currently in development are specifically designed to support DSP algorithms and dynamic reconfiguration, are programmable with high-level languages such as C, and offer extremely fast reconfiguration times. Research into these architectures was deemed beyond the scope of this investigation, but it is envisaged that they will play an important role in similar applications in the future.

The following investigation of the technical aspects of FPGA configuration techniques focuses solely on the Xilinx Virtex range of devices, being the market leader with widespread availability and support for dynamic reconfiguration.

3 Configuration

3.1 Comparison of FPGA programming techniques

There are broadly three ways to program a Xilinx FPGA [16]. Average times for all three methods are given in table 1.

Serial mode

Serial mode operates by loading the configuration bitstream at one bit per cycle of the configuration clock (CCLK), and can be performed in both master and slave arrangements. Master mode implies that the FPGA generates CCLK and thus controls the configuration, usually interfacing with an external PROM. In Slave Serial mode an external device controls CCLK; this device could be a microprocessor or another FPGA where devices are arranged in a daisy-chained configuration. The main limitation is storage capacity; in Master Serial mode the device receives the bitstream from an array of PROMS, which is not feasible with limited board space. If the device is being configured in Slave Serial mode then the possibility exists of the driving device having access to external storage (on say a PCI bus), this however is slower and less efficient than the equivalent SelectMAP mode.

SelectMAP mode

SelectMAP is an 8-bit parallel bus interface to configure the FPGA, which can also be performed in both master and slave arrangements. As with serial configuration, Master SelectMAP is for use with a PROM device, where the FPGA being configured controls the configuration clock CCLK. Slave SelectMAP mode allows several devices to be configured by the driving logic from a single bus, using a chip select signal (CS.B) to select the appropriate device. It is also possible to configure parallel devices simultaneously with the same bitstream. Slave SelectMAP is the most efficient way of a FPGA being configured by a microprocessor or another FPGA, although it would require the most complicated board-level routing. Slave SelectMAP mode also supports dynamic partial reconfiguration. The maximum configuration clock speed without handshaking is 50MHz, however through monitoring of the DONE signal higher speeds are attainable. The parallel nature of the SelectMAP arrangement allows a full byte of data to be loaded every clock cycle, hence the advantage of SelectMAP mode lies in its speed, being the fastest of the three programming methods.

JTAG boundary scan mode

Since it is presumed that the Boundary Scan chain will be in place for test purposes, this method is probably the least difficult to physically implement at the board level. The Boundary Scan standard allows custom instructions to be specified by the device manufacturer in addition to the standard JTAG instructions; in this instance the capability is used to implement Configure and Verify instructions through the JTAG chain. By utilising these instructions, the FPGA can be configured by itself or as part of a multiple-device chain. The JTAG mode also allows for partial reconfiguration. The main drawback with this method is its inherent slowness, operating at a maximum speed of 33MHz and requiring several cycles through the TAP state machine for each instruction or data set entered [17]. This system is not reliant on PROMs, and potentially unlimited data storage can be utilised by the device driving the scan chain.

3.2 Configuration management - System ACE

System ACE is a pre-engineered solution for managing the configuration process and storage of configuration data for Xilinx FPGA devices. It comes in two main forms: MPM and CF.

System ACE MPM

System ACE MPM (Multi-Package Module) is described in [18]. It consists of three integrated components: the System ACE controller (itself a small FPGA), a PROM containing code to configure the controller at boot, and Flash memory to store the configuration bitstream for the target devices. This solution offers storage of up to eight designs within 64Mb of storage, and can configure four FPGAs either sequentially or in parallel. Unfortunately, since a mid-size Virtex-II uses around 16Mb for a full configuration this capacity is a limitation when multiple bitstreams are required. System ACE MPM can program devices connected to it by either Slave Serial or Slave SelectMAP modes of configuration. The advantage of System ACE MPM lies in its size (just a single module) and potential speed benefits over the

alternative System ACE solution, System ACE CF.

System ACE CF

System ACE CF is a Compact Flash solution for FPGA configuration [19]. The interface of System ACE CF allows access to any standard Compact Flash module or IBM Microdrive, and allows up to 8Gb of storage to be used to potentially store an unlimited number of designs. It is designed for systems containing multiple reconfiguring FPGAs and offers a pre-engineered solution to such systems to fully conduct all (re)configurations. Configuration of target devices is carried out via the JTAG method of programming, as shown in figure 1.

The main drawback is that the designs must be stored in groups of up to eight known as collections. Only one collection may be active at run-time and so the actual number of designs available for reconfiguration is limited to this amount. A file in the root of the filesystem (`xilinx.sys`) has a parameter used to determine the active collection, the correct design subdirectory is then selected either through dedicated pins or register bits. The `xilinx.sys` file is created during the programming of the Compact Flash card.

The MPU interface is a microprocessor interface for controlling and monitoring the System ACE CF controller. This allows direct communication with the Compact Flash module and the FPGA device configuration chain, and consequently data can be used to configure the FPGA(s) that does not exist in the Compact Flash device. Through interaction with the System ACE controller, the MPU can initiate reconfigurations of the FPGA device. All communication by the MPU is via registers within the System ACE controller. Note that the MPU could be an embedded PowerPC processor within the FPGA fabric. A possible scenario would be to have System ACE configure an FPGA via the standard JTAG route at power up, and subsequently an embedded processor interfaces with the System ACE via the MPU interface to carry out further reconfigurations.

Besides the Compact Flash and MPU interfaces, a further reconfiguration path exists via the Test JTAG interface, offering a source of data from devices further up the JTAG chain. The data can be routed directly through the System ACE controller

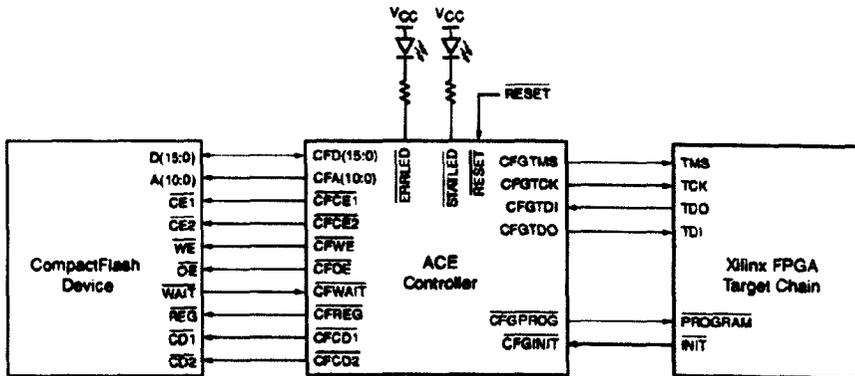


Figure 1: System ACE CF to CFGJTAG [19]

or around the boundary scan path if required.

The Compact Flash solution fits in with the three timescales of reconfigurability raised previously, having a plug-in COTS Compact Flash module as the centralised data store for the FPGA configuration bitstreams which can be changed before the deployment of the system in the field.

4 Partial reconfiguration

The Xilinx Virtex family of devices support partial dynamic reconfiguration, making it possible to reprogram sections of them whilst they remain operational. This requires partitioning the functionality of the system into discrete blocks that are either static (required during all configurations and thus resident throughout) or dynamic (configured in or out of the array as required). This partitioning is an important step in the design process, maximisation of static circuitry will minimise configuration overheads by reducing the amount of the chip that is reconfigured. This will result in improved performance and a more favourable comparison to fixed hardware. As stated by Hadley and Hutchings [20] in their analysis of static/dynamic partitioning, “*The designer resorts to dynamic circuitry only when functional commonality cannot be found between configurations*”.

The second important point to note about static circuitry on a dynamic array is the need for it to be grouped. By implementing all static circuitry in a similar area, this maximises free space for dynamic blocks. This will require manual intervention

during the design flow to ensure locations are constrained correctly.

Further complications exist in Virtex devices, due to the way in which they are programmed. The smallest unit of configuration in a Virtex device is the frame, 48 of which make up a column of Configurable Logic Blocks (CLBs) [21]. Although frames can be written to individually, a whole column must be reprogrammed in order to change the function of a CLB, which in effect makes the column the atomic unit of configuration. There are various types of column; the most common contains CLBs with IOBs at the top and bottom. Reconfiguration of the column will temporarily disable the IOBs within its boundaries.

The columnar nature of reconfiguration may become a hindrance when the FPGA is acting as a System on Chip (SoC) with several different tasks executing concurrently within the array. If tasks are positioned on the array such that they share columns, they cannot be individually reconfigured without causing disruption to the others. The most obvious method of overcoming this, by not placing multiple tasks within a single column, is inefficient of resources since a task must either use all logic within a column or else it is wasted. This problem is addressed by Carline and Coulton [22], who suggest that each task (or module) be comprised of three parts: a static section, a reconfigurable section, and a buffer. (The buffer section exists to hold input data during the time the reconfigurable section is being programmed.) By having such a predefined structure for modules, it allows the static regions of different tasks to overlap in the vertical plane and their reconfigurable regions to be combined, reducing the inefficiency imposed by the columnar nature of the device. Figure 2 shows this diagrammatically. Both Module A and Module B have separate buffer (ABr and BBr) and static (AS and BS) regions. The section Rc contains the reconfigurable logic for both A and B, and any change to the functionality of A or B involves the reprogramming of this section only.

This is a useful technique; sharing reconfigurable areas between tasks is necessary as failing to utilise a high proportion of the logic elements in a single column will not only waste resources but also ultimately limit design complexity. However, some drawbacks can be shown. Firstly, the technique depends on the modules within the device having clearly defined discrete static and dynamic functionalities. These

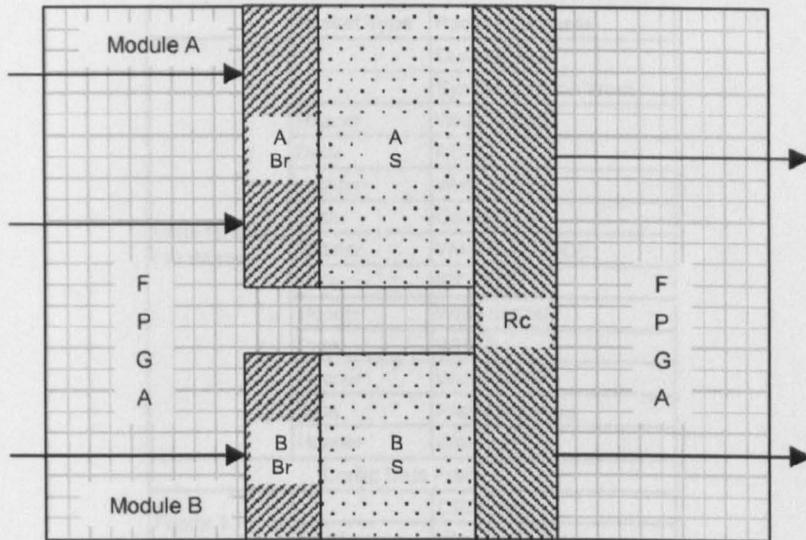


Figure 2: Column sharing modules with common reconfigurable area [22]

must be extracted from the requirements early in the design process, and the design must then be implemented accordingly. This will involve a great deal of manual intervention by the designer. Second, the technique is most suited to concurrently operating tasks that have no interdependence, i.e. operating on different sets of data. Tasks containing reconfigurable elements that operate serially on a set of data will require a more complicated design.

4.1 Partial reconfiguration sequence

The configuration sequence involves reading and writing of various options and commands, creating an overhead associated with each configuration. The bitstream sequence for executing a partial reconfiguration of a Xilinx Virtex-II is shown in table 2, with descriptions of the various commands available in [16].

This is the minimum number of operations required for reconfiguration of a section of the device. As shown, in addition to the configuration data itself there is an overhead of 20 words (640 bits) plus one frame's length of *no-op* (required to flush the internal configuration pipelines). Using this information, some approximate partial reconfiguration times can be calculated, see table 3 and figure 3. It is worth noting that these times are idealistic and do not account for the overhead involved when writing to non-sequential addresses, which becomes an issue when the unusual

	Packet Type	Packet Contents
Write Header (13 Words)		Dummy Word
		Synchronisation Word
	Header	Write to CMD
	Data	RCRC
	Header	Write to COR
	Data	Config Ops
	Header	Write to IDCODE
	Data	IDCODE
	Header	Write to CMD
	Data	WCFG
	Header	Write to FAR
	Data	Frame Address
	Header	Write to FDRI
Config Data Frames		
Footer 1 (3 Words)		CRC Value
	Header	Write to CMD
	Data	DGHIGH
1 Frame of No-Op		
Footer 2 (4 Words)	Header	Write to CRC
	Data	CRC Value
	Header	Write to CMD
	Data	DESYNCH

Table 2: Partial reconfiguration sequence

	No of Frames	Frame Size	Number of Configuration Bits			33MHz JTAG Download Time (s)			50MHz SelectMAP Download Time (s)		
			10%	50%	100%	10%	50%	100%	10%	50%	100%
XC2VP2	884	1472	131648	652736	1303360	0.0040	0.0198	0.0395	0.0003	0.0016	0.0033
XC2VP30	1756	6982	1160832	5795008	11582784	0.0352	0.1756	0.3510	0.0029	0.0145	0.0290
XC2VP125	3936	11072	4363008	21801408	43591104	0.1322	0.6606	1.3209	0.0109	0.0545	0.1090

Table 3: Programming times of Virtex-II Pro for varying configuration loads

addressing scheme of Virtex devices is considered. The start address of logic on a Virtex is in the centre of the device, addresses then alternate between the left and right sides, causing one side to have even addresses and the other odd. This creates an overhead in writes to the device as a single logic block on the FPGA may contain several columns, each of which must be individually addressed (an example of how the Virtex devices were not designed specifically for partial reconfiguration).

4.2 Modular design flow for partial reconfiguration

Dynamic partial reconfiguration is the process of reprogramming sections of the FPGA during execution, while other fixed sections are unaffected and remain opera-

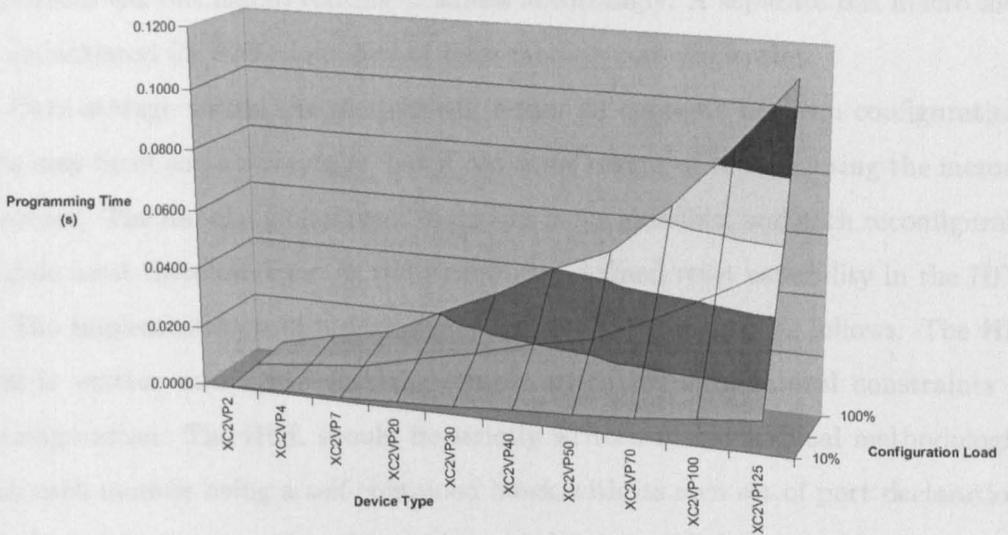


Figure 3: SelectMAP programming times

tional. For a Virtex device this can be carried out in two ways: module based partial reconfiguration and small bit manipulation [8].

The small bit manipulation partial reconfiguration process involves making small changes to the configuration of a device, usually carried out by hand using a tool such as FPGA Editor. The reconfiguration bit-stream is simply the difference between the two configurations, and since this is typically small the reconfiguration process completes very quickly. This method requires hand crafting of the mapping and placement of the design and so is deemed too specialised for the intended application.

Module based partial reconfiguration is the reprogramming of sections of the device, and is based around Xilinx's "Modular Design" flow [23]. These sections (known as modules) are of regular size and shape; their width always corresponds to a column edge, and they occupy the full height of the device. Any elements (such as Block RAM, IOBs and routing) contained within the boundaries of a module also need to be considered as hardware for reconfiguration. The routing acting as communication paths between a reconfigurable module and those around it must remain fixed. In order to facilitate communication between reconfigurable blocks, a technique known as a Bus Macro is used; this is a fixed routing bridge across the module boundaries. All modules making use of the bus macro must instantiate and

implement the bus macro routing channels accordingly. A separate bus macro must be instantiated for every four bits of inter-module communication.

Data storage within the module will retain its contents between configurations. This may be of some advantage, but if not some means of re-initialising the memory is needed. The use of a global reset to do this is not possible, and each reconfigurable module must therefore have an individual user-defined reset capability in the HDL.

The implementation of a partially reconfigurable design is as follows: The HDL code is written and synthesised to comply with the architectural constraints for reconfiguration. The HDL should be strictly written to hierarchical methodologies, with each module being a self-contained block with its own set of port declarations. All clock structures need to be implemented using global resources in order that they remain functional between configurations. Each module is then synthesised individually beginning with the top-level. Floorplanning is carried out to ensure that the modules fit into the boundaries defined earlier. At this point the design consists of an area-based floorplan, location constraints for bus macros and top-level logic, and global timing constraints. Following definition of any module-level constraints, a bitstream can be generated for each module. When these bitstreams are then combined into a complete FPGA design each module will retain its placement and routing structure. At least one assembly must be carried out with a configuration present for each module of the FPGA; this will then become the initial bitstream loaded into the device at boot up as a complete design. Designs after reconfiguration need not be assembled, however Xilinx recommend that every possible combination of static and reconfigurable blocks is assembled for visual inspection in order to detect any breach of design rules, and also for post-PAR simulation if required.

4.3 Design considerations

Protection against misconfiguration

Damage can occur to an FPGA device if it is configured incorrectly through creation of excessively high currents. This can happen through configuration of inputs as outputs, possibly causing a conflict in logic levels between the device and whatever

it is connected to, or alternatively contentions can be created within the device by causing multiple logic blocks to drive the same routing resource simultaneously [24]. In either case, a path for high currents is created and physical damage to the device can ensue. Normal configuration procedures for Xilinx devices prevent misconfiguration, firstly by checks performed by the programming software, and secondly through the use of the internal FPGA signal GHIGH. GHIGH holds the configuration logic in a shutdown state during programming, and does not release it until the configuration bitstream has been validated using a CRC at the end of transmission. However, if the device remains active throughout reconfiguration (active partial dynamic reconfiguration), an end of transmission CRC will not suffice and intermediary checks will be required, potentially after transmission of each frame (the smallest unit of reconfiguration) [25]. In [25], the solution to this problem is the creation of a block within the FPGA that performs the validation of the bitstream as it is presented to the configuration interface; if a CRC fails, the reconfiguration is aborted and the FPGA rebooted.

Dual-function pins

The three methods of configuring a FPGA each use varying numbers of pins, some dedicated to the configuration process whilst others double as I/O during the normal operation of the device. If these dual-function pins are required for I/O purposes, procedures must be in place to ensure that they are set to the required standard after configuration has taken place. Furthermore, in order to implement partial reconfiguration (using slave SelectMAP) the pins must be used as the configuration interface again, and hence two banks of I/O pins are unavailable during this time. This will have to be considered at design time. If the two banks are not required for I/O purposes during the normal execution of the system, then they can be set to retain their configuration function throughout. During configuration, all I/O banks are set for the LVTTTL standard. JTAG inputs use dedicated pins and so the interface is always present, however TDO is sourced from V_{cc0} , which should be set according to the TDI of the subsequent device in the chain.

4.4 Self-reconfiguration

A Virtex-II device may use an embedded PowerPC core (alternatively a “soft” Microblaze Processor) for the configuration controller so the device is actively reprogramming parts of itself, eliminating the need for external hardware. A particularly good overview of self-controlling dynamic configuration can be found in [25]; similar systems can also be found in more recent articles where it is termed self-reconfiguration [26], [27].

If the controller has access to a PCI bus through a core implemented on the FPGA fabric itself, a potentially unlimited amount of storage may be utilised for configuration data. A typical configuration sequence may involve the device being “booted” to initialise the PowerPC processor and PCI core using a full configuration stored on a PROM. Subsequent partial reconfigurations of the remaining FPGA fabric can then be performed as necessary with no external programming hardware. The method of loading the bootstrap code depends on how subsequent reconfigurations should be carried out. Two methodologies can be identified, configuration via JTAG or via the Virtex Internal Configuration Access Port (ICAP).

Configuration via JTAG

Figure 4 shows a method of using System ACE CF to boot the FPGA via the JTAG interface. A configuration controller, (here a PowerPC core within the FPGA) then controls the subsequent reconfigurations by drawing data from the PCI bus and writing it to the MPU interface of System ACE. System ACE will in turn process the data into the correct format for the programming logic of the FPGA and place it onto the JTAG chain for partial reconfiguration of the device. The FPGA is initiating the reconfigurations but requires external hardware, and is thus a self-controlling reconfigurable system.

Configuration via SelectMAP/ICAP

The second method requires System ACE MPM to boot the FPGA via the SelectMAP parallel interface, shown in figure 5. Subsequent partial reconfigurations are carried out completely within the FPGA through the ICAP interface, with con-

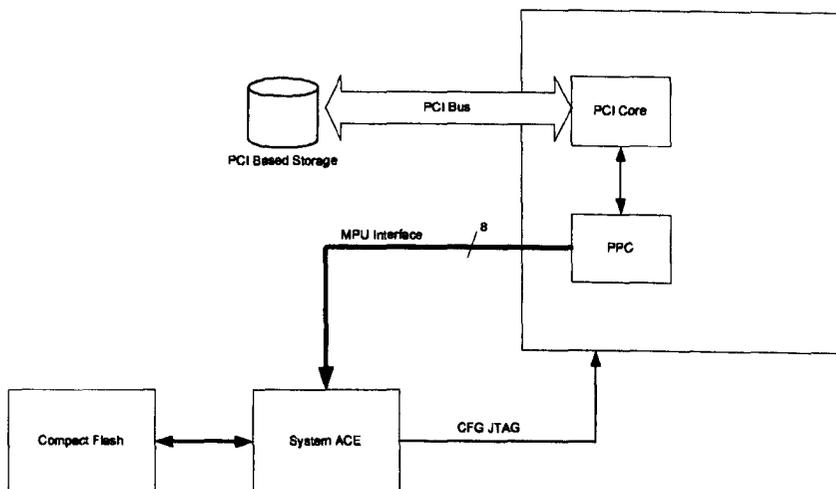


Figure 4: Partial reconfiguration via MPU interface to System ACE CF

figuration data being retrieved from a storage device on the PCI bus. The ICAP is essentially a multiplexed connection to the SelectMAP interface, allowing an embedded PowerPC core to directly drive the configuration registers with no external hardware required, in essence self-reconfiguration as proposed in [25], [26], [27]. The ICAP interface consists of the standard SelectMAP signals of separate 8-bit data paths for reads and writes, write and chip enables, a busy signal, and clock input [25]. Because both the initial (full) configuration and further partial reconfigurations are carried out using SelectMAP parallel buses this method has significant performance gains over JTAG, but is also the most sophisticated to implement.

Extensibility to dual FPGA system

The benefit of both of these methods of self-reconfiguration is their extensibility to a dual FPGA system. Using the JTAG method, the extra FPGA simply requires access to the JTAG chain, all configurations and reconfigurations are then handled by the FPGA with access to the MPU interface. Using the ICAP interface the connections from System ACE are used to boot both FPGAs in parallel, they then both have embedded PowerPC and PCI cores with which they control their own reconfigurations independently.

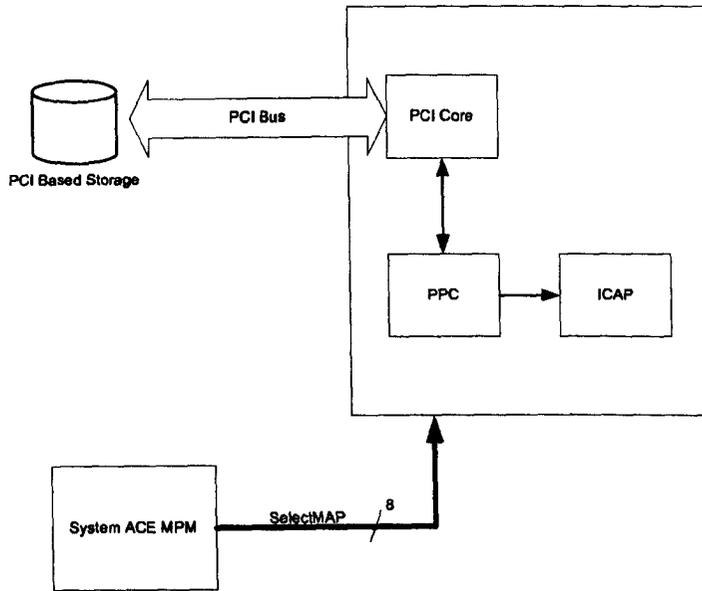


Figure 5: Partial reconfiguration via ICAP interface

5 CAD tools for dynamically reconfigurable logic

The following section describes two alternatives to the Xilinx Modular Design flow for partial reconfiguration, Dynamic Circuit Switching and JBits. Each method has its own advantages and disadvantages, and should be selected according to its merits for a particular application. A third option, known as PARBIT, is not described here as it is deemed too specialised; further information can be found in [28].

5.1 Dynamic Circuit Switching

Work carried out at the University of Strathclyde, Glasgow, on the subject of reconfigurable logic has largely concentrated on defining a CAD framework for dynamically reconfigurable systems. Lysaght & Stockwood [29] first proposed Dynamic Circuit Switching (DCS) in 1996 as a method of overcoming the additional complexity dynamic reconfigurability introduces into the design cycle. Subsequent publications report extensions to handle new device architectures [30], [31]; the addition of new modules [32], [33]; and improved functionality [34], [35].

To model the reconfiguration process, four distinct steps are identified in order to prevent erroneous interactions between a reconfiguring block and those around it.

These are:

1. Isolation of the array to be configured
2. Loading of a default (safe) configuration
3. Loading of the new configuration
4. Reactivation of the block and the connections to its neighbours

DCS works by allowing for identification of the dynamically reconfigurable components within a design and provides a means of specifying their behaviour that was previously unavailable. It does so in a way that allows integration with existing CAD systems without being dependent on proprietary features.

In early versions of DCS each dynamic task is allocated a reconfiguration schedule consisting of five attributes that specify the task's dynamic behaviour, including the conditions that cause a reconfiguration to take place and the times taken for loading and removing the task. These attributes are either textual (as in VHDL attributes) or represented schematically. A netlist post-processor is then used to insert virtual components, known as isolation switches, at the inputs and outputs of dynamic tasks. These are controlled by Schedule Control Modules (SCMs), which detect the stimuli that trigger a reconfiguration and switch the affected block in/out of the system as appropriate. The SCMs also provide status signals to allow monitoring of the condition of a reconfigurable block; i.e. whether it is active, inactive, or currently undergoing reconfiguration. The instability of a block undergoing reconfiguration is represented for simulation purposes by the isolation switches propagating the unknown signal 'X' from the block's outputs. (The model of the isolation switch was later redefined by Robinson & Lysaght [36]. When the task to which it relates is active, the output of the switch is equal to its input. When the task is inactive, the output is 'Z', the weakest signal in VHDL `std_logic` and thus overwritten by any other value. In this way the inactive task is 'invisible' to those it drives. As before, the task drives 'X' whilst in transition between the active and inactive states.)

Changes made by Robinson et al. in 1998 [32] include using a single file for the reconfiguration attributes, centralising the location of this information. This enables

HDL design to be used as the primary means of design entry, which encourages a hierarchical design approach to be taken. Other changes made are the introduction of a technique known as task grouping, and improved modelling of the reconfiguration port. Task grouping is applied where tasks are identified that share common logic but exist at different times due to reconfiguration. These groups, known as *mutex* (mutually exclusive) groups, can be checked as possible causes of contention for resources. Improvements to the modelling of the reconfiguration port, so that only one task can access the port at any one time, were added for the case where several reconfigurations are taking place concurrently, thus requiring a form of arbitration mechanism. The system is scalable to simulate multiple reconfiguration ports if required.

Also at this time two new modules were added: DCSEst and DCSTech, and the simulation aspect renamed to DCSim; the term DCS now covering the complete set of CAD tools for dynamically reconfigurable logic. DCSEst is a tool used to provide timing estimates that are back annotated into the original design; it can operate with design files at various stages of the design flow. DCSTech uses the information in the reconfiguration file to split the dynamically reconfigurable design into several static designs, making processing by technology dependent mapping tools possible. The output of DCSTech is a design file for each dynamically reconfigurable task within the system, plus an extra design file for all the static tasks. DCSTech handles dynamic tasks through the reservation of space for each mutex group, sufficient to cover all tasks within the group.

In order to cope with different forms of configuration controller, another virtual component was added in the same year [34]. Known as Task Status Registers (TSR), these are designed to give a generic interface to a controller regardless of its implementation, and allow for standardisation of the connection to the isolation switches and the reconfiguration status of the task to be presented in a predefined format. DCSim can take advantage of this to perform analysis and aid in the verification of the dynamically reconfigurable aspects of the design.

The DCS framework, in particular DCSTech, was extended in 2002 to cover the Xilinx Virtex devices. The complete design flow, from VHDL source to verification

and bitstream generation, can be achieved for a dynamically reconfigurable design, see figure 6. DCS_{Tech} takes as input a structural design, with dynamic tasks represented as components, and a reconfiguration file describing the dynamic aspects. The resulting bitstreams represent each dynamic task plus the static tasks in the design. These must then be manually altered into valid partial bitstreams using a tool such as JBits (see below).

The most recent developments to DCS came as a response to an alternative method of simulating dynamically reconfigurable logic published by Vasilko & Cabanis [37]. Known as Clock Morphing, the technique has the ability to represent the internal state of synchronous elements during reconfiguration, although with deficiencies in other areas most notably its usage being restricted exclusively to VHDL. The solution proposed by Robertson et al. [35] is the usage of DCS_{Sim} in conjunction with Clock Morphing to provide exhaustive simulation when it is required at the expense of portability. Improvements to DCS_{Sim} were also made at this time in remodelling the isolation switches as Dynamic Task Modellers (DTMs), and Dynamic Task Selectors (DTSs), effectively splitting the combined functions of the original switches. This change was made to ensure correct results in the special case of a static task being driven only by a dynamic task, thus having an invalid input of 'Z' when that task was not present on the array. The DTS overcomes this by driving 'X' when it receives only 'Z' on its inputs.

5.2 JBits

The lack of high-level design tools for partially reconfiguring systems has prevented widespread use in commercial applications. The JBits API (Application Programming Interface) is an example of a lower-level tool that attempts to address this lack of appropriate techniques. JBits is a set of Java classes that can be used to create and manipulate the bitstream of a reconfigurable Xilinx FPGA [38]. Because of its nature as a library of precompiled classes it can be used within a user-defined environment, or integrated into a custom design tool, to perform operations at the bitstream level. The use of JBits is through function calls which operate on the bitstream through an interface as part of the API.

make the modifications, but on a bitstream that has been preloaded into the JRTR cache.

Of the many publications reporting usage of JBits one of note, published by Dyer et al. (2002) [40], suggests a design flow that allows the designer to work at a higher level of abstraction. It does so by utilising standalone cores created using traditional industry-standard CAD tools; these cores comply with architectural constraints that allow them to be connected together with predefined connection structures. (This is similar in principle to the interconnection of IP blocks using ‘collars’ in ASIC SoC design). The application consists of a CPU core and dynamically reconfigurable coprocessor operating within the FPGA. These implement an audio decoder: the CPU unpacks data arriving over an ethernet connection, then passes it to the coprocessor which decodes the audio stream and sends to a D/A converter. Dynamic reconfiguration is used to allow different formats of audio encoding to be sent to the device, with the coprocessor reconfiguring to implement the various decoders as necessary.

The CPU and coprocessor blocks are designed and synthesised using standard high-level CAD tools, then manually placed on to the FPGA and connected together through a specially designed interface called a Virtual Socket, which acts as the boundary between static and dynamic portions of a design. A complete bitstream is produced using the normal Xilinx tools, which will implement the full configuration of the device. A secondary design can then be created for the coprocessor; its placement is constrained to connect to the Virtual Socket defined earlier. This is also implemented with the Xilinx tools into a full bitstream, but this is then operated on using a custom tool based on JBits named JBitsCopy, which can extract portions of a design from a configuration bitstream: here it is used to extract the bit sequence detailing the coprocessor. A standard JBits function is then used to merge this coprocessor with the original full bitstream, creating a full bitstream with the second coprocessor and all the other components. Now two full bitstreams exist, having different coprocessors but identical in all other respects. These can be compared and the difference extracted to create a partial configuration that will implement the change in coprocessor while the remainder of the design remains active.

Several difficulties exist that have prevented the widespread use of JBits. It

is still a tool in early stages of development, and an area of active research both in industry and academia. Because of this it is not supported by any high-level CAD tools, meaning that designs must be created through explicit statement of all details. Furthermore, a precise knowledge of the device architecture is needed in order to make manual changes. JBits does not support explicit definition of routing structures, so techniques must be adopted to constrain routing to certain areas (the above application uses feed-through components, simple elements that lock routing to a given point on the array). However JBits does show promise in the ability to automate certain aspects of reconfigurable design that have traditionally been done by hand, and offers the ability to develop architectures from within a software environment, which reduces the reliance on large CAD suites when making small changes to a design.

6 Conclusions and further work

The related applications that have been reported show that signal and image processing with dynamically reconfigurable FPGAs is an area of active research both in the industrial and academic worlds. However, the constraints imposed by the nature of the embedded systems Thales plans to implement, such as low size and low power for portability, alongside high performance, mean that no direct precedents exist.

FPGA implementations of signal processing algorithms and dynamic reconfiguration are both fully developed fields in their own right. Add to this the low size constraint which makes self-controlling partial reconfiguration look necessary, and the need for design methodologies to be used which are at present underdeveloped, and it becomes clear that the proposed system will require a convergence of several research areas.

Further research can be classified into three areas:

Algorithm implementation

How is an algorithm implemented in a predefined FPGA system with the minimum of manual intervention and hand crafting? Partial reconfiguration works best when

static functionality can be found between configurations, for which it will be necessary to find commonalities between algorithms. However, it should be possible to implement algorithms without the requirement for them to conform to a possibly restrictive preordained static framework. Therefore there is a trade-off between ease of algorithm implementation and system performance. This is further complicated by the introduction of “mostly static” blocks, a term introduced by Hadley and Hutchings [20], to describe functional blocks that share functionality but vary in some small way such as precision, or are architecturally equivalent but functionally different, for example a bit-serial adder and bit-serial subtractor.

Qualitative assessments need to be made as to how this trade-off should be balanced. In the extreme case, perhaps all the functionality of an algorithm will be reconfigurable. If the reconfiguration mechanism provides the necessary performance to keep reconfiguration times reasonable, this will place fewer demands on the algorithm design.

Self-controlling dynamic reconfiguration

Two main problems exist that prevent the use of a more general system architecture. Firstly, the hardware and space restraints imposed by the nature of the application mean that having a dedicated configuration controller is unfeasible. Second, current methods of managing configurations rely on banks of PROMS, which is not feasible when a large amount of configuration data is required. Xilinx’s System ACE CF is promising but has shortcomings both in the number of accessible designs and the fact that it uses the slow JTAG method of programming the FPGA. Until an effective configuration management solution exists that can accommodate more than eight designs over the SelectMAP interface, then storage of configuration data is also a system element that will require an innovative solution.

A means of overcoming these problems through development of a self-controlling partially reconfiguring system has been suggested, implementing a PowerPC processor with PCI functionality and utilising the Virtex ICAP interface for efficient partial reconfiguration. Similar proposals made elsewhere [25], [26], [27], [40], do not incorporate the PCI interface, but theoretically it is possible. If this system was

developed, it would offer a solution that is extensible to multiple FPGA systems, has potentially unlimited storage capacity, and has high performance reconfigurations. Furthermore, all dynamic reconfiguration functionality would be software based, i.e. the program running on the PowerPC, and easily upgradeable. In this case, the boot code contains the system core functionality, and all image processing algorithms reside on the PCI device.

There are potential drawbacks however. A self-controlling design will by definition contain a mixture of static and dynamic tasks, requiring constrained placement within the design. Also, problems can arise due to the fact that having separate controllers for initial programming and subsequent reprogramming mean that two controllers exist in the system. One example is the possibility of the initial programming device prematurely losing control of the configuration process due to contention [27].

These problems may be solved with careful design; however the deciding factor for self-reconfiguring designs may be the device utilisation after the processor, ICAP, and PCI core and any other reconfiguration control modules have been instantiated. If a large portion of the device is taken up by the reconfiguration mechanism alone, and this is required for each FPGA in the system, it would make more sense to centralise it to a dedicated device (board space permitting), and have all configurations performed over the more traditional SelectMAP method.

Design flow

Well established design tools and methodologies for specification and verification of dynamically reconfigurable systems do not exist at present. It is envisaged that thorough and comprehensive testing and verification will be required of a commercial system. However the tools available all have associated advantages and disadvantages, and further research and experimentation may be required before a design methodology can be chosen. The modular design flow, advocated by Xilinx, allows for simulation of individual modules within the larger design, but cannot model the reconfiguration process itself. DCS allows for such simulation within a comprehensive design flow, with support for other high-level analysis tools, but does not as yet support the latest devices. JBits bypasses the need for high-level tools by operating

directly on the bitstream, but requires high skill levels and custom software to be utilised.

Hardware/Software partitioning

Hardware/Software co-design is envisaged to be a key problem in the design of the system, particularly one that controls its own reconfigurations. For instance, a self-reconfiguring device will require decisions to be made as to how much of the configuration controller will be implemented in the hardware of the FPGA fabric, or as software executing on the embedded processor. Reconfiguration performance may dictate a hardware approach, but high device utilisation levels could mean a software approach is necessary to ensure sufficient free resources for data processing requirements. Many similar situations may become evident during development of the system.

Future work

The research carried out so far has highlighted the number of considerations involved in the development of the intended system. The results of this work show that a self-reconfiguring system looks to be the most likely direction for success, due to it implementing the required functionality with minimum external hardware. Future research intends to develop a prototype of a self-reconfiguring device, for which possible solutions to the issues described above can be assessed.

An FPGA incorporating PowerPC, PCI, and ICAP should be developed, initially using the Xilinx Modular Design technique (JBits and DCS can be experimented with later). Configuration control can initially be implemented in software, with subsequent mapping to hardware of critical functions for improved performance. Decisions can then be made regarding the viability of such an approach in the context of a commercial system. A secondary avenue of work should focus on FPGA implementations of DSP algorithms, aiming at reducing the amount of manual intervention and creating a design technique that considers dynamic reconfiguration as a property of the algorithm. The convergence of these two research areas will be the final stage in the development of the intended system.

References

- [1] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.
- [2] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, Apr. 1998.
- [3] S. Brown and J. Rose, "FPGA and CPLD architectures: a tutorial," *IEEE Design & Test of Computers*, vol. 13, no. 2, pp. 42–57, 1996.
- [4] J. R. Hauser, "Augmenting a microprocessor with reconfigurable hardware," Ph.D. dissertation, University of California, Berkeley, 2000.
- [5] R. Andraka, "Dynamic hardware video processing platform," in *Proc. SPIE Vol. 2914, p. 90-99, High-Speed Computing, Digital Signal Processing, and Filtering Using Reconfigurable Logic*, J. Schewel, P. M. Athanas, V. M. Bove, and J. Watson, Eds., Oct. 1996, pp. 90–99.
- [6] T. Bapty, J. Scott, S. Neema, and J. Sztipanovits, "Uniform execution environment for dynamic reconfiguration," in *Engineering of Computer-Based Systems, 1999. Proceedings. ECBS '99. IEEE Conference and Workshop on*, Nashville, TN, Mar. 7–12, 1999, pp. 181–187.
- [7] M. Barr, "A reconfigurable computing primer," *Multimedia Systems Design*, pp. 44–47, Sept. 1998.
- [8] *Two flows for partial reconfiguration: module based or small bit manipulations*, v1.0 ed., Xilinx Inc., 2002, XAPP290.

- [9] J. Villasenor, B. Schoner, K.-N. Chia, C. Zapata, H. J. Kim, C. Jones, S. Lansing, and B. Mangione-Smith, "Configurable computing solutions for automatic target recognition," in *FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on*, Napa Valley, CA, Apr. 17–19, 1996, pp. 70–79.
- [10] T. Moeller and D. Martinez, "Field programmable gate array based radar front-end digital signal processing," in *Field-Programmable Custom Computing Machines, 1999. FCCM '99. Proceedings. Seventh Annual IEEE Symposium on*, Napa Valley, CA, Apr. 21–23, 1999, pp. 178–187.
- [11] Xilinx Inc., "The role of distributed arithmetic in FPGA based signal processing," 1996.
- [12] J. Scalera, C. I. Jones, M. Soni, M. Bucciero, P. Athanas, A. Abbott, and A. Mishra, "Reconfigurable object detection in FLIR image sequences," in *FPGAs for Custom Computing Machines, 2002. Proceedings. IEEE Symposium on*, Apr. 22–24, 2002, pp. 284–285.
- [13] S. Donthi and R. Haggard, "A survey of dynamically reconfigurable FPGA devices," in *2003. Proceedings of the 35th Southeastern Symposium on System Theory*, Mar. 16–18, 2003, pp. 422–426.
- [14] C. Maxfield. (2002, Oct.) Silver bullet for ACMs. Accessed: 20/11/2006. [Online]. Available: http://www.eetimes.com/news/design/columns/max_bytes/showArticle.jhtml?articleID=17408002
- [15] PACT GmbH, "The XPP white paper," 2002, release 2.1.
- [16] *Virtex-II Pro platform FPGA handbook*, v2.4 ed., Xilinx Inc., 2003.
- [17] *Configuration and readback of Virtex FPGAs using (JTAG) boundary scan*, v1.4 ed., XAPP139, Xilinx Inc., 2002, xAPP139.
- [18] *Xilinx System ACE MPM solution*, v2.2 ed., Xilinx Inc., 2003, DS087.
- [19] *Xilinx System ACE solution data sheet*, v1.5 ed., Xilinx Inc., 2002, DS080.

- [20] J. D. Hadley and B. L. Hutchings, "Designing a partially reconfigured system," in *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, Proc. SPIE 2607*, J. Schewel, Ed. Bellingham, WA: SPIE – The International Society for Optical Engineering, 1995, pp. 210–220.
- [21] *Virtex series configuration architecture user guide*, v1.6 ed., Xilinx Inc., 2003, xAPP151.
- [22] D. Carline and P. Coulton, "Partial reconfiguration in Xilinx Virtex FPGAs: pitfalls and solutions for SoC implementations," in *IEE DSP enabled radio colloquium*, Livingston, Scotland, UK, Sept. 2003.
- [23] *Development system reference guide, Xilinx ISE 5*, Xilinx Inc., 2002.
- [24] I. Hadzic, S. Udani, and J. M. Smith, "FPGA viruses," in *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 1999, pp. 291–300.
- [25] R. Fong, S. Harper, and P. Athanas, "A versatile framework for FPGA field updates: an application of partial self-reconfiguration," in *2003. Proceedings. 14th IEEE International Workshop on Rapid Systems Prototyping*, June 9–11, 2003, pp. 117–123.
- [26] B. Blodget, S. McMillan, and P. Lysaght, "A lightweight approach for embedded reconfiguration of FPGAs," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 2003, pp. 399–400.
- [27] G. McGregor and P. Lysaght, "Self controlling dynamic reconfiguration: A case study," in *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 1999, pp. 144–154.
- [28] E. Horta, J. Lockwood, D. Taylor, and D. Parlour, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," in *Design Automation Conference, 2002. Proceedings. 39th*, June 10–14, 2002, pp. 343–348.

- [29] P. Lysaght and J. Stockwood, "A simulation tool for dynamically reconfigurable field programmable gate arrays," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 3, pp. 381–390, Sept. 1996.
- [30] G. McGregor and P. Lysaght, "Extending dynamic circuit switching to meet the challenges of new fpga architectures," in *FPL '97: Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 1997, pp. 31–40.
- [31] I. Robertson, J. Irvine, P. Lysaght, and D. Robinson, "Timing verification of dynamically reconfigurable logic for the Xilinx Virtex fpga series," in *FPGA '02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. New York, NY, USA: ACM Press, 2002, pp. 127–135.
- [32] D. Robinson, G. McGregor, and P. Lysaght, "New CAD framework extends simulation of dynamically reconfigurable logic," in *FPL '98: Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm*. London, UK: Springer-Verlag, 1998, pp. 1–8.
- [33] D. Robinson and P. Lysaght, "Modelling and synthesis of configuration controllers for dynamically reconfigurable logic systems using the dcs cad framework," in *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 1999, pp. 41–50.
- [34] —, "Verification of dynamically reconfigurable logic," in *FPL '00: Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 2000, pp. 141–150.
- [35] I. Robertson, J. Irvine, P. Lysaght, and D. Robinson, "Improved functional simulation of dynamically reconfigurable logic," in *FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference*

- on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 2002, pp. 152–161.
- [36] D. Robinson and P. Lysaght, “Methods of exploiting simulation technology for simulating the timing of dynamically reconfigurable logic,” *IEE Proceedings-Computers and Digital Techniques*, vol. 147, no. 3, pp. 175–180, May 2000.
- [37] M. Vasilko and D. Cabanis, “A technique for modelling dynamic reconfiguration with improved simulation accuracy,” *IEICE Transactions on fundamentals of electronics, communications, and computer sciences*, vol. E82-A, no. 11, pp. 2465–2474, Nov. 1999.
- [38] S. Guccione, D. Levi, and P. Sundararajan, “JBits: A Java-based interface for reconfigurable computing,” in *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, 1999.
- [39] S. McMillan and S. Guccione, “Partial run-time reconfiguration using jrtr,” in *FPL '00: Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 2000, pp. 352–360.
- [40] M. Dyer, C. Plessl, and M. Platzner, “Partially reconfigurable cores for Xilinx Virtex,” in *FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 2002, pp. 292–301.

Appendix B: Implementation of LRM Algorithm using Scheduling & Allocation

Implementation of LRM Algorithm using Scheduling & Allocation

April 2004

Summary: The LRM method of contrast enhancement was the first algorithm looked at during the research period. The method of implementing it in hardware is based on manually performing behavioural synthesis tasks, using ANSI C as an intermediate language between Matlab and VHDL.

Oliver Sims
EngD 2nd Year
Industrial Sponsor: Thales Optronics

1 Introduction

The benefits of executing algorithms on FPGAs rather than general-purpose processors are well known. However the implementation process requires specialist experience and skills, and is largely a manual procedure despite the prevalence of highly sophisticated CAD tools. In many situations bespoke solutions must be created for each algorithm, and the number of implementation decisions involved means that a large design space must be explored. Key decisions must be taken involving fixed-point representations, area/speed trade-offs, memory requirements etc., and it is not always clear when the optimal solution has been found. Because a sub-optimal design may negate the performance benefits resulting from the use of FPGA technology the realisation process may take in the order of months to complete.

In general terms the process of transforming a behavioural view of a system into a structural one is known as behavioural synthesis. Although a language such as VHDL contains the necessary constructs to describe both behaviour and structure, the limited subset of VHDL that is synthesisable makes it unwieldy for algorithm design. Algorithms are usually developed in a high-level language such as C or Matlab. Matlab is the language of choice for algorithm designers as it offers sophisticated analysis techniques and a wide range of powerful operators that can handle a variety of data formats. Some instructions are at a very high level, for instance '*fft*', which cause several problems when designing hardware. Often these relate to the use of matrices, which are ideal for image operations and allow powerful techniques to be utilised in just a few commands, but in hardware terms these may require several levels of branching and looping constructs. The advantages of designing algorithms at a raised level of abstraction are thus negated by the difficulties involved in realising them in hardware.

The following work demonstrates a methodology for transforming a typical image-processing algorithm from a high-level language into a synthesisable register transfer-level (RTL) model in a hardware design language (HDL) such as VHDL or Verilog, which may then be used directly in the production of hardware. The methodology is repeatable, and produces a single purpose custom processor that conforms to a well-known paradigm. The method uses C as an intermediate language in which

the transformation from behavioural-level Matlab code to structural-level HDL can take place in clearly defined stages. It is envisaged that by utilising a repeatable methodology the design process will become more tractable, and as the difficulties faced during the process become common to many algorithms the time to complete a design will shorten. There are also some issues that were encountered, such as the best methods of performing complex calculations and error modelling, that may be common to the implementation process of many algorithms. Some example solutions to problems of this nature are also demonstrated here.

2 Target algorithm

The algorithm being implemented is the Local Range Modification method of contrast enhancement, developed by Fahnestock & Schowengerdt [1]. This algorithm implements a standard linear stretch of the contrast at each pixel, however it does so using parameters that are derived from within the pixel's locality and not over the image as a whole. In this way the algorithm is adaptive to regional variations in contrast levels.

The algorithm operates by first subdividing the image into adjoining blocks. The performance of the algorithm is highly dependent on the size of block used: too small a block means that insignificant detail may be highlighted, whereas too large a block reduces the overall level of enhancement. The calculation required for each pixel varies with its location within a block, and the resulting value is dependent on not only the contrast range of the containing block but also of those blocks that surround it. The initial phase of the algorithm defines the block boundaries, and finds local minimum and maximum pixel values within each block. At the corner of each block exists a node, and a node may have associated with it one, two or four neighbouring blocks (figure 1). An image with $M \times N$ blocks will therefore have $(M + 1) \times (N + 1)$ nodes. The second phase of the algorithm uses the block maxima and minima to find node maxima and minima. The minimum and maximum at each node is found from the neighbouring blocks, this equates to using overlapping blocks of twice the width and height of the original partitions.

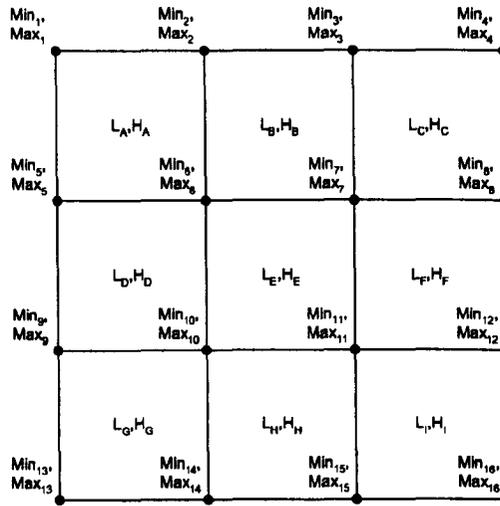


Figure 1: Partitioning of input image into blocks and nodes for LRM contrast enhancement.

Once this data is obtained, a maximum and minimum value for each pixel can be found by a bilinear interpolation of the node values. For example, the following two calculations give the range of output for a pixel in the upper left block of figure 1.

$$\begin{aligned} out_{\min} &= \left[\frac{x}{X} Min_5 + \left(\frac{X-x}{X} \right) Min_1 \right] \left(\frac{Y-y}{Y} \right) \\ &+ \left[\frac{x}{X} Min_6 + \left(\frac{X-x}{X} \right) Min_2 \right] \frac{y}{Y} \end{aligned} \quad (1a)$$

$$\begin{aligned} out_{\max} &= \left[\frac{x}{X} Max_5 + \left(\frac{X-x}{X} \right) Max_1 \right] \left(\frac{Y-y}{Y} \right) \\ &+ \left[\frac{x}{X} Max_6 + \left(\frac{X-x}{X} \right) Max_2 \right] \frac{y}{Y} \end{aligned} \quad (1b)$$

Where x and y give the pixel locations within each block, and X and Y give the block size in each direction. The desired pixel value is then found by a linear stretch:

$$out = \frac{in - out_{\min}}{out_{\max} - out_{\min}} \times 255 \quad (2)$$

The algorithm is ideally suited to a hardware rather than software implementation, as although mathematically fairly simplistic (the most computationally difficult stage being the several divide operations of the bilinear interpolation) it has a significant

throughput requirement. Two passes through the image data are required, once to find the minimum and maximum values for each block and then a second time to perform the stretch. Hence the algorithm must operate at a high speed to achieve satisfactory frame rates, and increasing its efficiency was a key consideration during implementation. The algorithm also requires substantial amounts of intermediate storage for block and node data, and optimisations were sought here also.

In an attempt to decrease the potential latency of the algorithm, efforts were made to reduce the complexity of the calculations involved. It can be shown that by placing certain constraints on the algorithm's operation it may be made more suitable for hardware implementation. Firstly, by forcing the blocks to be square, the number of divide operations may be reduced. In the original algorithm description both the horizontal and vertical block dimension are parameterisable, however non-square blocks would conceivably only be useful in very limited circumstances, and removing them greatly simplifies the calculations.

The effects of this constraint may be shown mathematically. A sample of the bilinear interpolation calculation is:

$$out_{\min} = \frac{[xMin_5 + (X - x) Min_1] (X - y) + [xMin_6 + (X - x) Min_2] y}{X^2} \quad (3)$$

This has the effect of reducing the number of divides from six to one. Since a divide operation in hardware is expensive in terms of gates and delay, this results in substantial savings in hardware terms. If the block size is then further constrained to be a power of two the single division may now be accomplished by a right binary shift of $2 \times \log_2$ (block size) places.

Another feature of the original algorithm was its ability to handle images that contained a non integer number of blocks. The final row and column was thus likely to contain blocks that were a different size to the majority within the frame, causing the calculations to vary depending on the current block's relative position. However, by introducing a further constraint that requires the image to contain an integer multiple of blocks all blocks are guaranteed to be same size. Although this is a more restrictive constraint the savings in hardware complexity are significant due to the

computations being identical regardless of block position.

3 Implementation process

As an intermediate step between the Matlab original and synthesisable VHDL the algorithm was ported to ANSI C. This was done with the aim of creating an adaptable model in a language that shares features with both Matlab and VHDL. C was chosen due to its flexibility and the availability of the GNU C compiler (GCC) and debugger (GDB), both of which are freely available and fully featured. Using C allows a progression to take place between the original model and the desired hardware description. The C code at any given stage represents an executable specification of the design. Because the model is executable, testing can be performed after each revision, using the original test cases applied to the Matlab algorithm. In this way errors introduced during the conversion process are not only quantifiable but may also be pinpointed to specific changes made.

The initial C implementation was functionally an exact copy of the Matlab M-code, but several syntactical changes had to be made to support Matlab instructions that do not have a direct C equivalent. For instance, in Matlab the maximum value of an array of any dimension may be found simply using '*max(array name)*'. In C this requires nested *for* loops (a level per array dimension), and temporary storage for the current maximum. A further problem exists due to Matlab not requiring variables to be declared before use, their size and type instead being inferred at run time. In C all variables must be declared before use, and therefore some analysis was required to determine the necessary details. Although in this instance this task was performed by hand, options exist to insert commands into the Matlab M-file to report the size and type of variables after use (using the *size* command for instance). This is an issue covered by the MATCH compiler (now commercially known as Accelchip), where annotations are inserted into the Matlab code and are used to instruct the compiler on the storage required [2].

Once the C version of the M-code algorithm is functionally identical to the original, testing may be performed on both versions to ensure their equivalence. By

writing the output of the C algorithm to file the output data may be loaded back into Matlab to take advantage of the rich analysis environment. This C code is denoted Version 1 and any changes that are subsequently made en-route to VHDL are given a new version number. By comparing the operation of a given version with that of its predecessor a traceable route from Matlab to VHDL is available, and the degree of conformance may be quantified. If unacceptable levels of error are introduced between two versions, then it is clear where and how the errors are occurring.

The purpose of the transformation process is to develop a route from Matlab equivalent C, to C that can be mapped to VHDL with the minimum of difficulty. Although less abstract than Matlab, C is still classed as a high-level language, and so in order to represent VHDL a coding style and subset of commands are adopted. It is the inherent flexibility of C that allows it to bridge the gap between high and low-level designs, or, more generally, the transformation from behaviour to structure. Whilst the C language contains both high and low-level constructs, the obvious and most fundamental difference between C and VHDL is that C is sequential, whereas VHDL models concurrent processes. The method of overcoming this problem used in several commercial behavioural synthesis tools is the use of non-ANSI C extensions, for example the Handel-C language used by Celoxica [3]. The method presented here allows for representation of concurrency in standard C.

3.1 Scheduling and allocation

By adopting a specific coding style, it is possible to introduce a suggestion of structure and concurrency into the standard C language. Scheduling and Allocation is a technique used in high-level synthesis to transform HDL from a behavioural representation to a structural one. The resulting hardware represents a custom single purpose processor, which conforms to the Finite State Machine with Datapath (FSMD) model [4]. Here the technique is adapted to apply to an algorithm described using a high-level language. Scheduling and allocation models hardware by splitting the original code into discrete time steps (scheduling), and then determining the best hardware usage in order to meet design constraints (allocation). These con-

straints may be in the form of minimum performance requirements or limitations in the amount of discrete hardware elements that are available. To perform the same process in a high-level language, statements present in the original description are grouped according to their data dependencies, and control signals are then generated to trigger the activation of these groups as appropriate. The mechanism used to generate these control signals takes the form of a finite state machine.

As an example, consider the C code fragment in program 1. This implements three simple assignment statements. It is clear that the order of execution of the statements is crucial in obtaining a correct output. This may be scheduled as shown in program 2. Here, the code has been implemented using two case statements. The upper one is the control mechanism, designated the control path, which produces the control signals necessary to correctly sequence the instructions contained in the lower case statement, the datapath. A new variable called state has been introduced which represents the control signals between the two parts of the program. There are two points worthy of note. Firstly, the allocation of assignment statements to case numbers is arbitrary, as it is the control path code that defines the order of execution and not their position in the code. Second, the control path has two states where no datapath activity is scheduled, state 0 and state 4, which represent a reset state and halt state respectively. The halt state is implemented by simply causing execution to loop back on itself with no corresponding datapath instruction.

Program 1 Basic sequential function.

```
void sequential()
{
    x = a + b;
    y = a - b;
    z = x * y;
}
```

An impression of concurrency may now be introduced into the code. From inspection of the original sequential code fragment, the assignments to x and y may be executed in any order without affecting the outcome of the program. These statements may therefore be scheduled to occur during the same time step. The assignment to z is dependent on the values of x and y and so must occur after their

Program 2 Sequential function with separate control and datapath structures.

```
void scheduled()
{
    state = 0;
    do{
        //Control Path
        switch(state){
            case 0:
                state = 1;
                break;
            case 1:
                state = 2;
                break;
            case 2:
                state = 3;
                break;
            case 3:
                state = 4;
                break;
            case 4:
                state = 4;
                break;
        }

        //Data Path
        switch(state){
            case 1:
                x = a + b;
                break;
            case 2:
                y = a - b;
                break;
            case 3:
                z = x * y;
                break;
        }
    }while(1==1);
}
```

assignment has taken place, on the next time step. This is shown in program 3. The assignments to x and y can now be considered to occur in parallel, and the number of control states is thus reduced accordingly.

Program 3 Independent datapath states are combined to suggest concurrency.

```
void scheduled()
{
    state = 0;
    do{
        //Control Path
        switch(state){
            case 0:
                state = 1;
                break;
            case 1:
                state = 2;
                break;
            case 2:
                state = 3;
                break;
            case 3:
                state = 3;
                break;
        }

        //Data Path
        switch(state){
            case 1:
                x = a + b;
                y = a - b;
                break;
            case 2:
                z = x * y;
                break;
        }
    }while(1==1);
}
```

The examples shown so far have been purely sequential, and have contained no complex control constructs. However, most programs contain loops and conditionals, and these require more sophisticated techniques in the interaction between the control and datapath sections. As an example consider the code fragment in program 4.

This simply assigns to z the greater of a or b . A choice must be made as to which

Program 4 Basic function with branching execution flow.

```
void sequential()  
{  
    if(a>b)  
        z = a;  
    else  
        z = b;  
}
```

path through the code should be taken in order to affect the correct assignment. This requires translation to retain the correct operation of the *if* construct but still conform to the syntax of the *switch* constructs that make up the control and data segments. Hence an extra flag variable is required to communicate the outcome of the relational operator between the control and data paths. The control path then uses this flag to decide on the correct datapath instruction to execute. The scheduled version of this code may be seen in program 5. Here, the Boolean variable *a_flag* is set during state 1 according to the outcome of the test $a > b$. The control path then sets the next state to either 2 or 3 as necessary. Control from state 2 will pass next to state 4, missing the unwanted assignment in state 3. Loops may also be implemented in a similar fashion, with a loop index being explicitly incremented in the data path. The flag is set before the end of the loop by testing whether this index has reached the desired value, and the control path uses this flag to decide whether execution should return to the beginning of the loop or exit by continuing to the subsequent state. (In order to correctly create the necessary indices and flags it is useful to transform all loops in the original code to '*while*' loops. This has no loss of functionality if performed correctly.)

The control and data paths can now be split into separate functions, which will become standalone VHDL modules. The variables that are local to each function represent internal signals, those variables that are passed between the control and data paths, such as flags and state information, are global and are instantiated in the top level of the VHDL hierarchy. Using these techniques, complex programs may thus be written in a style that exhibits both structural and behavioural information. Figure 2 shows a block diagram representation of how the completed code example

Program 5 Branching function with separate control and datapath.

```
void scheduled()
{
    state = 0;
    do{
        //Control Path
        switch(state){
            case 0:
                state = 1;
                break;
            case 1:
                if(flag)
                    state = 2;
                else
                    state = 3;
                break;
            case 2:
                state = 4;
                break;
            case 3:
                state = 4;
                break;
            case 4:
                state = 4
        }

        //Data Path
        switch(state){
            case 1:
                flag = (a>b);
                break;
            case 2:
                z = a;
                break;
            case 3:
                z = b;
                break;
        }

    }while(1==1);
}
```

of program 6 maps to structural hardware with implicit hierarchy. From this stage the translation to VHDL is fairly trivial, as there is a direct mapping between the

instructions used in C and those available in synthesisable VHDL. (There may be a slight change in format or syntax, for instance *switch* statements are known as *case* in VHDL and do not require *break* instructions on each branch.)

Writing code in this way can be viewed in hardware terms as a custom, single-purpose processing unit. In this sense, the state variable passed from the control path to the data path represents the instruction to be executed. The FSMD model can be considered as universal, in that it may represent all designs [4]. The relative sizes of the control and datapaths are directly representative of the type of application. Control oriented applications will have a large control path, whereas data processing applications will be datapath dominant.

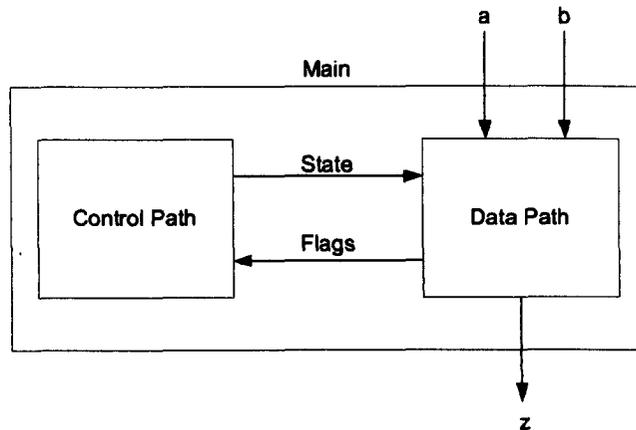


Figure 2: Block diagram of program 6.

4 Implementation details

4.1 Intermediate storage

The algorithm requires four arrays of data to store Block Min, Block Max, Node Min, and Node Max values. The sizes of these arrays are obviously dependent on the size of the input image and the block size. In the worst case, with a maximum image size of 640×480 elements and using 8×8 blocks, there will be $80 \times 60 = 4800$ blocks and $81 \times 61 = 4941$ nodes. These arrays will store 8-bit pixel values. In the original Matlab description the arrays were two-dimensional to correspond spatially to the

Program 6 Branching program with control and datapath implemented as separate functions.

```
void scheduled()
int controlpath(int state, boolean flag)
{
    switch(state){
        case 0:
            state = 1;
            break;
        case 1:
            if(flag)
                state = 2;
            else
                state = 3;
            break;
        case 2:
            state = 4;
            break;
        case 3:
            state = 4;
            break;
        case 4:
            state = 4;
    }
    return(state);
}

void datapath(int state, int a, int b, int z, boolean flag)
{
    switch(state){
        case 1:
            flag = (a>b);
            break;
        case 2:
            z = a;
            break;
        case 3:
            z = b;
            break;
    }
}
```

continued...

Program 6 Branching program with control and datapath implemented as separate functions (cont.)

```
void main()
{
    //These are internal signals
    int state = 0;
    boolean flag;

    //These represent ports to outside world
    int a, b;
    int z;

    while(state<4)
    {
        state = controlpath(state, flag);
        datapath(state, a, b, z, flag);
    }
}
```

input image. Although modern synthesis tools support multidimensional arrays it is not always apparent what form the resulting hardware will take, whereas single dimensioned arrays can be equated to blocks of memory with linear addressing. Of course, in making this modification, changes also had to be made to the calculations that provide indices into these arrays. These changes to the algorithm were fully tested during the C stage of the implementation process.

The process of determining node data requires the algorithm to select from a node's surrounding blocks the maximum and minimum values. The number of blocks that are connected to a node differs depending on its position within the image. There are therefore three stages to be completed: firstly the algorithm must decide whether or not a block exists in a certain direction from the node, then the block's corresponding address must be calculated, and finally the block data is retrieved from memory. This must happen for all four possible locations that a block may exist in relation to a node: above left, above right, below left, and below right.

In order to make the process more deterministic and simplify the resulting hardware, the sequence of instructions was made identical for each block position regardless of whether a block exists there or not. The reasoning behind this lies in the difficulty in scheduling the varying number of instructions required to select a

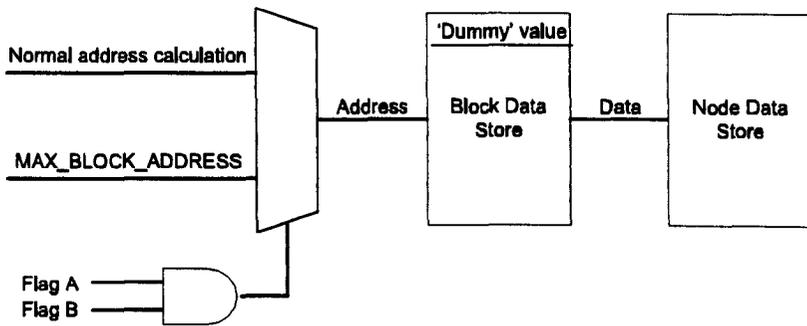


Figure 3: Flags indicating a particular block is at the edge of the image cause a dummy value to be addressed.

maximum or minimum value from one, two or four possibilities. A simpler solution is to supply dummy values for those positions where blocks do not exist, and providing appropriate dummy values are selected they will always be overwritten later by valid data. (When finding the maximum value, a dummy value of zero will be overwritten by any other data. Similarly a value of 255 will be overwritten when finding a minimum). The problem is then always one of finding the maximum or minimum value from four possibilities.

In order to supply the dummy values to the program they are stored in an extra location at the end of the memory space reserved for block data. An address is now calculated for each of the four locations surrounding a node. If a block does not exist in a certain location, then the predefined constant address `MAX_BLOCK_ADDRESS` is placed on the address lines, otherwise the row and column indices are used to find the actual block address. The resulting hardware can be seen in figure 3. The signals Flag A and Flag B represent the Boolean operators used to signify the beginning/end of a row/column. This information can be used to ascertain the presence of a block in a given direction from a node.

The memory blocks to store the block and node data were generated using Xilinx's Core Generator tool, which allows for manual instantiation of parameterisable cores for the Xilinx FPGA devices. The addition of the dummy value to the Block Min and Max storage means that $M \times N + 1$ locations are now required, where M is the number of blocks in the horizontal direction, and N is the number of blocks in the vertical direction. This equates to 4801 locations for 80×60 blocks. The storage

generated was thus two single port RAMs for the block data, at 4801×8 bits, and two single port RAMs for the node data, at 4941×8 bits.

During core generation initialisation files may be specified for memory blocks, which contain the data that the memory contents will default to at power-up. Using this facility the dummy value in the final location of the Block Min memory was set to 255, and all other locations were set to zero. The dummy value for Block Max is zero and so the whole memory may be initialised to this value. This initialisation data is incorporated into the netlists generated by Core Generator and will be read into the tools used during the generation of the FPGA programming bitstream.

4.2 Bilinear interpolation calculation

Using the modified version of the bilinear interpolation equation of equation 3, the calculation may be analysed to determine the best fixed-point representation of the resulting data. The coefficients used during the calculation all represent node data. In the worst case, these values will all equal $255 = Max_{max}$. Therefore:

$$\begin{aligned} out_{max} &= \frac{[xMax_{max} + (X - x) Max_{max}] (X - y) + [xMax_{max} + (X - x) Max_{max}] y}{X^2} \\ out_{max} &= \frac{XMax_{max} (X - y) + XMax_{max} y}{X^2} \\ out_{max} &= \frac{X^2 Max_{max}}{X^2} \\ out_{max} &= Max_{max} \end{aligned} \tag{4}$$

This indicates that when the coefficients are equal the output will match this value. Hence the linear ranges of out_{max} and out_{min} are equal to that of the image data, i.e. 8-bit.

The divide operation is a source of difficulty in implementing the algorithm in hardware. In this case, because the block size has been previously constrained to be a power of 2, the division may be performed by a right binary shift, which will incur a loss of accuracy in the value as digits are discarded. Therefore as a means of preserving accuracy in this value the shifting operation is not performed yet but the binary representation is changed; in effect the theoretical binary point is moved

$\log_2(X^2) = 2\log_2(X)$ places to the left. Now no bits are discarded, and therefore the error of this value compared to the Matlab original is zero.

The number of bits required to represent out_{max} and out_{min} is now dependent on the block size. There are 8 bits for the integer part and $2\log_2(X)$ bits for the fractional part, and hence in total $8 + 2\log_2 X$ bits are required. For example, when using 32×32 element blocks 18 bits are required, of which 10 bits make up the fractional part.

4.3 Contrast stretch calculation

The contrast stretch is performed using equation 2. As shown in Section 4.2 out_{max} and out_{min} refer to greyscale data, and after re-scaling the denominator of this expression will always equate to a value between 1 and 255. Because of the difficulties in implementing divide operations and the abundance of on-chip memory on the target device, it was decided that a lookup table of the reciprocals of integers in this range would be the best method of performing this calculation. The representation chosen for the reciprocal data was 18Q18 (an 18 bit word of 18 binary places). Spreadsheet analysis revealed this to possess a maximum representation error of 0.09%, which was deemed sufficiently accurate whilst making optimal use of the 18×18 bit dedicated multipliers present on the target device. The look-up table was implemented as a 255×18 Single Port Block ROM, generated using Core Generator. A text file containing the reciprocals of the integers from 1 to 255 scaled by 2^{18} was used as input to Core Generator to correctly initialise the memory.

Using a look-up table of reciprocals introduces error into the system because only a limited number of reciprocals are stored, which means that it is important the most appropriate value is selected. The reciprocal is chosen based on the result of $out_{max} - out_{min}$ after re-scaling back to the 0 - 255 range. The re-scaling process was originally performed by truncation of the fractional part. This results in a loss of accuracy as follows:

The precision being discarded is $2\log_2(X)$ bits, where X represents the block size

being used. The linear range of the fractional part is thus:

$$\begin{aligned} LR_{frac} &= 2^{2\log_2(X)} \\ LR_{frac} &= 2^{\log_2(X)} \times 2^{\log_2(X)} \\ LR_{frac} &= X^2 \end{aligned} \quad (5)$$

The smallest value that may be represented is therefore $\frac{1}{X^2}$. The maximum error occurs when these bits are all 1 before being discarded. In this situation the resulting integer has a maximum absolute error of:

$$MaxError_{trunc} = \frac{X^2 - 1}{X^2} \quad (6)$$

Using actual figures demonstrates this more intuitively. For example, when using 16×16 element blocks there are eight bits of precision that are discarded. The maximum error occurs when these bits are all 1, which is equivalent to $16^2 - 1 = 255$ in decimal. Because these bits represent the fractional part of the result, the maximum absolute error is $255/256$, or approximately 0.996. Truncation means that the error will always be negative as the fractional part is simply discarded. This value for maximum error is consistent throughout the 8-bit greyscale range and is insignificant at high greyscale values, but in relative terms it has a greater effect when the integer part is small.

Initial results gained from the algorithm showed that this error became unacceptable when the whole part of $out_{max} - out_{min}$ is of low magnitude, which occurs when out_{max} is approximately equal to out_{min} , i.e. when the local minima and maxima are closely bound. This is an indicator of areas of very low dynamic range. It therefore follows that the accuracy of the VHDL system compared to the Matlab equivalent is directly related to the contrast of the input image.

To decrease the error that is introduced at this stage the truncation operation described above was replaced by rounding, which resulted in significant observable improvements. In general terms, rounding means that if the fractional part is greater than 0.5 the least significant digit of the integer part is incremented. This is implemented practically by adding 1 to the MSB of the fractional part before it is dis-

carded. Because the MSB of the fractional part is now taken into consideration, the number of bits being discarded is $2\log_2(X) - 1$. The discarded linear range is thus:

$$\begin{aligned} LR_{frac} &= 2^{2\log_2(X)-1} \\ LR_{frac} &= \frac{2^{\log_2(X)} \times 2^{\log_2(X)}}{2} \\ LR_{frac} &= \frac{X^2}{2} \end{aligned}$$

The maximum error is now:

$$MaxError_{round} = \frac{\frac{X^2}{2} - 1}{X^2}$$

Continuing the example of 16×16 element blocks, the expression for $MaxError$ evaluates to $127/256$ (or approximately 0.496). This can be seen more intuitively by considering that now only seven of the eight bits that constitute the fractional part are discarded without consideration. It is important to note that although the maximum error is halved, it also becomes bipolar in that it may be positive or negative. The effect of rounding is not to decrease the total range of error, but to shift it so that it is roughly symmetrical around the integer.

This expression indicates that the error introduced into the system before the selection of an integer reciprocal value is a slowly changing function of block size, and when rounding is used its magnitude is always less than 0.5. Since this is the denominator of the stretch expression it may cause significant errors in the output when the integer part is small. For instance compare the relative error between 1 and 1.5 (33%) to that between 250 and 250.5 (around 0.001%).

From the fixed-point analysis above, it was found that the out_{max} and out_{min} values have $2 \times \log_2(BlockSize)$ binary places. This is then multiplied by the reciprocal data to give a value that has $2 \times \log_2(BlockSize) + 18$ binary places. The final stage of the calculation is to multiply by 255, then discard $2 \times \log_2(BlockSize) + 18$ bits of precision. Let the result of the reciprocal multiplication be X , and the excess

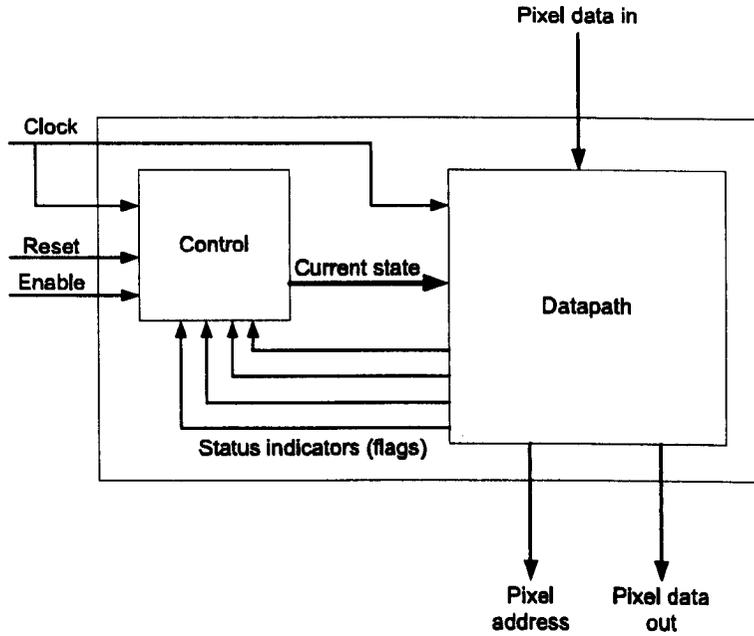


Figure 4: Overview of LRM design with FSM structure.

precision p . Then:

$$\begin{aligned}
 out &= \frac{255X}{2^p} \\
 out &= \frac{256X}{2^p} - \frac{X}{2^p} \\
 out &= \frac{X}{2^{(p-8)}} - \frac{X}{2^p} \\
 out &= X \gg (p-8) - X \gg p
 \end{aligned}$$

Where the \gg symbol denotes a right binary shift. Since a shift in hardware is easily implemented by slicing the bit array (and has zero cost in terms of gates), the final calculation has been reduced to the reciprocal multiply and a subtraction.

4.4 Design structure

The block diagram of the top-level design may be seen in figure 4, and figure 5 uses a timing diagram to display the relationship between the Control and Datapath sections. The Start signal is used to initiate processing of image data. The current state of Control represents the next state of the Datapath, or using the custom processor paradigm, the current state of the Control function represents the next instruction to be executed.

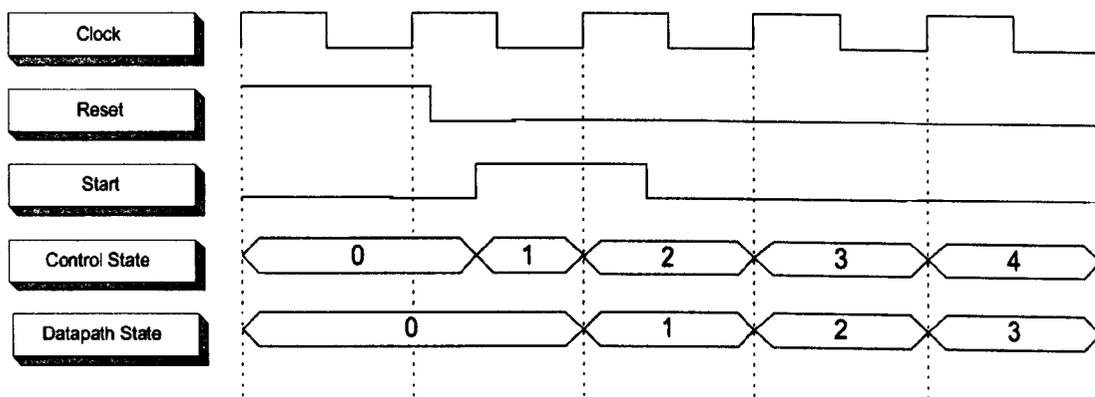


Figure 5: Timing diagram shows relationship between control and datapath elements of the design.

5 Verification and results

5.1 Methodology

Testing was first carried out on the Control module using a simple test bench that provided the clock and reset signals and simulated the flag signals from the datapath, then through monitoring of the control path's state output the design could be verified. Rather than write a testbench specifically for the datapath it was decided to write the top-level module, integrate the control and datapath blocks, and develop a testbench for the whole system. This decision was made on the basis that much of the signal generation that would be required of a datapath testbench already existed in the form of the verified control path, and the top-level module contains no logic so any errors in the functionality could be traced directly to the datapath.

Testing of the completed design was carried out using real image data. The image processing toolbox of Matlab provides capabilities for working with images as ordinary matrices, and so the manipulation and analysis of images is straightforward. Images may also be imported from recognised file formats such as JPEG and GIF, or read in as data from text files. Using these facilities a test procedure was developed that allowed the VHDL design to be tested alongside the original Matlab algorithm using real image data.

After an image has been imported into Matlab it is resized to fit into the 256×240 element input memory of the VHDL design. (This image size is purposefully small

to keep simulation times down during early ‘proof-of-concept’ testing.) Resizing an image may be performed in two ways: by simply removing columns and/or vectors from the matrix; or by the Matlab function *imresize* which uses interpolation to resize the image. In practise a mixture of these two methods was used: *imresize* was used first to reduce the overall image to approximately the correct size whilst retaining the original aspect ratio, then columns and/or vectors were removed to alter the aspect ratio to that of the target memory. Resizing the images in this manner minimises distortion. Once resized, the images were imported into the VHDL design by using a simple Matlab function that was developed to write matrices to a text file, in a format that could be read by Modelsim’s Memory Editor. As part of the VHDL testbench two RAMs were instantiated to act as input and output image storage. Because the testbench is not synthesisable this can be done without details of the available memory on the target system.

Upon completion of the Modelsim simulation, the Memory Editor was used to write the contents of the output frame store to a text file, which was then imported as matrix data into Matlab. A custom function is then used to reshape the matrix to the correct dimensions (required since Modelsim writes out Memory data as a vector), and cast the data values as image data. During this process the data is not affected in any way.

In order to create a benchmark against which the Modelsim output may be gauged the test image was also processed by the original Matlab version of the LRM algorithm. An ideal implementation would have zero difference between the VHDL and Matlab outputs.

5.2 Analysis

The initial test image was chosen from those supplied with the Matlab image processing toolbox as a good example of an image with poor contrast, and with several details that may be highlighted by the algorithm. The image is shown in figure 6.

The image was applied to both versions of the LRM algorithm using 16×16 element blocks. Figure 7 and figure 8 show the corresponding output images from the VHDL and Matlab methods of implementation respectively. To the human eye the



Figure 6: Original test image with poor contrast.

images appear nearly identical. A difference image between the two implementations is shown in figure 9 (for ease of presentation the image has been inverted and scaled to use the full greyscale range).

Greyscale images of the type used here may be easily assessed in the frequency domain using the *imhist* function of the image processing toolbox, which produces a histogram of the image data using a default of 256 bins representing each possible greyscale value. Figure 10 shows the histogram of the test image. The poor contrast of the image is reflected in the close proximity of the two peaks (which represent the mean values of the image subject and the background), and the narrow range of intensities that have a non-zero number of elements indicates a low dynamic range. The corresponding histograms from the enhanced images are shown in figure 11 (VHDL) and figure 12 (Matlab).

The improvements to the image's contrast and dynamic range are clearly visible in both cases. (In some ways the enhanced images have a contrast that is now too high, shown by the skew of the background mean towards dark pixel values.) Of more interest is the similarity of the methods, although the VHDL enhancement has resulted in a more jagged curve. The smoother curve of the Matlab approach is evidence of a more even distribution, which in turn would suggest that the resulting image has more gradual transitions between light and dark areas. In practise it is difficult to detect this difference visually. It is also worth noting that although



Figure 7: Test image after processing by VHDL implementation.



Figure 8: Test image after processing by Matlab implementation.



Figure 9: Difference image between Matlab and VHDL implementations.

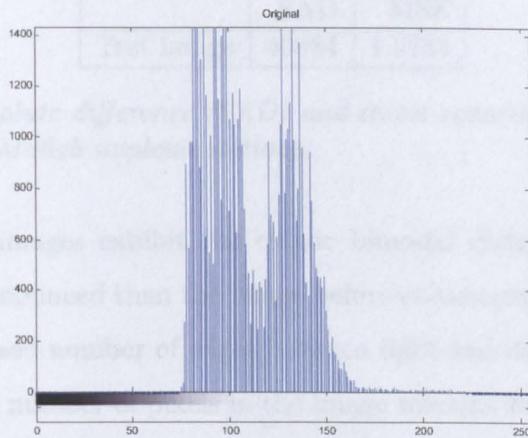


Figure 10: Brightness histogram of original image.

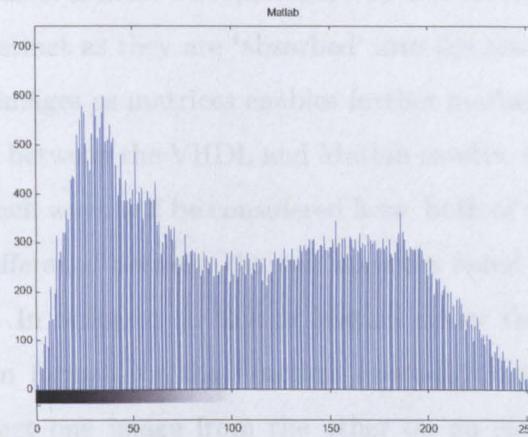


Figure 11: Histogram of test image after processing by VHDL implementation.

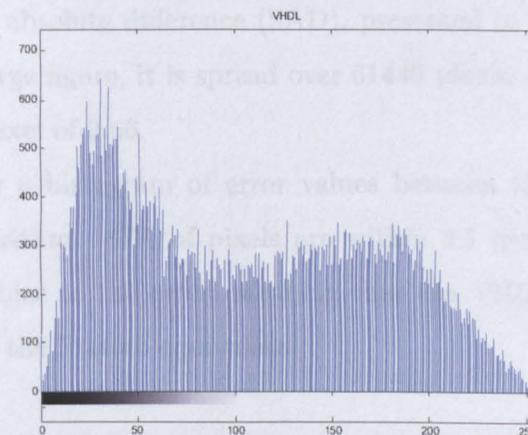


Figure 12: Histogram of test image after processing by Matlab implementation.

	SAD	MSE
Test image	40684	1.0784

Table 1: Sum of absolute difference (SAD) and mean squared error (MSE) metrics between VHDL and Matlab implementations.

both the enhanced images exhibit the classic bimodal distribution the peaks are considerably less pronounced than the image before enhancement, an effect probably caused by the increased number of edges between light and dark areas in the image. (Of course, since the number of pixels in the image remains fixed the envelope curve of the histogram must contain the same area both before and after enhancement. If the number of edges between the foreground and background mean values increases, the mean values themselves must be represented by a lower number of pixels. Hence the peaks are less distinct as they are ‘absorbed’ into the rest of the curve.)

Considering the images as matrices enables further mathematical analysis of the degree of conformity between the VHDL and Matlab results, which can be gauged in several ways. Two such ways will be considered here, both of which are derived from the mathematical difference between the two matrices found by simply subtracting one from the other. In order to do this in Matlab either the data must be recast into double precision format, or the function *imabsdiff* may be used, which will automatically subtract one image from the other on an element-by-element basis and return a matrix of absolute differences between corresponding elements. This matrix is then used to generate two measures of conformity, mean squared error (MSE), and sum of absolute difference (SAD), presented in table 1. Although the SAD may seem a large figure, it is spread over 61440 pixels, and equates to a mean absolute error per pixel of 0.66.

Figure 13 shows a histogram of error values between the VHDL and Matlab versions of the algorithm. 87% of pixels are within ± 1 greyscale value. Overall, there is a negative bias to the error, meaning that the VHDL output may appear slightly darker than the Matlab equivalent.

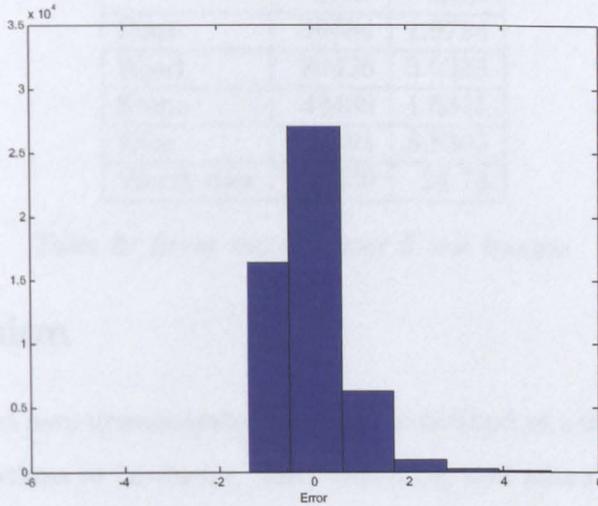


Figure 13: Histogram of error values between the images of figures 7 and 8.

5.3 Results over other images

The above tests were carried out on five images, with four of the five being examples of typical images (consisting of foreground and background areas) with poor contrast. The final image was artificially created in order to test the limits of the system by exploiting the known source of error: the reciprocal selection. This image was designed to have exceedingly low contrast and dynamic range, and was generated in Matlab as a matrix of elements all equal to 127 before application of zero mean Gaussian noise of low variance. The resulting image had mean value 127, maximum value 137 and minimum 115, providing the low input range necessary to amplify relative error. These results show that in this situation the SAD is 241330, which equates to a mean error per pixel of around 3.9. Although less than ideal, this artificially poor input image results in a VHDL output that is a fair approximation of the floating point Matlab equivalent. The resulting measures of conformance from all five images are shown in table 2.

These measurements of error between the Matlab and VHDL representations are in some cases fairly large. The source of the discrepancies can be pinpointed to the reciprocal selection. A discussion of ways to improve the accuracy of the design may be found in the conclusion.

	SAD	MSE
Pout	40684	1.0784
Road	80426	3.0333
Scene	43439	1.6315
Face	123101	8.8303
Worst case	241330	24.73

Table 2: Error metrics over 5 test images.

6 Conclusion

The work presented here demonstrates an effective method of realising high-level descriptions of algorithms in hardware. The scheduling and allocation method allows the sequential programming paradigm to be used as the basis for hardware design, but still leverage parallelism of execution where it is possible. The resulting FSM model may be applied universally to implement any hardware-design. It is a framework that is repeatable across different forms of algorithm and allows a consistent approach to be taken, using well-known and understood techniques. In most cases this improved tractability will drastically shorten the time take to realise algorithms in hardware. The main problem then becomes one of optimisation, as without it the resulting implementation may not be adequately efficient. The main areas for optimisation lie in optimising for speed by identifying the interdependencies between variables that prevent parallel execution of statements, and optimising for hardware cost by sequencing instructions to share functional units within the datapath.

In addition to using scheduling and allocation several hardware based techniques were used that facilitated the hardware design and allowed an assessment of the error within the system to be made. Although the way in which these techniques were applied here is specialised to this particular algorithm, the concepts may be used in a more general sense with whole classes of algorithms. For instance, careful application of constraints on an algorithm's parameters may greatly reduce the mathematical burden placed on the hardware.

In the case of the LRM algorithm, it was shown that the error of intermediate values could be virtually eliminated through careful manipulation of the fixed-point representation. In fact, had the interpolation division been implemented using a true

hardware divider the total error throughout the system would have been arbitrarily small and a function of the time allowed for calculating precision. In this application a look-up table of reciprocals was used instead to improve the speed of the system. The error introduced by the reciprocal multiply is caused not by the reciprocal storage or representation, but rather through the limited quantity of reciprocal values available, and therefore to reduce this error a larger look-up table containing a greater number of reciprocal values would be necessary. By doubling the number of available reciprocals and continuing to use rounding instead of truncation the representation error before the divide could be halved. In terms of hardware cost this would require an increase in reciprocal memory from 255×18 bits to 511×18 bits or a further 4.5Kb, which is minimal in terms of memory available on the target device. Each doubling of the number of reciprocals would halve the representation error before the divide, and therefore this represents a trade-off between accuracy and memory requirements.

When working with digital greyscale images the designer often has a measure of leeway, as inputs are usually integer values, and unless non-integer coefficients are introduced or divide operations are used then the data will remain as integers throughout the algorithm. Furthermore, the accuracy of the output is usually only required to the nearest integer. In these situations, providing fixed-point representations are used correctly, the need for floating-point arithmetic is greatly exaggerated. This is particularly true in the domain of FPGAs, where the constraints of fixed register sizes and word lengths are not present. In this work the fixed-point representations were largely determined by the nature of the algorithm itself, however this is not always possible and often requires some experimentation to find the best solution. Matlab may be useful in these circumstances as it contains functions such as '*quantizer*' which allow fixed point representations to be easily specified and applied to groups of data. The effects of changing these representations may then be easily observed, which promotes experimentation at the Matlab level before following the scheduling and allocation route.

The often-stated difficulty in high-level synthesis is that the time it takes to follow the design flow prevents the subtle experimentation that is required to find

the optimum solution. This problem may be lessened by using a form of coding that promotes a repeatable design flow between the high-level algorithm development language and the HDL RTL, and thus bridges the gap between behaviour and structure.

References

- [1] J. D. Fahnstock and R. A. Schowengerdt, "Spatially variant contrast enhancement using local range modification," *Optical Engineering*, vol. 22, no. 3, pp. 378–381, 1983.
- [2] P. Banerjee, "An overview of a compiler for mapping MATLAB programs onto FPGAs," in *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, Jan. 21–24, 2003, pp. 477–482.
- [3] Handel-C language reference manual. Accessed 8/12/06. [Online]. Available: <http://www.celoxica.com/techlib/files/CEL-W0410251JJ4-60.pdf>
- [4] D. D. Gajski and L. Ramachandran, "Introduction to high-level synthesis," *IEEE Design and Test of Computers*, vol. 11, no. 4, pp. 44–54, 1994.

Appendix C: Implementation of LRM Algorithm using System Generator

System Generator Implementation of LRM Algorithm

September 2004

Summary: A period of approximately three months was spent learning the System Generator tool and how it could be of use to Thales for implementing image processing algorithms. The work was based on the LRM method of contrast enhancement that had been previously implemented in hardware using hand-coded methods. Since the time of writing this report some of the features and capabilities of System Generator have changed, however many of the general conclusions presented here remain valid.

Oliver Sims

EngD 2nd Year

Industrial Sponsor: Thales Optronics

1 Introduction

System Generator [1] is a set of libraries for Mathworks' Simulink that offers the ability to model Xilinx FPGA hardware within the Simulink environment. In essence the libraries are simulation models of the IP cores available in Core Generator, the Xilinx tool for IP parameterisation and generation within traditional Hardware Design Language (HDL) synthesis design flows. The System Generator design flow reduces the need for manually crafted HDL, and the graphical design environment allows blocks to be connected together in a block diagram style, before the tool automatically generates the HDL for synthesis. The tool also includes some advanced simulation options such as hardware co-simulation and an interface to Mentor's Modelsim HDL simulator for incorporating System Generator's output within a larger design.

The process of realising a sophisticated algorithm in hardware may often be a more difficult and time-consuming process than the algorithm development itself. Furthermore, the algorithm's effectiveness may be diminished by hardware related issues, for instance quantisation effects. It may therefore be assumed that if the process of realising an algorithm in hardware is eased, the level of abstraction is raised, and the designer's efforts may be redirected towards performance and efficiency related issues.

The algorithm to be implemented is the Local Range Modification (LRM) method of contrast enhancement, developed by Fahnestock & Schowengerdt [2]. This algorithm was successfully demonstrated on FPGA hardware in an earlier project, after following a traditional hand-coded route through the implementation tools. The demonstration system also included clock management and composite video generation. The aim of the work with System Generator is to replace the original implementation of LRM with a version created with System Generator, whilst maintaining the same system framework, in order to demonstrate the functional equivalence of the two methods. It is important to note that the purpose of this work is not to assess the performance or efficiency of the product of a particular means of implementation, but rather to examine the difficulties faced by the designer during the implementation process. For this reason attempts were made to maintain similarities between the demonstration systems, and focus purely on the subject of design

realisation.

2 System Generator capabilities

2.1 Design features

The following are some of the features of the Simulink and System Generator environment that are used in the creation of a hardware design.

Xilinx Blockset

The libraries provided by System Generator contain the functions that may be later converted to hardware. There are two libraries of cores, named the Xilinx Blockset and Xilinx Reference Blockset. The standard Blockset contains mostly basic operators such as arithmetic, logic, and memory (RAMs, ROMs, and FIFOs of various types), and some common DSP functions such as Fast Fourier Transform or FIR filters. There are also some more sophisticated blocks such as a 'soft' processor and microcontroller (Microblaze and Picoblaze respectively), convolution encoder, time division multiplexer, etc. The Reference Blockset contains some more abstract functions, such as a more varied range of filtering options (including a basic 2D filter for imaging applications), CORDIC processing, and both Mealy and Moore type state machines.

Hierarchical design methodology

The Simulink environment is tailored towards hierarchical design methods. Subsystems are used to encapsulate lower levels of hierarchy, which may then be instantiated as often as required in the wider system. This is closely related to the way in which hardware is traditionally designed and promotes both efficiency and lucidity; it is especially useful in this environment as large Simulink designs may become cluttered and difficult to understand.

Handling of fixed-point representations

Deciding on the best representation of analogue data within the digital system is often one of the most difficult and skilled tasks facing the designer, and also one of the main factors in the effectiveness of the completed design. System Generator contains several components that aim to make this process easier. The conversion of floating-point data from the outside world into the fixed-point representation used by the system is handled through the use of 'Gateway' ports, which correspond to Input/Output Blocks (IOB) on the FPGA. These ports are parameterisable: the user may choose the total number of bits to be used in the representation, the number of binary places, and the format (i.e. unsigned, 2's complement, or Boolean). Because the parameters are easily adjustable, the designer may evaluate the effect of using different representations simply by making the alteration and re-simulating the design. Special blocks that measure the error due to quantisation are available to be inserted into a design at any point and help to assess the effects of different binary representations on the accuracy of the system. Because it is not always apparent how the design is affected by representation error before implementation in hardware, the ability to model the quantisation process in the simulation environment is one of the key strengths of using System Generator.

M-Code blocks

M-Code is the Matlab programming language used to sequence multiple commands to create fully featured functions or scripts. Because Matlab contains many high-level commands, particularly related to matrix and vector operations, it is a powerful language for algorithm design and as such M-files are a common algorithm representation. It would therefore be highly desirable if it was possible to synthesise hardware directly from M-code, and a great deal of effort is directed towards this cause in the wider research community. At present System Generator offers limited support for synthesis of M-code. The mechanism for implementing M-code is the M-Code Block, basically a block that has associated with it an M-file. The block automatically adopts the correct number of inputs and outputs depending on the parameters of its M-file, and so represents the custom Matlab function within the

Simulink environment.

The limitations of the M-code block stem from the way in which it is handled during the generation phase. The hardware resulting from such a block may form combinatorial logic only, which means that the function may only contain conditional and logical operators and simple arithmetic functions. Sequential logic cannot be generated, and this removes the possibility of using any type of looping or waiting constructs.

One use of the M-code blocks is in the implementation of the state transition function of a Finite State Machine (FSM), which is basically a set of conditional assigns. Most of the other capabilities of the M-code block are made redundant by the functions represented in the block libraries. It is hoped that the capabilities of this block will increase in future versions of the software.

2.2 Simulation capabilities

One of the strengths of the System Generator design flow is the rich set of simulation tools provided by Simulink that may be used in analysis of the design. In particular the possible signal sources, which include periodic signals, ramps, steps, and random noise, are all easy to generate and offer a level of sophistication not easily achieved using traditional HDL techniques. It is also possible to import data from Matlab in real time, and export outputs back to Matlab for analysis. This readily allows for examination of a Simulink design using Matlab's abundant analysis capabilities, and gives the designer a convenient but powerful way of comparing an algorithm's System Generator implementation against the original model.

When simulating a System Generator design before generation it is necessary to draw comparisons with the equivalent simulation environment for a HDL flow, i.e. using a HDL simulator such as Modelsim. Modelsim is more powerful and offers a greater degree of control to the user, however the nature of Simulink means that it is much simpler to generate stimuli and process the results. Nevertheless, there are some features of Modelsim that would assist in Simulink simulations, such as viewing of the contents of RAM blocks during simulation; being able to work more closely with waveforms using cursors and search functions; and the ability

to force signals to a desired value during the simulation. Many of the standard features of HDL simulators are unavailable or implemented differently, the net result is that functional verification of a design and fault diagnosis may be more difficult in Simulink, particularly for those engineers accustomed to HDL testbench methods. Of course Modelsim may be used later after generation of HDL, but any alterations to the design's function would almost certainly need to be carried out back in Simulink. This iterative approach may be time consuming, and would possibly lead to the adoption of incremental synthesis methods.

Another potential problem discovered during the implementation of the LRM algorithm was that simulation of large designs required considerable amounts of computing resources or would otherwise execute extremely slowly. This is a pretty unavoidable consequence of modelling such complexity, but with some models containing hundreds of blocks it can become necessary to use high-powered computers, or resort to using a modular approach to verification where individual subsystems are verified functionally correct before incorporation into the larger model.

2.3 Generation

Once the design is complete and functionally correct, System Generator offers several methods of progressing towards realisation in hardware.

- HDL generation, where VHDL or Verilog is generated as connected instantiations of the Xilinx cores.
- NGC Netlist, which generates HDL then automatically links to a synthesis tool to produce a netlist-level output.
- Bitstream, which takes the design to the FPGA programming stage, possible when the entire design is modelled in System Generator.
- EDK Export, for importing the design into an embedded system built using the Xilinx EDK tool.
- Hardware Co-simulation, which allows simulation of the System Generator design in FPGA hardware using Modelsim for providing stimuli and displaying

the results, and Simulink to provide the interface with the hardware.

The System Generator flow integrates seamlessly into the implementation tool chain. When generating HDL all the necessary VHDL or Verilog files are produced, along with testbenches, timing constraint files, ISE project files, and netlists for the Xilinx IP. When synthesis is also carried out as part of the generation process a script is automatically generated which contains the necessary commands to identify the design files, apply the necessary constraints, and perform the synthesis. Upon execution of the script the synthesis tool will produce a netlist without manual intervention by the user.

Post-generation verification

As part of the generation process the System Generator design is simulated, during which time the value of the inputs and outputs at each time step is recorded to a file. Once generation has completed (either to HDL or NGC) a Modelsim simulation script is automatically produced to import the necessary libraries, compile the design files, run a simulation, and check the results against those from Simulink. After the simulation has ended scores are displayed that show the number of errors and a percentage of correct samples, thus giving a simple measure of conformity between the pre- and post-generation design. This is an impressive feature that effectively allows the system designer, who may be proficient in Simulink but is inexperienced using a complex HDL simulator such as Modelsim, to perform verification of the model in Simulink and use Modelsim only to check the equivalence of the final implementation. It is also possible to perform a similar simulation after various stages in the implementation process, such as post-map and post-PAR.

Incorporating System Generator designs into a larger system

The results from a System Generator design may later be incorporated into a larger hardware design, either by direct instantiation of HDL or as a black box representation of a NGC netlist. The latter method allows an incremental synthesis approach to be taken, whereby the System Generator portion of the design is not re-synthesised

with the other HDL but incorporated at a later stage by the FPGA implementation tools, which merges the associated netlists into the complete design.

Simulation of the entire design is made possible through the use of special HDL files generated during the compilation procedure, along with initialisation files for any blocks of memory that may be needed.

2.4 Hardware co-simulation

Hardware co-simulation allows parts of a design to be compiled into hardware during a Simulink simulation so that the FPGA itself is used to calculate results. Simulink interfaces to the FPGA hardware via JTAG over the Parallel IV programming cable. During simulation, the inputs to a compiled block are fed to the FPGA, and as the results are generated they are read back into Simulink. This can provide a significant simulation speed up, and allows the designer to test out parts of design in hardware before completing the entire design. The hardware co-simulation feature may be used with any board with a Xilinx FPGA, but requires a Board Support Package (BSP). Template files are provided to create a custom BSP if necessary, which requires knowledge of the board's JTAG scan chain and FPGA pin-out.

An interesting use of the hardware co-simulation feature is to accelerate a design composed purely of HDL, i.e. with no Xilinx IP blocks. HDL designs may be imported into System Generator as a black box, which at simulation time will be simulated using Modelsim. If this black box is compiled for hardware co-simulation there exists a method of simulating a purely HDL design with Modelsim, but using the speed advantages of having the design running in hardware. During simulation, Modelsim will provide the stimuli (usually defined by a testbench) which is fed to Simulink; Simulink then interfaces with the FPGA to provide the stimuli on the appropriate inputs and retrieve the outputs when available, which are then returned to Modelsim for display.

3 LRM algorithm implementation

3.1 Implementation details

The LRM algorithm was implemented using System Generator to demonstrate the realisation process. The design was heavily influenced by the results of the earlier manual HDL synthesis flow, and it is difficult to estimate how the second implementation would differ without the earlier work, for instance, how much longer the process would have taken without the previously gained knowledge of the algorithm's operation. The two systems have virtually identical implementations, and this is evident in the resulting device usage.

The algorithm was divided up into three distinct phases: Block Extrema, Node Extrema, and Contrast Stretch. The three phases of the design run sequentially; this process is controlled by a simple FSM. Each phase of operation is represented by a subsystem in the top-level of hierarchy; a further subsystem contains logic for generating the addresses used to identify pixel locations, and acts to sequence the flow of data through the algorithm. Blocks of Single Port RAM are instantiated between the top-level blocks to store intermediate data, and the address lines to these RAMs are multiplexed as necessary. The system is shown in figure 1.

3.2 Implementation specifics

During the course of the design process there were several key problems that needed to be solved. In order to provide an insight into the subtleties of the System Generator design process the following section describes some of these problems and the methods used to overcome them.

System control - finite state machine

The FSM used to control the sequence of execution of the three phases of the algorithm was implemented in M-code using the M-code block. The system has four inputs and three outputs; one of the inputs represents the current state, the remaining inputs and the three outputs are the respective status and activation signals for the three subsystems under control. Also shown in the diagram is a register that

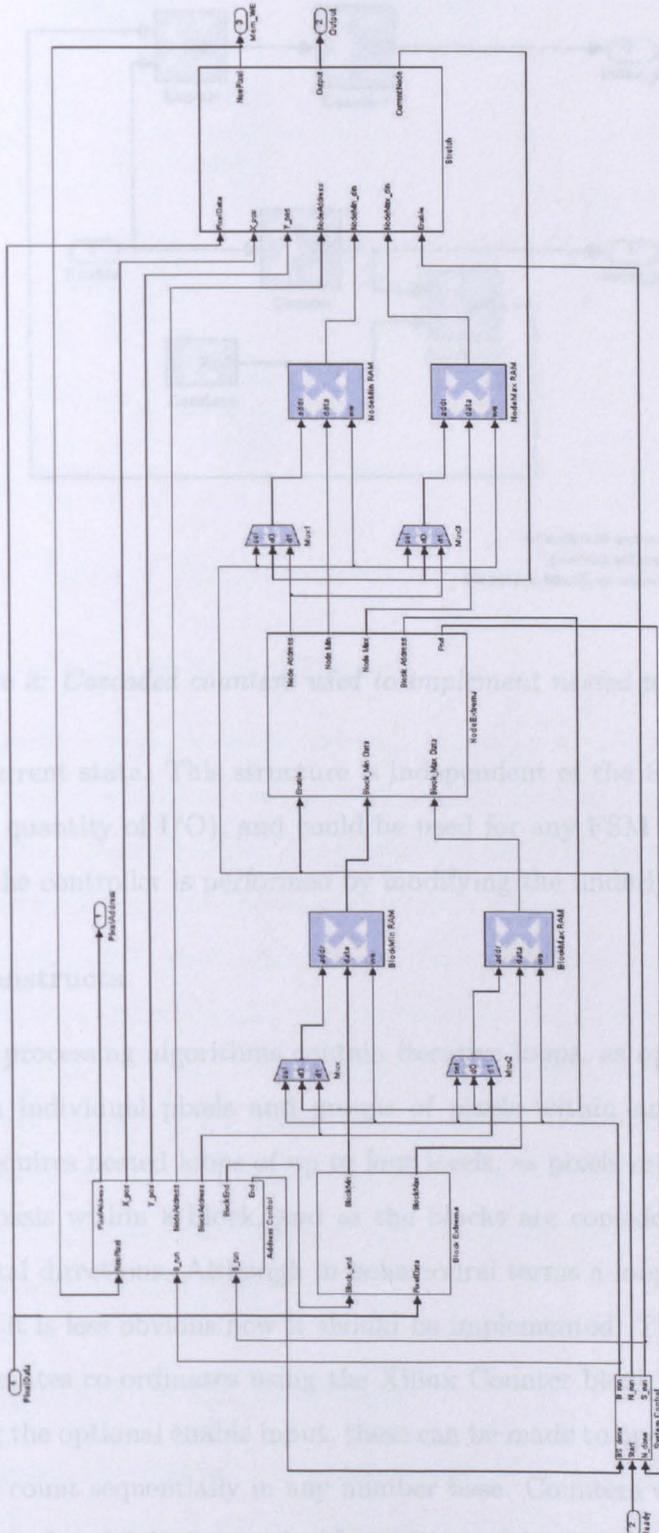
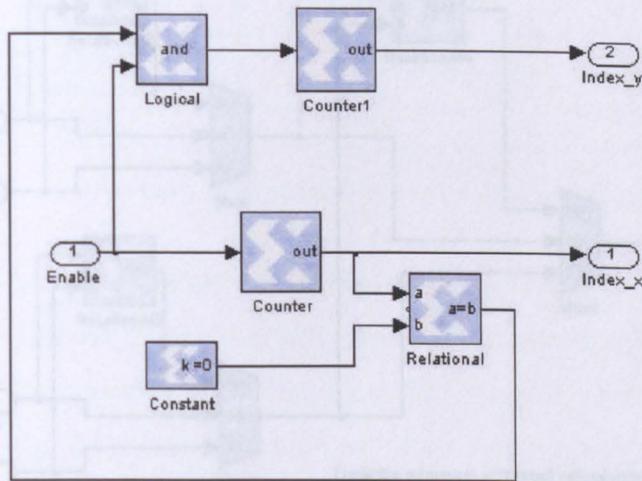


Figure 1: Overview of LRM algorithm in System Generator.



Counters to generate x and y pixel position within current block. Blocks are 16x16, so counters are 4-bit free-running.

Figure 2: Cascaded counters used to implement nested program loops.

The LRM algorithm requires the generation and storage of the current state. This structure is independent of the FSM that it contains (besides the quantity of I/O), and could be used for any FSM code. Changing the function of the controller is performed by modifying the underlying M-code.

Looping constructs

Most image processing algorithms contain iterative loops, as operations are carried out both on individual pixels and groups of pixels within an image. The LRM algorithm requires nested loops of up to four levels, as pixels are examined on a line by column basis within a block, and as the blocks are considered in both vertical and horizontal directions. Although in behavioural terms a loop is easy to describe, in hardware it is less obvious how it should be implemented. The solution shown in figure 2 generates co-ordinates using the Xilinx Counter block. By cascading them and utilising the optional enable input, these can be made to operate in a hierarchical manner and count sequentially in any number base. Counters were also used as the 'select' input of multiplexers to enable sections of hardware to be activated in a cyclical manner.

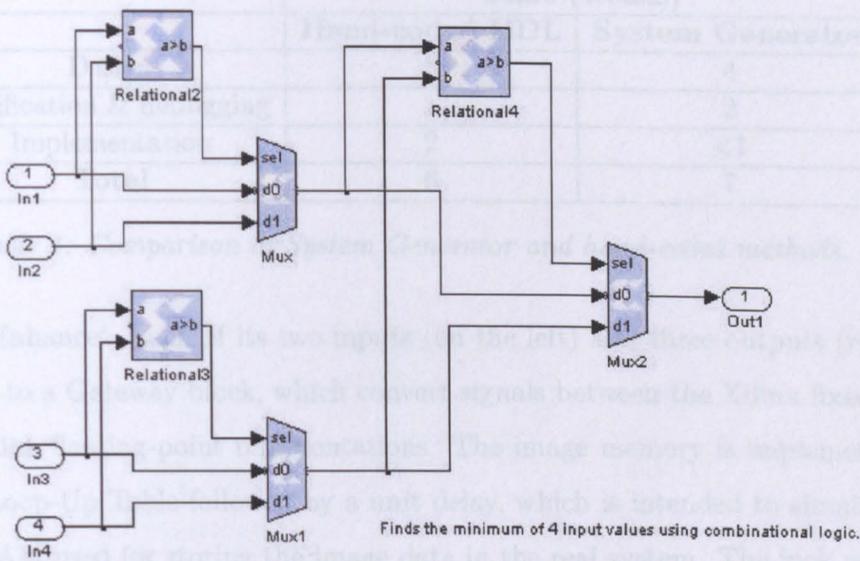


Figure 3: Parallel comparison of four data elements.

Comparison of parallel data elements

The LRM algorithm requires the maximum and minimum values from up to four sets of data to be found, which are used as coefficients in the contrast stretch. The ability to compare data values in parallel and extract the extreme values may be carried out using combinational logic, in the manner shown in figure 3. This consists of using relational operators to control multiplexers, which feedforward the appropriate value. In general, because the relational operator has a maximum of two inputs it requires $n - 1$ comparitors to reduce n inputs down to a single value. Furthermore, the comparison will require $\log_2(n)$ stages, or logic levels. Of course, blocks of logic that are deep because of cascaded stages may cause timing difficulties unless pipelined; however pipelining is easily achieved in System Generator due to most blocks offering the option to incorporate register delays.

3.3 Simulation of LRM design

The LRM design was simulated by encapsulating it within a subsystem, then providing stimuli and monitoring the outputs at this top level. The schematic is shown in figure 4. The diagram shows the algorithm contained within a subsystem named

	Time (weeks)	
	Hand-coded HDL	System Generator
Design	3	4
Verification & debugging	1	2
Implementation	2	<1
Total	6	7

Table 1: Comparison of System Generator and hand-coded methods.

‘ContrastEnhance’. Each of its two inputs (on the left) and three outputs (right) is connected to a Gateway block, which convert signals between the Xilinx fixed-point and Simulink floating-point representations. The image memory is implemented as a Direct Loop-Up Table followed by a unit delay, which is intended to simulate the on-chip RAM used for storing the image data in the real system. The look up table is preloaded with data by specifying the name of a Matlab vector, which in this case is a test image (the conversion of an image to a vector representation is fairly simple using a few Matlab commands). The output of the design was monitored using a scope block, and was additionally written back to Matlab for more detailed analysis. The icon in the top right of the figure is the System Generator token, which contains various settings used to control the simulation and generation of Xilinx cores.

4 Results

Table 1 shows the time taken, in weeks, to perform the various stages of design realisation using both the manual HDL design flow and System Generator. The results may be slightly misleading, as although the System Generator flow took longer to produce a working result the time learning to use the tool is indistinguishable from the time spent implementing the LRM algorithm. However the design process was undoubtedly quickened by the knowledge gained during the HDL flow, and as System Generator claims to improve design times the results begin to appear less favourable. The majority of the design time during the System Generator flow was spent forming functional units that would otherwise be automatically inferred from HDL by a synthesis tool. As would be expected, the time spent implementing a design using the ISE tools is cut significantly when using System Generator.

5. Conclusions

One of the strengths of designing FPGA hardware compared to ASIC is that the designer does not have to worry about physical design issues to the same extent. In effect, there is less requirement for the designer to be expert in the intricacies of semiconductor and VLSI techniques, which often require emphasis to be placed on the functionality of the design. The design work for FPGA hardware largely automates much of the physical design that would be required for ASIC. This saves step along the route towards a behavioural description of the design, which is a partly behavioural view of functional blocks. Behavioural synthesis is a design methodology that has been practised for several years, and although it is not a magic wand, it has produced a number of interesting results. Fundamentally, System Generator offers a high-level design environment for several years, and although it is not a magic wand, it has produced a number of interesting results.

Within the Simulink environment, a designer can create a high-level design of a system, which is then automatically converted into HDL, tested, and synthesized. In real hardware design, the designer must know the details of the hardware, the main skill lies in knowing how to use the available hardware resources.

Despite this, in many respects System Generator offers an elegant method of producing hardware. For data-flow applications, particularly communications, the graphical environment allows the designer to see the progression of data through the system, insert registers for pipelining and reuse if necessary, and observe the effects of varying fixed-point implementations with ease. In this situation the Simulink simulation apparatus is well suited to the task and simulating the System Generator design is the same process as verifying the original algorithm. However, when the algorithm does not fit up to the data flow paradigm, it becomes difficult to use the graphical environment.

The designer must resort to using low-level design techniques, building up the complex operators from fundamental elements such as registers, basic arithmetic units, and logic. It is this aspect of the skill and experience of the designer that is the most significant factor in the

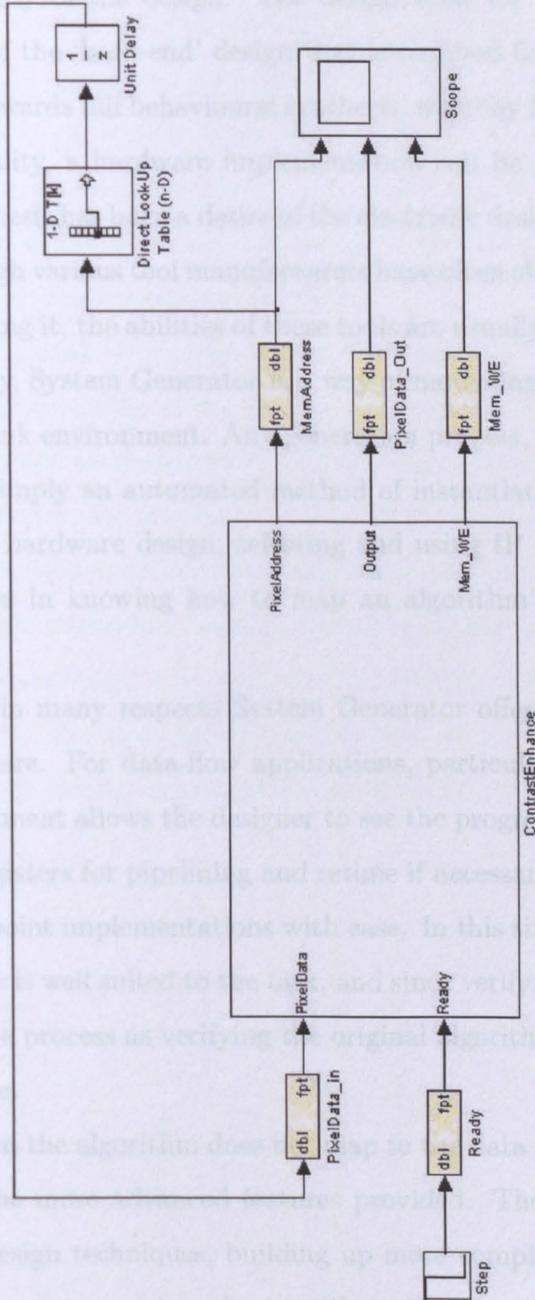


Figure 4: Simulation framework.

5 Conclusions

One of the strengths of designing FPGA hardware compared to ASIC is that the designer does not have to worry about physical device issues to the same extent. In effect, there is less requirement for the designer to be expert in the intricacies of semiconductors and VLSI techniques, which allows greater emphasis to be placed on the functionality of the design. The design tools for FPGA hardware largely automate much of the 'back-end' design that is required for ASIC. This is one step along the route towards full behavioural synthesis, whereby from a purely behavioural view of functionality, a hardware implementation can be generated automatically. Behavioural synthesis has been a desire of the electronic design community for several years, and although various tool manufacturers have often claimed to have produced a method of achieving it, the abilities of these tools are usually not without limitations.

Fundamentally, System Generator is a way of modelling pre-determined IP cores within the Simulink environment. Any generation process, whether to HDL, netlist, or bitstream, is simply an automated method of instantiating and connecting cores together. In real hardware design, selecting and using IP is only half the problem, the main skill lies in knowing how to map an algorithm's behaviour onto the IP available.

Despite this, in many respects System Generator offers an excellent method of producing hardware. For data-flow applications, particularly communications, the graphical environment allows the designer to see the progression of data through the system, insert registers for pipelining and retime if necessary, and observe the effects of varying fixed-point implementations with ease. In this situation the Simulink simulation apparatus is well suited to the task, and since verifying the System Generator design is the same process as verifying the original algorithm greatly reduced design times are possible.

However, when the algorithm does not map to the data flow paradigm it becomes difficult to use the more advanced features provided. The designer must resort to using low-level design techniques, building up more complex operators from fundamental elements such as registers, basic arithmetic units, and logic. In this situation the skill and experience of the designer is the most significant factor in the pro-

duction of a correct and efficient implementation, but the majority of the effort is merely replicating by hand what commercial HDL synthesis tools are designed to automate. It is worth remembering that the move towards logic synthesis tools occurred due to the difficulties faced by designers when attempting to use schematic capture techniques with systems of high complexity.

Much of this difficulty stems from the absence of IP that suits the application domain. In order to overcome these problems it may be necessary to create a library of IP, in the form of configurable subsystems, for use in the chosen domain. Configurable subsystems are an integral part of the Simulink environment, and in hardware terms represent parameterisable blocks. The custom made IP library would consist of various functional blocks which are designed specifically to carry out the tasks that are common to the application domain.

Matlab and Simulink offer a high powered environment for algorithm design, and utilisation of these strengths in the process of designing hardware could potentially be of great advantage to the designer. By linking the algorithm design directly to the design of hardware, the process may not only be completed more quickly, but the end results may exhibit both improved efficiency and accuracy. With the inclusion of some of the more interesting features already available in System Generator, such as hardware co-simulation, it is evident that System Generator could potentially be of great benefit. For now, for algorithms that do not conform to the data-flow paradigm, it is a promising but immature solution.

References

- [1] System Generator user guide. Xilinx Inc. Accessed: 03/01/2007. [Online]. Available: http://www.xilinx.com/support/sw_manuals/sysgen.ug.pdf
- [2] J. D. Fahnstock and R. A. Schowengerdt, "Spatially variant contrast enhancement using local range modification," *Optical Engineering*, vol. 22, no. 3, pp. 378–381, 1983.

Appendix D: Implementation of PID Servo Control Algorithm

System Generator Implementation of PID Servo Control Algorithm

July 2005

Summary: An opportunity arose to assist in the implementation of a PID control algorithm that was to form part of a focus mechanism on a commercial Thales product. The algorithm was designed and tuned in Simulink, before mapping to System Generator blocks and generating hardware automatically. The PID control algorithm presents some interesting challenges that are not usually present in the implementation of video processing algorithms.

Oliver Sims

EngD 3rd Year

Industrial Sponsor: Thales Optronics

1 Introduction

As part of a wider study into algorithm implementation tools and techniques, this report documents the implementation of an automatic control algorithm intended for the servomechanism in a commercial application. The algorithm is based on the common PID paradigm, whereby a summation of three terms, Proportional, Integral, and Derivative, provides the correct stimulus for the system under control given an input derived from the difference between the desired and actual state of the system. The algorithm is the standard method of performing automatic control, and can offer good performance control over a range of different processes.

System Generator is a tool that has been developed to assist users of the Simulink environment realise their designs in FPGA hardware. The majority of engineers designing in Simulink with FPGA target hardware will be working with signal processing algorithms, and the System Generator block libraries reflect this, with an emphasis on functions such as complex transforms (FFT, DCT), and buffer memories (FIFOs and RAMs) etc. However, Simulink (and its parent application Matlab) is also the application of choice for development of control algorithms. Control algorithms pose slightly different problems to standard signal processing algorithms: they are usually required to operate at much slower data rates (which can pose problems in an FPGA), and must have very little or ideally zero latency. FPGAs are probably not created with the PID control market in mind. However, in the sort of systems that Thales produce, which have stringent constraints both on physical size and power, it makes sense to combine functionality into the fewest number of devices possible. When an FPGA is also being used in the system for a variety of other purposes, it is an understandable aim to utilise spare logic and I/O capacity for the control algorithm.

A problem caused by the constraint for zero latency in the algorithm means that there will be resulting long combinatorial paths. This may cause problems later with meeting timing, though the low internal clock speeds may go some way to prevent this. Even generating low clock speeds can be troublesome, as in most systems the global clock will be running in the tens of megahertz, and the on-chip DCMs have a low bound in the hundreds of kilohertz, anything less than this will require

manual clock division. The sample period will also have an effect on coefficient representations, and so operating at a minimal clock frequency will help to improve internal precision levels.

2 Implementation details

The continuous form of the PID algorithm is as follows:

$$u(t) = K_c \left(e(t) + \frac{1}{T_i} \int_0^t e(t) dt + T_d \frac{de(t)}{dt} \right) \quad (1)$$

K_c is the gain of the controller, T_i is the integral time (or reset time), and T_d is the derivative time (rate time). $e(t)$ is the current error term of the system to be controlled. The discretized versions of the three terms will now be discussed.

2.1 Proportional term

The proportional term is equivalent to

$$u_P(t) = K_c e(t) \quad (2)$$

The discretized form is:

$$u_P(k) = K_c e(k) \quad (3)$$

The proportional term is thus implemented as a simple gain applied to all samples. The proportional coefficient represents the reciprocal of a measurement known as the proportional band, which is the range of deviation between the process variable and the set point that will produce a proportional response in the controller's output. A proportional term that is too large will cause the controller to oscillate, too small and the output may drift away from the set point.

2.2 Integral term

The integral term is defined as:

$$u_I(t) = \frac{K_c}{T_i} \int_0^t e(\tau) d\tau \quad (4)$$

Using a trapezoidal numerical approximation for the integration:

$$\int_0^t e(\tau) d\tau \approx T_s \sum_{i=0}^k \frac{e(i) + e(i-1)}{2} \quad (5)$$

The discrete integral term is thus:

$$u_I(k) = \frac{K_i T_s}{2} \sum_{i=0}^k e(i) + e(i-1) \quad (6)$$

where $K_i = \frac{K_c}{T_i}$ is the integral term coefficient.

Trapezoidal integration differs from standard rectangular integration as it averages the input value over the last two samples. This has the effect of helping to avoid sharp changes in integral action caused by a sudden change in the error signal, and will thus help to minimise overshoot.

The integral coefficient will determine the amount of integral action in the controller's output. Too high and the controller may become unstable and oscillate. Too low and the controller may have steady state error. Figure 1 shows the discrete integral calculation mapped to the System Generator library.

2.3 Derivative term

The derivative term is:

$$u_D(t) = K_c T_d \frac{de(t)}{dt} \quad (7)$$

A numerical approximation for the derivative term is given by:

$$\frac{de(t)}{dt} \approx \frac{e(t) - e(t-1)}{T_s} \quad (8)$$

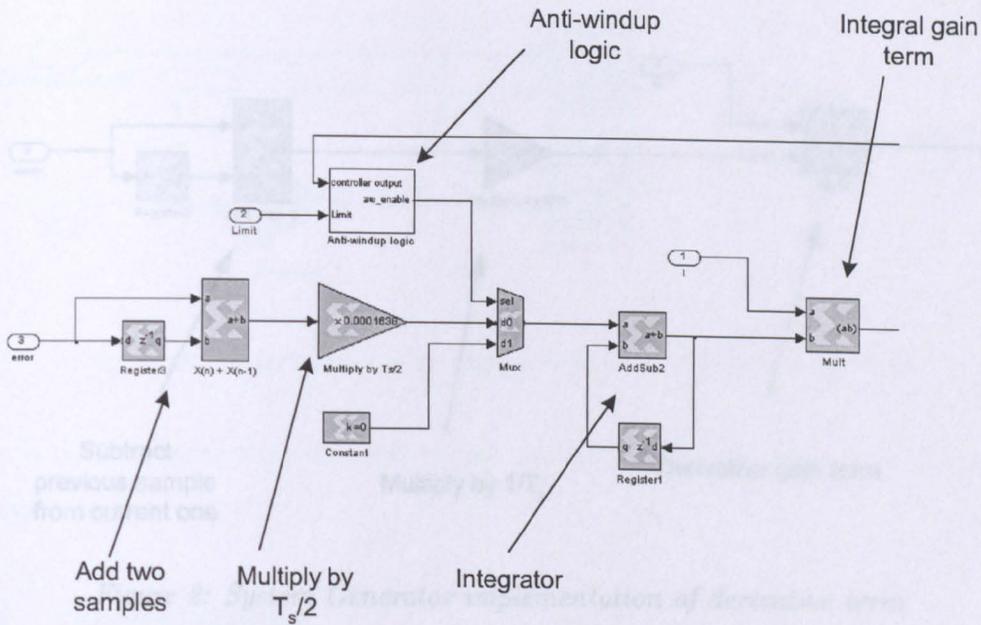


Figure 1: System Generator implementation of integral term

The discrete derivative term is thus:

$$u_D(k) = \frac{K_d}{T_s} [e(k) - e(k - 1)] \tag{9}$$

where $K_d = K_c T_d$ is the derivative term coefficient.

The input to the derivative term is calculated by subtracting the previous sample from the current one, before dividing by the sample period, hence calculating the gradient of the input function. The System Generator implementation is shown in figure 2.

3 Design features

3.1 Generic, parameterisable design

The coefficients for the three terms are all fed in as variables at the top level, which enables tuning of the controller without the having to regenerate the design files. Large scale changes in the coefficients may require the fixed point representations to be adjusted. Customisation and optimisation have not been extensively performed in

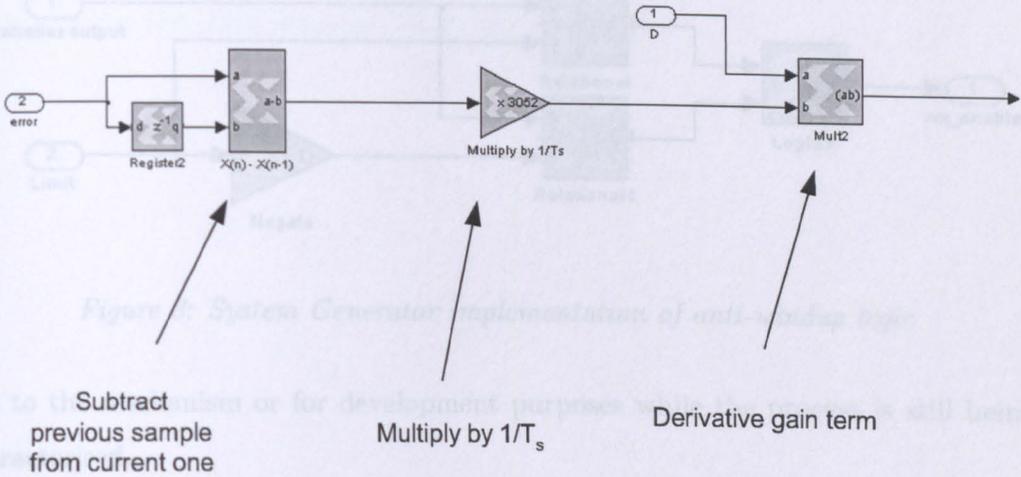


Figure 2: System Generator implementation of derivative term

order to keep the design as generic as possible, but savings are possible by trimming unnecessary precision from internal word lengths. If accuracy is an overriding constraint then the maximum number of bits can be used for fixed-point representations, in order to minimise quantisation error.

3.2 Anti-windup logic

The PID controller is the standard form with first order integrative and derivative terms. The integrator is augmented with a simple form of anti-windup logic, which enables a more effective recovery when the controller's output has become saturated. This is implemented by setting the input to the accumulator equal to zero when the output limit has been exceeded in either positive or the negative direction. This prevents the accumulator from increasing its internal value far outside the range of the controller's output, and works in conjunction with the output limiting logic.

3.3 Output-limiting

The output may be limited to less than its full range through setting a limit parameter at run-time. This will restrict the output through use of relational blocks. This provides the ability to reduce the maximum absolute output to prevent dam-

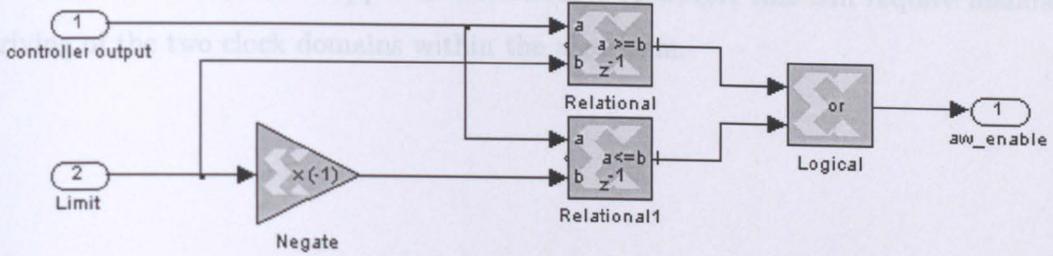


Figure 3: System Generator implementation of anti-windup logic

age to the mechanism or for development purposes while the process is still being characterised.

3.4 Downsampling

To reduce the internal sample rate of the controller the input data stream is downsampled. To observe the constraint of zero latency this is carried out without incurring register delays, using the purely combinatorial form of the Xilinx downsample core. Using a smaller internal sample rate (i.e. a larger sample period) helps ensure that fixed-point representations of internal data on the Integral/Derivative paths (where the sample period is used as a multiplier/divisor) can be represented using a smaller number of bits.

4 Implementation results

The design was synthesised with Precision Synthesis 2005a.69 and implemented with Xilinx ISE 7.1i, and was found to utilise 651 Xilinx Virtex slices. This is approximately 8 per cent of a medium sized Virtex-2, or 4 per cent of a medium Virtex-2 Pro. Throughput issues are typically not a problem, and low internal clock speeds mean that timing constraints are easily met within the control loop. It should be noted that System Generator will usually generate a "clock wrapper" to contain and drive the algorithm with the correct control signals, and the presence of a downsample block within the algorithm will cause the tools to infer the need for a DCM. Since the PID algorithm will usually be used within a larger system, also presumably with

a DCM, often the clock wrapper is unnecessary. However, this will require manual driving of the two clock domains within the algorithm.

Appendix E: Implementation of Richardson-Lucy Deconvolution

Implementation of Richardson-Lucy Deconvolution

September 2005

Summary: The Richardson-Lucy deconvolution algorithm is an important method in removing blur from images, by using statistical estimation techniques based on Bayesian probability theory. A Matlab version of the algorithm was available within Thales, and this presented an opportunity to investigate the issues surrounding implementation of a mathematically quite complex algorithm in FPGA. The algorithm is an iterative process that includes some mathematical operators that are not straightforward to implement in hardware, including several large 2D convolutions. These solutions found to these problems and the resulting design is presented in the following report.

Oliver Sims

EngD 3rd Year

Industrial Sponsor: Thales Optronics

1 Introduction

Observing and recording a scene is, in practise, an imperfect process. The measurement of physical quantities is always limited by the inherent characteristics of the measurement device, for instance finite resolution, or flaws in the imaging components. Naturally occurring phenomena such as spreading, smearing, and blurring also impact the accuracy of the observation. All sources of degradations introduced during the observation process reduce sharpness and obscure detail in the final image. In most cases the processes that impair measurement of physical quantities may be represented mathematically by a convolution operation, and the problem of recovering the true data given a limited observation is the justification for efficient and robust deconvolution techniques. Many of these techniques were developed in the field of astronomy, but have since spread to such diverse areas as medical tomography, seismology, spectroscopy, magnetic resonance imaging, and others [1]. Deconvolution techniques are not confined to problems in one- and two- dimensions, and application to higher dimensional problems involving reconstruction from projections have also been reported. The ideal solution is the recovery of the same data that would be observed by a hypothetical, perfectly resolving instrument [2]. The degradation process may be characterised as:

$$I_n = \sum_{m=-\infty}^{\infty} S_{n-m} O_m \quad (1)$$

where I represents the image, S is a Point Spread Function (PSF), and O is the object. The PSF represents the spreading or smearing of a single point source and in a perfectly resolving instrument would be equivalent to a two-dimensional Dirac function [2], so that I becomes equal to O . A PSF with a wide spread will result in a severely blurred image. In other cases the PSF may be quite compact with a sharp peak, and this may lead to a situation where the primary concern is reduction in dynamic range and sensitivity, rather than loss of spatial resolution. The application presented here describes a deconvolution process where the PSF is known a priori, having been determined empirically or through modelling of the factors influencing the observation, but deconvolution with an unknown PSF is also possible. This is

then known as blind deconvolution, which uses a process of revising estimates to the PSF; an overview of blind deconvolution techniques may be found in [3].

The degradation that occurs during the observation process may be thought to correspond to the null-space of the observing system [4], and the resulting data as incomplete (in the mathematical sense). The task is therefore an inverse problem that is ill posed, and as such cannot be solved directly. Most techniques rely on statistical estimation methods to generate best-fit solutions. The Richardson-Lucy algorithm [5], [6] is one such method that came to prominence during the early 1990s, when it emerged as the de-facto standard for restoration of images from the Hubble Space Telescope, before an aberration present in the imaging mirror was corrected in a repair mission three years after launch [7]. The algorithm was developed independently by Richardson and Lucy in the 1970s, and uses Bayesian probability theory to seek the image of highest probability given the data and the PSF [2].

Use of a general-purpose processor to implement a real-time implementation of the Richardson-Lucy algorithm is unfeasible, due to the performance constraints implied by working with real-time video signals. FPGAs are an ideal medium for the implementation of complex image processing algorithms due to their ability to realise massive parallelism. In this paper an FPGA implementation of the Richardson-Lucy algorithm will be discussed. The paper is organised as follows: Section 2 will introduce the algorithm. Section 3 describes a software implementation developed in Matlab and efforts to improve the speed of convergence. Section 4 describes some of the issues faced and methods used to overcome these issues in translating the algorithm into a hardware description. Section 5 provides results of the implementation, and some conclusions are drawn in Section 6.

2 The Richardson-Lucy algorithm

The following mathematical description of the Richardson-Lucy algorithm uses the notation used by Lucy in his original paper. Unlike Lucy, Richardson uses a 2D notation from the start, but the link to the original convolution operation is slightly

less clear.

The convolution of a scene with a PSF may be modelled in the 1D sense as follows:

$$\phi(x) = \int \Psi(\xi) P(x|\xi) d\xi \quad (2)$$

where $\phi(x)$ is the blurred image, $\Psi(\xi)$ is the true image and $P(x|\xi)$ is the PSF. It is important to note at this stage that both images and the PSF may be considered probability distributions, with $P(x|\xi)$ being the probability that x' will fall in the interval $(x, x + dx)$ when it is known that $\xi' = \xi$. The PSF is thus a probability distribution describing the potential destinations of a single unit of energy originating from a given point in the input.

Define $Q(\xi|x)$ to be the probability that ξ' comes from the interval $(\xi, \xi + d\xi)$ when it is known that $x' = x$. This is in effect an inverse PSF.

From Bayes's Theorem:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)} \quad (3)$$

we can infer:

$$Q(\xi|x) = \frac{P(x|\xi) \Psi(\xi)}{\phi(x)} \quad (4)$$

Hence

$$P(x|\xi) \Psi(\xi) = \phi(x) Q(\xi|x) \quad (5)$$

Since $P(x|\xi)$ is a probability distribution, it follows that:

$$\Psi(\xi) = \int \phi(x) Q(\xi|x) dx \quad (6)$$

This has the appearance of being the inverse of the original equation, but it cannot be used in this form directly since $Q(\xi|x)$ is dependent on Ψ , the true image. We can however use this to formulate an iterative scheme for generating estimates to Ψ^r .

If $\tilde{\phi}$ is the original observation, and Ψ^r is the r -th estimate of the true image, the $(r + 1)$ -th estimate is:

$$\Psi^{r+1}(\xi) = \int \tilde{\phi}(x) Q^r(\xi|x) dx \quad (7)$$

Program 1 Matlab implementation of accelerated Richardson-Lucy algorithm

```

for i = 1 : fctrl
    implied_observation = filter2(PSF,reconstructed);
    factor = observation./(implied_observation + 1e-12);
    correction = filter2(PSF,factor);
    reconstructed = reconstructed.*(correction.^pwr);
    reconstructed = min(ones(a,b),reconstructed);
end

```

where

$$Q^r(\xi|x) = \frac{P(x|\xi)\Psi^r(\xi)}{\phi^r(x)} \quad (8)$$

and

$$\phi^r(x) = \int \Psi^r(\xi) P(x|\xi) d\xi \quad (9)$$

so that

$$\Psi^{r+1}(\xi) = \Psi^r(\xi) \int \frac{\tilde{\phi}(x)}{\phi^r(x)} P(x|\xi) dx \quad (10)$$

This shows that the iterative scheme will converge if

$$\tilde{\phi} = \phi = \int \Psi^r(\xi) P(x|\xi) d\xi \quad (11)$$

i.e. when the original observation equals the current estimate convolved with the PSF. This may be viewed as a goodness-of-fit measure.

The application of this algorithm to images may be represented as follows, assuming a symmetric point spread function:

$$I^{r+1} = I^r \times \left(\text{PSF} * \frac{\text{observation}}{\text{PSF} * I^r} \right) \quad (12)$$

3 Software implementation

A Matlab implementation of the iterative scheme of 12 was the starting point for hardware development [8]. Due to the high-level nature of the Matlab command environment the main loop of the algorithm could be represented by five instructions.

The main loop body, with a slight modification to be discussed below, is shown in program 1. In the literature there is a lack of general consensus regarding the number

of iterations the algorithm will need before it converges. Some results, particularly those based on applications in astronomy, state in the region of 20-30 [1], though this could be particular to the nature of astronomical images and the Poisson noise model used in that field. Other publications highlight the semi-converging nature of the algorithm; i.e. that it approaches a solution, typically before 10 iterations, and then diverges again [9]. To improve the rate at which the algorithm approaches a solution a further parameter may be introduced, denoted in the following equation:

$$I^{r+1} = I^r \times \left(\text{PSF} * \frac{\text{observation}}{\text{PSF} * I^r} \right)^\beta \quad (13)$$

Typically $1 \leq \beta \leq 3$, and is shown in [10] to reduce the required number of iterations by a factor of β . Values greater than three tend to introduce instability and cause the algorithm to fail to converge. The eventual divergence of the algorithm may be attributed to the effects of noise, and excessive iterations will cause deterioration in image quality as the algorithm tries to fit the estimation too closely to the noisy input data. This effectively causes the algorithm to fit to the noise in the image rather than the useful data. The algorithm will thus tend to amplify any noise bumps, as the smoothing effect of the PSF means that noise errors in the observation can only be approximated by large noise spikes in the original [7]. In software implementations of Richardson-Lucy the characteristic of semi-convergence is unimportant, as a trial and error approach may be taken to find the best stopping point. In a real-time hardware implementation this approach is less realistic, especially with overriding constraints on logic usage, latency, and throughput. Some experimentation was therefore necessary to determine the optimum number of iterations and degree of acceleration, whilst keeping the amount of hardware required to a minimum.

Five test images were chosen that were deemed to represent a variety of situations, with varying brightness and contrast levels. The images were treated with zero-mean Gaussian noise, before being blurred with a suitable two-dimensional PSF. The Matlab implementation of Richardson-Lucy was then applied to each image individually. In order to measure the performance of the deconvolution process an error matrix was calculated after each iteration by subtracting the current image from

the original reference image. The edges of both images were discounted to remove the edge-effects caused by convolution (the number of lines removed from each edge is equivalent to $\frac{N-1}{2}$, where N is the width of the convolution kernel). A performance metric was then generated by finding the mean squared error over all values in the error matrix. This value is subsequently normalised, so that the initial blurred image has an error value equal to unity, and a figure of zero represents a perfect reproduction of the original image before blurring. Consequently, reductions of the error value demonstrate improvements in image quality. A graph showing average results over the five images is shown in figure 1, where the solid line represents the Richardson-Lucy algorithm with no acceleration ($\beta = 1$), and the dashed line the accelerated algorithm ($\beta = 2$). The improvement caused by acceleration is clearly visible: the minimum of the accelerated algorithm is reached after a single iteration, compared with two or three iterations for the standard algorithm. It should also be noted that, although slower, the un-accelerated algorithm achieves a marginally lower minimum value. This may be explained by considering the Richardson-Lucy process to be an implementation of a steepest-descent method, with parameter β equivalent to the step size [11]. In this sense the un-accelerated algorithm may find a smaller error value by taking smaller steps towards the minima, and thus decreasing the possibility of overshoot. This raises the possibility of using the accelerated algorithm for the initial iteration, followed by the standard algorithm for subsequent iterations. figure 2 shows the average results over the five test images when this method is used, and indicates that the variable exponent method finds lower minima than either the standard or accelerated versions of the algorithm when the exponent is constant. The minimum is found after two iterations in four of the five test cases.

4 Hardware implementation

The following section describes some of the design challenges faced during the implementation of the algorithm in FPGA hardware. The system was developed using the Xilinx System Generator tool for Simulink, and was carried out as part of a wider study into algorithm implementation tools and techniques. The diagram in figure 3

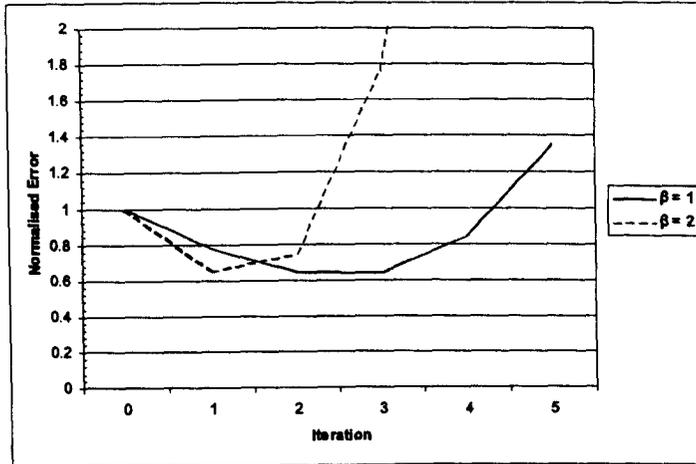


Figure 1: Comparison of standard and accelerated algorithm

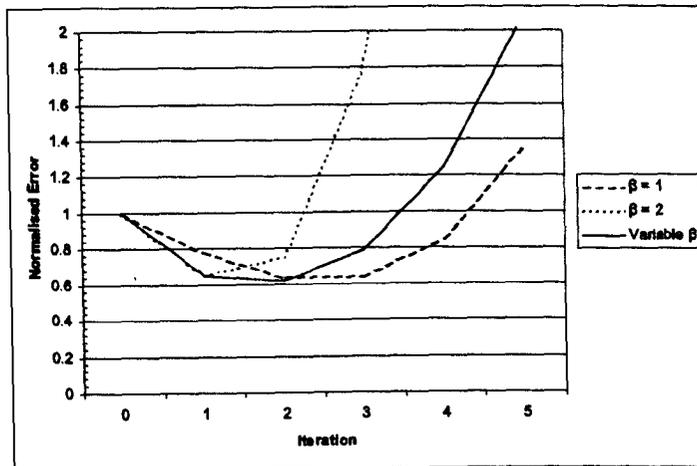


Figure 2: Improvement gained by a variable acceleration exponent

shows a top-level block diagram representing a single iteration of the Matlab code in program 1. The delay line running parallel to the main computational pipeline is necessary to align the observation data with the correction factor being computed.

4.1 Two-dimensional filtering

Two of the stages shown in the single iteration of figure 3 require a two-dimensional convolution to be performed. In a single dimension FIR filters are generalised structures commonly used to carry out this task. The extension to two dimensions is however not straightforward, as a direct implementation quickly becomes expensive

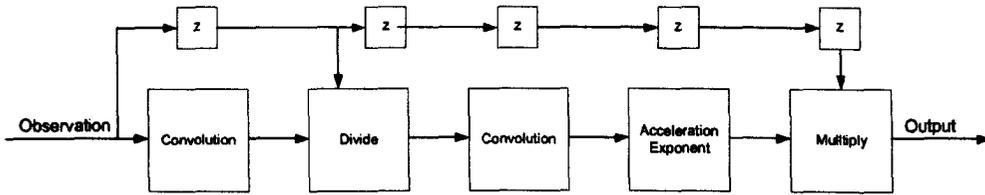


Figure 3: Block diagram depicting a single iteration of the Richardson-Lucy method

in hardware terms as the size of the kernel increases. In some circumstances, these difficulties may be solved by reducing the two dimensional kernel into a combination of two single dimensional convolutions, which may then be implemented by two FIR filters. A kernel that can be decomposed in this way is said to be separable. Separable filter kernels are a special case of general two-dimensional filter kernels where the kernel matrix can be expressed purely as a combination of two vectors. Unfortunately, separable kernels are limited in their applicability, and in most circumstances a kernel will not be separable, largely due to the difficulties involved in factoring polynomials in two independent variables [12]. This often prevents a simple realisation using cascade or parallel structures. Despite this there are alternative methods of decomposing the kernel, based on the fact that a parallel connection of separable filters results in a non-separable response [12]. This raises the possibility of using sets of easily implementable orthogonal one-dimensional filters in parallel to create the desired two-dimensional filter, as shown in figure 4. In this way the response of a non-separable filter may be accurately reproduced as a combination of separable filters, each of which is formed by two cascaded 1D convolutions that act individually in the horizontal and vertical directions. It can be shown that for square kernels when the required number of parallel filter sets is less than half the order of the filter, this structure will still utilise less hardware than a direct implementation of the 2D convolution. In some cases, as demonstrated later, this condition can be relaxed further.

The extraction of separable filters is based on a technique by Treitel & Shanks [13], who use the term multistage separable planar filters. Their method utilises a technique known elsewhere as the Singular Value Decomposition (SVD) [14], and has previously been shown to be successful in the implementation of arbitrary two-

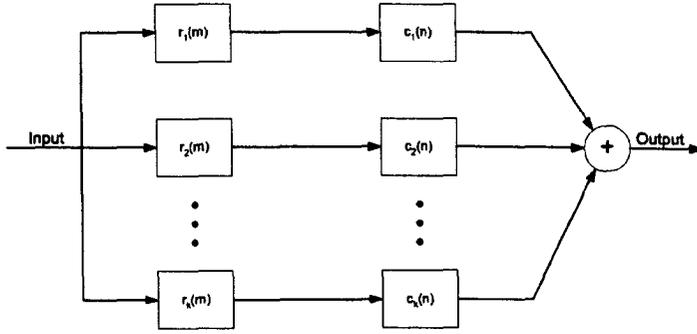


Figure 4: Summation of separable filters to produce a non-separable response

dimensional filters [15]. The SVD of a matrix, X , is given as:

$$X_{m \times n} = U_{m \times n} \Sigma_{m \times n} V_{m \times n}^T \quad (14)$$

(where superscript T denotes the matrix transpose operation) and is found by calculating the eigenvalues and associated eigenvectors of the square matrices $Q = XX^T$ and $S = X^T X$. It is demonstrated in [13] that the eigenvalues for both Q and S will always be the same. The square root of these eigenvalues are known as the singular values of X , and are listed in the diagonal matrix S . The associated eigenvectors from Q and S form the columns in the orthonormal matrices U and V respectively. The rank of S , which is equal to the number of non-zero elements, determines the number of separable filters required to exactly reproduce the original kernel. Since the values in S are the eigenvalues of Q and S , it follows that the rank of S will be equivalent to the number of linearly independent columns in the original matrix X . This can be particularly advantageous in image processing applications, where filter kernels with a linear phase response are often a necessity, and the resulting kernel is symmetrical. Thus an N^2 kernel will have $\frac{N+1}{2}$ linearly dependent rows and columns. When the SVD is calculated the eigenvalues corresponding to these rows/columns vanish, meaning that any linear phase kernel can be exactly reproduced with, at most, $\frac{N+1}{2}$ separable filters.

Further savings can be made by making approximations to the original filter design rather than reproducing it directly. By convention the eigenvalues in S , known as singular values, are listed in descending order, which suggests that the corresponding

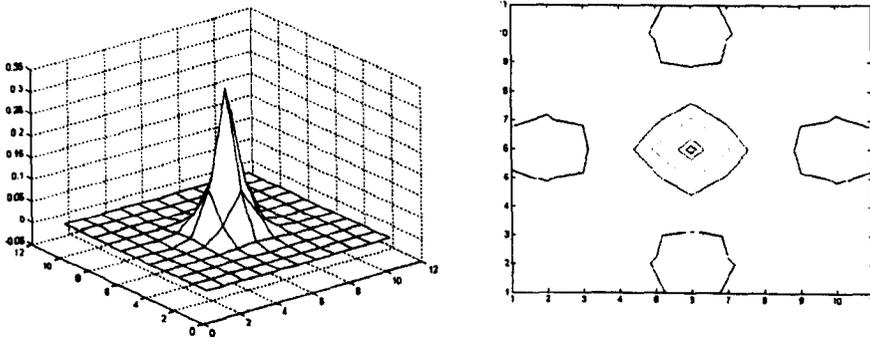
columns of U and V are of decreasing importance to the final reproduction. Using a reduced number of singular values will produce an approximation to the original matrix X , with an error that is equal to the ratio of the sum of the discarded eigenvalues to the sum of all eigenvalues in S [13]. Thus:

$$\varepsilon_k = 1 - \frac{\sigma_1 + \sigma_2 + \dots + \sigma_k}{\sigma_1 + \sigma_2 + \dots + \sigma_n} \quad (15)$$

gives the error ε_k from using k singular values from a maximum of n , where $k \leq n$, and σ denotes the individual singular values in the matrix Σ . When all singular values are used, $k = n$ and the error ε is reduced to zero. The aim is to balance the need to keep the error matrix ε to a minimum, whilst reducing the necessary hardware as far as possible. Since each additional singular value will result in another separable filter pair in the final implementation, potentially large savings in hardware may be achieved by keeping the number of singular values included in the design to a minimum.

This may be best demonstrated with a real example. The following work uses the empirically obtained PSF that will be the focus of the subsequent implementation of the Richardson-Lucy algorithm. The PSF, shown in figure 5 using both surface and contour plots, is equivalent to an 11×11 normalised filter kernel that exhibits the linear phase characteristics discussed earlier. Application of the SVD to this kernel results in six eigenvalues (i.e. $\text{rank}(\Sigma) = 6$), and thus, to exactly reproduce this response using separable filters, six filter pairs would be needed. Table 1 shows how successive singular values become less significant, and the resulting error term decreases with each additional stage. The results indicate that in this instance two filter stages will reproduce the original kernel to $> 99\%$ accuracy, and any more than four is almost certainly unnecessary in most applications of this particular kernel.

Once the separable filter coefficients have been determined there are still a number of design decisions regarding the hardware structure of the filter. FIR filters may be implemented in an FPGA device in numerous ways, depending on whether the overriding constraint is in terms of throughput, logic area, or some other factor. Parallel FIR architectures, such as direct and transpose forms, use the greatest

Figure 5: 11×11 PSF

k	Singular value σ_k	Normalised contribution $\frac{\sigma_k}{\sum_{i=1}^k \sigma_i}$	Accuracy $\frac{\sum_{i=1}^k \sigma_i}{\sum_{i=1}^n \sigma_i}$	Normalised error $1 - \frac{\sum_{i=1}^k \sigma_i}{\sum_{i=1}^n \sigma_i}$
1	0.400661	0.972346	97.2346%	0.0277
2	0.009133	0.022164	99.4510%	0.0055
3	0.001951	0.004736	99.9246%	0.0008
4	0.000285	0.000692	99.9938%	6.2×10^{-5}
5	2.44×10^{-5}	0.000059	99.9997%	2.7×10^{-6}
6	1.10×10^{-6}	0.000003	100.0000%	0.0

Table 1: Significance of the n singular values ($n=6$) of the filter kernel shown in Figure 4.3

amount of logic resources but have high throughput and low latency. Alternatively Multiply and Accumulate (MAC) FIR filters may use just a single MAC element and so are highly efficient in terms of hardware, but must operate at a multiple of the data rate and will thus reduce throughput in most applications. A trade-off solution in FPGAs also exists in the form of a distributed arithmetic filter, which is nearly as hardware efficient as a MAC FIR but maintains throughput at higher filter lengths. FIR filtering in the vertical direction requires slightly altered architectures to allow the pixel data, which is normally streamed line by line, to be filtered with corresponding data that are vertically adjacent in the image. To achieve this line buffers must be inserted between multiplier elements that delay each pixel by the width of one line. Consequently a vertical filter will require sufficient storage for the

number of elements in a single row multiplied by the number of filter coefficients. For multistage filters such as those developed here this becomes an unrealistic amount of storage, and to save memory the filters were designed to share line buffers, as shown in figure 6 with two filter stages present. The multiplier structures inside the vertical FIR subsystems are all transpose direct-form parallel filters, an inherently pipelined architecture that allows high clock speeds and thus high throughput. MAC FIRS are not easily implemented for vertical filtering applications due to the need for line buffers.

4.2 Other stages

The divide operation in the deconvolution algorithm determines a correction factor from the ratio of the observed image to the image obtained by filtering the current best guess. System Generator does not provide a native divide operation, but one may be implemented using the black box function to import soft IP cores, in this case the Pipelined Divider v3.0 core provided by Xilinx and customised through the Core Generator tool. Importing the VHDL and EDIF netlist produced by Core Generator into the black box module results in the block format shown in figure 7. The core was parameterised to provide a quotient output every clock cycle, with a fractional remainder that was subsequently concatenated to the quotient. The resulting latency of the core is proportional to the word lengths of the output data, in this instance equal to the number of quotient bits + number of remainder bits + 4. Later stages of the deconvolution algorithm are both multiply operations and are easily implemented with standard System Generator blocks. The acceleration exponent (with a value of two) squares the current value by feeding it through both inputs of a multiplier simultaneously. The second multiplier acts on the squared correction factor and the delayed observation data. The multiplier stages are shown in figure 8. All stages were connected together in the format shown in the block diagram of figure 3. The design is fully pipelined throughout to enable high clock speeds, and can produce an output value every cycle after an initial latency proportional to the image width.

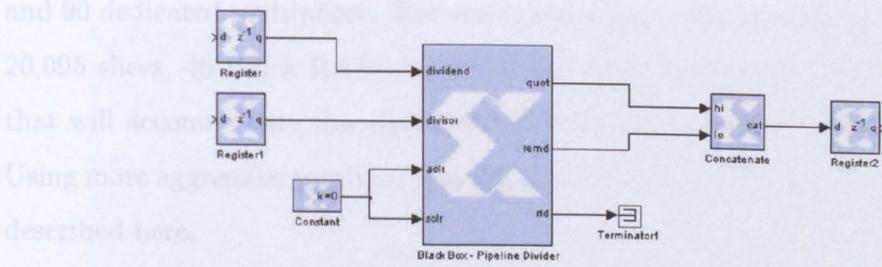


Figure 7: Pipelined divider in System Generator using a Black Box module and a Core Generator netlist

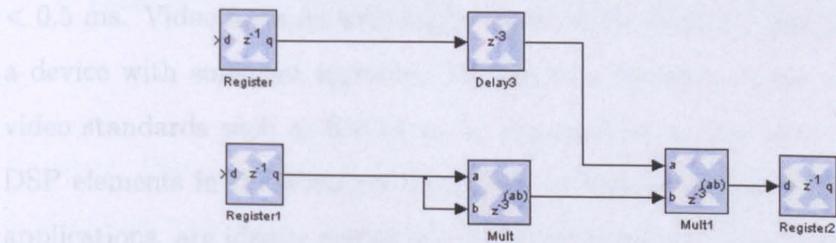


Figure 8: Output stage of a single iteration of the accelerated deconvolution algorithm

5 Results

Because the design is extensible, the figures presented here are specific to this implementation with an 11×11 PSF and designed for an 8-bit greyscale VGA image of 640×480 elements. The two dimensional filters are implemented with two separable filter stages, which approximates the true kernel to 99%. The parameters that will influence the amount of logic usage are the size of the input image, the dimensions of the PSF, the number of iterations that are required, the desired accuracy of the filter stages (and hence the number of separable filters required to reproduce the kernel), and the structure of the horizontal FIRs. Using single MAC FIRs will significantly reduce the amount of logic required but will also reduce the throughput accordingly. A larger PSF will require an increased number of line buffers for the two-dimensional filters, of width proportional to the width of the input image.

The design was synthesised using Precision Synthesis 2005a.69 before being implemented with ISE6.3i targeting the Virtex-2 family of devices. A single iteration of the algorithm uses approximately 12,000 slices of the device, 20 Block RAMs,

and 90 dedicated multipliers. The variable exponent format with two iterations uses 20,095 slices, 40 Block RAMs, and 179 dedicated multipliers. The smallest device that will accommodate the algorithm in this configuration is a Virtex 2 XC2VP50. Using more aggressive synthesis and place and route settings may improve the figures described here.

The maximum clock speed suggested by ISE with minimal synthesis/place and route effort is 63 MHz. A frequency of just 20MHz will allow video in 8-bit greyscale VGA format to be processed in real time at 60 frames per second, with a latency < 0.5 ms. Video formats with higher bandwidth could be processed comfortably in a device with sufficient memory. Moving to a Virtex-4 device could allow modern video standards such as SXGA to be deconvolved in real time. The abundance of DSP elements in the Virtex-4 SX family, intended specifically for signal processing applications, are ideally suited to the filtering stages involved in this application.

Optimisations in the fixed-point word lengths used throughout the implementation may also produce valuable savings in slice usage. The current implementation uses a minimum of 12 binary places for all internal fixed-point representations. Optimisation of internal word lengths was not performed here as it was desired to leave the implementation as general as possible.

Sample images have been produced for comparison: on the left of figure 9 is an aerial image that has been blurred by the 11×11 PSF; the same image after two iterations of deconvolution is shown to the right. The increase in spatial resolution is noticeable in several areas, and edges are generally sharper.

6 Conclusion

An extensible implementation of the Richardson-Lucy deconvolution algorithm has been presented, that will perform full scene deconvolution in real time. The design is modular, to allow as many iterations as necessary or hardware will allow, with no significant changes to the design of each module. Hardware efficiency is promoted throughout, particularly in terms of the two-dimensional convolutions, where multi-stage separable filters have been used to reproduce the arbitrary PSF to a custom

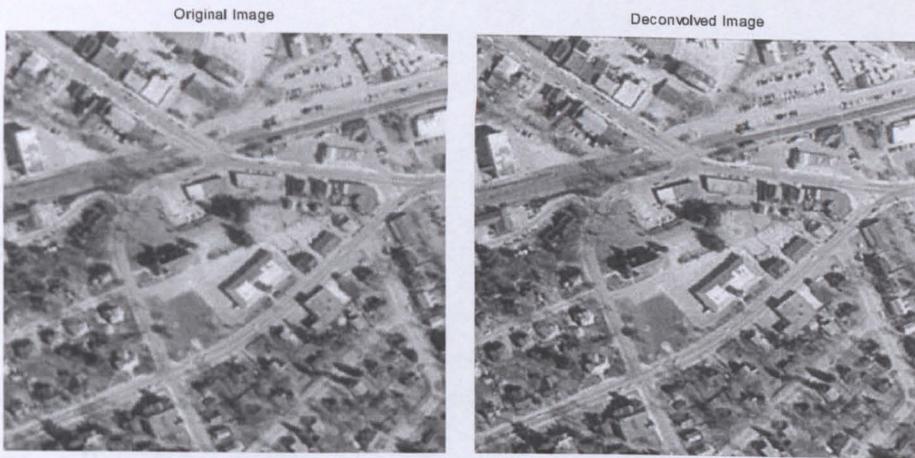


Figure 9: Example blurred and deconvolved images

accuracy. This approach allows a trade-off to take place between accuracy of response and hardware utilisation. In order to improve the performance of the algorithm over a limited number of iterations, a modified acceleration technique has been incorporated. The variable exponent technique further increases the probability of finding a desirable solution when there are hardware limitations. The design is fully pipelined and will produce an output element every clock cycle. Initial results show that a design offering two iterations of accelerated deconvolution may achieve a throughput of over 60 million pixels per second using Virtex-2 hardware.

There are a number of directions future work could take to improve on the results shown here. Investigations into the effects on the output image of reducing the accuracy of the filtering stages may allow further hardware savings to be made. In particular it is unknown whether using reduced accuracy when implementing the filtering stages in the earlier iterations will severely impair the deconvolution process, and it is possible that, during this initial search of the solution space, a more heavy handed approach may be adopted.

[8] O. Thomas,

Richardson-Lucy

[9] M. Bertolo and R.

imaging nature

References

- [1] R. Molina, J. Nunez, F. J. Cortijo, and J. Mateos, "Image restoration in astronomy: a Bayesian perspective," *IEEE Signal Processing Magazine*, vol. 18, no. 2, pp. 11–29, Mar. 2001.
- [2] P. A. Jansson, *Deconvolution of Images and Spectra*. Academic Press, 1996.
- [3] D. Kundur and D. Hatzinakos, "Blind image deconvolution," *IEEE Signal Processing Magazine*, vol. 13, no. 3, pp. 43–64, May 1996.
- [4] K. M. Hanson, "Bayesian and related methods in image reconstruction from incomplete data," in *Image recovery - theory and application*, H. Stark, Ed. Academic Press, 1987.
- [5] L. B. Lucy, "An iterative technique for the rectification of observed distributions," *Astronomical Journal*, vol. 79, no. 6, pp. 745–754, June 1974.
- [6] W. H. Richardson, "Bayesian-based iterative method of image restoration," *Journal of the Optical Society of America*, vol. 62, no. 1, pp. 55–59, Jan. 1972.
- [7] R. J. Hanisch, "Deconvolution of Hubble space telescope images and spectra," in *Deconvolution of images and spectra*, P. A. Jansson, Ed. Academic Press, 1997.
- [8] O. Thomas, "Product enhancement study - Matlab implementation of Richardson-Lucy deconvolution," Thales Optronics, Tech. Rep., 2002.
- [9] M. Bertero and P. Boccacci, "Image deconvolution," in *From cells to proteins: imaging nature across dimensions. Proceedings of the NATO Advanced Study*

- Institute*, V. Evangelista, L. Barsanti, V. Passarell, and P. Gualtieri, Eds., Pisa, Italy, Sept. 2004.
- [10] H. Lanteri, M. Roche, O. Cuevas, and C. Aime, "A general method to devise maximum-likelihood signal restoration multiplicative algorithms with non-negativity constraints," *Signal Processing*, vol. 81, no. 5, pp. 945–974, 2001.
- [11] T. S. Zaccheo and R. A. Gonsalves, "Iterative maximum likelihood estimators for positively constrained objects," *Journal of the Optical Society of America*, vol. 13, no. 2, Feb. 1996.
- [12] D. Dudgeon and R. M. Mersereau, *Multidimensional digital signal processing*. Prentice-Hall, 1984.
- [13] S. Treitel and J. L. Shanks, "The design of multistage separable planar filters," *IEEE Transactions on Geoscience Electronics*, vol. 9, no. 1, pp. 10–27, Jan. 1971.
- [14] V. Klema and A. Laub, "The singular value decomposition: its computation and some applications," *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, Apr. 1980.
- [15] M. Andrews, "Architectures for generalized 2D FIR filtering using separable filter structures," in *Acoustics, Speech, and Signal Processing, 1999. ICASSP '99. Proceedings., 1999 IEEE International Conference on*, vol. 4, Phoenix, AZ, Mar. 15–19, 1999, pp. 2215–2218.

Appendix F: Implementation of Pyramidal Image Fusion

Implementation of Pyramidal Image Fusion

April 2006

Summary: This is a modified version of a paper submitted to the FPL2006 conference, describing an implementation of image fusion using a multiscale decomposition known as gradient image pyramids. The method presented here uses multiple sample rates to allow the different frequency ranges of the decomposed image to be processed concurrently. The resulting design exhibited a speedup of over 100 times compared to an equivalent software implementation. The design has since been found to be erroneous and may produce incorrect output data. Some discussion of the problems will be given and an estimate of a correct implementation is provided.

Oliver Sims

EngD 4th Year

Industrial Sponsor: Thales Optronics

1 Introduction

Image fusion allows multiple observations of a scene to be combined, in order to increase the information content presented in a single image, and make the image more effective for its intended application. The images to be fused may originate from a single sensor, perhaps taken with different points of focus, or from multiple sensors that are sensitive to different spectral regions. The resulting composite image may subsequently be used by a human observer or, increasingly, used in machine vision applications. Image fusion has been used widely in medical, manufacturing, military, and security applications, amongst others [1]. One modern example where image fusion techniques are proving useful is in the detection of concealed weapons by using a composite of thermal and visible-range observations [2].

There are several methods of performing image fusion, with a successful implementation being one that retains all useful information from the source images into a single composite image, without introducing artefacts. Basic methods take no account of the image content and perform simple merging of the image data, for instance averaging. More sophisticated methods work at a higher level by identifying detail in the source images and using a selection process to determine the elements that will be used in the final composite.

Several key methods in image fusion rely on the multiscale image pyramid [3]. Image pyramids are a decomposition of a single image into a series of images of varying resolutions, with each image containing data representative of detail at a particular scale. The advantages of a multiscale representation lie in its localisation in both spatial and spatial-frequency domains. It is thought that such a representation has similarities with the operation of the human eye [3]. For a more rigorous explanation of image pyramids see papers by Burt [4] [5], who developed much of the theory describing image pyramids such as the Gaussian and Laplacian pyramids used in image fusion. An example of an image pyramid may be seen in figure 1.

The fusion algorithm that will be presented here uses pyramids that have been further decomposed into orientation specific pyramids [5], [6], [7]. It uses simple edge filters (gradient filters) to identify details in the source images along four orientations. These edges are compared, and the most useful features are selected

according to some measure of saliency and then carried forward into the composite image. Using gradient pyramids has been found to reduce the artefacts that fusion with straightforward Laplacian pyramids can introduce [5].

The pattern-selective fusion algorithm is complicated and requires thousands of calculations to be performed in order to produce a single output image. For this reason a microprocessor implementation is inherently slow, and real-time processing unfeasible. However, like many image processing algorithms, there are opportunities to exploit parallelism in the algorithm's operation that make an FPGA implementation an attractive option.

This paper describes an implementation of a pattern-selective fusion algorithm on a single Virtex-2 FPGA. The implementation uses several novel approaches to enable a design that can fuse dual greyscale VGA images in real-time. Extension of the methods presented here would permit fusion of multiple images without major design modifications. Note that the implementation described here uses a method of generating image pyramids that exhibits increased levels of blur and is liable to cause aliasing. It has been used here for image fusion with passable results, but in general the design is incorrect and should not be used. The implications of this approach and a comparison with the correct method of generating image pyramids will be described in a later section.

It should also be noted that image registration is not covered in this work, and all source images are assumed to be pre-aligned. If the sensors are not perfectly aligned this may be achieved using automatic registration techniques, which are beyond the scope of this paper.

The paper is organised as follows. Section 2 gives an outline of image pyramids and their construction. Section 3 looks at the processing of these pyramids to extract the salient features, and the subsequent formation of a composite image. Section 4 briefly discusses the inverse pyramid transform; section 5 provides implementation results for the design, and includes some quantification of the speedup achieved by using FPGAs compared to software methods. Section 6 discusses the method of generating pyramids that has been used here, and produces a comparison with traditional methods. Finally, section 7 concludes the paper.



Figure 1: Gaussian pyramid of the well-known image "Lena".

2 Pyramid generation

The Gaussian or low pass pyramid and Laplacian bandpass pyramids were introduced by Burt in 1983 [4]. These methods have since been used in a wide variety of applications besides fusion, and were a precursor to the development of more general multiresolution methods, in particular the study of wavelets.

The image to be decomposed forms the bottom level of a notional pyramid. Standard methods generate higher levels of the pyramid by low-pass filtering and then two-dimensionally subsampling (by a factor of two) the pyramid level beneath it. The act of low-pass filtering reduces the band limit by one octave, and hence, according to the sampling theorem, subsampling can take place without any loss of information. In reality the generating filter is not "ideal", which means that the following subsampling may result in some aliasing; however these effects are usually slight and can be disregarded for these purposes [8]. The low-pass filter is usually chosen to be a 5×5 Gaussian, which has the added advantage of being separable.

Hence, the pyramid generation process can be described as:

$$G_k(i, j) = \sum_{m=-2}^2 \sum_{n=-2}^2 w(m, n) G_{k-1}(2i + m, 2j + n) \quad (1)$$

for $k=1, \dots, N$; $G_0 \equiv I$, the original image; w is the filter kernel.

The following implementation is based on Matlab code that differs from equation 1 by performing the subsampling before filtering at each stage. This implementation thus has no means of preventing aliasing, and higher levels of the pyramid are



Figure 2: Image pyramid of the Lena image where subsampling is performed before filtering at each stage.

likely to become distorted. Subsampling before filtering also exaggerates the effect of the filtering process and thus causes pyramids to be produced with an increased level of blur. Figure 2 shows the pyramid of figure 1 when subsampling is performed before filtering. With this particular image it is difficult to discern any artefacts due to spatial aliasing; however increased blur is clearly noticeable at the higher pyramid levels. Because of these effects, subsampling should not be performed before filtering in general applications of image pyramids. The hardware implications of performing the process in this way and the effect it has on the fused image will be discussed in a later section.

The process of equation 1 is usually referred to as *REDUCE*, when considering the 2D image as a whole:

$$G_k = REDUCE(G_{k-1}) \quad (2)$$

Since each image is half the size in each dimension of the one below it, it therefore consists of one quarter the number of pixels.

The alternative pyramid type is known as the Laplacian. This is formed as a bandpass pyramid rather than a low pass, and is obtained by subtracting a level of the Gaussian pyramid from the level directly beneath it. Each level of the Laplacian pyramid can thus be thought of as a difference image between two corresponding levels of the Gaussian pyramid. Because two levels of the Gaussian pyramid are different sizes, in order to subtract one from another the resolution of the image at level $k + 1$ must first be increased to the resolution of the image at level k . In order

to do this we use the *EXPAND* operation, which upsamples the smaller image (by inserting zeros), and then interpolates the missing values by a further application of the generating filter kernel, i.e.

$$EXPAND(G_{k+1}) = 4 \sum_{m=-2}^2 \sum_{n=-2}^2 w(m, n) G_{k+1} \left(\frac{2i+m}{2}, \frac{2j+n}{2} \right) \quad (3)$$

then:

$$L_k = G_k - EXPAND(G_{k+1}) \quad (4)$$

There must be one fewer levels in the Laplacian pyramid than in the Gaussian. Typically the Gaussian and Laplacian pyramids are generated with three levels above the base image. At levels higher than this the resulting images may become too small to be useful. Another consideration is that with each successive reduction in size, the edge-effects of the convolutions with the generating kernel become more detrimental. It is possible to perform fusion with less than four pyramid levels, but the ability of the algorithm to distinguish features of different sizes is compromised.

The Gaussian pyramid is the one used in the process of extracting detail from the source images to be used for fusion. The Laplacian pyramid described here, and a variant of it known as the FSD Laplacian [9], is used in the inverse pyramid transform to reconstruct the composite image.

2.1 Hardware implementation

There is an inherent parallelism in the pyramid generation process that can be used to generate multiple levels of the pyramid concurrently. A block diagram of the system for generating image pyramids where the subsampling is performed before filtering is shown in figure 3. (Note that for the reasons outlined above this process is conceptually incorrect and should not be used as a general method of producing Gaussian pyramids.) As image data flows into the design, it is immediately downsampled in two dimensions. Horizontal downsampling occurs by discarding every other sample; vertical downsampling occurs by discarding every other row. The samples that are not discarded are stored in a FIFO. When there is sufficient data in the FIFO

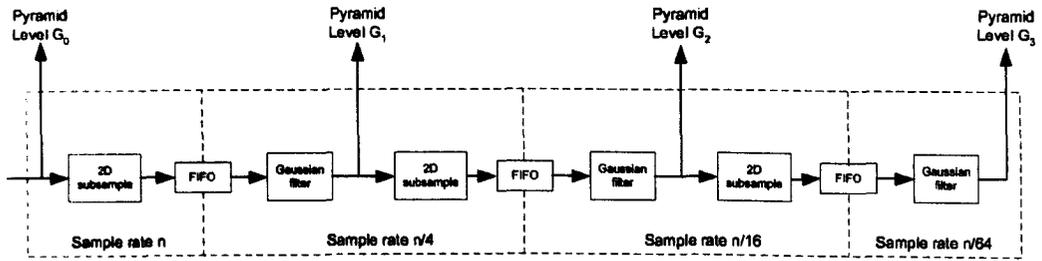


Figure 3: Block diagram of pipeline for generating image pyramids where subsampling is performed before filtering.

the next pyramid level starts to read data out of it. This pyramid level operates at a clock rate that is one quarter that of the level below it, to match the fact that it is receiving one quarter the number of samples. In this way the higher level can run concurrently with the lower level, without emptying the FIFO. This structure is repeated for each level of the pyramid, with each level operating at one quarter the clock speed of the level below it. By using this approach all levels can be generated near simultaneously, and the time to generate the whole pyramid is only 1.1 times that needed to read a full frame of data.

Because the filters used to generate the pyramids are separable and the coefficients are inverse powers of two, they can be implemented using only shift and adds, meaning the design is resource efficient.

This approach allows a fully pipelined method of generating this modified form of image pyramids. Table 1 gives a comparison of the design presented here and the two notable instances of hardware pyramid generation that have been reported. Although the implementation presented here differs from the two referenced implementations in that subsampling is performed before filtering, a consequence of using a fully pipelined structure means that swapping the position of filtering and subsampling blocks would only affect the latency of the design and throughput would remain as shown in the table. Swapping the filtering and subsampling blocks would be necessary to produce Gaussian pyramids with an acceptable level of aliasing and blur, but could be performed whilst still using a multiple clock rate design.

	Image Size	Max Throughput (fps)
PYR [8]	512 × 480	44
Splash 2 [10]	512 × 512	30
Proposed Architecture	640 × 480	100

Table 1: Comparison of proposed and previously reported architectures for image pyramid decomposition.

3 Detail extraction and fusion

The following section will briefly describe the fusion process, but for a more complete discussion of the algorithm see [6]. The implementation presented here is an exact implementation of that published method. To extract detail from the levels of the source pyramids four gradient operators are each applied to each level of the source pyramids via a simple convolution. The operators represent derivatives in the horizontal, vertical, and two diagonal directions, and essentially act as edge detection filters in the four orientations. The gradient filters are as follows:

$$\begin{aligned}
 d_1 &= \begin{bmatrix} 1 & -1 \end{bmatrix} \\
 d_2 &= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \frac{1}{\sqrt{2}} \\
 d_3 &= \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\
 d_4 &= \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \frac{1}{\sqrt{2}}
 \end{aligned} \tag{5}$$

The resulting set of images is known as a gradient pyramid [6], and can completely represent the original image [10]. The gradient pyramid is essentially a set of gradient maps of the source images at varying scales. As illustrated in figure 4, the gradient pyramids constitute a large amount of intermediate data: each level of the two source pyramids is now represented by four gradient maps. Another way to conceptualise this is that the two source pyramids have now been decomposed into four further pyramids each, giving a total of eight full image pyramids to be handled

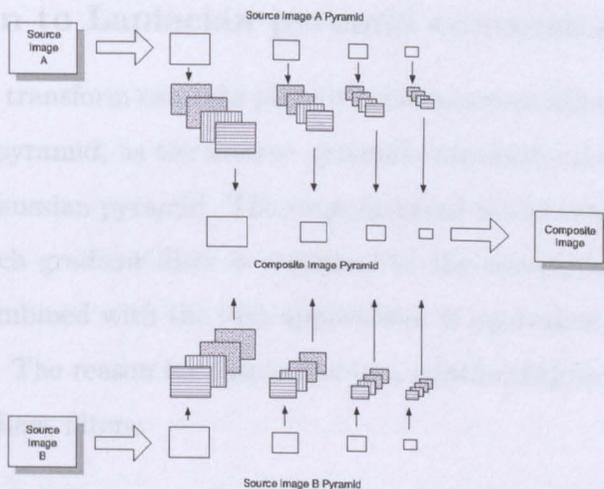


Figure 4: Gradient pyramid decomposition and fusion

and processed concurrently. Obviously, the ability to work with this amount of data on chip is one of the ways in which an FPGA may achieve large performance gains over a microprocessor based implementation.

When applying the gradient operators, the source pyramids are first convolved with $(1 + \dot{w})$, where \dot{w} is a 3×3 filter with binomial coefficients, and chosen such that $\dot{w} * \dot{w} = w$ (where w is the Gaussian filter used in the creation of the Gaussian pyramids). Let D_{kl} be the k^{th} level and l^{th} orientation gradient pyramid image for I . Then:

$$D_{kl} = d_l * [1 + \dot{w}] * G_k \tag{6}$$

$$D_{kl} = d_l * [G_k + \dot{w} * G_k] \tag{7}$$

Fusion of the gradient pyramids then takes place by selecting the most prominent detail from each level. In this application the elements with the greatest absolute value are chosen through a simple comparison, this is an implementation of the simple measure of saliency given in [5]. Other, more complex measures of saliency (also known as activity-level measurements [11]), based on texture criteria and other higher order attributes, may give better results in some specific circumstances, but the amplitude based measure has been shown to provide good results in the general case [5].

3.1 Gaussian to Laplacian pyramid conversion

Before the inverse transform can take place it is necessary to adjust the format of the composite image pyramid, as the inverse pyramid transform relies on the Laplacian rather than the Gaussian pyramid. The steps involved in the conversion are detailed below. Firstly each gradient filter is reapplied to the corresponding oriented data stream, which, combined with the first application, is equivalent to application of a second derivative. The reason for this is due to a relationship between the binomial filter and the gradient filters:

$$1 - \dot{w} = -(d_1 * d_1 + d_2 * d_2 + d_3 * d_3 + d_4 * d_4) \frac{1}{8} \quad (8)$$

Thus each gradient pyramid level D_{kl} can be converted to a second derivative pyramid (or oriented Laplacian [6]) level through a second application of the gradient filters multiplied by $-\frac{1}{8}$:

$$\vec{L}_{kl} = -\frac{1}{8} d_l * D_{kl} \quad (9)$$

Substituting for D_{kl} from (6)

$$\begin{aligned} \vec{L}_{kl} &= -\frac{1}{8} d_l * d_l * (1 + \dot{w}) * G_k \\ \vec{L}_{kl} &= (1 - \dot{w}) * (1 + \dot{w}) * G_k \\ \vec{L}_{kl} &= (1 - w) * G_k \end{aligned} \quad (10)$$

which is the equation for the FSD Laplacian pyramid [6]. Hence by combining the four orientations we arrive at the FSD Laplacian:

$$L_k = \sum_{l=1}^4 \vec{L}_{kl} \quad (11)$$

Conversion from the FSD Laplacian to the RE Laplacian can then be performed by the following approximation, accurate enough for these purposes [6]

$$\tilde{L}_k \approx [1 + w] * L_k \quad (12)$$

3.2 Hardware implementation

A schematic of the fusion portion of the design at a single pyramid level is shown in figure 5. When implementing this process in hardware, at each level of the pyramid the eight gradient operators (four per source image) run in parallel on the source data. The most salient detail in each orientation is then selected from each image through a set of comparators. This detail is fed forward, through a second application of the gradient filters and an adder tree structure for (11), before being filtered again for (12). This structure is repeated at each level of the pyramid, and as with the pyramid generation logic, each level runs at one quarter of the clock speed of the level below it, to account for the fact that the image at that level has a quarter of the number of pixels. The output of each fusion section will form a single level of the composite pyramid, which is subsequently inverse transformed.

4 Inverse pyramid transform

The method of reconstructing an image from its Laplacian pyramid uses the *EXPAND* operation given in (4). The starting point for the inverse transform is the top level of the Gaussian pyramid (in this case G_5). This is formed by a simple averaging of the top level of the source pyramids. Then, from (5):

$$\tilde{G}_k = \tilde{L}_k + EXPAND(G_{k+1}) \quad (13)$$

This process is performed repeatedly to expand each pyramid level. The addition of \tilde{L}_k represents the incorporation of detail data at each scale. The complete fused image lies at the bottom of this pyramid, level G_0 . This process mirrors the decomposition process described in section 2, and again uses FIFOs to store data between pyramid levels, with two-dimensional upsampling occurring as data are read from the FIFOs through insertion of zero value samples.

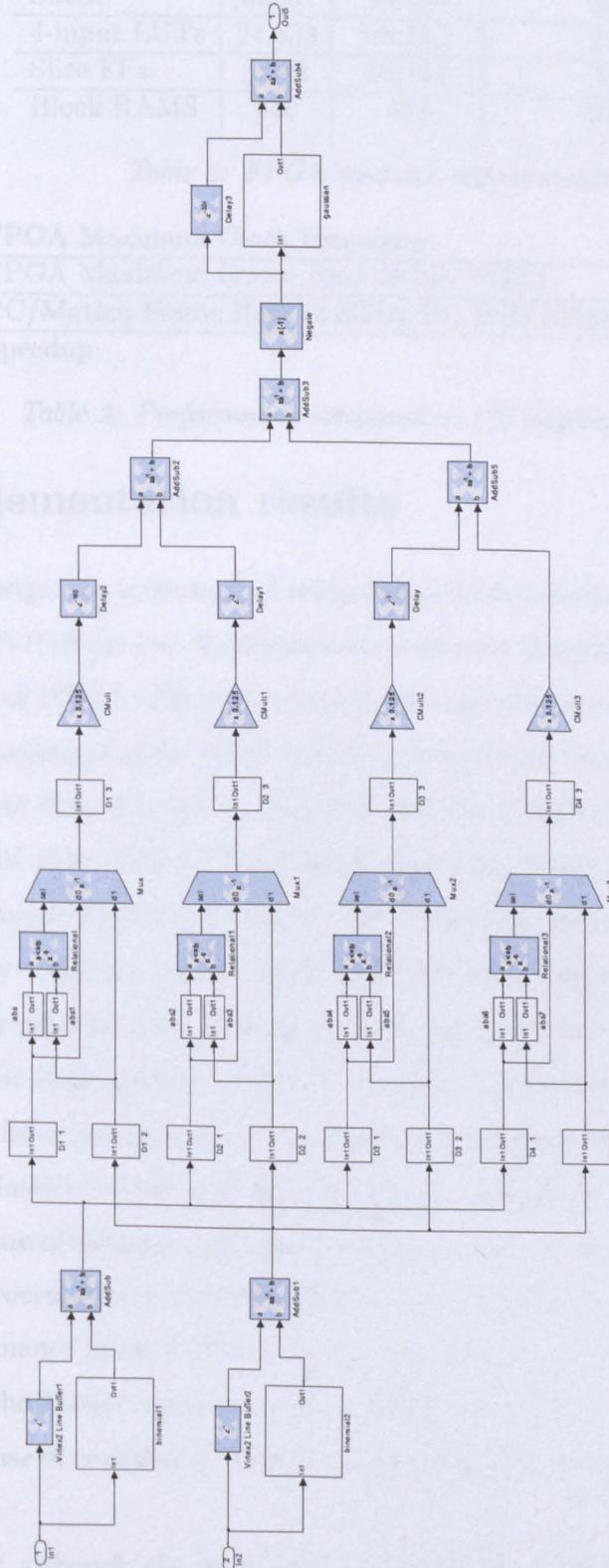


Figure 5: Schematic of the detail extraction and fusion process. This structure is repeated at each level of the pyramid.

Resource	Used	Available	% of XC2VP100
Slices	13,287	44,096	30
4-input LUTs	24,533	88,192	27
Slice FFs	5,784	88,192	6
Block RAMS	430	444	96

Table 2: FPGA resource requirements

FPGA Maximum Clock Frequency	31 MHz
FPGA Maximum Frame Rate (8-bit, VGA)	101 fps
PC/Matlab Frame Rate (2.8GHz P4, 1GB RAM)	0.91 fps
Speedup	111x

Table 3: Performance compared to PC implementation

5 Implementation results

The entire design was implemented using Xilinx System Generator and mapped to a Virtex-2 XC2VP100 device. Synthesis was carried out using Xilinx's proprietary XST tool, as part of ISE8.1. The logic usage results are presented in table 2. The main hardware requirement is for RAM, which is understandable given the large amount of intermediate data that has to be held within the FPGA during the processing of a single pair of input frames. The RAM is essentially being used as long delay lines, and there is scope to retarget some of this requirement towards use of the flip-flops on the device. This may enable the design to fit into a slightly smaller device. The design is fully pipelined and capable of producing an output pixel every clock cycle. The maximum clock speed reported by the place and route tools is 31MHz.

As a means of comparing the system's performance with a software implementation, the Matlab Profiler was used to measure the speed of execution of a Matlab implementation of the same algorithm. Processing a single frame of data on a 2.8GHz Pentium-4 processor with 1GB RAM takes, on average, 1.1 seconds. A comparison of the performance of both FPGA and PC-based versions of the algorithm is shown in table 3. The Matlab version of the algorithm is used in its original (interpreted) format, and use of compiled code may improve the results from the PC-based implementation.

Note that although the maximum reported clock speed is 31MHz, a speed of



Figure 6: Example of source and fused images using the described algorithm and implementation.

10MHz would allow 8-bit greyscale VGA video at 30fps to be processed in real-time, with a latency of <50 ms. Sample images have been produced for comparison and are shown in figure 6. The source images are from a TV camera and thermal (IR) camera respectively, and are pre-registered. Both source images accentuate different features about the scene being observed. The composite image contains the significant details from both source images. A side effect of producing pyramids by subsampling before filtering is that the output images are more blurred than they would be otherwise and are likely to suffer from distortions caused by spatial aliasing. Reversing the ordering of the subsampling and filtering blocks in the pipeline structure of figure 3 would produce clearer images, at the expense of increased latency.

A difference image (after inversion and scaling) between the Matlab and FPGA implementations is shown in figure 7. Forty lines have been removed from each edge to remove most of the edge-effects caused by the convolutions. (Although the filters are only 5×5 , the upsampling and downsampling involved in the pyramidal algorithm means that the edge-effects are spread over a wider area than in typical convolution applications). Over the region shown the maximum error between corresponding pixels of the two implementations is equal to 1.35, and the mean-squared error is 0.02. The small error value is explained by the simple mathematics of the algorithm (just basic convolutions with easily representable coefficients), which pose no problem to the hardware implementation.

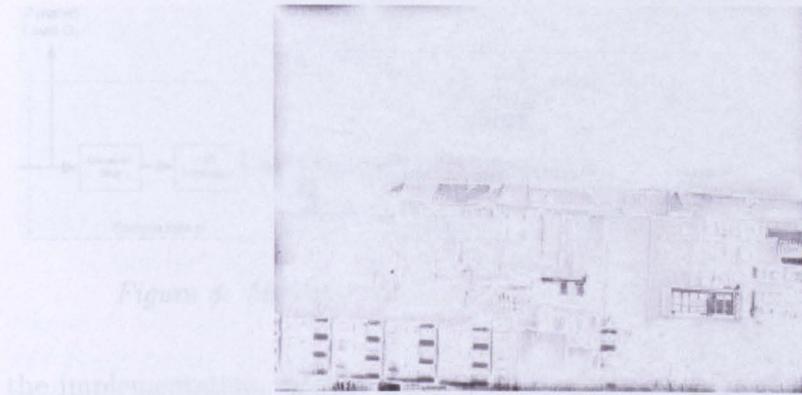


Figure 7: Inverted and scaled difference image between Matlab and hardware implementations of pyramidal image fusion, after removing the edge pixels affected by the convolutions.

6 Implications of the alternative pyramid generation process

The method of generating image pyramids that has been used here is conceptually incorrect, due to subsampling being performed before filtering at each stage of the process. Depending on the bandwidth of the source images this method will therefore produce image pyramids with varying levels of distortion due to spatial aliasing. As was shown in figure 2, the resulting image pyramids also exhibit an increased level of blur. The implementation described here demonstrates that a degree of image fusion can be performed with image pyramids generated in this way, but the method should not be used in a general sense because of the erroneous data it produces. In order to generate true Gaussian pyramids the processing pipeline of figure 3 could be modified by swapping the positions of the subsampling and filtering blocks, as shown in the modified pipeline of figure 8. The multiple clock rate design would still be an effective method for the same reasons that are described above.

This change has hardware implications due to the increased size of the images being filtered at each stage, meaning that the line-buffers in the two-dimensional filters must now be twice as long. Calculations show that for two 640×480 video streams and filters with internal 16-bit representations an implementation of image fusion would have an increased memory requirement of 70kb. This would require an additional four of the 18kb Block RAMs available on the Virtex-2 FPGA used for

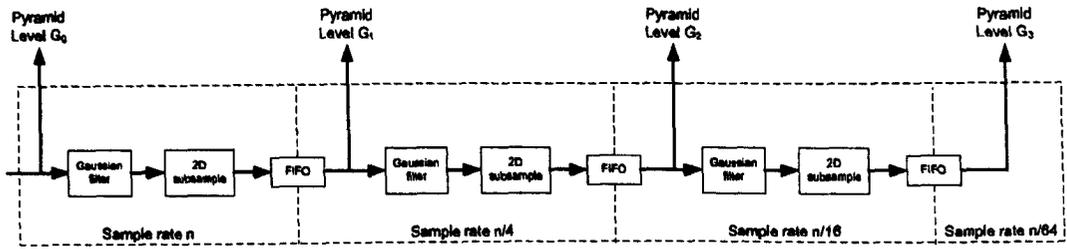


Figure 8: Modified pipeline for generating true Gaussian pyramids.

the implementation; table 2 shows that there are sufficient remaining resources to accommodate this on the same device.

Although there is an increased memory requirement the size of the filters is unchanged and so there is no additional computational expense. However the latency of the pyramid generation process is increased by a factor of two. Due to the pyramid generation process constituting about a third of the total implementation, it is estimated that the total latency for the alternative design would be in the region of 60-70ms.

The section of the design for performing detail extraction and fusion process is not affected by the change in how pyramids are generated. The inverse pyramid transform implemented here is also an exact implementation of that published by Burt in his original paper.

Comparison images between the implementation presented here and fusion with true Burt pyramids is presented in figure 9. Although it is difficult to discern aliasing artefacts from the original implementation, the increased level of blur is evident. With these test images the two implementations appear to select the same details from the source images, but with images that use the full bandwidth the aliasing artefacts caused by the original pipeline of figure 3 may be identified as detail by the gradient filters and therefore be noticeable in the composite image. The difference image shows that the most severe differences are found where features have high contrast relative to the surrounding region.

For these reasons the processing pipeline of figure 8 is the method that should always be used, despite the increased resource requirements.

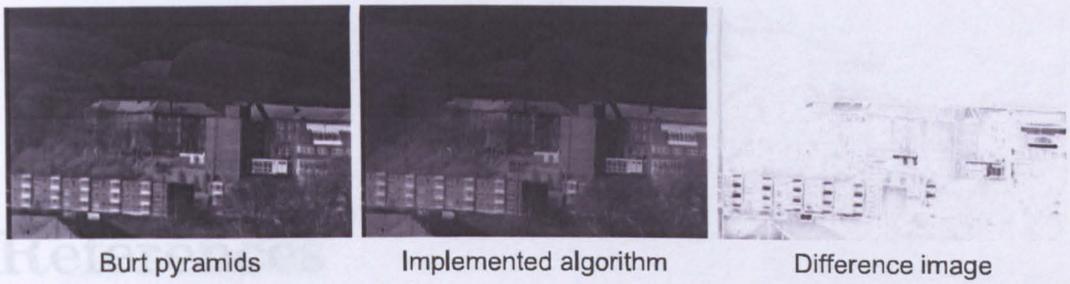


Figure 9: Comparison of fused images when Burt Gaussian pyramids are used versus the implemented pyramid algorithm. The difference image has been inverted and scaled to use the whole greyscale range.

7 Conclusion

An implementation of a pattern-selective image fusion algorithm has been presented that utilises aspects of FPGA technology to enable dual video streams to be processed in real-time. The results from the implementation showed that four levels of pyramidal decomposition, with four separate gradient operators, can all run on a single FPGA with no requirement for off-chip memory. The modular nature of the design means that pyramids with less, or more, levels could be added without major modification. Use of an FPGA has enabled a design that can process images at a rate over 100 times faster than a similar PC-based Matlab implementation. The method of generating Gaussian pyramids used here is based on an incorrect implementation of the Burt algorithm, which is prone to aliasing and increased blur, and is in general an unacceptable method. An analysis has been given of the cost of implementing the true Burt algorithm using the same pipelined structure, concluding that the modified design could fit on the same device with only a small decrease in performance.

Berlin, May 4-7, 2005.

- [7] V. S. Pappas and J. A. Stiles, "A fast algorithm for generating Gaussian pyramids," *IEEE Transactions on Image Processing*, vol. 13, no. 12, pp. 1517-1524, Dec. 2004.
- [8] G. S. Yang and S. J. Ko, "Fast algorithm for generating Gaussian pyramids using image analysis," *IEEE Transactions on Image Processing*, vol. 13, no. 12, pp. 1525-1532, Dec. 2004.
- [9] C. H. Anderson, "Fast algorithms for generating Gaussian pyramids and synthesizing images," *IEEE Transactions on Image Processing*, vol. 13, no. 12, pp. 1533-1540, Dec. 2004.

References

- [1] P. K. Varshney, "Multisensor data fusion," *Electronics and Communication Engineering Journal*, vol. 9, no. 6, pp. 245–253, Dec. 1997.
- [2] Z. Xue, R. S. Blum, and Y. Li, "Fusion of visual and IR images for concealed weapon detection," in *Proc. 5th Int. Conf. Information Fusion*, vol. 1, Sept. 2002.
- [3] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA Engineer*, vol. 29, no. 6, pp. 33–41, Nov. 1984.
- [4] P. Burt and E. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 532–540, Apr. 1983.
- [5] P. J. Burt, "A gradient pyramid-basis for pattern-selective image fusion," in *Proc. Society for Information Display Conf.*, 1992.
- [6] P. J. Burt and R. J. Kolczynski, "Enhanced image capture through fusion," in *Computer Vision, 1993. Proceedings., Fourth International Conference on*, Berlin, May 11–14, 1993, pp. 173–182.
- [7] V. S. Petrović and C. S. Xydeas, "Gradient based multiresolution image fusion," *IEEE Transactions on Image Processing*, vol. 13, no. 2, pp. 228–237, Feb. 2004.
- [8] G. S. van der Wal and P. J. Burt, "A VLSI pyramid chip for multiresolution image analysis," *Int. Jour. Computer Vision*, vol. 8, no. 3, pp. 177–189, 1992.
- [9] C. H. Anderson, "Filter-subtract-decimate hierarchical pyramid signal analyzing and synthesizing technique."

-
- [10] A. L. Abbot, P. M. Athanas, L. Chen, and R. L. Elliot, "Finding lines and building pyramids with Splash-2," in *Proc. IEEE Workshop on FPGAs for custom computing machines*, Apr. 1994, pp. 155–163.
- [11] Z. Zhang and R. S. Blum, "A categorization of multiscale-decomposition-based image fusion schemes with a performance study for a digital camera," *Proceedings of the IEEE*, vol. 87, no. 8, pp. 177–189, Aug. 1999.

Appendix G: Video Processing Demonstration Platform

Video Processing Demonstration Platform

August 2006

Summary: Several algorithms presented in this portfolio were implemented on a custom FPGA circuit board designed by Thales for a commercial product. The board was ideal for algorithm implementation purposes, as it featured a medium sized Virtex-2 FPGA, video D/A and A/D converter ICs, and six banks of SDRAM each with independent address and data buses.

In order to make the algorithm implementation process easier, a suite of automated scripts and batch files were produced by the RE that enabled a System Generator algorithm design to be synthesised and implemented on the board with no manual intervention from the user. A VHDL 'wrapper' was also developed to handle low-level issues such as memory accesses and clocking.

Use of this system would enable a person with little or no hardware design experience to trial and demonstrate video processing algorithms in hardware, and it may therefore be useful for algorithm development purposes. This short report documents the board and implementation process and how it should be used.

Oliver Sims

EngD 4th Year

Industrial Sponsor: Thales Optronics

1 Introduction

The following document describes a system developed to provide a standard platform for video processing algorithms. Use of this demonstration platform allows for rapid deployment of video processing algorithms on FPGA hardware, and provides an opportunity for the designer to trial algorithms quickly using high-speed logic.

The platform essentially provides a VHDL wrapper for algorithms implemented using Xilinx System Generator. The wrapper provides clock management, memory interfaces, and interfaces to video A/D and D/A chips that manage the data acquisition and display.

System Generator allows use of Xilinx IP cores in a high-level graphical domain, and allows the designer to utilise the whole range of advanced Simulink stimulus and verification tools to test the design. Once the algorithm is operating satisfactorily a push-button process generates HDL code. A set of scripts and batch files have been produced that will translate this HDL code into an FPGA programming file. In this way an algorithm design can go from a high-level block diagram to a working demonstration in hardware in minutes, with no manual intervention required. The automated implementation process will in some cases be a quicker alternative to performing a PC-based simulation, and will give a more immediate means to display and compare the efficacy of different algorithm models.

This report documents the hardware design and software environment that makes up the demonstration platform, and the format required for algorithms to be demonstrated.

2 Hardware

The system uses the main board and portions of firmware from the Thales Joint Target Acquisition System (JTAS) project. The JTAS project is a handheld system with dual video input channels and interfaces for a variety of peripheral equipment, for instance navigational systems. On-board it features six independent SDRAM devices, with each device containing 8M 16-bit locations. Each device has dedicated address and data lines, meaning they can be accessed individually and independently

of each other.

The video input channels are controlled by SAF7113H video input processors from Philips Semiconductor. These devices synchronise to an analogue video stream at the input, perform A/D conversion, before outputting the active video data in ITU656 4:2:2 format. The devices are highly configurable and can handle a wide range of input video standards. They are programmed using an I2C interface. The JTAS board outputs analogue video data via an ADV7179 video encoder from Analog Devices. The device converts 8-bit 4:2:2 video into a configurable analogue output. The device handles all timing issues and generates the synchronising and blanking periods as necessary for the output standard being used. The device is also configured using an I2C interface.

It should be noted that the JTAS platform is designed for full colour video whereas this video processing demonstrator is greyscale only. Greyscale is sufficient to demonstrate the majority of image and video processing algorithms, and it cuts down the complexity of the system and provides more spare logic for the algorithm.

3 System architecture

A block diagram of the system is shown in figure 1. There are multiple clock domains throughout the design, with asynchronous FIFOs acting as buffers between domains. At the ends of the pipeline the clock rate is 27MHz, as determined by the 4:2:2 format for digital sampling of analogue video (ITU-R 656). In the 27MHz domains the video input and output processor modules interface with the video decoder and encoder devices respectively. These modules are taken from the original JTAS design with only minor modifications to handle greyscale only images.

On either side of the algorithm the video data is buffered in large SDRAM stores. The SDRAM blocks are controlled by two separate SDRAM controllers, whose function is to read/write data from/to RAM as necessary to prevent the FIFOs from filling or emptying. The SDRAM controllers are subtly different to each other in the way they operate: the input side memory controller is designed in such a way as to crudely deinterlace the video data on the input side of the system through weaving

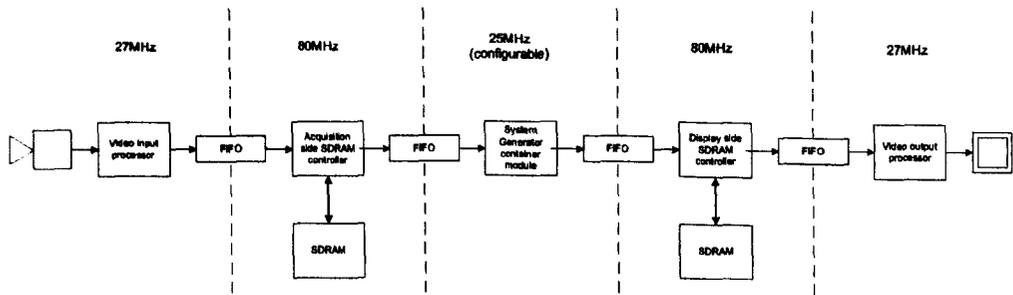


Figure 1: System block diagram.

of alternate frames as they are written to memory; this operation is reversed on the output side. Although this form of deinterlacing may introduce artefacts for fast moving objects, it is chosen here for its simplicity.

The SDRAM controllers both operate at a clock frequency of 80MHz, this is a remnant of the JTAS design where this frequency was chosen to accommodate SVGA video formats. It remains unchanged here, as 80MHz provides sufficient responsiveness to perform the SDRAM reads/writes given that the clock frequency of the algorithm is user definable and may be increased.

The System Generator container module connects the System Generator netlist to the surrounding hardware. The clock rate of this module is variable, through modification of the top-level VHDL netlist and/or reconfiguration of the Digital Clock Management (DCM) units on the FPGA. Increasing the clock speed of the algorithm may be necessary in situations where the algorithm requires multiple clock cycles of processing time per pixel, and a clock speed of 25MHz is insufficient to process the incoming video data in real-time. The algorithm in the System Generator module does not run continuously, but instead operates in bursts by processing whole lines of image data one at a time. This is controlled by the VHDL container module, which detects when a full line of data is available in the input FIFO, and then enables the algorithm for sufficient time to process that data and write it to the output FIFO.

The controller and interface logic has the resource overhead described in table 1. The remaining logic is available for the algorithm.

Resource	Used	Total on Device	% of XC2VP20
Slices	923	9,280	9%
4-input LUTs	1,382	18,560	7%
Slice FFs	990	18,560	5%
Block RAMs	6	88	6%

Table 1: Resource overhead

4 System Generator model format

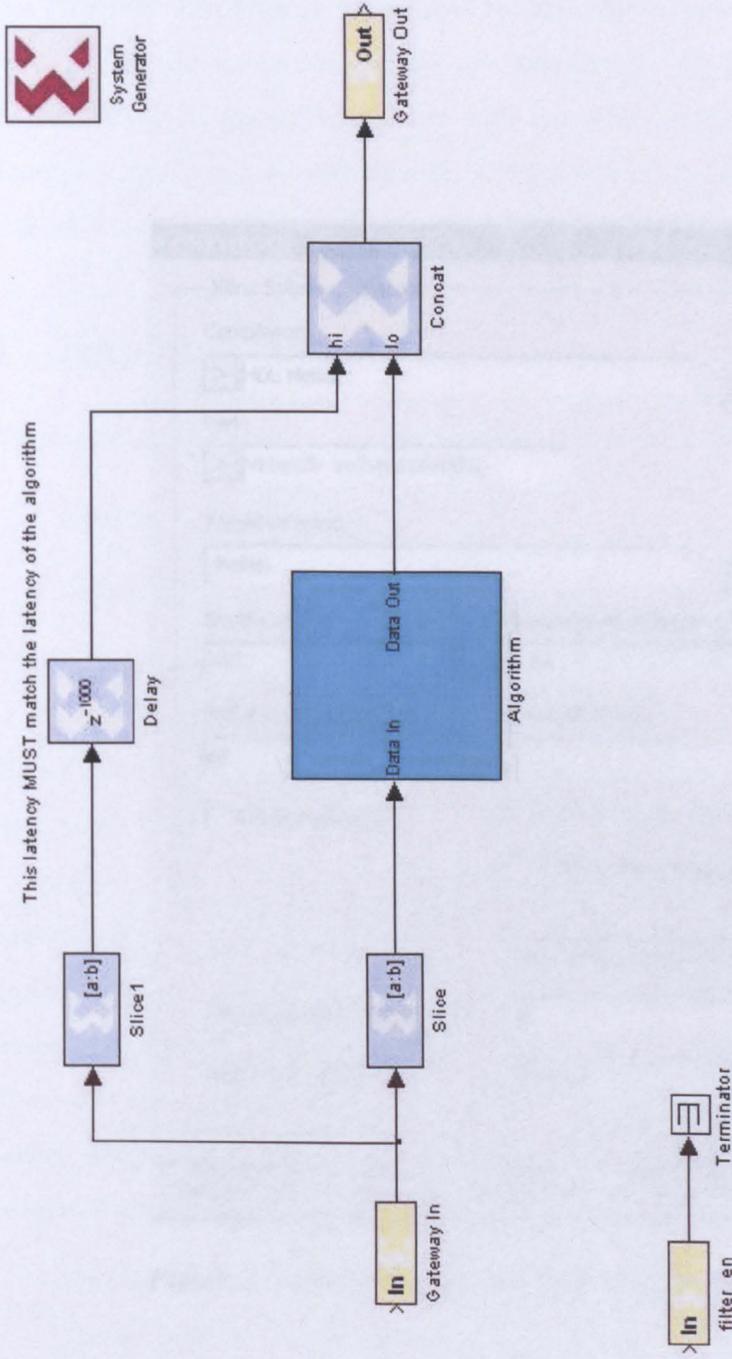
A copy of the System Generator model to be used as a basis for algorithm development is shown in figure 2.

The designer is free to implement any algorithm using the System Generator block set, but there are certain constraints to ensure compatibility with the surrounding hardware:

- The algorithm will only operate on video data arriving in a raster format.
- There must be a parallel delay line (3-bits wide) that is used to send sync information alongside the data. This delay line must be the same length as the latency of the algorithm being deployed. The latency of the algorithm can usually be determined by counting the number of register stages (denoted z^{-1} by System Generator) between the input and output.
- The filename used for the System Generator model must be *alg.mdl*, and the generated code must be saved into a sub-folder of the SysGen folder called *netlist*.
- The image size is currently imposed by the PAL input as 575 lines of 720 pixels. Modification to operate on smaller images (for instance VGA) is possible but requires alteration of the SDRAM controllers.

There is a user input labelled *filter_en*, which can be used within the algorithm, this is connected to the *fire* push-button on the JTAS test harness. One way in which this could be used is in selecting between a processed and unprocessed version of the video stream, implemented using a multiplexer at the output of the algorithm. In addition the centre button of the five directional buttons on the test harness acts

as a master reset. The HDL netlist is generated using the Generate button in the System Generator block. A copy of the System Generator dialog box is shown in figure 3; the settings shown in the figure should not be altered.



This filter enable input is connected to a push-button input (labelled "Fire") on the JTAS test bed.

It can be used to provide user input to algorithm, while it is running, for instance controlling a Mux that selects between processed and unprocessed data

Figure 2: System Generator top-level wrapper.

5. Directory structure

The directory structure in listing 1 is required for correct operation of the scripts. Some of the important design files are also listed. All sub-directories in the structure may be placed on a drive with any letter as the name. The System Generator model file, named alg.mdl, is the first file in the directory. This is the file that

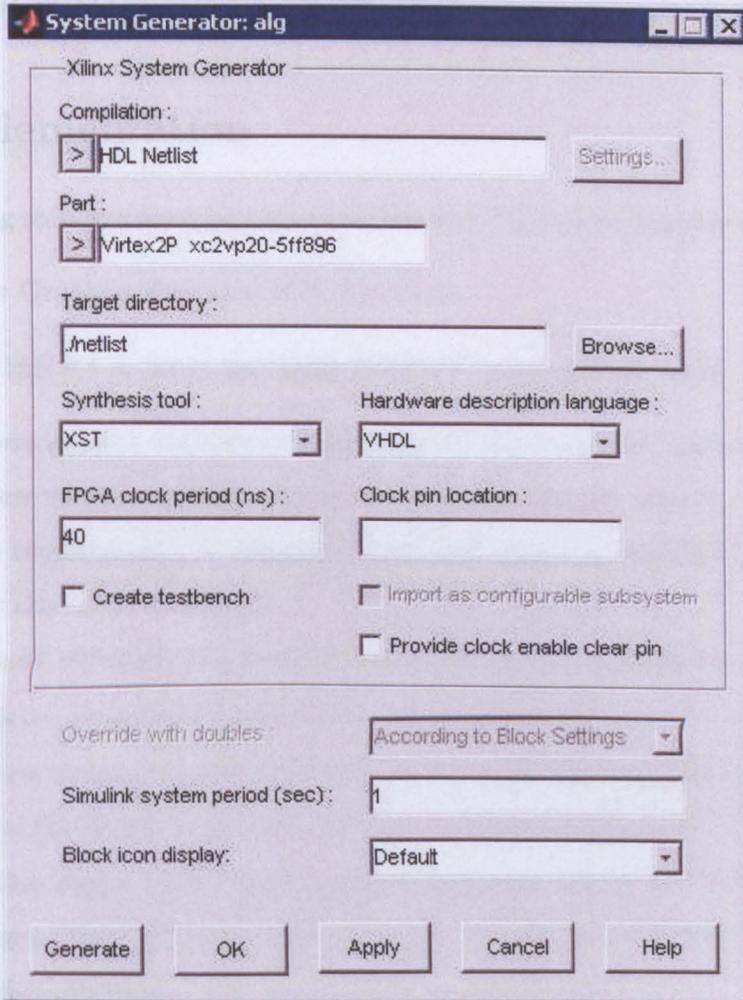


Figure 3: Correct settings for System Generator dialog box

5 Directory structure

The directory structure in listing 1 is required for correct operation of the scripts. Some of the important design files are also listed. All paths are relative, so the structure may be placed on a drive with any letter as its name. Note the System Generator model file, named *alg.mdl*, in the SysGen subdirectory of Videoproc. This is the file that should contain the algorithm to be deployed.

6 Implementation

The following software must be present on the host PC before implementing a design:

- Mentor Graphics Precision RTL Synthesis
- Xilinx ISE 8.1 or later, including iMPACT programming tools

The Path environment variable on the host PC must also be correctly configured to allow access to these programs from anywhere in the file system. In addition to the software requirements, a Compact Flash card reader is required to transfer the bitstream to the target hardware.

The process of taking the System Generator netlist through to an FPGA bitstream has been automated through the use of TCL scripts and a batch file. The implementation process is controlled from the XFLOW directory (XFLOW being the name given to the Xilinx command used for running the ISE tools in batch mode). There are three stages to the implementation process: synthesis, Xilinx implementation, and generation of the programming file. There are two batch files that can be used for this process:

- *Go.bat* prompts the user before executing each program. This is useful to check the output of a program before executing the next one.
- *Go_all.bat* executes all stages without requiring user input.

The synthesis stage calls Precision Synthesis in batch mode and synthesises the design. The commands are TCL scripted to collate and compile the necessary VHDL files and apply synthesis constraints.

Listing 1 Directory structure

```
\VIDEOPROC
|  ADV7179IIC.vhd
|  AlgSim.vhd
|  CONFIGrst.vhd
|  Debounce.vhd
|  PALout.vhd
|  SAF7113IIC.vhd
|  SDRAM_Acq.vhd
|  SDRAM_Disp.vhd
|  SURAC_A.vhd
|  VideoChain.vhd
|  VideoIn444.vhd
|
+---Cores
|  |
+---SysGen
|  |  alg.mdl
|  +---netlist
|  |
+---Test
+---work
|  |
\---xflow
|  bitgen_opts.opt
|  go.bat
|  go_all.bat
|  ISE_opts.opt
|  precision_rtl.sdc
|  precision_settings.tcl
|  precproj.tcl
|  precsynth.tcl
|  SURAC.ucf
|
+---impact
|  \---VidProCF
|      |  xilinx.sys
|      \---VidProCF
|
+---Reports
|
+---ISE
|
\---Synthesis
    \---Precision
```

The Xilinx tools translate the netlist into Xilinx hardware and then perform placement and routing. These processes are controlled from the XFLOW command. Finally, the Xilinx iMPACT tool is called to generate the programming files. On completion the programming files (a folder called VidProCF and a *xilinx.sys* file) should be copied to the root of the System ACE Compact Flash card being used to program the device.

During the implementation phase several reports and log files are generated. These are all automatically copied to the `/xflow/Reports` subdirectory. These reports are useful to check for any warnings or errors detected by the synthesis tool, and can also be used to check device usage and timing information as reported by ISE. The device usage information is contained in the Map report, named `SURAC_map.mrp`.

7 Notes

There are several areas where the demonstration platform could be developed further:

- Cater for adjustable image sizes. For instance, VGA would be a popular resolution but currently requires modification of the VHDL firmware design.
- The constraints of the ITU601 standard mean that although the video is in 8-bit format, the necessity of sync levels etc. means that this translates to 200 quantisation levels not 255. The black level corresponds to 16, and the white level is 235. Algorithms that exceed this range in their output will still work without causing problems as the logic has been designed to impose these limits before outputting data to the video encoder. Unfortunately this means there is a reduction of dynamic range available to the algorithm.

Appendix H: Implementation of Image Registration by Polynomial Warping

Image Registration by Polynomial Warping

September 2006

Summary: An introduction to an algorithm that uses polynomial equations to generate spatial maps, which describe the affine transformations required to register an image to a reference image. Registration is a necessary step before image fusion may be performed. The implementation adapts an SDRAM controller to warp an image as it is read out of memory.

Oliver Sims

EngD 4th Year

Industrial Sponsor: Thales Optronics

1 Introduction

Image registration is required in a variety of fields where information contained in multiple images must be compared or combined. Some example uses of image registration are in medical imaging (for instance monitoring tumour growth), updating of map imagery, computer graphics, etc. Due to the wide range of applications, there has emerged an abundance of methods of performing image registration. These are increasingly becoming domain-specific, such that the popular algorithms in one area of image registration may not have similar success in other areas.

The problem is essentially one of transforming an image, such that when it is compared with another image of the same scene any common features are in the same relative position. This will enable identification of any differences between the images, whilst removing the discrepancies caused by the processes involved in capturing the image data. These discrepancies may be caused by using different sensors for each image, movement of a sensor between captures, or a time delay between captures, amongst other things.

The application domain of this work is in registration of images that are to be used as input to an image fusion algorithm, operating on data obtained from two different sensors. The purpose of image fusion is to combine images such that all useful information from both source images is retained into the final composite, without introduction of unnecessary noise or artefacts. The information content of the fused image is thus greater than either source image, and may be deemed more effective for either human or machine interpretation. The two sensors that provide the data for fusion will be operating simultaneously, and will usually be adjacent to one another; the difference in viewing angle between the two sensors is known as the boresight error. The sensors may be also be of differing resolutions. The objective of the registration algorithm is to harmonise the two images such that they are the same resolution and appear to be captured from the same viewpoint. It is equally important that no detail in either image is destroyed or lost by the registration process.

The fact that the two sensors are adjacent means that the amount of translation between the two images can be assessed once before processing begins, and is assumed

to remain constant. This means that automatic registration techniques (whereby the amount of transformation that must be applied is calculated in real-time) are unnecessary. Rather than mathematically modelling the transformation it is common to rely on manual identification of multiple control points, i.e. points in each image that correspond to one-another. These points may then be used to calculate the transformation that is required.

This document describes a hardware implementation of one such image registration algorithm known as polynomial warping. The target hardware is a video processing demonstration platform, comprising the necessary hardware for analogue video acquisition, FPGA processing, and subsequent D/A conversion and display. The resulting implementation is a small module that can warp the image data as it is being read into the FPGA ready for further processing, for instance for image fusion.

2 Polynomial warping

The key task of image registration for fusion applications is the removal of the variation in boresight between two (or more) sensors. This may be classed as a rigid-body transformation, i.e. one that is formed of a translation, rotation, and a scale change. Rigid-body transformations are global (the same transformation applies over the whole image) and fall into the category of affine transformations [1]. This makes them suitable for implementation using polynomial warping methods; an example of the kind of warping possible with affine transformations is shown in figure 1.

Polynomial warping produces a non-linear spatial map that describes the plane-to-plane mapping between the two images [2]. This mapping can be used to select the pixel in the unregistered image that corresponds to a given pixel in the registered image. There will not always be a direct one-to-one mapping, so interpolation is usually required to calculate values at a sub-pixel level.

Given two images, $f(x_1, x_2)$ and $f(u_1, u_2)$, the task is to find a coordinate transform between the two, i.e. functions that translate an input pixel location (x_1, x_2)

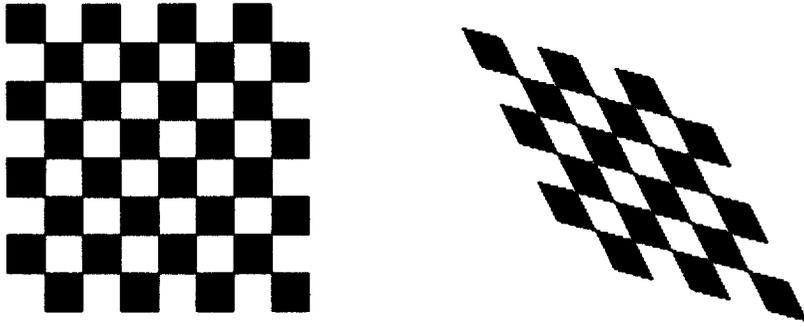


Figure 1: Example of an affine transformation

to an output pixel location (u_1, u_2) :

$$x_1 = g_1(u_1, u_2) \quad (1a)$$

$$x_2 = g_2(u_1, u_2) \quad (1b)$$

Rather than attempting to mathematically model the transforming functions g_1 and g_2 , it is usually sufficient to approximate them using polynomials of the form:

$$x_1 = \sum_{i=0}^N \sum_{j=0}^{N-i} K_{ij}^1 u_1^i u_2^j \quad (2a)$$

$$x_2 = \sum_{i=0}^N \sum_{j=0}^{N-i} K_{ij}^2 u_1^i u_2^j \quad (2b)$$

where K^1 and K^2 are sets of constant coefficients. N gives the order of the polynomials; low orders are usually sufficient and all affine transformations can be described by first order ($N=1$) equations [3]. In this case the polynomials expand to:

$$x_1 = K_{00}^1 + K_{01}^1 u_2 + K_{10}^1 u_1 \quad (3a)$$

$$x_2 = K_{00}^2 + K_{01}^2 u_2 + K_{10}^2 u_1 \quad (3b)$$

The coefficients K^1 and K^2 may be determined through a process based on identification of control points, which are used to form systems of simultaneous equations

that when solved give the values for the coefficients. In order to determine six unknown coefficients, as required for the first order equations above, it is necessary to identify at least six control point pairs. (More than six control points may also be used, resulting in an over-determined system of equations). The control points may be defined as:

$$\{x_{1i}, x_{2i}, u_{1i}, u_{2i}\} \text{ for } i = 1 \text{ to } 6 \quad (4)$$

Solution of the system of equations can then be performed using standard matrix methods, i.e.

$$\begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \end{bmatrix} = \begin{bmatrix} 1 & u_{21} & u_{11} \\ 1 & u_{22} & u_{12} \\ 1 & u_{23} & u_{13} \end{bmatrix} \begin{bmatrix} K_{00}^1 \\ K_{01}^1 \\ K_{10}^1 \end{bmatrix} \quad (5)$$

Hence:

$$\begin{bmatrix} K_{00}^1 \\ K_{01}^1 \\ K_{10}^1 \end{bmatrix} = \begin{bmatrix} 1 & u_{21} & u_{11} \\ 1 & u_{22} & u_{12} \\ 1 & u_{23} & u_{13} \end{bmatrix}^{-1} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \end{bmatrix} \quad (6)$$

will provide the solutions for the K^1 coefficients, and the similar operation can be performed for K^2 .

Once the coefficients have been calculated the polynomial equations can be used to identify the pixel locations in the input image that are required for the output image. The values generated by the polynomials will not usually be integers, so that the point they refer to in the input image may be between pixels, and interpolation will be required to approximate the necessary value.

3 Implementation

The process of calculating the coefficients for the warping polynomials only needs to be performed once for a particular transformation between two sensor positions, so that if the sensors do not move relative to one another there is no need to repeat the calculations. These calculations can thus be performed beforehand (using for instance Matlab) in order to generate the polynomial coefficients that can then be hard coded into the design.

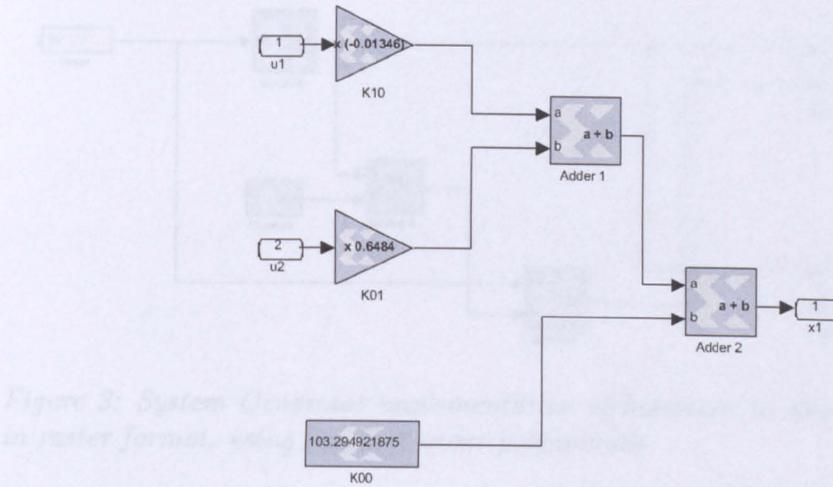


Figure 2: System Generator implementation of first-order polynomial equation

System Generator was used to implement the polynomial equations that utilise the coefficients. System Generator is ideally suited to applications such as this: it is designed to facilitate experimentation with binary precision and word-lengths, and the polynomials are formed from add and multiply operations which map directly to the standard block libraries. For these reasons, and the inherent ease in designing systems graphically, implementing a design such as this is much quicker in System Generator than in manually written VHDL.

Figure 2 shows the implementation of one of the polynomial equations. The coefficients are hard-coded as constants for the two gain terms and as an input into the second adder block. Figure 3 shows how this model of a polynomial equation is used to generate the spatial map. Two counters are used to generate the input addresses; the counters should be able to represent all pixel locations in the input image (in this case a 640×480 element VGA image), so the first counter represents the pixels per line (up to a maximum of 640), and the second counter represents the current image row (up to 480). In this way all pixel locations are generated in raster format, one per clock cycle.

Sixteen bits are used to represent the constants in signed or unsigned format, with a fixed binary point that may be positioned as appropriate for each coefficient. Subsequently the values that are naturally output by the system are real numbers and are rarely integers. The input image is comprised of discrete pixels that lie on

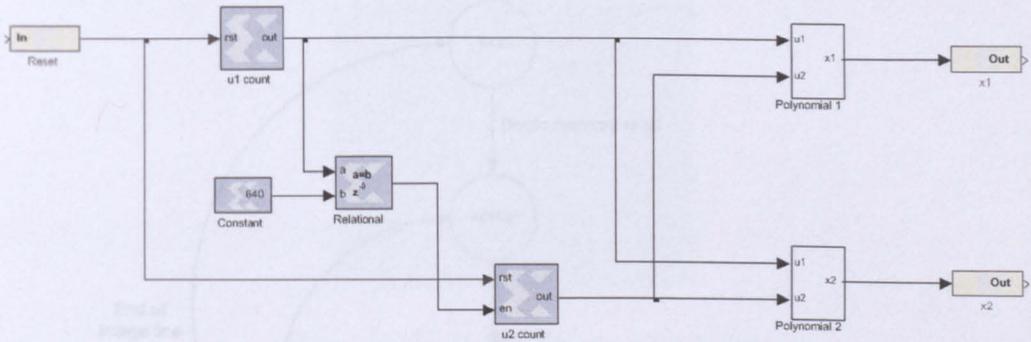


Figure 3: System Generator implementation of hardware to generate a spatial map in raster format, using two first-order polynomials

a sampling grid, and the coordinates generated by the polynomials generate a new sampling grid with pixel locations that do not correspond exactly to the pixels in the input image [3]. Interpolation is required to approximate the output pixel values. Bilinear interpolation is commonly used in these situations, however nearest-neighbour interpolation has the advantage of being significantly easier to implement as it may be achieved simply by rounding the output to the nearest integer. The disadvantage of nearest-neighbour interpolation is that it may introduce some blockiness in places in the output image. However, bilinear interpolation is also problematic (it has an inherent smoothing effect that can reduce detail), and so the reduced complexity of nearest-neighbour interpolation is preferred for demonstration purposes. More sophisticated methods of interpolation could be used at a later stage if necessary.

3.1 SDRAM controller

The video processing system used for the registration process buffers image data arriving from a PAL analogue camera in an SDRAM store. As the data are stored in memory the alternate fields are weaved together as a simple form of deinterlacing. SDRAM memory is organised into rows and columns, and the image is stored with one image line split over two rows of memory.

The spatial maps generated by the polynomial equations are utilised during the process of reading the data out of SDRAM and into the FPGA for processing. The read addresses for the SDRAM are formed from the coordinates generated by the

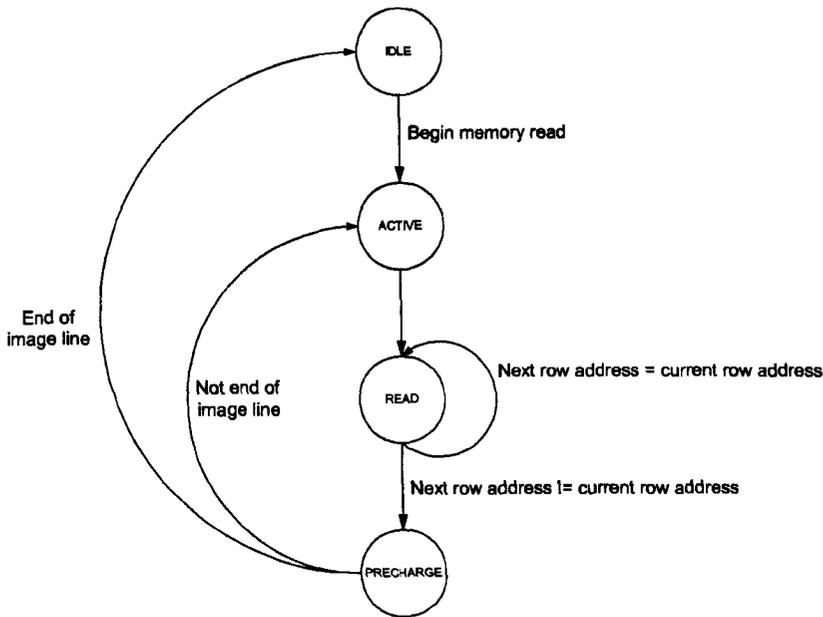


Figure 4: State machine implementation of memory controller for image registration

System Generator polynomials, with x_1 representing the required memory row and x_2 representing the required memory column. As coordinates are generated in a raster format, the column address changes significantly more frequently than the row address. Due to the way in which SDRAM operates, when the row address changes a Precharge command must be issued to the row currently being accessed, and then the next row must be activated before reads can continue. This introduces extra complexity into the SDRAM controller; a state machine was necessary to detect the changes in row addresses and issue the correct sequence of commands, this is shown in figure 4. The issuing of Active and Precharge commands adds latency, which may become problematic if the warping includes a significant rotational aspect since the number of row changes required per image line will be high. In this situation the clock rate of the SDRAM controller may need to be increased in order to meet real-time constraints imposed by the source video.

4 Results

The design is mostly comprised of the System Generator implementation of the polynomial equations, and the extra control logic needed in the SDRAM controller

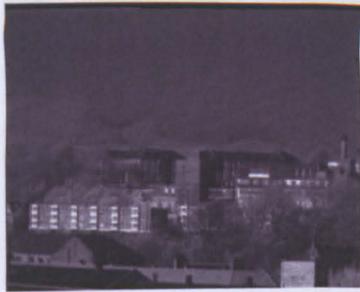


Figure 5: TV image before registration

to be able to perform the active/precharge commands as required by the addresses as they are generated. For this reason the registration design consumes a minimal amount of FPGA resources.

Resource	Used	Available	% of XC2VP20
Occupied slices	242	9,280	2
4-input LUTs	469	18,560	2
Block RAM	0	444	0
Multipliers	0	444	0

Table 1: Resource usage of registration design

Some example images are presented below. Figure 5 shows an image of a scene captured with a standard TV camera. Figure 6 shows the same scene captured by a high-resolution thermal camera. This image is the reference image, to which the TV image should be registered, however the TV image is capturing a much wider area. Figure 7 shows the TV image from figure 5 after it has been warped to match the thermal image from figure 6. The warping includes an inherent magnification of the desired region. It now shows the same area, and common features are in the same relative position.

5 Conclusion

This report has presented a simple method of performing a spatial warp of an image as it is read from an off-chip SDRAM frame store into an FPGA. To do so it uses polynomial equations, implemented in System Generator, to generate the pixel locations required in the unregistered image. The warp is predetermined according



Figure 6: Reference image (thermal)



Figure 7: Registered TV image

to manual selection of control points in the unregistered and reference images. A finite state machine is used to generate the necessary sequence of commands to the SDRAM memory. The application is a simple example of separating control and datapath elements of a design, and implementing them with the appropriate tools.

References

- [1] L. G. Brown, "A survey of image registration techniques," *ACM Computing Surveys*, vol. 24, no. 4, pp. 325–376, 1992.
- [2] P. A. Kenny, D. J. Dowsett, D. Vernon, and J. T. Ennis, "A technique for digital image registration used prior to subtraction of lung images in nuclear medicine," *Physics in Medicine and Biology*, vol. 35, no. 5, pp. 679–685, 1990.
- [3] G. Wolberg, *Digital image warping*. IEEE Computer Society Press, 1990.

