



<https://theses.gla.ac.uk/>

Theses Digitisation:

<https://www.gla.ac.uk/myglasgow/research/enlighten/theses/digitisation/>

This is a digitised version of the original print thesis.

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

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

This work 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

Enlighten: Theses

<https://theses.gla.ac.uk/>  
[research-enlighten@glasgow.ac.uk](mailto:research-enlighten@glasgow.ac.uk)

# **Smart Vision in System-on-Chip Applications**

Cade Cenric Wells MEng (Wales)

A thesis submitted to

The Universities of

Edinburgh

Glasgow

Heriot-Watt

Strathclyde

for the Degree of

**Doctor of Engineering in System Level Integration**

Copyright © Cade Cenric Wells 2005

ProQuest Number: 10754010

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10754010

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code  
Microform Edition © ProQuest LLC.

ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 – 1346



## **Abstract**

In the last decade the ability to design and manufacture integrated circuits with higher transistor densities has led to the integration of complete systems on a single silicon die. These are commonly referred to as System-on-Chip (SoC). As SoCs processes can incorporate multiple technologies it is now feasible to produce single chip camera systems with embedded image processing, known as Imager-on-Chips (IoC). The development of IoCs is complicated due to the mixture of digital and analog components and the high cost of prototyping these designs using silicon processes. There are currently no re-usable prototyping platforms that specifically address the needs of IoC development.

This thesis details a new prototyping platform specifically for use in the development of low-cost mass-market IoC applications. FPGA technology was utilised to implement a frame-based processing architecture suitable for supporting a range of real-time imaging and machine vision applications. To demonstrate the effectiveness of the prototyping platform, an example object counting and highlighting application was developed and functionally verified in real-time. A high-level IoC cost model was formulated to calculate the cost of manufacturing prototyped applications as a single IoC. This highlighted the requirement for careful analysis of optical issues, embedded imager array size and the silicon process used to ensure the desired IoC unit cost was achieved. A modified version of the FPGA architecture, which would result in improving the DSP performance, is also proposed.

# Acknowledgments

I would like to thank:

- STMicroelectronics, Edinburgh and the EPSRC for their financial support during the doctoral research.
- My academic supervisor, Dr David Renshaw, for his support and guidance during the last four years.
- My secondary academic supervisor, Dr Iain Lindsay.
- My industrial supervisor and mentor, Ed Duncan, for his encouragement and advice during the doctoral research and the publication of papers.
- STMicroelectronics Imager Development Director, Graham Townsend, for his support during the research project.
- The STMicroelectronics Imager IP design team: Marie Torrie, Andy Kinsey, David Grant, Gary McGillivray, Brian Paisley, William Mair, David Storrar and Ewen Brown for the help provided using development tools.
- The STMicroelectronics Hardware design team: Martin Turner, Jonathan Adler and Russell Simpson for their guidance on PCB design.
- The STMicroelectronics Image Sensor design team: Dr Graeme Storm, Dr Mathew Purcell, William Holland for their advice on CMOS image sensor design issues.
- My wife, Claire, for her everlasting encouragement, understanding and for proof-reading my Thesis.

## **Declarations**

Some parts of the work presented in this thesis have been published in the following articles:

1. Wells CC, Duncan E, Renshaw D. An FPGA based prototyping platform for imager on chip applications. Proceedings of the IEEE International Conference on Field Programmable Technology (FPT04). 2004 Dec 6-8; Brisbane, Australia. p307-310.
2. Wells CC, Duncan E, Renshaw D. A model for imaging system-on-chip manufacturing costs. Proceedings of the IEEE International Symposium on System-on-Chip (SoC 2004); 2004 Nov 16-18; Tampere, Finland. p53-56.

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
<b>2</b>	<b>Literature Review .....</b>	<b>7</b>
2.1	History of Machine Vision and Related Image Processing .....	7
2.2	Current Market and Applications.....	8
2.3	Emerging and Future Applications .....	11
2.4	Board-Level Integrated Smart Vision Systems.....	13
2.5	System-Level Integrated Smart Vision Systems.....	15
2.5.1	Biological and Architecturally-Implicit Processing based Vision Sensors 16	
2.5.2	On-Chip Explicit Computational Processing Based Vision Sensors .....	19
2.5.2.1	Mesh Processing Architectures .....	20
2.5.2.2	Linear Processing Array Architectures .....	25
2.5.2.3	Frame Processing Architectures.....	29
2.5.3	Summary .....	34
2.6	Vision System Prototyping .....	37
2.7	Summary .....	38
<b>3</b>	<b>System Requirements, Analysis and Specifications .....</b>	<b>41</b>
3.1	Requirements .....	41
3.2	Specifications .....	43
3.2.1	FPGA Backplane Selection.....	43
3.2.2	Daughter Board Analysis and PCB Component Selection .....	45
3.2.3	System Buses Analysis and Specification .....	49
3.2.4	Detailed IMPBUS Specification .....	51
3.2.5	FPGA Design Analysis and Specification .....	53
3.2.6	System Firmware .....	61
3.2.7	Cost Issues.....	62
3.3	Summary .....	63
<b>4</b>	<b>FPGA System Design, Test and Results.....</b>	<b>64</b>
4.1	FPGA Design Flow.....	64
4.2	FPGA Top Level Mapping .....	65
4.3	Core Architecture .....	66
4.3.1	Register Bank .....	67
4.3.2	Memories .....	69
4.3.3	SDRAM Decoder .....	70
4.3.4	Sensor Interface.....	73
4.3.5	Video Generator .....	76
4.3.6	Ping-pong Unit.....	80
4.3.7	System Control Unit.....	83



4.3.8	Network Control .....	87
4.4	DSP IP Block Library .....	89
4.5	Simulation and Functional Verification .....	101
4.6	System Programming .....	103
4.7	FPGA Conversion to SoC .....	104
4.8	Results .....	105
4.9	Summary .....	111
<b>5</b>	<b>IoC Manufacturing Cost Modelling .....</b>	<b>112</b>
5.1	CMOS Imager Area Issues .....	112
5.2	Cost Model .....	114
5.3	Application of Model to Example IoC .....	116
5.4	Summary .....	119
<b>6</b>	<b>Discussion and Conclusions .....</b>	<b>120</b>
6.1	Discussion .....	120
6.1.1	PCB Work .....	121
6.1.2	FPGA Work .....	123
6.1.2.1	The Instruction Set .....	123
6.1.2.2	The DSP IP Block Library .....	127
6.1.2.3	Available Processing Time .....	129
6.1.2.4	FPGA Resource Usage .....	130
6.1.2.5	Minimum Object Size and Maximum Object Velocity .....	133
6.1.2.6	Processing Architecture Comparison with Other Systems .....	135
6.1.3	The Complete Prototyping System .....	136
6.1.4	IoC Manufacturing Cost Modelling .....	140
6.2	Recommendations for Future work .....	142
6.3	Example Re-Design of Prototyping Platform .....	145
6.3.1	Scalability of Current Platform to Higher Image Resolutions .....	145
6.3.1.1	Daughter Board and FPGA Backplane .....	146
6.3.1.2	FPGA Memory Sub-Systems .....	146
6.3.1.3	Sensor Interface, Video Generator and Ping-pong Unit .....	151
6.3.1.4	System and Network Control Units and DSP IP Blocks .....	152
6.3.1.5	Available Processing Time .....	153
6.3.1.6	Summary .....	154
6.3.2	Re-designed Architecture .....	154
6.3.2.1	Overview of New Architecture .....	155
6.3.2.2	Memory Sub-system .....	158
6.3.2.3	Microcontroller & I <sup>2</sup> C Interface .....	162
6.3.2.4	Ping-Pong Unit .....	165
6.3.2.5	Configurable DSP block .....	167
6.3.2.6	Bus Network Controller .....	172
6.3.3	Analysis of Suggested New Architecture .....	173
6.3.3.1	DSP Performance .....	174
6.3.3.2	Cycles Available for DSP .....	174

6.3.3.3	Microcontroller Instruction Execution Speed and Code Size.....	176
6.3.3.4	Pin Count, FPGA Logic Cell and Memory Usage.....	178
6.3.3.5	Summary .....	182
6.3.4	Colour Processing .....	182
6.4	Conclusions.....	184
<b>References .....</b>		<b>186</b>
<b>Appendix A.....</b>		<b>201</b>
<b>Appendix B.....</b>		<b>202</b>
B.1	Design and Fabrication Process Overview.....	202
B.1.1	Schematic Entry .....	202
B.1.2	Layout and Routing .....	205
B.1.3	PCB Manufacture .....	206
B.1.4	Component Population .....	206
B.1.5	Functional Testing.....	207
B.2	Results .....	208
B.3	Summary .....	211
<b>Appendix C.....</b>		<b>212</b>

## List of Figures

Figure 2.1 Machine Vision Revenue Generated by Geographic Region in 2002.....	9
Figure 2.2 Major end-user industries for machine vision in 2002 by revenue, Europe (left) and North America (right).....	9
Figure 2.3 Adaptation of image sensors to lighting, conventional (left) and extended dynamic range (right).....	11
Figure 2.4 Micrographs of three different CMOS fovea image sensors, (a) a disjoint fovea array and peripheral ring structure, (b) a disjoint fovea array and peripheral array and (c) a continuous fovea and retinal ring structure .....	18
Figure 2.5 Architecture of S3PE .....	20
Figure 2.6 Architecture of (a) Single APE (left) and (b) the SCAMP architecture with a single APE marked (right).....	23
Figure 2.7 Architectures of (a) IVP MAPP2500 (left) and (b) Chen et al.'s PASIC (right) .....	26
Figure 2.8 Top-level architecture of the Philips Xetal Vision Chip .....	27
Figure 2.9 Top-level architecture of Ni and Guan Smart CMOS Image Sensor .....	28
Figure 2.10 Top-level architecture of Fang et al.'s Smart Vision SoC.....	30
Figure 2.11 Top-level architecture of Neuricam's VISoc based smart camera .....	31
Figure 2.12 The three approaches to general purpose processing on vision chips ....	34
Figure 2.13 Top level block diagram of Ateame Digital Media Evaluation Kit 6414	38
Figure 3.1 STMicroelectronics Backplane.....	44
Figure 3.2 Top-Level Daughter Board Bus Diagram .....	51
Figure 3.3 FPGA memory address map.....	55
Figure 3.4 Top-level IP block architecture in the FPGA .....	59
Figure 4.1 Top-level pin mappings for the FPGA .....	66
Figure 4.2 The three phases of development for the core architecture .....	67
Figure 4.3 Single instance of an example 8-bit register from the register bank .....	69
Figure 4.4 Co-processor's SDRAM read (top) and write (bottom) timing requirements.....	72
Figure 4.5 Sensor VGA image frame and timings at 25FPS .....	75
Figure 4.6 Bayer colourisation pattern used in STMicroelectronics' colour image sensors.....	76
Figure 4.7 ITU-R BT.656 PAL video timing requirements for a 27 MHz clock .....	78
Figure 4.8 Control mechanism for Video generator's line memory and FIFO .....	80
Figure 4.9 Top-level block diagram of Video generators FSM.....	80
Figure 4.10 Test patterns visible for test_sel=1 (left) and test_sel=2 (right).....	83
Figure 4.11 Top-level interfaces of system control unit .....	84
Figure 4.12 I/O interface for DSP IP co-processor blocks .....	90
Figure 4.13 3x3 neighbourhood filter operation on an image.....	94
Figure 4.14 Sobel vertical mask (left), input Lenna image (middle) and processed image (right).....	94
Figure 4.15 Representation of the 3x3 filter's datapath .....	96
Figure 4.16 Preference of connected pixels in Getobjs algorithm .....	97
Figure 4.17 Object database item format .....	97
Figure 4.18 Getsobjs algorithm flow chart .....	99
Figure 4.19 Binary input image (top) and processed image using Getobjs Matlab script (bottom).....	100
Figure 4.20 Functional verification and simulation flow.....	102

Figure 4.21 Programming method for the STV0674 co-processor.....	104
Figure 4.22 Three types of available time slot for uninterrupted image processing	107
Figure 4.23 Distance an object must travel across the lens field of view .....	109
Figure 5.1 Typical CMOS imager architecture.....	112
Figure 5.2 An example IoC to support the demonstration application .....	117
Figure 5.3 Manufacturing cost graph for example IoC.....	118
Figure 6.1 Clock cycles required to process each pixel using the 3x3filter DSP IP block .....	128
Figure 6.2 Methods of reordered pixel sub-sampling for improve performance.....	139
Figure 6.3 Two different FPGA memory maps taking into account the use of off-chip SDRAM .....	150
Figure 6.4 Reordered FPGA memory map .....	151
Figure 6.5 New proposed FPGA architecture.....	157
Figure 6.6 I/O interface for the SDRAM controllers .....	160
Figure 6.7 New proposed FPGA memory map.....	161
Figure 6.8 DP8051 microcontroller sub-system .....	163
Figure 6.9 DP8051 Memory Map .....	164
Figure 6.10 New DSP sub-system .....	167
Figure 6.11 DSP address generator I/O interface .....	168
Figure 6.12 Example DSP pipeline configuration .....	170
Figure 6.13 I/O interface for new DSP IP blocks .....	170
Figure B.1 Peak current at rising clock edge .....	204
Figure B.2 Sub system locations on topside of PCB .....	210
Figure B.3 A photograph of the complete PCB attached to the FPGA backplane ..	210
Figure C.1 Screenshots of a coffee mug and Dilbert toy detected and highlighted by the system.....	213

## List of Tables

Table 2.1 Execution times of sample programs for the SPARSIS architecture.....	21
Table 2.2 Time of execution of several algorithms on the SCAMP vision chip .....	24
Table 2.3 Comparison of Neuricams VISoc and proposed SmartPupilla.....	32
Table 2.4 Comparison of the three approaches to processing in vision chips .....	34
Table 3.1 Interconnections between sub-systems, DSP IP blocks and memories on the FPGA.....	60
Table 4.1 Minimum set of I/Os for the FPGA Register Bank .....	68
Table 4.2 Minimum register map for FPGA Register Bank .....	69
Table 4.3 Top-level memory interface to system architecture for an Image bank ....	70
Table 4.4 I/O interface for the SDRAM decoder .....	71
Table 4.5 SDRAM commands and associated signals to start read and write transactions.....	72
Table 4.6 I/O interface for the sensor interface .....	74
Table 4.7 I/O interface for the Video generator .....	77
Table 4.8 SAV and EAV code sequence .....	79
Table 4.9 I/O interface for the Ping-pong unit.....	81
Table 4.10 I/O interface for the system control unit .....	84
Table 4.11 Instruction list supported by prototyping system.....	85
Table 4.12 Supported operations by DSP block library.....	92
Table 4.13 Instruction execution time.....	105
Table 4.14 Processing times and cycles per pixel for the DSP IP block library.....	106
Table 4.15 Resource usage by FPGA system component .....	108
Table 4.16 FPGA memory utilisation .....	109
Table 4.17 Maximum object velocity in m/s to still guarantee detection .....	109
Table 4.18 Minimum detectable object size .....	110
Table 5.1 Values used for cost model .....	117
Table 5.2 Area size for imager and complete IoC .....	118
Table 5.3 Cost of the IoC at a given process and unit quantity .....	118
Table 6.1 Worst case instruction execution time .....	123
Table 6.2 Improved instruction worst-case execution time .....	126
Table 6.3 LC usage and percentage of total FPGA LC by IP block.....	132
Table 6.4 Number of LCs used by each sub-system type of the FPGA architecture .....	133
Table 6.5 Platform's Memory Requirements (in Bytes) to Support Higher Image Sizes .....	146
Table 6.6 Platform's Memory Requirements to Support Higher Image Sizes .....	147
Table 6.7 Interconnect Address Scheme.....	169
Table 6.8 Shared memory access priorities allocation.....	172
Table 6.9 Number of 96MHz clock cycles to perform an SDRAM Read and Write .....	176
Table 6.10 Logic Element and ESB usage in the new architecture .....	179
Table A.11 IMPBUS pin mappings for the STV0674 .....	201
Table C.1 Application code for object count and highlight demonstration .....	214

## List of Equations

Equation 2.1 Dynamic Power Dissipated by CMOS logic .....	35
Equation 4.1 The calculation of protection bits P0 to P3.....	79
Equation 4.2 Threshold equation .....	92
Equation 4.3 Copy equation.....	93
Equation 4.4 Getcoords Equations.....	93
Equation 4.5 Absdiff equation .....	93
Equation 4.6 Equation for the spatial domain neighbourhood operator 3x3filter .....	93
Equation 4.7 Maximum number of cycles to perform Threshold DSP operation ...	106
Equation 4.8 Maximum number of cycles to perform Filter3x3 DSP operation .....	106
Equation 4.9 Maximum number of cycles to perform Rectangle operation .....	106
Equation 4.10 Maximum number of cycles to perform Getcoords operation .....	106
Equation 4.11 Maximum number of cycles to perform Absdiff operation.....	107
Equation 4.12 Maximum number of cycles to perform Copy operation .....	107
Equation 5.1 IoC system cost model.....	114
Equation 5.2 Die cost calculation .....	114
Equation 5.3 Die yield model .....	114
Equation 5.4 Area of die .....	115
Equation 5.5 Area of sensor .....	115
Equation 5.6 Area of pixel array .....	115
Equation 5.7 Area of digital logic.....	115
Equation 5.8 Area of analog address units.....	115
Equation 5.9 Area of miscellaneous analog components .....	116
Equation 5.10 Inter-process scaling factor.....	116
Equation 6.1 Maximum cycles available for processing per image frame .....	175

## List of Abbreviations

<b>A/D</b>	Analog-to-Digital
<b>ADC</b>	Analog-to-Digital Converter
<b>ALU</b>	Arithmetic Logic Unit
<b>ASIC</b>	Application-Specific Integrated Circuit
<b>ATM</b>	Automatic Teller Machine
<b>CAS</b>	Column Address Select
<b>CCD</b>	Charge-Coupled Device
<b>CIF</b>	Common Image Format
<b>CMOS</b>	Complementary Metal-Oxide Semiconductor
<b>DAC</b>	Digital-to-Analog Converter
<b>DRC</b>	Design Rule Checks
<b>DSP</b>	Digital Signal Processing
<b>EAV</b>	End of Active Video
<b>EEPROM</b>	Electrically erasable and programmable ROM
<b>EOF</b>	End Of Frame
<b>ESB</b>	Embedded System Block
<b>FIFO</b>	First In First Out
<b>FPGA</b>	Field Programmable Gate Array
<b>FPS</b>	Frames Per Second
<b>GLU</b>	Global Logic Unit
<b>GOPS</b>	Giga-Operations Per Second
<b>GPIO</b>	General-Purpose Inputs and Outputs
<b>GPMV</b>	General-Purpose Machine Vision
<b>I/O</b>	Inputs and Outputs
<b>IC</b>	Integrated Circuit
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>IoC</b>	Imager-on-Chip
<b>IP</b>	Intellectual Property
<b>LC</b>	Logic Cells
<b>LE</b>	Logic Elements
<b>LED</b>	Light Emitting Diode
<b>LPA</b>	Linear Processing Arrays
<b>LUT</b>	Look-Up Table
<b>MAC</b>	Multiply and Accumulate
<b>MCM</b>	Multi-Chip Module
<b>MIPS</b>	Million Instructions Per Second
<b>MPW</b>	Multi-Project Wafer
<b>NASA</b>	National Aeronautics and Space Administration
<b>NLU</b>	Neighbourhood Logic Unit
<b>NTSC</b>	National Television System Committee
<b>OCNN</b>	Optimisation Cellular Neural Network
<b>PAL</b>	Phase Alternation Line
<b>PBD</b>	Platform-Based Design
<b>PCB</b>	Printed Circuit Board
<b>PCI</b>	Peripheral Component Interconnect
<b>PDA</b>	Personal Digital Assistant
<b>PE</b>	Processing Element

<b>PLL</b>	Phase-Locked Loop
<b>PLU</b>	Pixel Logic Unit
<b>RAM</b>	Random Access Memory
<b>RISC</b>	Reduce Instruction Set Computer
<b>ROI</b>	Region Of Interest
<b>ROM</b>	Read Only Memory
<b>SAV</b>	Start of Active Video
<b>SDRAM</b>	Synchronous Dynamic Random Access Memory
<b>SIMD</b>	Single-Instruction Stream Multiple-Data Stream
<b>SoC</b>	System-on-Chip
<b>SRAM</b>	Static Random Access Memory
<b>UART</b>	Universal Asynchronous Receiver-Transmitter
<b>USB</b>	Universal Serial Bus
<b>VGA</b>	Video Graphics Array
<b>VLSI</b>	Very Large-Scale Integration
<b>WSTS</b>	World Semiconductor Trade Statistics



# 1 Introduction

The visual system forms an important part of the human sensory system. The capacity of the human eyes to adapt to selectively focus on areas of interest in the field of view provides an effective means to collect visual information. The connection of the eye to the brain enables further processing to be performed and comparisons to be made with previous observations using relevant prior knowledge. This combined effort results in the valuable ability to process, analyze and interpret the world around us. Examples demonstrating our flexibility to interpret different situations include; the subtleties of lip reading to the ability to strike a fast moving cricket ball with a bat. These seem relatively simple tasks, given enough practice, and are taken for granted but in almost all cases the complexity of the processing required is not apparent. It is only when these everyday tasks are analyzed in detail that their mathematical complexity is revealed.

“Machine vision is concerned with the automatic interpretation of images of real scenes in order to obtain information and thereby to control or monitor machines or processes” [1]. Despite the complex nature of most vision tasks there are several advantages of using machine vision system rather than a human operator, namely consistency, precision, cost, flexibility and operational speed. Whilst there are a wide range of applications that exist for computer vision to be applied to, there are usually 4 stages common to all, these are:-

1. Acquisition – obtain image from camera.
2. Processing – apply algorithms to enhance parts of the image that are of interest.
3. Feature extraction – identify and quantify important features.
4. Decision and control – make a decision given a set of known data or rules and then output a control signal to change a process or output relevant information.

An example of a typical application is the identification of manufacturing faults for items on a production line. An algorithm is implemented to compare a snapshot

image of each item with a stored image of a known defect free item. This is often referred to as comparison with a golden reference. If one or more of the visual characteristics of the item differ from the golden reference the production line supervisor can be notified and the issue addressed.

The last 20 years has seen an increase in the adoption of machine vision systems. In 2003 the European vision market was estimated at \$420 million and the North American vision market at \$1.6 billion [2,3]. In 2004, the world machine vision market was estimated at \$6.6 billion [3].

STMicroelectronics holds a world leading position in CMOS image sensors, with primary markets in web cameras and digital stills cameras. There is an expectation that an ever increasing proportion of future mainstream imaging applications will require some form of embedded vision processing. Examples of this kind of embedded processing include the ability to track or recognize objects in real-time, thus enabling numerous possible applications, such as, airbag deployment or security via face recognition. Embedded vision processing differs from traditional vision processing in the sense that some or all of the processing is performed within the camera unit rather than on a separate host computer. The main advantage of this partitioning of processing is that there is less latency between capturing an image and its interpretation. It also reduces the demands of communication bandwidth between camera and any connected external systems as the data transmitted is usually a control signal rather than a stream of images. Systems based on this processing paradigm are called smart cameras.

In the last decade the increasing ability to integrate complete systems onto a single die has resulted in the manufacture of very dense integrated circuits comprising of over 100 million transistors [4, 5, 6]. These complex system-level integrated circuits are commonly referred to as System-on-Chip (SoC). As SoC design flows can incorporate multiple technologies into a single package, it has become feasible to produce single chip camera systems. In this thesis these SoC camera systems are defined as Imager-on-Chip (IoC). The imaging and machine vision industry has recognised the opportunity to configure IoCs to perform vision tasks. As a result, IoC

based autonomous smart camera systems are gradually becoming commercially available [7].

Unfortunately, while the geometry of transistors is continuing to shrink in line with Moore's Law, allowing more complex circuits to be realized, the ability to design new devices based on these new silicon technologies is not improving at the same rate. It is generally accepted that this design capability gap is steadily increasing due to the technology improving by 60% a year while design methodology improves by only 20% [8]. Therefore without dramatically increasing the size of design teams, the design and verification time for new products will also increase with SoC complexity. To address this issue, companies in the mid 1990s started adopting design re-use programmes. In such a programme, new design components are created with re-use in mind by parameterising interfaces and functionality. These flexible design components are often referred to as virtual components or Intellectual Property (IP) blocks. IP blocks are assembled together within a database system to provide system designers with a library of pre-verified and well-documented re-usable designs for future projects. It is generally accepted that the benefits of using such a methodology are mid to long term given that an IP block for re-use can require 10 to 100 times the development effort as a design for one off use [9]. Traditional software design flows have also been changed as it is generally no longer possible to develop the software once design, manufacture and debugging of the hardware has occurred. As a result, hardware and software must be co-developed adding to the complexity of the overall system design flow. The requirement for co-design has led to the extension of re-use programmes to include a platform-based design (PBD) methodology.

Platform-based design is an evolutionary step in IP re-use. Several different definitions for a platform have been summarised in recent literature [10]. PBD can be generically defined for this body of work as the use of a defined architectural platform with a library of associated pre-qualified hardware and software IP, which can be configured in a variety of forms to meet the requirements of a new product. These platforms have predominantly been domain specific in order to provide system optimisation for the required range of possible derivative products, such as for multimedia or wireless applications [11].

Traditionally, prototyping platform-based designs would have initially involved manufacturing multi-project wafers (MPW) at the intended or older lower cost silicon process. These MPW consist of several different circuit designs on a single silicon substrate and hence manufacturing costs are apportioned over all the projects that have submitted a design for prototyping. Unfortunately the viability of this approach has reduced with the shift to processes with feature sizes of  $0.1\mu\text{m}$  or less and given the low volumes of ICs fabricated. At these small feature sizes the cost of MPW runs increases dramatically when compared to MPW run at around  $1\mu\text{m}$  feature size. A further complication is caused by the slots available for MPW runs in fabrication plants, as usually these need to be booked in advanced. Even a large company with in-house fabrication facilities may have to accept long turn-around times due to extra steps such as packaging of ICs. These facets to multi-project wafer production for the purpose of early prototyping can further exacerbate the existing time-to-market pressures and project costs for a new product.

Fortunately, the improvements to silicon technologies have allowed programmable logic based devices to realize large capacities. This has occurred to such a degree that the latest 90nm process Field-Programmable Gate Arrays (FPGAs) have up to 10Mbits of programmable memory and approximately 200K programmable logic cells, providing millions of useable gates [12]. As a result, many companies use FPGA technology to perform real-time or near full clock rate functional verification of new platform designs using these large devices. Indeed, Xilinx is now marketing several platform FPGAs not only for verification purposes but as a replacement for platform-based application specific integrated circuits (ASIC) for high value and medium to low volume applications [13]. The key advantage of these FPGAs is their ability to be rapidly re-programmed, providing turn-around times of hours compared to the several weeks required by MPW runs. The approximate cost for these large capacity FPGAs is in the region of \$1K to \$10K which compares favourably with a typical cost of a typical MPW run. For example a  $25\text{mm}^2$  die size using UMC processes through Europractice costs \$17K at  $0.25\mu\text{m}$  feature size and \$85K at a  $0.13\mu\text{m}$  feature size [14].

The disadvantage of the utilisation of FPGA technology is apparent when attempting to prototype IoC architectures. The analog components of CMOS and charge-coupled device (CCD) sensors cannot be mapped onto the array of logic elements found in current FPGAs. Even the latest Field Programmable Analog Arrays (FPAA) are unsuitable due to the fact the high density structures essential for the instantiation of the photosensitive array cannot currently be provided.

The aim of this research was to investigate and develop a new prototyping platform for low-cost mass-market IoC applications. This was achieved by designing a system architecture suitable for supporting a wide range of applications. A library of image processing IP blocks was created to interface both with this system architecture and future ST Imaging products. To calculate the cost of manufacturing a prototyped application as a single integrated circuit, a high-level IoC cost model was formulated.

In this thesis, it is proposed that a frame-based processing architecture is the most applicable means for image processing, when prototyping IoC applications. This architecture has the distinct advantage of separating the processing elements from the CMOS image sensor. This is unlike mesh-based or linear processing arrays, which are generally tightly coupled with an image sensor. Each architectural component can be developed and tested independently as an IP block before integrating the complete system into an IoC. Custom IP blocks can also be functionally verified in real-time using reusable FPGA technology and optimised for specific applications.

The thesis is organised as follows. Chapter 2 reviews research performed in the field of vision system design and prototyping systems, with particular emphasis on single chip vision system implementations. A summary of the image processing and machine vision market and current, emerging and future applications driving current research is also presented. Chapter 3 details the system requirements for the types of IoC that are envisaged to be prototyped using the developed platform. In addition, the system partitioning scheme used and system specifications are defined. The printed circuit board (PCB) design, fabrication and test flows for new systems are described in Chapter 4, followed by the FPGA-level hardware and software design in Chapter 5. An outline of an example application is provided in Chapter 6. An IoC

manufacturing cost model is explained in Chapter 7 and applied to a single chip system that could support the example application. Chapter 8 critically discusses the results obtained with a conclusion provided in Chapter 9, including suggestions for future work.

## **2 Literature Review**

As the research would encompass several different topics a broad literature search was initially performed. A brief history of the uses of image processing and machine vision was obtained followed by an assessment of the current market. Emerging and future applications were investigated in addition to current applications of these technologies. Three specific areas were then studied, namely; smart camera implementations, highly-integrated vision IoC and prototyping system design. This provided the necessary understanding of the relevant fields in order to address the architectural design of the new IoC prototyping platform.

### **2.1 History of Machine Vision and Related Image Processing**

The progress of machine-vision research can be separated into 3 stages [15]. These three stages demonstrate that machine vision was driven by available computing power. The first stage started in the 1970s with the main applications area of factory automation. General-purpose mainframe computers were linked to image acquisition devices, however their limited processing power and high cost made it very difficult to justify their large scale use outside high-value markets, such as PCB and semiconductor production [16]. As a result, their penetration into industry was limited [17]. The research at this time, was performed in areas of detecting positions, shapes and defects using windowing, pattern matching and feature extraction techniques. Computing power limited most of the processing to binary images. Increasing computing power with a reduction in system cost led to the start of the second stage in the 1980s. The advent of Reduced Instruction Set Computer (RISC) based systems meant grey-level image processing could now be applied to image processing and vision problems. Office automation was now becoming feasible with many applications, for example, ATMs, mail sorting machines and document readers. Research primarily focused on text and image recognition, using technologies such vector conversion, data structuring and context analysis. The third and current stage started in the 1990s. According to Ejiri the main application area in the 1990s was social automation which included the domains of traffic

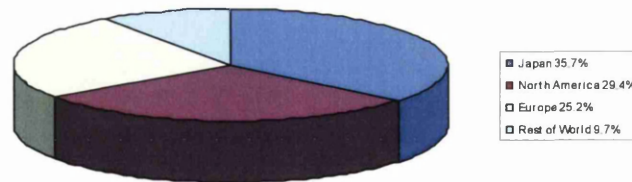
management, communications, welfare, medical systems and the environment [15]. The vision research topics at that time included; biometric identification/ recognition, abnormality monitoring and behaviour understanding using real-time video analysis, sensor fusion and networked machine vision systems. This has been realized by the introduction of more embedded systems and systems that are capable of colour processing.

Many technology companies continue to manufacture vision systems focused on narrow niche applications where repeat business is possible [17]. To survive in such niche markets usually requires these companies to have global market penetration. It has been noted recently however that with availability of low cost embedded vision systems, or ‘smart’ cameras, machine vision is entering new manufacturing areas where previously the technology would have been too expensive to implement [18]. If this trend is to continue and for machine vision to migrate into other new applications areas, uptake of greater levels of integration, programmable processors and use of technologies such as CMOS image sensors will be required to provide flexible systems with greater processing power at increasingly lower costs.

## **2.2 Current Market and Applications**

In 2002, the worldwide machine vision was estimated at \$5.2 billion [19]. This was the first significant decline, in recent history, from the previous year and was attributed to the worldwide recession at that time. The worst impacted sales figures were from the semiconductor industry but these still attributed to over a third of the total revenue for Europe and North America. The effects of this decline were limited by the growth in the adoption of general-purpose machine vision systems for manufacturing, which were sold to a wide range of end-user industries. At this point in time, the percentage of the total world machine vision revenue generated by geographic region was as show in Figure 2.1.

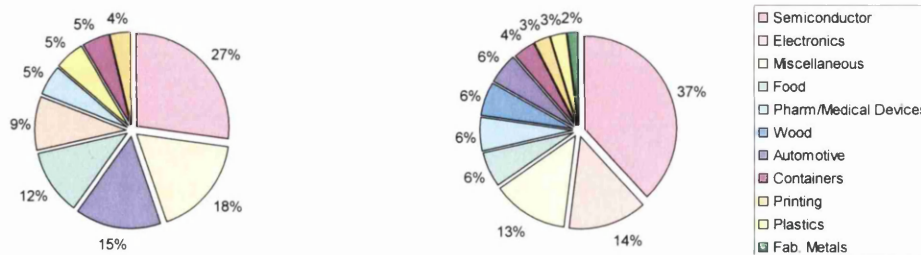




**Figure 2.1 Machine Vision Revenue Generated by Geographic Region in 2002 [19]**

The machine vision market has steadily continued to grow throughout 2004. This revenue however, has not yet matched that generated in 2000. The number of units shipped has increased over the last two years by 27 % resulting in a gain of 13% in revenues. This has led to sales of \$6.6 billion. This is mainly attributed to the increased use of machine vision in the Pacific Rim region, in particular Japan and China [20]. The large discrepancy between revenue accumulated and units shipped, can be explained by the increased adoption of lower cost smart cameras, vision sensors and embedded vision processors. A clear example of this can be seen in the European market in 2003. Smart cameras accounted for an estimated 10250 of a total of 30500 general purpose vision system units shipped. This only accounted for less than 6% of the \$261.5 million general purpose machine vision market [2]. Forecasts expect approximately a 9% annual growth in revenue through to 2008 [20]. This predicted due to smart cameras entering new markets currently not addressed by current vision systems and by evidence that the worldwide downturn in capital equipment expenditure finished in 2003.

Revenues generated in 2002 from the major end-user industries can be seen below for Europe and North America in figure 2.2. Europe's figures for wood and metal fabrication industries were not available.



**Figure 2.2 Major end-user industries for machine vision in 2002 by revenue, Europe (left) and North America (right) [19]**

As can be seen in both charts above, the end-user industries for machine vision are indeed varied. Though appealing to focus on one particular market segment e.g. the semi-conductor industry representing 37% (North America) and 27% (Europe), this can be dangerous given the volatile nature of some of the end-user industries. Traditionally, the semiconductor industry followed a 4 yearly boom-bust cycle whilst displaying a compound annual growth rate of approximately 17%. Evidence indicates however that this is unsustainable in the long-term and that the growth rate is beginning to slow [21]. Recent figures by the Semiconductor Industry Association (SIA) and research organizations, such as Gartner Inc. and the World Semiconductor Trade Statistics (WSTS) group, suggest that there will be another decline in 2006 [22, 23, 24]. However, some forecast that the boom-bust cycles are no longer predictable and the period of time between a boom and bust is decreasing, hence leading to a potentially more volatile market.

Less semiconductor fabrication facilities are likely to be built by individual companies since the average cost of a state-of-the-art plant has risen from \$0.7 billion in 1994 to \$3 billion in 2003 [25]. It is also estimated that a company would require annual revenues of over \$6 billion to support the latest 300mm wafer technology plants, leaving only a select few able to run such a plant without partnerships with other semiconductor companies [26, 27]. This is unlikely to improve if the trend over the last 10 years reported by a Goldman Sachs continues, where build costs are increasing by a factor of 7 while the semiconductor industry's revenues have grown fivefold [25]. Despite these problems, there is still a current transition to fabrication plants capable of processing 300mm wafers and this is likely to require increasing numbers of automated materials handling systems [28]. This transition is expected to be a contributor to a 55% rise in capital expenditure in 2004 [29]. Although the long term prospects for the semiconductor market are uncertain, the rise in capital expenditure could lead to a short term increase in revenue for machine vision systems in this sector.

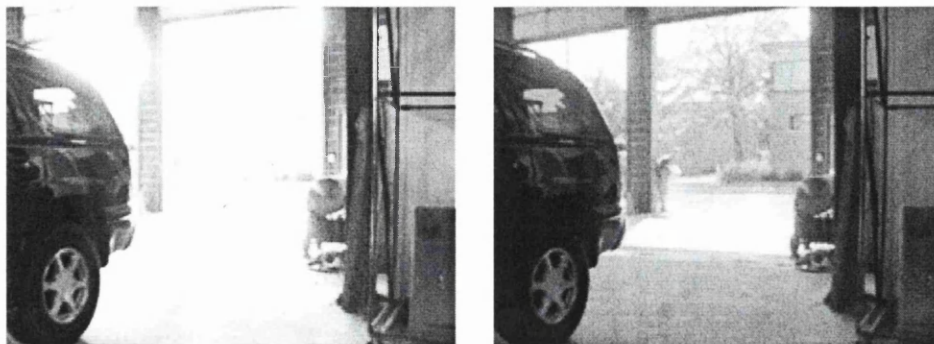
As has been demonstrated in the semiconductor industry, relying on a single market for income when developing machine vision systems can be a great financial risk. The lack of flexibility within these machine vision systems would lead to a product which is unsuitable for other markets. Hence, developing general purpose machine

vision systems results in a product that is flexible and independent of one single market.

### 2.3 Emerging and Future Applications

Current applications for general purpose machine vision systems (GPMV) are found in a wide range of industries. Typically the majority of these have been industrial inspection applications, for example, grading, sorting, fault detection and automated handling. Security applications have also benefited for GPMV systems and have included, intruder monitoring and vehicle registration plate identification. One particular aspect which is limiting the uptake of machine vision is the typical system cost. The average cost of a GPMV system in Europe in 2003 was approximately \$10K. While this does not indicate the range of prices it does possibly indicate the typical high-cost of a GPMV system [2]. Understandably, this high cost is a concern for industries looking to use general machine vision as a way of adding value to a low-cost product, rather than as a cost reduction tool or essential addition.

CMOS image sensor technology has improved and resulted in the expansion of GPMV systems into new emerging applications. Extended dynamic ranges of around 120dB for logarithm photo transfer-based image sensors is facilitating their use in outdoor applications where lighting can not be controlled [30]. Figure 2.3 below demonstrates how linear photo transfer-based conventional imagers cannot adapt to spatially varying lighting conditions.



**Figure 2.3 Adaptation of image sensors to lighting, conventional (left) and extended dynamic range (right) [31]**

Further efforts to extend the dynamic range of CMOS imagers have included the amalgamation of logarithmic and linear mechanisms into a single imager to provide a range of up to 140dB [32].

Two market sectors benefiting from this recent technology are transport applications. An example of the application of systems with a high dynamic range is Honeywell Airport Systems camera-guided aircraft-docking system [33]. As a newly arrived aircraft approaches the gate, a monitor on the wall facing the pilot, displays the correct stopping distance to the pilot for the plane to align with jetway. This system has to operate 24 hours a day in all weather conditions as well as be able to cope with reflections and varying lighting conditions. Another example can be found in the automotive industry where the dramatically changing light conditions and speeds of vehicles in motion, require high-performance real-time systems. Wide-spectrum, high-speed camera systems are used with permanent scene illumination from near-infra red headlights. Pedestrians can be detected from over 100m away while oncoming cars with headlights at full-beam can be recorded in the same image [33]. These are yet to become mainstream products due to their high cost. As the cost of vision systems fall, it is expected that the automotive industry will begin the integration of systems for vehicle guidance, collision avoidance, intelligent headlights and smart airbag deployment.

Recent fears of global terrorism have raised interest in the use of machine vision for security purposes. Visual biometrics, such as face or fingerprint recognition, have been a common application of such technology. In particular, camera based surveillance systems are becoming increasingly popular. The increased use of CCTV has required the transition from analog to digital systems capable of being networked together. This has created greater bandwidth requirements. As smart cameras are lower in cost, it is quite likely the industry will start adopting these systems as they provide two main advantages as a result of integrated processing. The processing of images for exceptional events means only images of interest need to be sent to a human operator, hence reducing the need for multiple video screens to be observed simultaneously and reducing the communication bandwidth requirement between camera and operator. Current algorithm research in the area is looking to identify not just when someone should not be present but what their exhibited

behaviour may indicate, by using posture recognition [34]. For example, distinguishing between someone walking along a footpath by a fence and someone climbing over a fence.

A factor which may increase the number of new applications using GPMV systems is the recent proliferation of wireless technology as shown by the uptake of low-cost mobile phones with imagers. To date, this coalescence of technologies has largely been unexplored, yet there are notable exceptions such as the Nokia Observation Camera [35].

## **2.4 Board-Level Integrated Smart Vision Systems**

In the early 1990s, the advent of low cost microprocessors and digital signal processors resulted in the ability to develop board-level integrated vision systems. An Edinburgh University spin-out company, VLSI Vision Ltd (VVL), developed the Imputer, a general purpose smart camera based on their low-cost CMOS imaging technology. Unlike the majority of the solutions available at that time, it integrated the complete system onto a single motherboard and did not require a host computer to operate. Initially, a 256x256 image sensor, frame grabber, frame store, 8-bit 8052 microcontroller and external IO were integrated onto a PCB measuring 100 x 50mm [36]. The Imputer was programmed using the C language and a windows-based software development suite. A large library of image processing functions was also provided. These included correlators, transforms, convolvers, morphological filters, logical operators and image segmentation algorithms. The Imputer also had the option of the addition of a 16-bit Motorola 56002 DSP co-processor. This DSP has the capability to provide up to a 3000 fold performance increase for some image processing algorithms. The Imputer was later updated to replace the sensor, microcontroller and DSP with a single 32-bit Intel i960 and a 512x512 CMOS image sensor. This updated product resulted in similar performance but reduced the design complexity of the systems PCB [37]. The Imputer family of products were used for a range of applications, from people tracking to component inspection and remote vision until it ceased production in the late 1990s when the Motorola DSP was no

longer available [36][38]. VVL was shortly after acquired by STMicroelectronics to enrich their intellectual property portfolio.

The two processing architectural approaches used by the Imputer family of products are still popular with vision system design companies today. The well-known vision company, DVT, uses both approaches in its 500 series of smart cameras. The lower end cameras use general purpose microprocessors, such as the Hitachi SH4, to achieve performance in the range of 60-360 million instructions per second (MIPS) [39]. The latest 550 series utilise the popular Texas Instruments DSPs to provide a boost in processing performance up to 3600 MIPS. Other high-profile smart camera producers, such as Vision Components, Cognex and PPT Vision, tend to opt for solely microprocessor or DSP based product lines. In the case of the Vision Components, the smart camera lines are DSP based using either Texas Instruments or Analog Devices parts to obtain performance in the region of 1200MIPS and 31-375MIPS respectively [40][41][42]. On the other hand, PPT Vision uses a 1000MIPS PowerPC microprocessor for its Impact T series of smart cameras [43].

The other devices used in smart vision cameras typically include FPGAs, ASICs and neural network processors. A specific commercial example of the use of neural networks for vision products is the Pulnix ZiCAM, with its Zero Instruction Set Computer (ZISC) patented technology. Each ZISC processor has 78 neurons with 64byte inputs. These processors can be linked together to perform parallel processing. Currently cameras with up to four ZISC processors are available [44]. Smart cameras that use FPGA technology are usually provided with a microprocessor or DSP, as is the case Wintriss Engineering's 5150-Pixel Line-Scan Vision Processing Camera. The 5150-pixel model combines a FPGA to perform some initial pixel processing followed by image analysis by the Motorola PowerPC [45].

Performing a paper-based detailed analysis on these smart cameras would be difficult as companies typically do not provide comprehensive benchmark results for different image processing functions. This is, in part, due to a lack of common benchmarking techniques and metrics for smart camera systems. Irrespective of this problem, some broad observations can be made, these are;

- A typical smart camera unit with lens assembly has a unit cost in the range of \$2K-10K.
- Most systems use CCD imagers except in the case of the low cost systems which use CMOS imagers.
- On-board RAM is between 2MB and 128MB.
- Resolution available is usually between 640x480 and 1280x1024 pixels.
- Continuous acquisition frame rate span from 10 to 110 frames per second (fps).
- Some use real-time operating systems, such as VxWorks.
- The great majority are aimed at industrial vision applications

An interesting evolution has been in the way in which these smart cameras are set-up for an application. Traditionally, the systems would be programmed in an assembly language or a high-level language, like C, with a supplied set of image processing functions. More recently, smart camera suppliers, namely Pulnix and Cognex, have provided software which learns by example, or uses techniques suitable for non-software engineers such as spreadsheet entry. The vision application developer's perspective of these camera systems may be heavily influenced by this type of product differentiation, given its benefits of less complexity and potentially faster development times, rather than raw instruction processing performance and system cost.

## **2.5 System-Level Integrated Smart Vision Systems**

The next developmental step for smart vision system design has been the integration of the complete system onto a single integrated circuit, also known as vision sensor or vision chip. The emergence this type of system was during the early 1980s. One of the first examples to be designed and implemented on silicon was Richard Lyon's optical mouse IC [46]. This chip used a simple digital motion tracking algorithm to determine the direction and distance traveled by the mouse over a mouse mat with a fixed patterned surface. Most of these early systems were application specific and only capable of spatial processing techniques as a result of the complexity of

implementing delay or on-chip storage for previous images. As the remit of the project includes the use of the sponsor array-based CMOS sensor technology, biological and architecturally-implicit processing based vision sensors will only be briefly covered from a processing point-of-view.

### 2.5.1 Biological and Architecturally-Implicit Processing based Vision Sensors

Vision sensors that use inherent processing provided by biological human or animal vision structures or, indeed, by the underlying sensor technology itself are commonly known as retina-based or fovea-based sensors. These sensors almost exclusively implement spatial image processing functionality from simple local smoothing operations to global object orientation detection.

Retina-based sensors, often referred to as silicon retina, are based on a model of human or animal vision. Research interest has particularly been focused on detecting local or global changes in light intensity and the ability to detect edges. The structures implemented on silicon usually include the use of an approximation of a Gaussian filter. This smoothing filter reduces the noise within the image before any further image processing is performed. Resistive networks are commonly used to create this smoothing effect.

Resistive networks act in an exponential manner by diffusing charge over the complete network depending on the value of the resistors. The simplest example of this is Mahowald and Mead's silicon retina, upon which others have been based [47, 48, 49, 50]. A 1-D version of this was used for matching images for a stereo retina image sensor [51].

Delbrück also used Mahowald and Mead's silicon retina elements for derivative system. This derivative system was designed to use feedback to control a lens assembly to focus the image onto the retina. This was based on the premise that an image will be in focus when there is a maximum difference between itself and its spatially smooth version. Once a value for the maximum difference value had been found, a signal was sent to change the lens control [52]. Standley also showed how



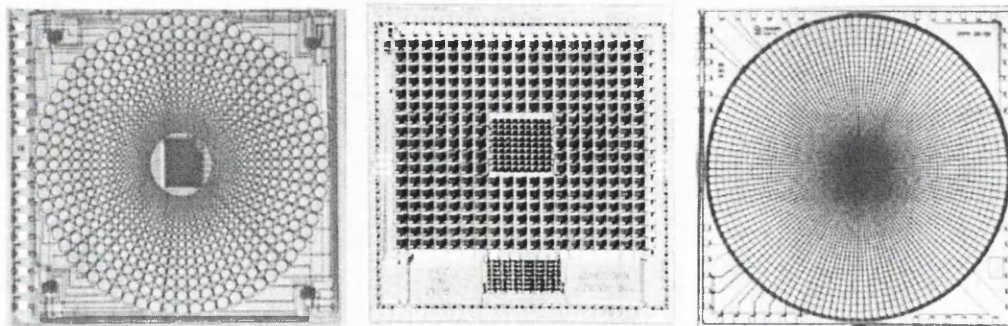
similar resistive networks could be used to detect the position and orientation of an object [53].

Another type of retina sensor technology is based on diffusive networks. Examples of this technology include the Solar Illumination Monitoring Chip by Venier et al. which could measure the azimuth, intensity and elevation of the sun [54]. The layout of the pixels for the system were linear-polar coordinate based. Each pixel's photocurrent was compared to the global average. If the intensity was higher than the global average the angular and radial currents were outputted. The summations of the angular currents were taken for all the other pixels on the same angle and separate summations were also taken for polar currents for all the other pixels on the same polar coordinate. These two sets of summations were processed to provide the azimuth and the elevation of the Sun.

Andreou and Boahen's silicon retina also used diffusive networks. A compact two layer hexagonally arranged network structure of 210x230 pixels was implemented. The system was inspired by vertebrate visual systems and as such, was suitable for processing acquired images to reduce noise while enhancing edges [55][56]. Meitzler et al. used Andreou retina cells for a spatial-temporal vision chip capable of detecting movement in one dimension [57]. This was achieved by sampling and holding an image and then, on subsequent images, each retina cell passed its value to the two adjunct cells. Each cell then performed two absolute differences using the value from the right cell and the sampled and held value and then repeated the process with the value from the left cell. The currents corresponding to left and right motion were then summed on global lines ready to be differenced off-chip to calculate a movement vector. Meitzler further developed the system to form a 50x50 two dimensional array version of the vision chip, capable of two computational processes, namely centroid and displacement computation [58]. The chip was aimed at the constrained application of sun tracking in high altitude balloons. This task was simplified by the fact there exists high contrast between the sun and the dark sky at an altitude of 35km. The rationale for developing a vision chip for this application was due to the limits of a low video bit-rate link from the balloon to a base station and the low power system requirement of less than 1 Watt. The allocation of 1 Watt

for powering the vision chip was as a result of the use of solar power to power the balloons electronic systems [59].

Fovea-based or log-polar sensors, which can incorporate retinal characteristics, have an organisation of photoreceptors similar to that of the human visual system. Unlike silicon retina based systems, the photoreceptors are not laid out in a linear density structure but a variable density and variable photoreceptor size, in a polar formation. The densest region of photoreceptors is at the centre of the sensor called the foveal region. Outside the foveal region, the density of the photoreceptors decrease and their size increases with distance from the centre. This structure allows for high spatial resolution in the region-of-interest (ROI), whilst allows the peripheral area to also be surveyed therefore using selective data reduction and hence reducing the video bandwidth requirements and the power consumption when compared to a large array-based sensor. This capability makes this type of sensor particular useful for robotics applications or tracking objects where changes in the peripheral area can be used to direct the foveal to a new ROI. Unfortunately these structures have disadvantages; a lower resolution compared to conventional imagers and increased complexity of design and use [60]. Figure 2.4 shows 3 examples of the common structures implemented in silicon.



**Figure 2.4 Micrographs of three different CMOS fovea image sensors, (a) a disjoint fovea array and peripheral ring structure, (b) a disjoint fovea array and peripheral array and (c) a continuous fovea and retinal ring structure [61, 62, 63]**

The key problem with (a) and to a lesser extent (b) in figure 2.4, is the discontinuity from the fovea to the peripheral photosensitive structure. This has been a common problem in early sensors [64] and exacerbates the mapping of the image into Cartesian coordinate array. The example in (c) has a more complex layout towards

the centre of the fovea which results in implementation problems due to the scaling of transistors and the small geometry effects.

The advantages that retina and fovea sensors provide are generally not required for the needs of the majority of current mainstream applications. The extra complexity required, such as mapping images to Cartesian coordinates, also reduces their attractiveness to the end-user. The problems due to the spacing between photosensitive elements, caused by the supporting architectures, also reduce the overall proportion of the sensor array which contributes towards photosensitivity. This is known as a low fill factor [65]. These are partly the reason that the majority of CMOS and CCD based image sensors use simple support structures with tightly packed photosensitive Cartesian array structures, which are more conducive to traditionally image processing techniques and provide potentially higher fill factors.

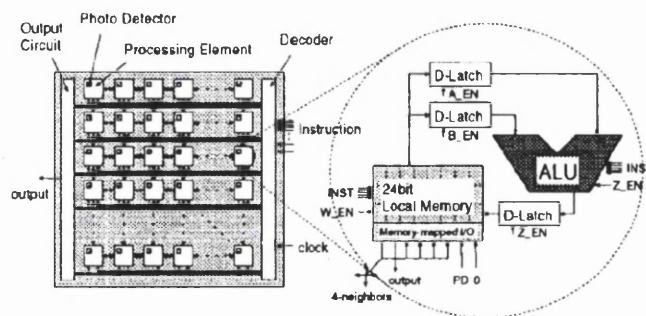
### 2.5.2 On-Chip Explicit Computational Processing Based Vision Sensors

On-chip explicit computational processing based architectures implement the photosensitive elements separately from the image processing elements. Vision sensors of this nature can be found to span a wide spectrum of architectures, from the integration of algorithmic logic units (ALU) at pixel level, to the separation of a sensor and processor by a communication bus on a multi-chip module (MCM). As has been the situation with biological and architecturally-implicit processing based vision sensors, the majority of the early literature on vision sensors has been application specific. Typical examples come from the biometric, automotive and automated visual inspection domains, such as fingerprint verification and vehicle monitoring [66, 67]. The ever decreasing transistor geometries however now mean it is possible to implement larger general purpose computational structures on-chip. In this section, three different architectures for general-purpose vision chips will be reviewed, they are mesh processing, linear processing array and frame processing.

### 2.5.2.1 Mesh Processing Architectures

A mesh processing architecture is constructed of a two dimensional array of processing elements each directly connected to a photo-sensitive element. As the processing for every pixel can occur concurrently, high frame rates over 1000fps are possible, as has been shown by Komuro et al.'s S<sup>3</sup>PE architecture [68, 69], Eklund et al.'s NSIP [70] and Kagami et al.'s SPARSIS system [71]. This high speed processing ability enables its use for a wide range of applications not possible with other systems, such as robotic applications requiring very high speed tracking and object manipulation.

The S<sup>3</sup>PE (Simple and Smart Sensory Processing Element) architecture, upon which the SPARSIS is also based, uses bit-serial processing techniques. Bit-serial processing is a method that performs a multi-bit operation on each bit sequentially. Although processing using this method is slower than bit-parallel processing, it results in the realisation of compact circuits and the ability to perform variable bit length operations. Each processing element (PE) consists of a bit-serial ALU, 3 latches and a 24bit local memory with bitwise addressing. The ALU has been implemented using a full-adder, a carry register and some multiplexers providing the ability to perform one operation per clock cycle, from a selection of 18 logical and arithmetic operators. It communicates with its 4 neighbouring PEs, a zero signal and its associated photo-detector using an 8 bit-serial I/O port, figure 2.5.



**Figure 2.5 Architecture of S3PE [72]**

The 24bit local memory and I/O port are allocated to the same address space. As all processing and I/O operators are performed by accessing local memory, the

instruction code is simplified. Each instruction is divided into 4 parts; read addresses A, B, (5bits each), operation code (5bits) and write address (5bits).

The S<sup>3</sup>PE architecture has been extended by Kagami et al. to form a new dual pipeline architecture called SPARSIS [71]. The extension was developed to provide an efficient control structure of the single instruction multiple data (SIMD) PE array. This was achieved by connecting a 32-bit reduced instruction set computer (RISC) processor via an integrated control and data path shared with the existing vision chip architecture. It's specific functions within the vision chip are :-

1. integer processing
2. program control operations, including branches and function calls
3. transmitting SIMD instructions to the PE array
4. parallel data I/O with the PE array and feature value extraction

The addition of the RISC processor also provides the opportunity for the system to be re-configured for different light levels and dynamic ranges using a software-controlled A/D conversion.

The SPARSIS system was prototyped by implementing the PE array and RISC processor separately. A 64x64 pixel PE array was fabricated using a 0.35µm CMOS technology. The controller incorporating the RISC processor was synthesized and fitted to a Xilinx FPGA and consisted of approximately 27000 logic gates. Providing a clock rate of 40 MHz for the controller allowed an instruction rate of 10 MHz for the PE array. A set of execution times for several sample programs are provided in Table 1.

<b>Program</b>	<b>Steps (time)</b>
Dilation or erosion (binary input)	5 (0.5 µs)
4-neighbour x,y edge detection (binary input)	9 (0.9 µs)
4-neighbour smoothing (8-bit input)	99 (9.9 µs)
Centroid detection (binary input)	1716 (171.6 µs)

**Table 2.1 Execution times of sample programs for the SPARSIS architecture**

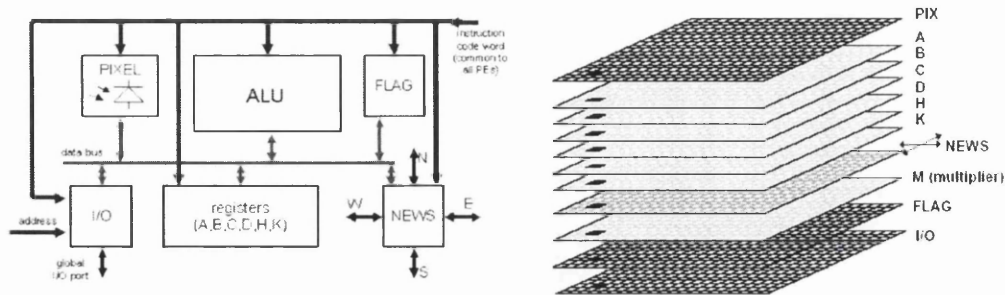
Despite providing low processing times for local operators, such as edge detection, the time required to perform global operations, as with most SIMD array-based processors, is poor. This can be explained by the situation whereby using pixel information from a PE on a distant pixel requires pixel information to be passed PE by PE until reaching the current PE. As a result, the required communication time is proportional to the distance the information must travel.

The NSIP (near-sensor image processing) architecture reduces this problem by incorporating a global logic unit (GLU) in every pixel. This combined circuit allows image data to be passed between a cardinal neighbouring GLUs by selecting one of four direction bits. As the GLUs can communicate without using the system clock for sampling data, pixel data can be transmitted to a distant GLU within a single clock period [72].

Komuro et al. highlighted the communication problem between distant PE in the S<sup>3</sup>PE architecture [73]. A more flexible architecture was developed where the PE network was reconfigurable in a way which allowed cumulative multiple global operations to be performed by chaining PEs together. This was achieved by connecting the output of one PE's 1-bit ALU to the input of another PE's 1-bit ALU and hence creating an n-bit ALU.

Using the new architecture, the centroid of a 256x256 pixel binary image was calculated using 440 local instructions and 65 non-local instructions. Using a 10Mhz instruction clock, as used with SPARSIS, a total processing time of 64μs was achieved compared to 3.4ms for the simple mesh-connected version and 171.6μs for an image size 16 times smaller using SPARSIS [73].

An interesting example of a novel mesh-based architecture is that of the Dudek and Hicks' SCAMP vision chip. Unlike the previous example discussed in this section, this architecture uses an array of analogue processing elements and processes local data as analogue samples in a SIMD fashion. Each APE (Analogue Processing Element) includes an ALU, photodetector, registers (A to K), activity flag register and I/O port, as shown in figure 2.6.



**Figure 2.6 Architecture of (a) Single APE (left) and (b) the SCAMP architecture with a single APE marked (right) [74,75]**

Image acquisition is achieved by the use of a special purpose register plane called PIX. The value held in this register plane corresponds to the state of the photodetector array. As with the SPARSIS architecture the exposure time is controllable and due to the non-destructive read-out mechanism multiple exposure times can be achieved. Each register plane (A-K) is constructed from switch-current memory cells and can hold a grey-level image. Arithmetic operations can be performed pixel-wise on these data planes. As the analogue bus is operating in current mode, summation can be directly performed on the bus using several source planes concurrently. Multiplication is performed using the multiplier register M. SCAMP also supports register plane to register plane transfers. Inter-APE communication is enabled using the NEWS register which provides access to the four nearest cardinal neighbours in the array. The activity flag register, FLAG, is set or reset depending on the result of a comparison operation. This enables local autonomy of an APE from the array and prevents an APE performing SIMD instructions without the FLAG register being set.

A prototype SCAMP chip with a array of 21x21 APEs, random-access I/O logic, on-chip D/A converter and control logic was fabricated using a digital 0.6 $\mu$ m CMOS process technology. In addition, an off-chip program store and instruction sequencer was implemented to provide instructions to the SCAMP vision chip. The maximum performance was over 1.1 GIPS (Giga Instructions Per Second) with a peak power dissipation of below 40mW and APEs clocked at up to 2.5Mhz. Several execution times for algorithms implement on the SCAMP chip are provided in Table 2.

Algorithm	Execution Time
Smoothing using 3x3 convolution template	5.6 $\mu$ s
Sharpen using 3x3 convolution template	6.0 $\mu$ s
Edge detection with Sobel template	11.6 $\mu$ s
Median Filter in 3x3 neighbourhood	61.6 $\mu$ s
Histogram with 64 Bins	205.6 $\mu$ s

**Table 2.2 Time of execution of several algorithms on the SCAMP vision chip [74]**

Since the first prototype a new SCAMP 2 architecture has been developed. The SCAMP 2 optimises the SCAMP architecture to reduce the area of the APE to approximately 25% its original size and reduce power consumption from 85 $\mu$ W to 12 $\mu$ W per APE. This has partly been achieved by lowering the instruction processing rate from 2.5MIPS per APE to 1MIPS per APE [74][76]. The tradeoffs of processing performance for area and power reductions were deemed necessary for the architecture to become more scalable, as another prototyped chip, the SCAMP 3, had been envisaged. The SCAMP 3 was fabricated with a 128x128 APE array.

Unlike the example of mesh systems that used digital processing, the SCAMP family has the advantage of only having to use a single capacitor to store analogue variables. Also, as the data buses within the APE are analogue, only 1 wire is required unlike N wires for N-bit wide digital data. As analogue processing is utilised there is no need to implement an A/D convertor in each APE. The disadvantages with the SCAMP family include a low fill factor of 5.6% to 8.4% [77]. The other main disadvantage is that of the accuracy of analogue processors as they are limited by errors and inherent noise. The accumulative effect of the errors within the SCAMP's analogue circuitry reduces the accuracy to below the equivalent of 7-bits, although this has been deemed sufficient for many low-level image processing operations [74].

Several general limitations of the pixel mesh processing technique have meant only a relatively small number of research groups have been working on this architecture. As the number of transistors used in the PE affects its area, the complexity of the PE is restricted to retain a suitable fill factor for each pixel. The programmability is, in many cases, a problem due to the SIMD instruction compilation and the timing requirements to achieve parallel instruction delivery to each PE. The proximity of PE to each other and the photosensitive circuit can also lead to cross-talk effects and



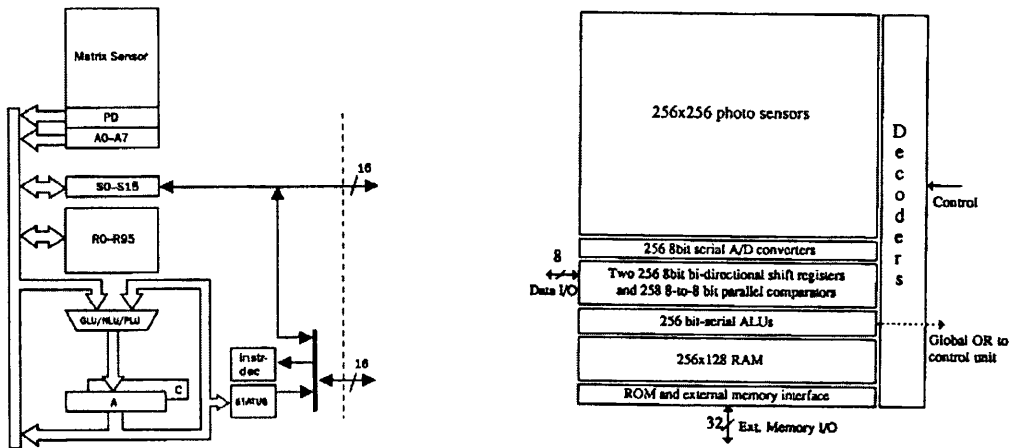
the emergence of fixed pattern noise (FPN) affecting the acquired images. Although for applications which require high processing rates and low power requirement, this form of architecture is an optimal solution. The ability to parallel process, reduces the need to implement very complex circuitry within a pixel as the cumulative processing power of all the processing elements can be immense. This provides the opportunity to reduce the processing frequency to a level that is sufficient for the application and therefore in the process reduce the vision chips power requirements significantly. A further advantage of the proximity of processing elements to the photosensitive elements is that of the easy implementation of fast global and local adaptation to light levels through close control of sampling and the A/D control process.

#### 2.5.2.2 Linear Processing Array Architectures

Linear processing array architectures typically only use one processing element and analog-to-digital converter per column. When compared to the pixel mesh processing architecture this approach has the advantage of the ability to integrate a more compact and uniform CMOS imager with a high fill factor while still retaining a level of parallel processing and low power consumption. It has also been shown that the linear processing arrays used can provide greater computational performance than mesh based pixel processing arrays with careful architectural enhancements [78].

Two examples of smart vision sensors that use this technology are Integrated Vision Products' MAPP2x00 and Chen et al.'s PASIC. MAPP2x00 relates to a family of products developed from 1987, with the latest member of the family being the MAPP2500. Both the PASIC and MAPP2500 use bit-serial processing within their ALU to reduce the complexity of implementations in much the same way as the pixel mesh processing architecture previously presented. Also, temporary bit-line memory exists in each to allow for interline and neighbourhood image processing. In the case of MAPP2500, this consists of 96 general purpose registers (R0-R96) capable of storing 512 bits and 256bits by 128 lines SRAM for the PASIC [79, 80]. The PASIC

and MAPP2500's A/D converters are capable of a grey-level resolution of up to 8-bits. Figure 2.7 shows the architecture for both the PASIC and MAPP2500.

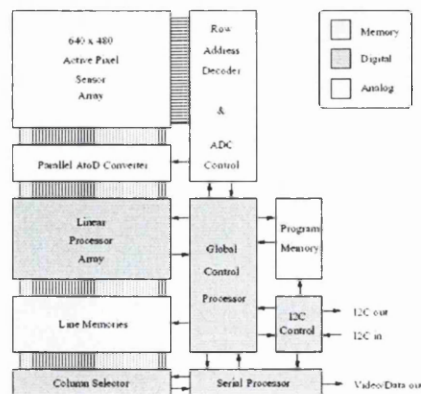


**Figure 2.7 Architectures of (a) IVP MAPP2500 (left) and (b) Chen et al.'s PASIC (right) [79, 80]**

Typically both systems are aimed at spatial applications, although the PASIC does have access to frame buffers via an external memory interface which could allow the implementation of temporal algorithms, for example inter-frame differencing. The implementation of the ALU structures also differs between the two. The PASIC approach is to use 256 ALUs consisting of three 1-bit registers feeding a full adder with sum and carry output. Each ALU has access to 128-bit memory and two 128x8-bit bi-directional shift registers via a bus. The ALUs have also been extended to provide 6 Boolean functions for the 3 inputs from the registers in addition to the ability to load each of the three registers with both inverted and non-inverted data. There are a total of 16 ALU address and control instructions. The MAPP has 80 different instructions to control 3 dedicated 512-bit logic units, PLU, NLU and GLU. The PLU performs bitwise Boolean algebraic operations on local pixel data, conversely, the NLU processes pixels in a way that is dependant on the pixel nearest neighbours allowing, for example, template matching, edge enhancement and filtering operations. Eight of the instructions control the GLU and provide operations, such as, marking vertical connectivity between objects in two image line and a fill operation that sets the bits between two objects active. Both PLU and NLU logic units operate on complete lines and complete operations in one clock cycle whereas the GLU requires two clock cycles.

The MAPP vision sensor is primarily aimed at, and hence primarily optimised for, range imaging systems. In such applications high frame rates are required to obtain a 3D profile of an object. The coarse-grained instruction set allows such an application to be performed in only 4 instructions and two status registers readouts [79]. In contrast the PASIC is a general purpose vision system with a collection of a small number of simple instructions from which more complex algorithms can be constructed.

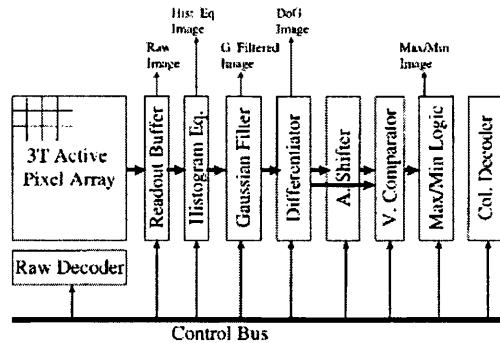
The Philips Xetal chip is one of the most recent and advanced linear processing array based architectures. It is especially designed for pre-processing images acquired through its integral 30fps VGA format CMOS image sensor. Currently no frame memory is provided which limits the Xetal to non-temporal tasks. Whilst the architecture of the Xetal seems similar to the MAPP and PASIC, in that it has an active pixel array connected to a parallel A/D converter, linear processor array and line memories, there are two distinct differences between the two. Compared to the previous two examples, the linear processing array does not work bit-serially but with 10-bit word widths of data. This has been shown to increase power efficiency [81]. Also, the linear processing array (LPA) only consists of 320 processing elements rather than the full width of the sensor array of 640, as this was deemed the most efficient with dealing Bayer type colour filters. Akin to the PASIC, the pixel and neighbourhood processing is performed by the linear array where as global computations are executed by the global control processor, such as exposure-time control and white-balancing (figure 2.8). Another task performed by the global control processor is that of passing instructions to the LPA.



**Figure 2.8 Top-level architecture of the Philips Xetal Vision Chip [82]**

Each processor in the LPA contains an accumulator, an adder and a multiplier. Unlike the previous examples, data can be obtained directly from 6 different columns facilitating effective convolution algorithms. It is this data transfer mechanism and its embedded multiplier per processor that enables the Xetal to reach a performance of 5000 million operations a second at 1.6W at 30fps [81]. The program memory can contain up to 1024 instructions and has been shown to be sufficient for a range of low to medium level real-time vision tasks, such as optical character recognition and skin-region detection [82].

Another approach to LPA architecture has been the implementation of a 256 x 256 pixel CMOS imager and fast on-chip analog image processing functions for line-based stereo vision applications. Rather than mapping algorithms onto an array of ALU, Ni and Guan's Smart CMOS Image Sensor implements 4 separate processor arrays; Histogram Equalisation to allow adaptation to changing illumination, Gaussian Filter to remove noise, spatial-temporal differentiation and minima and maxima extraction to detect edges (see figure 2.9).



**Figure 2.9 Top-level architecture of Ni and Guan Smart CMOS Image Sensor [83]**

In terms of processing, the Gaussian filtering, spatial-temporal differentiation and max/min extraction can be performed on a line of pixel within 12 $\mu$ s [83]. This is significant as all the operations, with the exception of histogram equalisation, can be done with the television's line blanking period. In order to generate a continuous television video signal, the histogram equalisation is pipelined with the other operations by using the capacitors within the Gaussian filter as temporary storage. The worst case power consumption of this sensor chip has been measured at 0.2W.

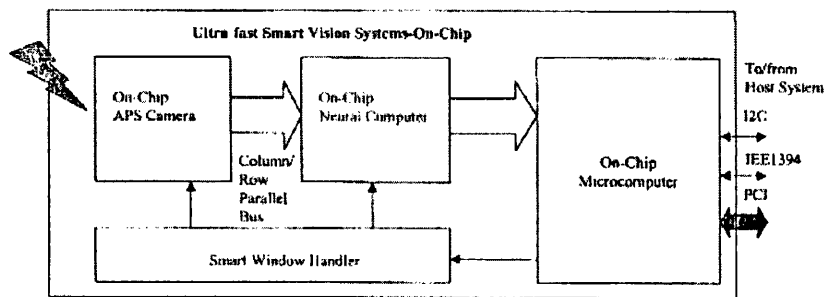
The largest difference between the mesh and LPA architectures is that of the LPA's partitioning of the sensing array from the processors. This has the inherent advantage of optimising the sensing array to a greater extent in relation to its fill factor, silicon area and hence sensitivity range. This further enhances the opportunity to develop the sensor array and processing arrays separately before integration, which could reduce the costs of prototyping. This method of prototyping the two components separately was used in developing the Xetal based vision sensor. A parallel processing paradigm is utilised, although to a less extent than mesh processing, allowing a low processing frequency and hence low power while still achieving high performances. The linear processor also simplifies algorithms that process within a column as communication is not required between the processing elements in the same column as pixel data is read a line at a time. Unfortunately, LPA architectures generally require higher power levels due to circuits such as the A/D converters running at 128-512 times the frequency of pixel processing architectures. The area of the processors in a linear array is usually restricted by the column width and hence limits their complexity. This in part confines the LPAs applicability to low to mid-level vision tasks. The ability to program these processor arrays in a high-level language such as C is difficult as a further result of the simplicity of the processing elements. This is seen with the Xetal, which is programmed using a custom assembly language and the MAPP2500 which is provided with a specific set of algorithms suitable for its expected application domain of range imaging.

### 2.5.2.3 Frame Processing Architectures

Frame processing architectures are typically found in conventional board-level integrated smart vision systems. This architecture separates the processing element(s) from the sensor array by a serial or limited parallel communication bus. Typically, a whole image frame is transmitted by the sensor array to a processing pipeline which performs the necessary operation on-the-fly or to a frame store for further processing. The main argument for not using this approach for system-level integrate circuits is that of the communication bottleneck between the processing element(s) and the sensor array. As a result, there are less examples of this form of

vision chip architecture in the recent literature than those based on mesh and linear processing arrays.

Fang et al.'s proposed a smart vision SoC integrating a 10-bit 1024x1024 pixel image sensor, a smart image window handler, microprocessor and neural processor into a single 30mm by 30mm SoC in a 0.18 $\mu$ m CMOS technology, see figure 2.10 [84]. The system was designed for the use in NASA scientific missions although it has been deemed suitable to other military, industrial and commercial vision applications.

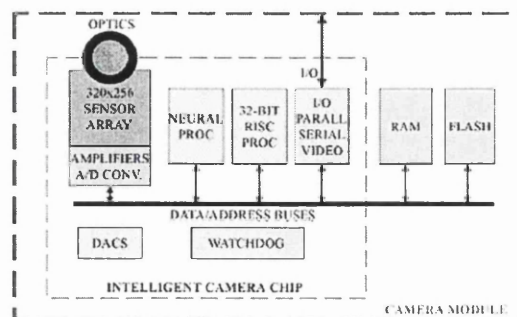


**Figure 2.10 Top-level architecture of Fang et al.'s Smart Vision SoC [84]**

The on-chip microcomputer controls the sub-systems and enables communication with an external host system, if required. It is constructed from an embedded PowerPC 750 processor with a data memory, a program memory, a UART and a multi-bus interface unit. The multi-bus interface unit conforms to IEEE 1394, commonly known as iLink or Firewire, I<sup>2</sup>C and PCI standards. The connection of Smart Window Handler to the image sensor allows the control of data acquisition and data transmission to the on-chip neural computer. This occurs by requesting an N by N sub-window of data from the sensor array at a rate up to 30 frames a second. This sub-sampled data is then read from the array, depending on the required row/column addressing scheme and transmitted, on a 10-bit wide data bus to the neural network computer. The programmable neural computer is based on an optimization cellular neural network (OCNN), with memory for synapse weights and a learning and post operation co-processor. The OCNN consists of a 1024 by 1024 cell matrix providing suitable computational power for a wide range of tasks.

Despite the system performance estimate, which is in the region of  $10^{12}$  operations per second, the die size of the complete SoC would result in a very high cost per unit and hence be unsuitable for a wide range of applications. This die size can be attributed to the 64MB of embedded DRAM and approximately 7MB of embedded SRAM for frame store, data store and neural network related memories. Fang et al. prototyped a version of this system on a multi-chip module (MCM). To accommodate the complete system on the same substrate, the MCM's dimensions were required to be 50mm by 100mm and had a mass of 0.2kg. Using the proposed 3D die stacking techniques combined with a sub 0.5 micron silicon-on-insulator CMOS process technology, the neural array's dimensions were estimated at of 30mm by 30mm by 5mm. At 4 MHz, the MCM was found to consume 1W [85].

A smaller example of a frame processing architecture is that of the VISoc, produced commercially by the Italian company Neuricam. The complete system, with the exception of off-chip SRAM and FLASH memories, has been fabricated in a  $0.35\mu\text{m}$  CMOS process and occupies an area of  $5.93 \times 5.98 \text{ mm}$ . The typical power dissipation has been measured at 1.3W at an operational speed of 60 MHz [84]. The VISoc comprises of a  $320 \times 256$  Greyscale CMOS imager, a 32-bit embedded integer RISC processor with 0.5KB instruction cache, a neural processor called TOTEM, a parallel and serial I/O and a memory interface. Externally from the chip is a 512KB FLASH memory for storing program code and initial set-up, such as sensor calibration data, and a 1MB SRAM for storing temporary data and for use as a frame buffer, see figure 2.11.



**Figure 2.11 Top-level architecture of Neuricam's VISoc based smart camera [86]**

The integrated CMOS imager has a logarithmic response to light with a dynamic range of over 120 dB. This increases its suitability for applications which require a

system that can adapt to changes in lighting, such as automotive applications. The pixel addressing for the sensor array is controlled by the RISC processor via a 17-bit address bus. On selection of a pixel, a signal is sent to an A/D converter (ADC) to generate a 10 bit word. This word is then transmitted to the frame buffer in off-chip SRAM using a direct memory access (DMA) channel without intervention from the RISC processor. The performance bottleneck for this operation is caused by the maximum sampling rate for the ADC, of 15 Mpixel/s. This limits the full frame (320x256 pixels) read-out to 180fps. Sub-sampling pixels from the array by windowing a smaller region of interest will result in a higher achievable frame rate.

The TOTEM co-processor has an array of 32 parallel integer multiply-and-accumulate (MAC) units that have access to on-chip weight memory. This allows the architecture to act as a multi-layer perceptron neural network and hence can implement learning and recognition operations. The performance of this co-processor is in the order of 300 Million MAC per second. Although the RISC processor is primary used for controlling the VISoc sub-systems, it is also capable of general purpose processing and has an additional processing power of 60 MIPS.

McBader of Neuricam has proposed an extension to the VISoc architecture called SmartPupilla [87]. It is aimed at serving a wider range of applications though the use of a larger imager, a large on-chip memory, the addition of a parallel pre-processor and a faster system clock rate. Table 3 below shows the relative differences between SmartPupilla and VISoc.

	<b>VISoc</b>	<b>SmartPupilla</b>
CMOS Sensor Array Size	320x256	640x480
Maximum Full Frame Rate	180 frame per second	130 frames per second
Pixel Resolution	10-bit	10-bit
On-chip Processing	32-bit RISC	32-bit RISC+DSP
On-chip Memory	93.5 Kbits	~32 Mbits
Off-chip Memory	1MB SRAM	Not required
Neural Network	32-node	Parameterisable
Clock Frequency	≤ 60 MHz	≥100 MHz
Processing Power	60 RISC MIPS	100 RISC MIPS
Power Dissipation	≤ 1.3 W	≤ 2W
Process	0.35 μm	0.18 μm or 0.13 μm
Die Size	< 36 mm <sup>2</sup>	~ 150 mm <sup>2</sup>

**Table 2.3 Comparison of Neuricams VISoc and proposed SmartPupilla**



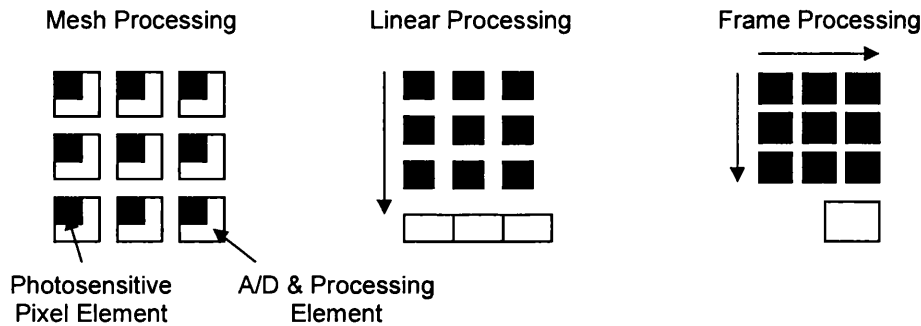
Unlike the VISoC, the SmartPupilla architecture features an image pre-processor based on a parallel array of 16 programmable processing elements. A combined DMA controller enables an image stored in embedded RAM to be addressed in 25 modes. These modes have been chosen as they are commonly used in image processing algorithms. The addressing sub-system also detects overlaps in images distributed to the parallel processor and reduces the need for redundant pixel data reads. The image pre-processing architecture was prototyped on a Xilinx XCV200E FPGA and achieved data throughputs of up to 667fps at 50 MHz, using a 256x256 pixel image. The SmartPupilla's peak performance is estimated to reach 3.23 GOPS.

Typically, frame processing architectures have several distinct advantages. The primary advantage is that there are no limitations on the design of the processing elements implemented. The use of frame buffers, in many systems, also allows image storage and the application of complex multi-inter-frame operations to a range of frames, hence extending its flexibility and allowing a wider selection of temporal algorithms to be realised. The fill factor of the embedded image sensor can be high and the pixel arrays can be optimised for size. As the embedded image sensor is separate from the processing logic, it can also be more easily shielded from noisy digital components by using techniques such as guard rings and careful floorplanning to position supply and ground lines [86].

In terms of disadvantages, frame processing architectures typically consume more power than that of LPA and mesh processing based vision chips. In frame processing architectures that do not use any image storage, a fast processing element or complex pipeline is required to process incoming images. Frame processing architectures that use image storage require additional power for the memory sub-systems. The use of image storage also has the disadvantage of the latency of a frame memory write and a very large silicon area if the memory is implemented on-chip. It is for these reasons that MCM and off-chip memory are used to allow the implementation of the memory in the most cost efficient process, which is often a different process than that used in the vision chip itself.

### 2.5.3 Summary

A wide variety of chip-based vision systems have been presented in the last two subsections. The general purpose vision systems discussed have displayed different levels of system costs, flexibility, power requirements and processing performance. The three approaches used for general purpose processing in vision chips are summarised in figure 2.12 and their attributes summarised in table 2.4 below. Grey highlighting in table 2.4 indicates the most favourable approach for each attribute and the arrows in figure 2.12 indicate the direction of pixel read out to the processing element(s).



**Figure 2.12 The three approaches to general purpose processing on vision chips**

Attribute	Mesh	Linear	Frame
Typical clock speed of processing elements	Very low	Low	High
Flexibility in design of processing elements	Very poor	Poor	Good
Suitability for temporal processing	Very poor	Poor	Good
Typical fill factor	Poor	Average	Good
Ability to shield photo-sensitive pixel array from processing element switching noise	Poor	Average	Good
Ability to prototype processing elements and pixel array separately	Poor	Average	Good
Typical level of processing performance	Very good	Good	Average
Power requirements per operation	Good	Good	Average
Ease of programming	Poor	Poor	Average

**Table 2.4 Comparison of the three approaches to processing in vision chips**

As can be seen in figure 2.12, the three approaches differ with respect to the level of parallel processing occurring and the locality of the processing elements to the photosensitive pixel array. Whilst there are quite distinct differences between the attributes for mesh processing and frame processing, linear processing with processing elements separated from pixel elements while still performing parallel processing, has mid-range values.

Examining table 2.4 would indicate that in situations that require high performance at low power, mesh processing is the best approach. This can be explained by the fact that the combined performance of an array of ALU units, often allows the processing clock frequency to be reduced while still fulfilling performance requirements. As frequency is directly related to the power dissipation of CMOS devices (see equation 2.1), huge power consumption savings can be made. This may make the device suitable for low power systems and mobile battery powered applications.

$$P = \alpha V_{DD}^2 f C$$

where P is power dissipated (W)

$\alpha$  is percentage of gates switching

$V_{DD}$  is supply voltage (V)

f is switching frequency (Hz)

C is switching capacitance (F)

**Equation 2.1 Dynamic Power Dissipated by CMOS logic [88]**

Conversely, in situations which require a flexible or complex processing element to be designed and/or temporal processing to be performed, a frame based approach is the most suitable. While the frame based approach does not typically have the large array of parallel processing elements as with mesh and linear array processing, parallel processing can also be used in frame-based processing architecture. Despite the lack of a wide communications channel between the sensor and processing element, parallel processing can be achieved by implementing parallel processing within the processing element itself. This is a popular approach as most image processing and machine vision algorithms can be mapped onto an array of multiplier and

accumulator-based processing elements. An example of this is the neural network in Neuricam's VISoc [86].

Another advantage of frame based processing architectures is that of the flexibility in designing the photosensitive pixel array. As the processing element is detached from the photosensitive pixel array, the dimensions of the processing element do not affect and limit the design and compactness of the pixel array. Without the extra structures for processing around each pixel element, the pixel elements can be closely packed together shrinking the pixel array area. This has the effect of increasing the ratio of photosensitive area to non-photosensitive area, i.e. increasing the fill factor. An alternative to shrinking the pixel array area would be to use the area gained to increase the size of each photosensitive pixel and make it more sensitive to light. This would potentially make the frame processing based architecture more suitable for low light applications than a mesh based architecture. The separation of the processing element and pixel array is also particularly conducive to vision chip development, as the processing element can be prototyped separately from the pixel array. This has the advantage of potentially reducing the need for prototyping on silicon in the early stages of product development and hence reduce the overall development costs.

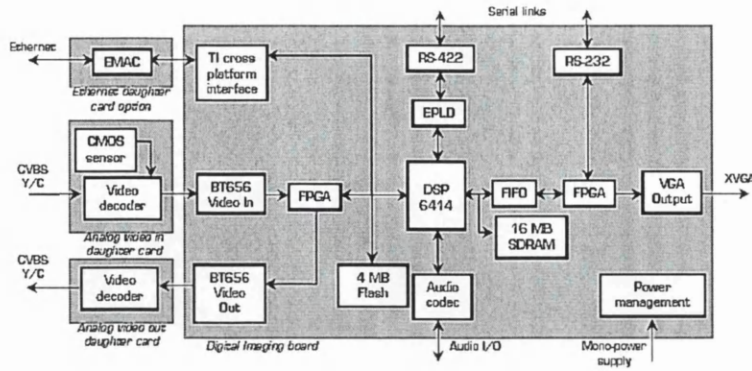
The majority of processing architectures must be programmed in some form of assembly language. Generally with traditional PCB-level GPMV systems, the C programming language is used due to its international adoption for many platforms and its relative ease of use. The use of assembly language, especially machine specific such as Xetal, complicates the use of the vision system by an end user. Only frame based systems such as Neuricams VISoc and SmartPupilla and Fang et al.'s Vision Chip, currently provide the opportunity to be programmed with a higher-level language due to the inclusion of an embedded RISC processor for system control. The reason behind this apparent flaw is that compilers often do not produce optimised or even suitable code for SIMD arrays of processing elements. It is likely that software development packages will have to be improved or vision architectures adjusted before the mass-adoption of vision chips for developing complex vision applications.

As has been already explained, the lack of on-die memory for image storage is due to the current cost of implementing enough memory for complete images on-chip. The proposed SmartPupilla vision sensor has embedded SRAM, which in a 0.18 $\mu\text{m}$  process, would occupy 115mm<sup>2</sup>, approximately 76% of the total die area [87]. As transistor geometries shrink, large capacity integrated memories will probably start to become an appealing option in conjunction with frame based processing, especially with the need for deeply embedded single-chip vision systems.

## **2.6 Vision System Prototyping**

The prototyping of integrated circuits has developed rapidly within the last decade to result in the availability of a myriad of solutions. These range from large flexible prototyping platforms for SoC designs to domain specific platforms with emphasis on software and minimum of hardware prototyping. One of the largest of these systems is that of the Aptix System Explorer family. These platforms provide a solution for verifying complex SoC designs. This is achieved using the provision of software to map SoC designs onto a multi-FPGA based reconfigurable hardware platform. The hardware also supports the connection of multiple system modules to a large complex PCB backplane via their patented programmable interconnects. The latest version of the family, the MP4CF, includes enough interconnects for up to 20 FPGA modules, each with a Xilinx Virtex V2000E FPGA providing more than 3 millions ASIC-equivalent gates and 10 Mbits of RAM [89]. The interconnects can also be used with other Aptix modules to allow the addition of microprocessors, memory devices and analog and digital converters.

The French company Ateame produces a development kit suitable for prototyping imaging systems. Its entry level product, the Digital Media Evaluation Kit (DMEK) 6414, consists of a backplane PCB with a 500Mhz Texas Instruments TM320C6414 DSP, 16MB of SDRAM, 4MB of FLASH, 30Kgate and 50Kgate Altera FPGA, as well as a selection of audio and digital I/O [90]. A selection of daughter boards can be connected to the backplane providing access to Analog I/O, Ethernet and a VGA CMOS Image sensor, see figure 2.13.



**Figure 2.13 Top level block diagram of Atmel Digital Media Evaluation Kit 6414 [90]**

While the large, more complex systems, provide the most flexibility they can also be extremely expensive. On the other hand, domain specific prototyping systems may be obtainable at a much lower cost but the choice of components available to the end-user may be very limited. This also extends to the problem of integrating a complete vision system onto a single chip. If the hardware IP blocks for each discrete component cannot be licensed from their respective provider, it may not be possible to migrate the end product to a single chip. It is often the case that even if the licences can be obtained, the high cost of royalty schemes for third-party IP may make the end-product unsuitable for use in many low-cost applications. It is therefore incredibly important to carefully select the discrete ICs and IP used in FPGA to ensure an easy path to migrate to a greater level of integration. The extreme of this situation is that the imaging development platform has to be designed in-house, using only components for which the company owns the necessary usage rights.

## 2.7 Summary

This review has covered the history of machine vision and its current, emerging and future applications. Board-level integrated smart vision systems have been also discussed followed by a broad array of system-level integrated smart vision systems and suitable prototyping systems.

The approach taken for this doctoral research was the design of a prototyping system consisting of a general-purpose reconfigurable FPGA backplane PCB and custom

daughter board with the required set of peripherals for image processing and machine vision applications. This provides the flexibility of a FPGA device for implementing custom logic while fixed function devices, common to all application are included on the custom daughter board. This solution is similar to that of the Ateme Digital Media Evaluation Kit but shares the Aptix System Explorers Family's ability to increase the available logic resources by changing the FPGA backplane. This is an important feature as the size of custom logic for future vision chip prototypes is unknown. This method of prototyping has the additional advantage of significantly lower costs fabrication cost when compared to using multi-project wafers. As the FPGA can be reconfigured the system can also be functionally tested at each stage of the development cycle.

As the research was based within the imaging division of STMicroelectronics it was required that the company's Cartesian array based CMOS imager technology would be used. This point and the fact that STMicroelectronics CMOS imager outputted images in a serial pixel stream meant the frame processing architecture would be the most effective implementation. Typically better fill-factors can also be achieved than mesh arrays and LPAs as the processing elements are not integrated with the sensor array. This provides better photosensitivity which important for many machine vision application. The selection of frame-based processing is conducive to implementing numerous hardware co-processors which could be optimised for common image processing and machine vision operators and complex algorithms. This multiple co-processor concept is unlike the vast majority of vision systems, with the notable exception of Ni and Guan Smart CMOS Image Sensor. The primary rationale for not using this concept is that potentially more area is consumed than the use of a general ALU and unless most or all the co-processor are used concurrently the system implementation could be seen as unoptimised. But there are several reasons why this co-processing paradigm is valid in this situation, they are;

- Quick exploration of design space and rapid prototyping for the developer of new products as it may not be necessary to have to write large amounts of program code. This could be particularly useful when demonstrating a product still in development to a potential customer.

- Potentially reduce the complexity of the end product users programming model and code size, as these co-processors could be operated by simply making function calls.
- Careful design of the hardware co-processors would allow their use in the large portfolio of other products that use frame-based processing within the division, i.e. useful in lowering development time, and hence cost, by re-use.
- Co-processors can be optimised for speed and/or power to be more efficient than a single ALU or neural network.
- Only a small subset of the library of co-processor related to the processing critical path may need to be included and hence providing a very compact solution to an application.
- Possibility of concurrently processing several pixel streams from different images using several co-processors or forming pipelines of co-processor to perform several operations on one pixel stream.
- Most suitable for applications using temporal and spatial processing.

It could also be argued that the ever increasing number of gates available as a result of new technology processes and the fact many new mass-market application require portability, that designs are being more constrained by power requirements than area. Therefore, to some extent, extra area to support the co-processors is becoming less of an issue. The use of these co-processors and a frame-based processing approach, enables each architectural component to be developed and tested independently as an IP block before being mapped onto a SoC. The final general point for using a frame-based approach is that in applications that require low to medium volumes, the system could stay as a multi-chip FPGA-based solution rather than going to the time and expense of integrating the complete system into a SoC.



### **3 System Requirements, Analysis and Specifications**

The initial stages in designing an electronics system consist of 2 key steps. The first of these steps is the capture of a set of requirements which the end system must fulfil. In the second step, it is then possible to produce system specifications based on these requirements. This allows the designer to concentrate the design effort to engineer a system that exactly meets the specifications. Integral to the specifications is the partitioning of the system into software and hardware sub-systems. These 2 steps are discussed in this chapter for the new prototyping system.

#### **3.1 Requirements**

The primary source of requirements was from the original project brief provided by STMicroelectronics and hence were only very broadly defined. These were;

1. Lay the groundwork for future realisation of tracking and recognition of object in real-time.
2. Produce a core set of re-usable operators.
3. Ideally, provide generic architectures capable of addressing many applications.

In addition to the project brief, the literature for the original VLSI Vision Imputer Family was examined. The typical requirements of applications that used the Imputer architecture were;

1. Support for image resolutions of 256x256 pixels.
2. Support for 8-bit greyscale processing.
3. Ability to program user-define algorithms in C.
4. Real-time operation for image processing.

The second source of system requirements was from a Managing Director of a local machine vision company. The information provided was useful as the company

could eventually become a customer and user of the prototyping system. The company's requirements were;

1. Support for image resolutions to CIF standards, i.e. 352x288 pixels.
2. Support for 8-bit greyscale unsigned processing.
3. Protect the end-user from the complexity of the underlying hardware architecture.
4. Ability to program user-define algorithms in C.
5. Support for fixed point processing and floating point processing in special cases.
6. Operate at 20fps at CIF resolution.
7. Provision of memory capacity for 8 CIF images.

The third source of requirements was that of current, emerging and future mass-market applications. Trying to encompass all the requirements from a very wide range of applications would lead to over specification in order to support specific applications, for example, support for processing thousands of frames per second as required by an optical mouse. Hence a final constrained superset of requirements were generated. These were;

1. Provide a greater opportunity for system migration to a single IC by using devices where STMicroelectronics owned their intellectual property rights.
2. Provide image capture support of up to an 8-bit greyscale 640x480 pixel (VGA) image.
3. Operate at up to 25fps.
4. Provide the opportunity for direct connection to a host computer.
5. Provide video output to a PAL video monitor.
6. Provide control of several actuators and indication devices, such as motors and LEDs.
7. Provide user definable switches for prototyped applications.
8. Maximum bill of materials cost for the prototyping system of \$1500.

## **3.2 Specifications**

During the system specification and system partitioning steps, important design constraints may be placed on the system development. Producing a detailed specification for the implementation of the system would usually need mathematical modelling, simulation and evaluation of algorithms to be performed. This would indicate the suitability of the specified system and may raise any issues regarding the partitioning of the system. As no set of algorithms were provided or any requirements in terms of data processing or instruction execution speeds, it was not possible to accomplish this process by modelling or simulation. As a result, the system components were selected with cost, intellectual property ownership and datasheet-specified performance in mind.

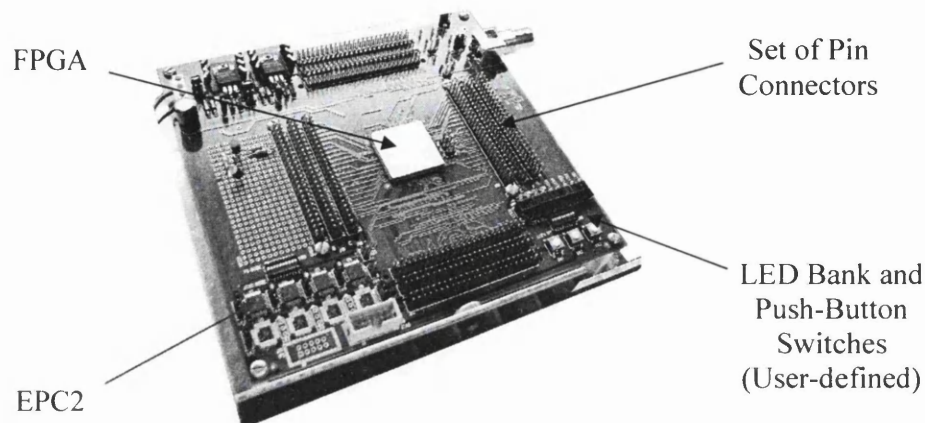
STMicroelectronics required the use of their CMOS image sensors within the project. This requirement prevented the development of mesh or linear array based processing architectures, as the image sensor provided would be physically separate from its associated processing element. This dictated that a frame-based processing approach had to be adopted. Also, although the original project brief did not specify the explicit development of a new hardware prototyping platform, a decision was made to implement a prototyping platform first and then a set of re-usable operators in the form of hardware IP blocks. The primary reason for this initial design decision was that without a prototyping platform, operations would have been required to be implemented in software, e.g. SystemC, or as mathematical simulations using a tool such as Matlab. This would have prevented the operations from being tested in real-time conditions.

The analysis, specifications and system partitioning are listed in the following subsections.

### **3.2.1 FPGA Backplane Selection**

Image Processing and machine vision algorithm processing was to be performed within a reconfigurable FPGA device. The FPGA device chosen was an Altera APEX 20K600E SRAM-based FPGA, mounted on an in-house available

STMicroelectronic Backplane, see figure 3.1. This backplane was chosen as there were no other in-house solutions available at the start of the project. An additional development time of 3-6 months would have been required to develop a new backplane and was deemed not have added further value to the project.



**Figure 3.1 STMicroelectronics Backplane**

The FPGA provided a typical usable gate count of 600000, 311296 bits of RAM, 4 phase-locked loops (PLL) and 508 user I/O outputted to 4 sets of pin connectors. As the FPGA retained its configuration using embedded SRAM, at power-off its configuration was lost. The inclusion of four non-volatile Altera EPC2 memory devices allowed the configuration to be stored after power-down and the FPGA to be reconfigured at power-up. Whilst a STMicroelectronics backplane with a larger APEX 20K1000E FPGA, providing a further 400K useable gates, was available, advice from the STMicroelectronics Imager IP team, was that timing closure would have been more difficult to achieve. This is explained by the increased size of the FPGA die and hence increased distances between pins and logic blocks giving rise to increase pin-to-pin and logic block to logic block transmission delays.

The use of an FPGA for processing is only feasible with a frame based processing architecture. Although linear array based processing architectures separate the processing elements from the photosensitive array, the communications bus between the photosensitive array and FPGA would have to be immense. For example, a linear array architecture connected to a VGA (640x480 pixel) image sensor with an 8-bit bus per pixel column, would require a 5120 wire bus connected to the FPGA. There

are currently no FPGA devices with this many I/O pins. Even if pixel data was sent to the FPGA bitwise, a 640 wire bus still would be required. A bus of 640 wires would still lead to the requirement of a very complex PCB to maintain signal integrity and timing requirements. The only other alternative for the implementation of a mesh or linear processing based architecture would have been to prototype designs in silicon. This was not a feasible option, given the costs and timescales involved with the silicon implementation of designs.

### **3.2.2 Daughter Board Analysis and PCB Component Selection**

As STMicroelectronics required the use of their CMOS image sensors, a new daughter board with a sensor interface needed to be designed. All other hardware components to support the image sensor were to be implemented on a single daughter board PCB, as it was not possible to implement these components on the backplane. This PCB, connected with the pin connectors on the backplane, would provide the necessary data paths, control signals and clock signal to the FPGA, while obtaining power from the backplanes voltage regulation circuitry.

Several options were available for implementing the daughter board PCB architecture. These were:

1. Implement the daughter board PCB with minimum support structures and implement a direct connection between the sensor and FPGA for the transmission of pixel data and control signals.
2. Implement the PCB with a general purpose microprocessor or microcontroller to act as an intermediary between the sensor and FPGA for sensor control purposes.
3. Implement the PCB using the associated STMicroelectronics sensor co-processor which would provide sensor control via an embedded microcontroller and initial image processing/enhancement in hardware.

The decision was taken to opt for the third choice, as implementing the sensor attached to its co-processor, would provide the greatest flexibility. The advantages of this approach were:

- It was a tried and tested in-house solution and could be implemented at a low cost and with relative ease.
- A full development suite existed for the sensor co-processor and in-house support for its use was available.
- The co-processor would aid the testing of the sensor.
- The co-processor offered embedded hardware providing USB communications support.
- The co-processor could perform initial image processing.
- No FPGA resources would be required for sensor control, unlike option 1.

The use of a sensor co-processor has disadvantages. These were that it was a more complex solution at PCB level than option 1 and that many of the co-processors functions may not be used. A DSP option was not applicable as STMicroelectronics asked that the library of operators be implemented as hardware IP blocks rather than software. The DSP would also not have provided a more efficient solution for the control of the sensor than the sensor co-processor or option 2's microprocessor.

As a result of the initial design analysis and design decisions, the specified core components of the PCB were;

1. A STMicroelectronics 100-pin (TQFP) STV0674 sensor co-processor designed for use with CIF and VGA ST CMOS image sensors and provides full exposure control, colour processing and sensor mode control. Using JPEG compression of image data, the co-processor can operate in three modes; as a USB Webcam camera; streaming audio and video at up to 30fps to a PC, a CIF or VGA stills camera; storing images to external memory, 'camcorder'; streaming audio and video to external memory for uploading later to a PC. The co-processor also contains an 8-bit 8052 microcontroller capable of running program code from on-chip 32Kbyte ROM and 32Kbyte

RAM [91]. This co-processor is required to act as a system ‘housekeeper’. As the housekeeper, the co-processors main role in the prototyping platform would be power-up device configuration, sensor control, image capture, image transfer to/from the FPGA backplane and a host PC via a 12Mbit USB 1.1 connection, if required. The microcontroller would also allow applications to be programmed in C and control of other board-level devices using its general purpose inputs and outputs (GPIO). All the PCB support components will be implemented to allow USB communications to a PC, audio in, via a board-mounted microphone, audio out, via a board-mounted piezoelectric buzzer, loudspeaker connector and earphone connector. Although not essential, these extra components may enable the implementation of a wider range of end applications and simplify any system or application debugging. An ST sensor socket is provided to allow interchange of different 36-pin sensor modules as well as a connector for flexible ribbon cable-based sensor modules , although it is expected that a 410 series CIF sensor or 500 series VGA sensor will be used in most applications developed with the system.

2. Four memory devices should be supported, providing a range of volatile and non-volatile memory options to the end-user of the prototyping platform.

These are;

- i. Two 54-pin (TSOP(II)) 128Mbit Samsung Single Data Rate Synchronous Dynamic RAM (DRAM) ICs. Part No. K4S281632D with maximum frequency of 133 MHz [92]. These provide a 16-bit wide data bus, compatible with the SDRAM interface on the STV0674 co-processor and capable of storing up to 54 VGA images each. These two memory devices are directly connected to the FPGA and as such, the co-processor does not have direct access. These are expected to be used for low cost high capacity short-term image storage.
- ii. One 44-pin (TSOP(II)) 64Mbit Samsung NAND FLASH IC. Part No. K9F6408. The FLASH is required to be directly connected to the STV0674 and the FPGA, via an 8-bit wide data bus. A 64Mbit

FLASH is capable of storing up to 27 VGA images. This is expected to be used for long-term non-volatile image or program code storage.

- iii. One 8-pin (DIP8) 256Kbyte STMicroelectronics Electrically Erasable & Programmable ROM (EEPROM) IC. Part No. 24C256. This part should be to directly connected to the I<sup>2</sup>C port on the STV0674 and hence provide non-volatile storage of application specific program code via a serial interface.
  - iv. One 3.3V compliant Smartmedia memory card slot connector, supporting up to 128Mbit removable memory cards. Connected to the FPGA and STV0674 by sharing the NAND FLASH 8-bit wide data bus. Note: Smartmedia and Flash device cannot be utilised concurrently for any application due to bus contention. This is expected to provide a useful alternative to a PCB-fixed FLASH device.
3. Video signal generation suitable for output to a video monitor will be supported using the STMicroelectronics 28-pin (SO28) STV0119a Video Encoder IC. This IC generates the analog composite video output from 8-bit wide time multiplexed 4:2:2 chrominance and luminance in an ITU-R BT.656 format. The video data supplied to this video encoder will be sourced directly from the FPGA in an 8-bit greyscale form.
  4. Actuators will be controlled by the switching of STV0674's GPIO connected to a STMicroelectronics 16-pin (PowerDIP) ULN2064B Quad Darlington switches IC. This IC can output 1.5A to each Darlington output at a sustainable voltage of at least 35V. As the switch IC also has integral suppression diodes, they are suitable for driving inductive loads, such as electric DC motors, stepper motors and solenoids [93]. The four outputs from the switch IC must be connected to screw terminals to provide the most flexible method of interfacing external devices
  5. Test and debug facilities should be provided in the form of a JTAG connector linked to the STV674 and a user-define port on the FPGA, in addition to the 2<sup>nd</sup> JTAG connector on the backplane which will be primarily used for FPGA



programming. All input and output signals to and from the video encoder and the sensor connector should be made accessible at the edge of the PCB via pin connectors. Three different coloured 1.8mm LEDs (Red, Yellow and Green) and three jumper switches toggling between ground and 3.3V, should be in a combined configuration in order to only occupy 3 GPIO connections to the STV0674.

6. Two board-level clock domains are required to drive the STV0674 sensor co-processor, STV0119a video encoder and the FPGA on the backplane. A 12 MHz and 27 MHz clock signal are required and will be generated using a crystal oscillator with the 12 MHz clock signal sent to the STV0674 and the 27 MHz clock signal sent to the STV0119a but both to the FPGA. Clock drivers should be used to transmit all clock signals to the necessary components, with resistor-capacitor networks providing the ability to balance clock delays between daughter board components and the FPGA.
7. Three voltage supplies are required for the daughter board's systems. These are 1.8V, 3.3V and 5V. 1.8V and 3.3V are supplied via the connection to the FPGA backplane, whereas 5V should be generated using a voltage regulator. Two layers of the PCB should be dedicated to a power plane and a ground plane. To reduce the possible effects of crosstalk, analog voltage supplies should be separated from the digital supplies via star-points on the power plane. For the same reason, the analog ground plane should also be split using star-points to provide separate grounds to the audio circuitry of the sensor and the co-processor's PLL.

### **3.2.3 System Buses Analysis and Specification**

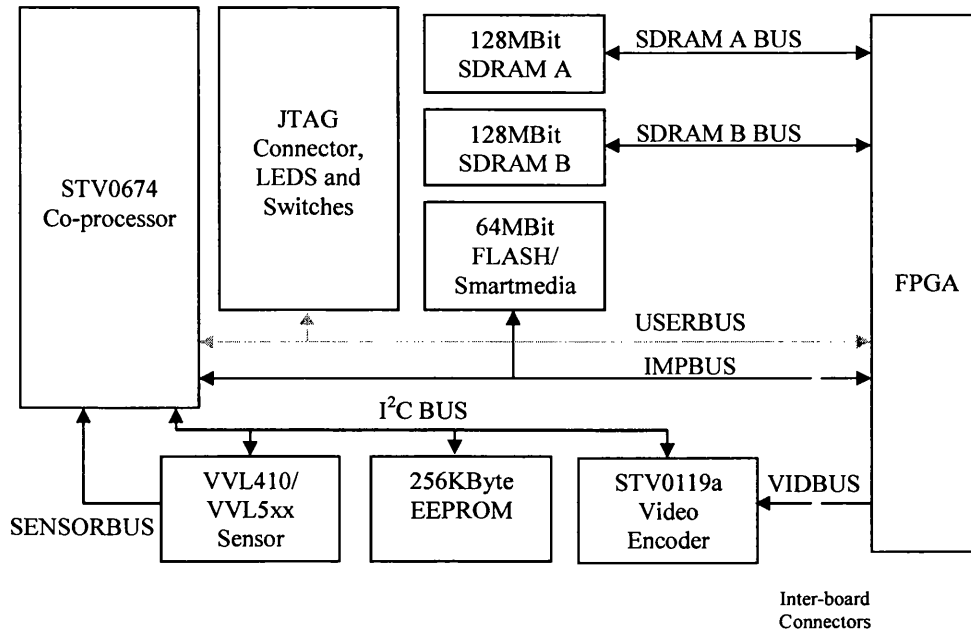
The decision to use a sensor co-processor to control the sensor and relay pixel data to the FPGA, had the effect of limiting the types of buses used due to the limitations of the co-processor's interfaces. The initial unavoidable bottleneck within the system was as a result of the data out interface on the available image sensors. This interface was only of a width of 5 wires and a maximum speed of 24MHz, providing

a bandwidth of 120Mbits/s. The second bottleneck existed between the co-processor and FPGA. As the co-processor only had general-purpose I/O (GPIO), I<sup>2</sup>C interface and USB interface, the bandwidth was limited to 2Mbit/s per pin, 400kbits/s and 12Mbits/s respectively. Fortunately, the GPIO could be driven by the co-processor's hardware SDRAM control module, providing a bandwidth of a maximum of 384Mbits/s at a fixed SDRAM bus speed of 24MHz. This was chosen as the main communications bus between the co-processor and FPGA. As the bandwidth was more than 3 times the bandwidth of the sensor's data bus, the option to use the co-processor between the sensor and FPGA was deemed just as effective as a direct connection from the sensor to the FPGA. This bus was referred to as the IMPBUS to distinguish it from the two SDRAM buses from the FPGA to the two SDRAM modules. Due to the fixed interface requirements on the other PCB level components, there was no choice in the selection of the remaining buses.

The prototyping system had 6 specified buses as follows;

1. IMPBUS – a shared 39 wire Memory/Communication bus operating at a maximum frequency of 24 MHz connecting the co-processor with the FLASH memory, Smartmedia connector and FPGA. This bus is the only pathway for communication between the co-processor and FPGA.
2. SENSORBUS – a uni-directional 5 wire data bus operating at a maximum frequency of 24 MHz between the sensor video output port and the video input port of the co-processor.
3. VIDBUS – a uni-directional 8 wire data bus operating a maximum frequency of 27 MHz connecting the video out port of the FPGA to the video encoder.
4. USERBUS – a shared 12 wire bus operating at a maximum frequency of 12Mhz facilitating control of the 3 LEDs, 3 input switches, 4 actuator switches and JTAG debugging provision for the co-processor.
5. I<sup>2</sup>C Bus- a shared 2 wire bus operating at a maximum frequency of 400 kHz for co-processor control of the video encoder, sensor and EEPROM.
6. SDRAM A/B Bus – two 38 wire combined SDRAM data, address and control buses operating at an expected frequency of 24 MHz. These two buses provide a direct path from the FPGA to the SDRAM ICs on the daughter board.

Figure 3.2 illustrates how these buses interconnect the different components in the prototyping system.



**Figure 3.2 Top-Level Daughter Board Bus Diagram**

### 3.2.4 Detailed IMPBUS Specification

Unlike the other buses, the USERBUS and IMPBUS were custom bus specifically designed for the prototyping system and hence did not have any publicly available datasheets. As the USERBUS was used to control on/off switching of devices, JTAG test signals and user-defined signals, no communication protocol was defined. Conversely, the IMPBUS had an important multi-purpose role within the system, which included inter-chip communications and therefore warrants further specification. The table A.1 in Appendix A lists the STV0674 ports, special function pins, actual pins and their mappings for the IMPBUS.

Three different modes of operation existed for the IMPBUS. The first mode was FLASH or Smartmedia card access and could operate separately or concurrently with the third mode, which was 16-bit bi-directional communication between the FPGA

and STV0674. The second mode had exclusive access over the bus for SDRAM transaction between the STV0674 and FPGA. The IMPBUS was designed for clock speeds of up to 2 MHz when communicating directly to and from the STV0674 and FPGA. This is due to the limitation that GPIO on the STV0674 could only toggle at a maximum of 2MHz. As a result, the maximum possible bandwidth was 32Mbits/s. SDRAM operations across the bus could be performed at 24 MHz, using the clock from the SDRAM controller located on the STV0674 and hence achieve a maximum bandwidth of 384Mbit/s. It was expected that the embedded memory in the FPGA and the two SDRAM attached to the FPGA, would be mapped into the co-processors addressable SDRAM space. This would provide the opportunity to access the FPGAs embedded RAM by implementing an FPGA resident SDRAM address decoder and using the same memory control commands as used when communicating directly with an SDRAM. The key advantage of this mechanism was that the maximum bandwidth was over a magnitude larger than that which the IMPBUS communications could provide, making it the best method for transmitting a large amount of data between the FPGA and STV0674 co-processor. Another advantage over IMPBUS communications was that SDRAM transactions were controlled by a dedicated hardware core and did not require any microcontroller bus control protocol emulation overhead, unlike the 16-bit communication mode. But a disadvantage was that only the co-processor could initialise and control the SDRAM transactions.

A specific IMPBUS communication protocol was not selected as the use of the bus may need to differ between applications. In some circumstances the constraint of a fixed bus protocol could have increased the complexity of the embedded software on the co-processor. This fixed protocol could have lead to unsatisfactory bus control execution times when a simpler and faster bus protocol could be devised by the application developer. Three extra wires were provided for a parity signal, bus clock and bus ownership indication to allow a greater control for error correction, synchronisation and bus contention, while reserving 16 wires for data transmission.

### 3.2.5 FPGA Design Analysis and Specification

Initial analysis of the prototyping system's requirements indicated that several FPGA design issues needed to be addressed. The digital encoder IC required video data to be encoded to the ITU-R BT.656 digital video standard and outputted on the 8-bit bus at 27 MHz to sustain the correct video timings for a PAL 50 fields per second video format. Each of these fields consisted of half an interlaced video frame, hence combining a field with its subsequent field would form one frame. As the sensor produced a video output at only 25 frames a second, each frame had to be outputted twice to meet the PAL video timing requirements. This indicated the need to use a frame buffer. As the sensor and video encoder operated asynchronously to each other and their timing requirements were hard deadlines, the frame buffer mechanism had to cope with potential skews between the two data domains.

The specification of the use of greyscale images for imaging processing at the beginning of the project had the advantage of significantly reducing the memory requirement for the frame buffer. Limiting the system to greyscale was not seen as a problem, as the majority of image processing can still be done without using colour data [94]. The sensor pixel was outputted in a YUV format, where each pixel is made from a luminance sample, Y, and two chrominance samples, U and V. Each sample was 8-bit wide, creating a 24-bit pixel with 16.7 million possible values. Using the luminance sample only, reduced the memory requirement to a third of the original value. For example, a single 8-bit greyscale VGA (640 by 480 pixels) frame had memory storage requirements of 2457600 bits. As the FPGA used on the backplane had 311269 bits, only a maximum 12.7% of a VGA image could be stored on-chip. Taking into account that the system was aimed at low-cost applications, this problem was overcome by sub-sampling the VGA image to create an image of 80 by 60 pixels. Where more memory is required, the external SDRAMs or a backplane with a FPGA containing greater memory sources could be used. The option of a smaller image size, allowed up to 8 whole images to be stored within the FPGA. Sensor video stream decoding and sub-sampling was performed in a sensor interface IP block fed from an asynchronous FIFO. At this point, the 80x60 pixel images were to be outputted to a monitor. The image would be scaled-up horizontal and vertically by

a factor of 8 to reconstruct a VGA image. This scaling process duplicated each pixel a further 7 times to increase the line length from 80 to 640 pixels.

The Altera FPGA provided the option of implementing the frame buffer as a single-port memory, i.e. read or write at one time, or a pseudo dual-port memory, allowing a reading from one port and concurrently writing to the other port. A true dual-port memory would have two independent ports, each capable of being read and written independently of the other port. As with all the Altera 20K FPGAs, true dual-port memories cannot be implemented. The use of pseudo dual-port memory creates more complex control structures than a single memory and has the potential to reduce the memory's maximum operating frequency. Other limitations include the following:

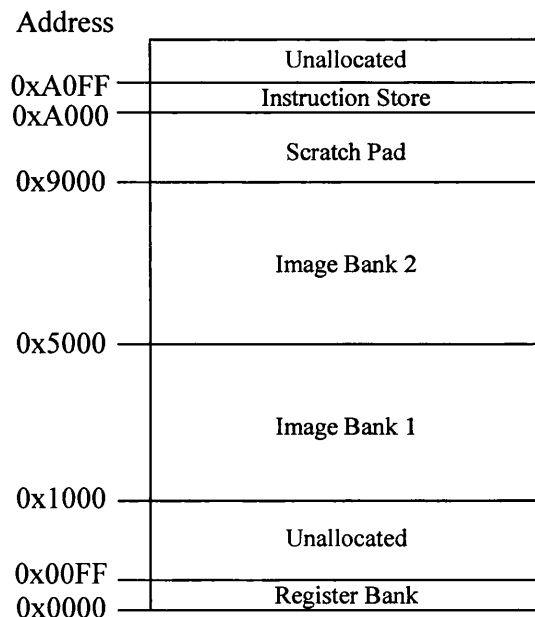
- There may be instances where an image is read out during or before it has been processed resulting in a partially processed or unprocessed image being visible on the monitor.
- Where memory requirements exceed the FPGA's resources, external SDRAM memories would be required. SDRAM are single-port memories and therefore it would be necessary to re-design the FPGA's memory interfaces.

To bypass these limitations, a two single-port RAM configuration was chosen for implementation of the image memory banks. A limitation of using only two memory banks for image storage, is that during cycles when an image is being read out to the video encoder, operations such as absolute differencing, are complicated due to sharing of the same memory bank. Adding a further image bank would alleviate the problem by always providing an image bank solely for uninterrupted image processing. This would of course be at the cost of adding latency on the image read out to the video encoder by one frame period. An extra image bank was not implemented, as applications initially developed using three on-chip image banks could not be as easily implemented later, using the two off-chip SDRAM ICs.

An IP block was specified to interleave or "ping-pong" image data between the two image banks. This allowed the system to write new image data into one image bank, while reading image data from the other image bank to the video generator. Using

the interleaving mechanism ensured that the frame rate could be maintained at 25fps. The size chosen for these image banks was 16Kbytes, each capable of storing 3 images. Three further memory banks were required, one for the storage of temporary data, one as a register bank and another for storing instructions for the system controller. Temporary data was stored in a 4KByte scratch pad as it was expected that some image processing IP blocks may need concurrent access to image banks and another separate bank for variable data. The register bank was used to control the different functions within the system, e.g. activation of image processing functions or test modes. A total of 256 Bytes were allocated for the register bank. The instruction store was allocated 256 Bytes providing enough capacity for 256 instructions or literal values.

The combined size of all the memory allocated in the FPGA was 37376 Bytes. This figure was 96% of the total number of memory bits available on the Altera FPGA used. Some of the remaining memory was used for FIFO structures. The memory map for the FPGA was specified as in figure 3.3.



**Figure 3.3 FPGA memory address map**

Each block of allocated memory was allocated to start on hex addresses that would be simple to remember and easily decoded. As the pixel data used in the system was a byte in size, the memory widths chosen for the memories were also byte-wide. This design decision would simplify the design of data ports on IP blocks and memories and reduce the internal FPGA wiring requirements for buses. A reduction in bus widths generally can result in less possibility of skew between data bus wires and can potentially reduce problems in meeting timing requirements. The disadvantages of using byte wide memories are that it reduces the number of pixels readable per cycle to one and limits the width of instructions within the instruction store.

A system controller was implemented in conjunction with the instruction store, to provide a programmable mechanism to control all the sub-systems within the FPGA. As the memory instantiated in the FPGA was 8-bit wide, the system controller had to use specialized coarse-grained instructions to perform the necessary operations. The system controller could access the instruction store independently from the rest of the system. This was advantageous as the programs were separate from the data memory allowing simultaneous access. This provided higher performance and ensured the constant flow of instructions to the system controller without interfering with other memory operations. This form of memory partitioning is known as Harvard architecture and is found in most modern DSPs [95].

A different method for providing a programmable mechanism could have been the inclusion of a microprocessor or microcontroller on the FPGA instead of the custom system controller. It has become popular for FPGA system designers to use either the Nios family of microprocessors when using Altera FPGAs or the Microblaze family when using Xilinx FPGAs. Both 32-bit microprocessors are provided by the FPGA vendor as soft IP blocks optimised for the vendor's FPGA architectures. Although these IP blocks are low cost and relatively easy to implement, licensing for use in silicon designs such as IoCs can be complicated by licensing issues and high costs. Generally these IP blocks are only offered for use in the vendors FPGA, encouraging the use of the vendor's FPGAs for the end-product. As end-products from the prototyping platform are likely to be a single chip, these devices probably would not be cost effective.



Tensilica provides the 32-bit Xtensa family of soft IP block microprocessors. These have the advantage that the company is fabless and not tied to any particular silicon technology. The development suite provided by Tensilica allows processors to be customized for a particular application, by changing the processors architecture and allowing the instruction set to be extended using new instructions. These instructions control optimized logic that results in more efficient processing performance. As a third party vendor there are still potential licensing and cost issues with the use of their microprocessors.

A similar microprocessor to Tensilica's Xtensa which is available freely for commercial use under a GNU GPL licence, is the SPARC compatible Leon 2, developed originally for the European Space Agency. This is also a configurable 32-bit soft IP block microprocessor which may be extended by the addition of up to a further 5 instructions. Although this may seem the most appealing option for a microprocessor, as with the majority of open source software and hardware, there is no specific company offering user support.

All four processors discussed have the advantage that compared to the system controller, they are far more flexible and more computationally powerful. These processors could have been used for some image processing functions that would be more efficiently implemented in software than as a hardware DSP IP block. As well as controlling the sub-systems in the FPGA, they could also be programmed to control the sensor from the FPGA, removing the need for the sensor co-processor. The disadvantages of all the processors are:

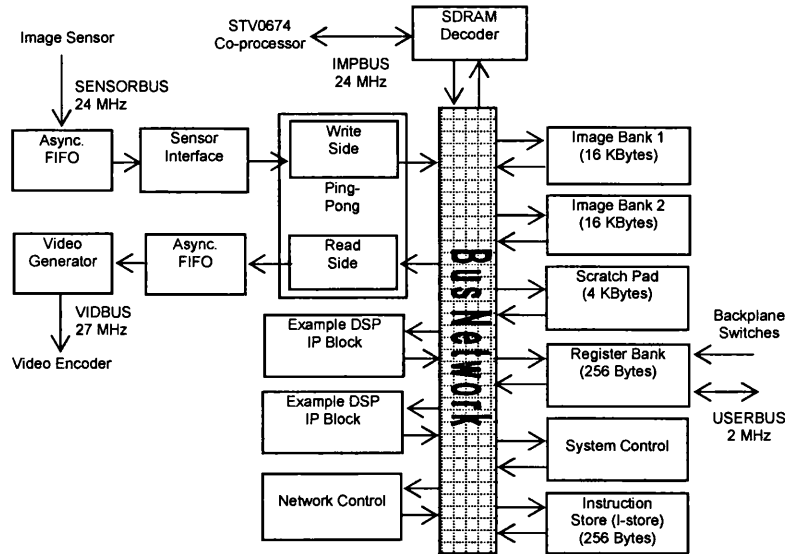
- All are 32-bit and would consume larger amounts of FPGA resources when compared to the system controller.
- The complexity of the IP block microprocessors could reduce the overall system clock frequency attainable, due to their affect on the place and route of the design onto the FPGA.
- The pixel data and memory architecture is only 8-bit and therefore the 32-bit microprocessors are unlikely to be more efficient than the 8-bit optimised system controller for many functions.

- The design flow is likely to become more complex for the prototyping platform's software, as new tools may be required and programming code would need to be ported or re-written for the new microprocessor.
- STMicroelectronics does not have IP ownership of these microprocessors and therefore there may be problematic or expensive licensing issues.

Irrespective of the advantages of the inclusion of a soft IP block microprocessor, STMicroelectronics requested that the FPGA design should not use any third party IP. This left no other option than to design the system controller, as no in-house processors were made available. If a mesh and linear array processing approach had been adopted, it would have been essential to add a microprocessor onto silicon die. Its inclusion would have been to perform the high-level image processing functions that both architectures are incapable of performing.

External access to the FPGA was provided by the IMPBUS and a SDRAM decoder. The SDRAM decoder converted SDRAM read and write transactions into internal reads and writes to the banks of memory embedded in the FPGA. It supported two data widths from the IMPBUS. These were 8-bit or 16-bit wide data which was transformed into multiple internal 8-bit wide data transactions. The SDRAM transactions provided flexibility in allowing the FPGA part of the prototyping system, to be used with processors or co-processors, other than the STV0674 provided. The disadvantage of this approach was that in a situation where the co-processor required information from the FPGA, it would still expect valid data to be returned from the FPGA within a given time window. The reason for this was that the co-processor's SDRAM controller communicated to the FPGA, as if it was an SDRAM module. This could cause problems if latency in the FPGA architecture resulted in strict memory timing deadlines for outputting data not being met. It was expected that a high system clock frequency for the FPGA system would prevent these timing issues from occurring. Without the implementation of the SDRAM decoder and the use of the SDRAM bus, all communications between the sensor co-processor and FPGA would be at the slow rate of 2Mbits/s per pin.

Figure 3.4 shows the simplified top-level IP block architecture specified for implementation within the FPGA, including the SDRAM decoder and its connection to the rest of the sub-systems via a bus network.



**Figure 3.4 Top-level IP block architecture in the FPGA**

The bus network specified for the FPGA architecture consisted of multiple point-to-point buses synchronized with the system clock. As only a small number of interconnections existed within the architecture, it was seen to be unnecessary to implement a traditional bus structure, such as, STMicroelectronics' ST-Bus, IBM's CoreConnect or Arm's AMBA, or a network-on-chip structure, given their associated control overheads. These point-to-point buses provided low latency communications links from sub-systems and DSP IP blocks to memories. As the register bank was used as a means for one sub-system or DSP IP block to indirectly control another, minimal direct wiring was required between DSP IP blocks and/or sub-systems. Hence, all sub-systems only required a direct connection to the register bank rather than a traditional shared bus providing full connectivity between all sub-systems. A small network controller was implemented to provide arbitration over the network of point-to-point buses, using multiplexer structures and a priority mechanism based on system status. This network controller primarily controlled the access to the different shared memory banks. Table 3.1 show the possible interconnections between the different parts of the architecture.

	<b>Image Bank 1</b>	<b>Image Bank 2</b>	<b>Scratch Pad</b>	<b>Register Bank</b>	<b>Instruction Store</b>
<b>SDRAM Decoder</b>	Yes	Yes	Yes	Yes	Yes (*)
<b>System Controller</b>	Yes	Yes	Yes	Yes	Yes
<b>DSP IP Blocks (Each)</b>	Yes	Yes	Yes	Yes	No
<b>Ping-Pong Unit</b>	Yes	Yes	No	No	No

(\*) In limited situations

**Table 3.1 Interconnections between sub-systems, DSP IP blocks and memories on the FPGA**

The limitation of using architecture with IP block interfaces designed for point-to-point buses, is that it may be more complex to re-use the IP blocks within a traditional shared bus structure. This issue could be addressed by using a ‘wrapper’ around each IP block to translate signals from the point-to-point bus interface, into a format suitable for a new bus structure, for example Arm’s AMBA bus.

The original project requirements from STMicroelectronics requested a core set of re-usable operators. This was extended during the project to specify that the operators ideally should be useable in other future STMicroelectronics products. This requirement further justified a frame based approach to processing, as processing elements for mesh and linear array processing architecture are specifically designed for a single architecture, i.e. generally not for reuse in a wide range of potential architectures. As a result, it was deemed applicable to develop optimised hardware DSP IP blocks with standardised interfaces, to interface with a point-to-point bus structure. Each DSP IP block could be optimised for a specific operation, unlike processing elements in mesh and linear array based processing arrays. This had the distinct advantage that future STMicroelectronics products could use just the DSP IP blocks they required, rather than general-purpose unoptimised processing elements.

As the emphasis for these DSP IP blocks was to lay the foundations for the future realisation of tracking and object recognition, several operations were selected. The selection of these operations was as a result of the analysis of demonstration applications, implemented on the original VLSI Vision Imputer. The specific operations selected were:

- Draw rectangle
- Threshold image
- Get maximum and minimum coordinates of all active pixels
- Absolute difference between two images
- Copy an ROI or Image to a new memory address
- 3x3 neighbourhood convolution filter
- Find all active objects within an image and store their attributes

Although these were selected for their suitability for tracking and object recognition, since they are low-level operations they would be used in a wide range of applications or form the basis of more complex algorithms and operators. For example, absolute differencing is also used in image compression techniques and the 3x3 filter for a whole host of low pass, high pass and edge detect operations found in most image enhancement applications. The limitation of the DSP IP block operator selection, is that some of the more simple operations, such as ‘Add literal value to pixel’, would have to be performed by using the microcontroller in the sensor co-processor. Despite this limitation, the selection of DSP blocks provides a range of reusable frameworks for the development of new DSP IP blocks which maybe required by the end user of the system. The major difference between the frameworks of the DSP IP blocks, is the number of connections provided to the point-to-point bus network and the memory addressing scheme supported.

### **3.2.6 System Firmware**

Initially, it is expected that the application developer will start development with the standard STV0674 Webcam camera firmware patch as a base for new applications. This will provide the opportunity to perform functionality tests on the sensor/co-processor pairing and USB communications. Following preliminary testing the standard patch will be modified to initialise the video encoder, switches, LEDs and configure the sensor, if required. On completion of any modifications of FPGA-resident image and machine vision processing architecture, the firmware can be augmented with suitable application code.

### **3.2.7 Cost Issues**

During the analysis and specification of the two PCB prototyping platforms, the issue of the relative cost of each design decision was addressed. In particular, the decision to use a frame-based architecture allowed the use of a reprogrammable FPGA rather than being forced to implement a mesh or linear array based architecture in silicon. If a multi-project wafer production run was used, the cost would have been likely to be in the region of two magnitudes different compared to the relatively low cost of the FPGA and daughter board. Also, given that the imager would have to be integrated onto the silicon die, this would add further risk of higher costs due to the complex implementation requirements of a mixed-signal CMOS imager architecture.

Four other cost aware design decisions were:

1. The selection of memory devices supported by the prototyping platform only included low-cost solutions such as non-volatile FLASH memory and high capacity SDRAM ICs instead of high cost, low to medium capacity SRAM ICs.
2. The use of the sensor co-processor removed the requirement for valuable FPGA resources for a sensor control IP block or the cost of licensing a general purpose FPGA-embeddable microprocessor.
3. Using a two PCB approach to prototyping reduces the need to redesign both PCB designs if the FPGA's support architecture needs future changes to support a new FPGA device. This is in addition to the cost benefit of using an in-house solution for the FPGA backplane rather than using a third party solution.
4. The decision to use a STMicroelectronics video encoder IC instead of a D/A convertor IC, reduced the requirement for a more complex video generator IP block to be design for the FPGA. The reduction in design and implementation time would be expected to result in a reduction in overall development cost.

The main cost disadvantage as a result of a design decision, was that of using a separate co-processor and system controller. This decision would typically require two different software design flows, rather than a single unified approach. The complexity of two design flows could lead to longer development times for application developers and hence larger development costs.

### **3.3 Summary**

This chapter has contained the requirements, analysis and specification which the design of the prototyping system was based upon. Further justification has been provided in the adoption of a frame based processing architecture, with emphasis on STMicroelectronics requirements of the use of their frame based sensor technology and the need for re-usable hardware based operators in the form of DSP IP blocks. The main PCB components have been listed and the primary bus structures between them defined. Particular attention has been paid to the IMPBUS due to its unusual configuration, flexibility and given it is the primary bus between daughter board and FPGA backplane. Details have also been provided with respect to the image processing architecture to be implemented within the FPGA on the backplane.

## **4 FPGA System Design, Test and Results**

This chapter concentrates on the methodology used to create the on-chip architecture for use in the FPGA on the backplane. The description of the architecture has been separated into two sections, firstly the core architecture which is essential for the correct functionality of the prototyping system and secondly an IP block library providing further flexibility. The simulation and functional verification flows have also been discussed. Details regarding the development of the PCB daughter card connected to the FPGA backplane are located in appendix B.

### **4.1 FPGA Design Flow**

The FPGA system required a design flow which would ensure that every IP block developed for the core architecture and IP block library had undergone the same development process. Given that the FPGA system would be used by application developers, the IP blocks had to be carefully integrated into the complete architecture. This was achieved by individually simulating each new IP block. Each block was then simulated as part of the system and functionally verified at full operational frequency within the prototyping system on the bench. The top-level FPGA design flow used is outlined below.

#### **1. FPGA architectural analysis and Verilog implementation**

- i. Assess the number of FPGA I/O pins required to support the necessary off-chip data and control buses in addition to clock and reset signals.
- ii. Produce Verilog HDL code for mapping the FPGA pins to the internal interfaces of an instantiated core module, representing the architecture to be developed in the FPGA.
- iii. Produce a new file containing the core level description of the architecture and the I/O interface to the FPGA pin mappings.

#### **2. IP block design (perform steps for each IP block)**

- i. Specify the I/O interface for the IP block in a new design file.



- ii. Implement the IP block in synthesizable Verilog in the new design file.
- iii. Perform checks on the correctness of the Verilog code and check that it meets re-usability, readability and synthesizability requirements.
- iv. Write a Verilog test algorithm that performs a series of tests on the IP block by applying different combinations of values to the IP block inputs.
- v. Perform a Verilog simulation by instantiating the IP block under test and the test algorithm and executing the Synopsys VCS simulator. If functioning correctly move on the IP block integration stage.

### **3. IP block integration (perform steps for each IP block)**

- i. Initiate the IP block within the core level description file.
- ii. Perform the necessary wiring to the core level's I/O interface.
- iii. Increment/create the system's Verilog test bench to include the new IP block test algorithm. This differs from testing during the IP block design as the test bench should only apply signals to the core levels I/O interface.
- iv. Perform Verilog simulation using Synopsys VCS simulator.
- v. Synthesize the complete FPGA design to a netlist using Synplify.
- vi. Compile and fit netlist design to create a FPGA bit stream programming file using Quartus II.
- vii. Perform timing analysis and check that there is sufficient slack along all internal FPGA wire paths.
- viii. Download design to FPGA/EPC2 using the Altera Byte Blaster cable and Altera MAX+PLUS II software programming tool.

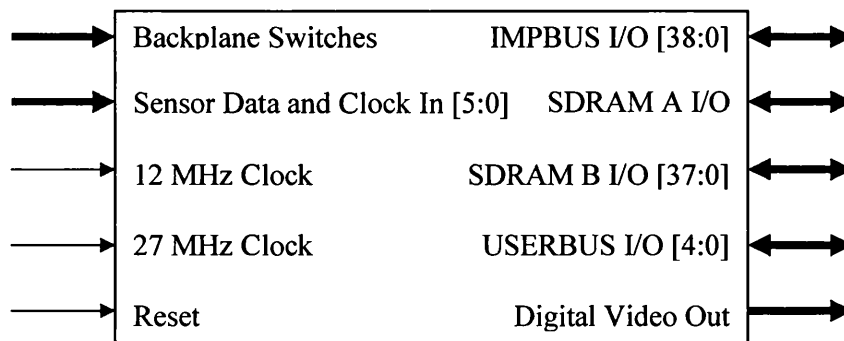
### **4. Post integration system verification (perform steps for each IP block)**

- i. Generate a test plan for the FPGA which can be executed at PCB level, either automatically or using user input from switches.
- ii. Verify the functionality of whole prototyping system with the new IP block using the test plan.

## **4.2 FPGA Top Level Mapping**

The first step in an FPGA design is the definition of the top level mapping of the external I/O pins of the FPGA to internally defined wires. This mapping takes the

type of signal into account to ensure that the FPGA's pins are be correctly configured. The original daughter card PCB was modified it to directly connect the output of the sensor access port to the FPGA. The reason for this modification was to reduce any processing overhead on the co-processor, reduce latency of the transfer of image frames to the FPGA for processing and reduce the utilisation of the IMPBUS. The addition of this sensor data input port on the FPGA, resulted in an increase in the total number of pins defined to 140, excluding ground and power supply pin mappings. The top level pin mapping for the FPGA is shown in figure 4.1.



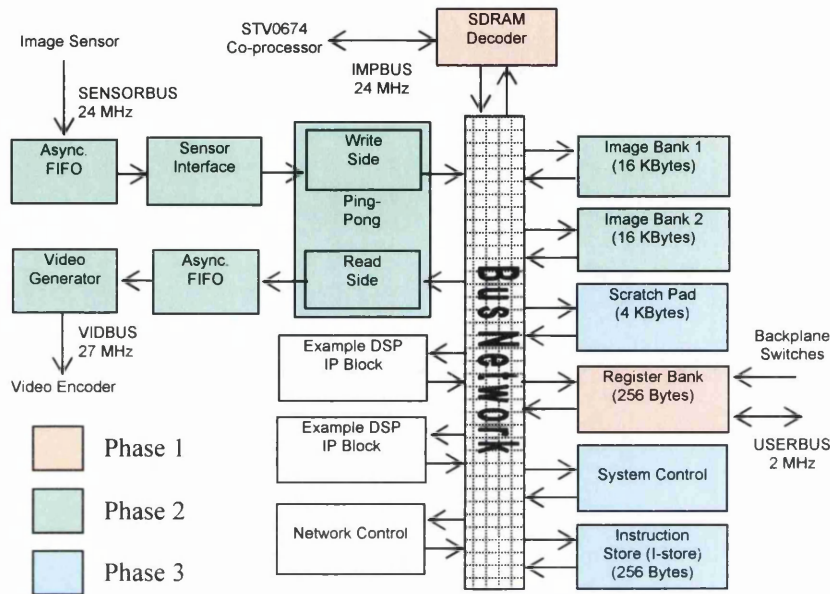
**Figure 4.1 Top-level pin mappings for the FPGA**

### 4.3 Core Architecture

The development of the core architecture connect to the top-level pin mapping was divided into 3 separate phases. Each of these phases resulted in a functional system with the new architectural extensions easily demonstrable on the bench. The three phases were:

1. The construction of the SDRAM decoder and register bank (marked in orange in figure 4.2). This phase demonstrated that data could be written from the STV0674 co-processor to the FPGA at a given address within the register bank
2. The development of the sensor interface, video encoder, ping-pong unit and two image banks (marked in green in figure 4.2). The completion of this phase demonstrated the architecture's ability to meet the required hard deadlines for the sampling of sensor data and its storage and output in a suitable format, at 25fps to the video encoder IC.

- The implementation of the system controller, instruction store and scratch pad memory (marked in blue in figure 4.2). This enhanced the FPGA-based architecture allowing instruction storage and autonomous processing to control the various sub-systems without the intervention of the co-processor.



**Figure 4.2** The three phases of development for the core architecture

The DSP IP block library, bus network and network control unit (marked in white in figure 4.2) were developed throughout the three phases and hence converged on an optimal solution for each phase. The following sub-sections provide detail on each of the core architectural components at the end of phase 3.

### 4.3.1 Register Bank

The register bank was the key component in the FPGA architecture as it stored the system configuration and current status in a readable, and in many cases writable, byte-wide configuration. The activation of new operations from the system controller required the use of the register bank as an intermediate means of controlling the ping-pong unit and DSP IP blocks. This provided a mutual exclusion mechanism that ensured the ping-pong unit or DSP IP blocks could only be configured by one other sub-system in any given clock cycle. Data stored in the register bank structure could be accessed in two ways, namely, directly, where a wire

was permanently connected to the output of the register, or indirectly using a memory-like read request. Typically, all system control and DSP IP block execution requests were controlled by the direct method to ensure the lowest latency. A write to a register used a synchronous memory-like write request, except in the case of a DSP IP block execution acknowledgment. This exception allowed the register bank to be freely readable and writable using the memory-like interface during the acknowledgement of the execution of a DSP IP Block. This was particularly important as after a DSP IP block received an execution request, it may need to access the register bank to obtain configuration data. The minimal recommended configuration for the register bank required the input/output interface as shown in table 4.1. Connections on the left side, indicate inputs and connections on the right, the outputs. Buses of wires are indicated by <n:0>, where n is the number of wires, minus one, in the bus.

I/O Name	Description
pc_val <7:0>	Program counter value
regdin <7:0>	Register data in
reg_addr <7:0>	Register address select
clk48	System clock
env_active	Active pixel data flag
ip_matrix_busy	DSP IP blocks busy
odd_even	Odd/Even frame flag
rdstrobe_n	Read request (active low)
reset_n	Asynchronous System Reset (active low)
switch_value	Switch value flag
wrstrobe_n	Write request (active low)
ping_pong_cfg <7:0>	Ping-pong configuration
regdout <7:0>	Register data out
test_sel <7:0>	Test mode select
config_complete_n	I-store access select (active low)
dsp_ctrl_rst	System controller reset
pingpong_dsp_ctrl_val	Ping-pong deactivate

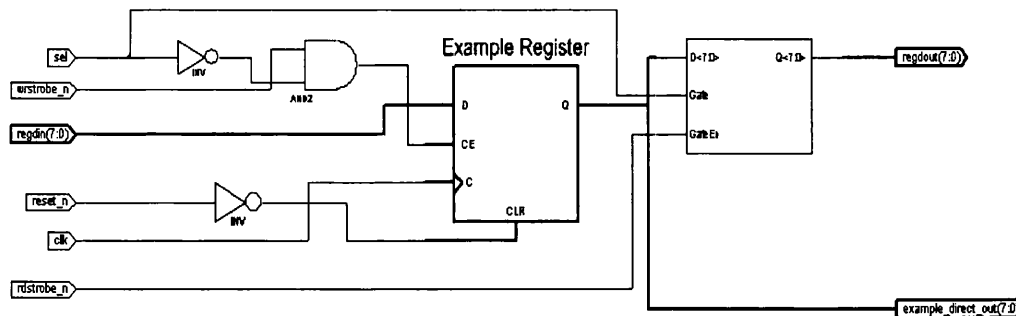
**Table 4.1 Minimum set of I/Os for the FPGA Register Bank**

These I/O were mapped to the first 10 addresses in the register bank and are listed in table 4.2.

Address	Name	Type	Function
0000h	device_id	Readable / Writable	Device Identification Number
0001h	rev_number	Readable / Writable	Device Revision Number
0002h	blank_0	Readable / Writable	Not Used
0003h	blank_1	Readable / Writable	Not Used
0004h	dsp_ctrl	Readable / Writable	System Controller and Program Counter Reset, I-Store Configuration Control and Ping-Pong Deactivate
0005h	pc	Readable	System Controller's Program Counter value
0006h	bp_ctrl	Readable / Writable	Status of 3 Backplane push-button switches
0007h	ping_pong_ctrl	Readable / Writable	Image bank selection and freeze-frame control
0008h	test_select	Readable / Writable	Test mode select
0009h	status_out	Readable	Active image data grab flag, DSP IP blocks busy flag and odd/even frame flag.

**Table 4.2 Minimum register map for FPGA Register Bank**

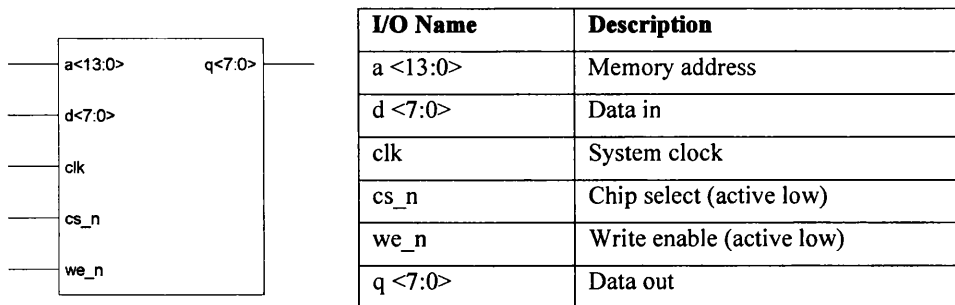
The typical structure of the register bank involved the repetition of similar register structures to create an array of register instances. Due to the complex nature of the schematics produced during synthesis of the register bank, only one register instance is provided in figure 4.3, which illustrates a typical structure.



**Figure 4.3 Single instance of an example 8-bit register from the register bank**

### 4.3.2 Memories

The image banks, I-store and scratch pad single-port memories in the core architecture possessed the same interface, internal structure and operated synchronously with the system clock. To all intent purposes, these single-port memories acted as static RAMs (SRAM). Table 4.3 shows the common top-level interface for these memories. The width of a memory address input varied depending on the size of the memory.



I/O Name	Description
a <13:0>	Memory address
d <7:0>	Data in
clk	System clock
cs_n	Chip select (active low)
we_n	Write enable (active low)
q <7:0>	Data out

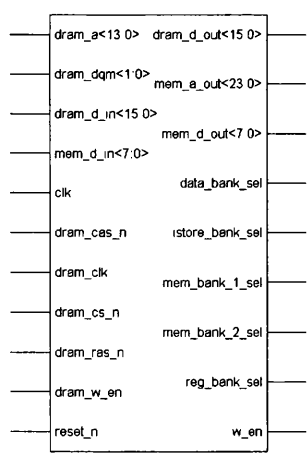
**Table 4.3 Top-level memory interface to system architecture for an Image bank**

A single parameterisable Verilog memory model was used to instantiate and synthesize these four memories. As the design decision was taken to implement two image banks, this limited the size of each image bank to 16Kbytes, as it was not possible to implement two 32Kbyte image banks within the FPGA due to lack of memory resources. The Altera 20K FPGA used in the prototyping platform, constructed memories from embedded systems blocks (ESB), each containing 2,048 programmable bits. Therefore, each 16KByte Image bank, 4KByte Scratch pad and I-store used 64,16,4 ESBs respectively. Once an address had been set-up on the memory interface ready for a rising clock edge, valid data for that address would be outputted on the data output ready for the next rising clock edge. As the memory bank data I/O interface had been specified to be byte-wide, the resultant pixel read/write rate was fixed at one pixel per clock cycle.

### 4.3.3 SDRAM Decoder

A SDRAM decoder was used to convert 16-bit data read and 8-bit data write SDRAM transactions from the IMPBUS. The SDRAM transactions were converted to 8-bit data read and write transactions suitable for the on-chip SRAM memories and Register Bank. The implementation of the decoder was as a direct result of the design decision to use a sensor co-processor as an intermediate between the sensor and FPGA. As such, this SDRAM transaction conversion process was used as the primary form of communication between the co-processor and the FPGA-based architecture. The SDRAM decoder could only access one of the image banks, I-Store or scratch pad or register bank at one time.

A strict set of timing requirements had to be adhered to for each transaction to be correctly completed. These timings were dictated by the SDRAM hardware control block within the co-processor. The co-processor supported single 16-bit word reads and byte writes and burst (4 words) writes and reads. The reads incurred a fixed two cycle latency from setting up the address on the bus, until the first valid data was available. Unfortunately, the burst modes were not available when controlling the SDRAM bus with the co-processor embedded micro-controller. As it was expected that the micro-controller would be used to control SDRAM transactions between it and the FPGA, only word reads and byte writes were supported with the SDRAM decoder. The SDRAM decoder also supported DQM data masking during write transactions to ensure data was written to the correct on-chip address. The I/Os for the decoder are summarised in table 4.4. The timing requirements for the co-processors SDRAM controller are shown in figure 4.4.



I/O Name	Description
dram_a <13:0>	SDRAM address
dram_dqm <13:0>	SDRAM data mask
dram_d_in <15:0>	SDRAM data in
mem_d_in <7:0>	On-chip memory data in
clk	System clock
dram_cas_n	SDRAM column select (active low)
dram_clk	SDRAM synchronization clock
dram_cs_n	SDRAM chip select (active low)
dram_ras_n	SDRAM row select (active low)
dram_w_en	SDRAM write enable (active low)
reset_n	Asynchronous System Reset (active low)
dram_d_out <15:0>	SDRAM data out
mem_a_out	On-chip memory address out
mem_d_out	On-chip memory data out
data_bank_sel	Data Bank select flag
istore_bank_sel	Instruction Store select flag
mem_bank_1_sel	Image Bank 1 select
mem_bank_2_sel	Image Bank 2 select
reg_bank_sel	Register Bank select
w_en	On-chip memory write enable (active low)

**Table 4.4 I/O interface for the SDRAM decoder**

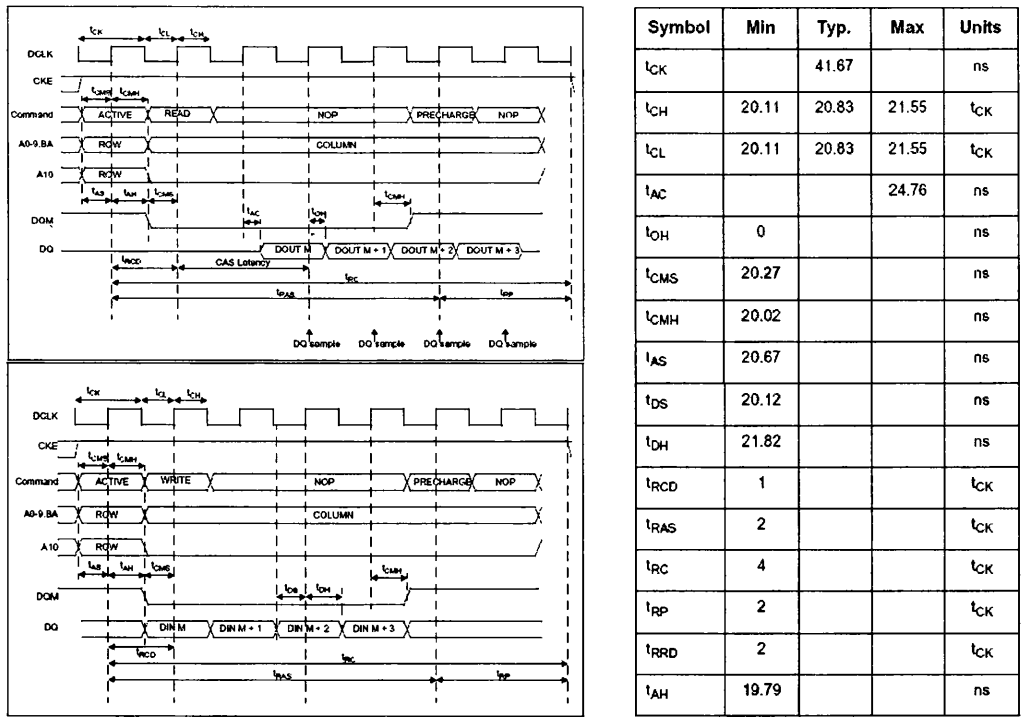


Figure 4.4 Co-processor's SDRAM read (top) and write (bottom) timing requirements [91]

Although Figure 4.4 shows multiple word read and writes, the timings are also applicable for single word or byte transactions. SDRAM transactions started with an "ACTIVE" command which involved the assertion of the SDRAM addresses most significant bits. This was followed on the next SDRAM clock cycle by either a "READ" or "WRITE", with the remaining least significant 9-bits of the SDRAM address. The commands given by the co-processor were constructed using the control lines of the IMPBUS. These commands and associated control signals are listed below in table 4.5.

	ACTIVE	READ	WRITE
dram_cs_n	Low	Low	Low
dram_cas_n	High	Low	Low
dram_ras_n	Low	High	High
dram_w_en	High	Low	High

Table 4.5 SDRAM commands and associated signals to start read and write transactions

Implementing the write transaction decoding within the SDRAM decoder was relatively simple as no signals were transmitted back to the co-processor following a write. This functionality had one main timing requirement which was that the



system must be able to process and store the byte of data in a single cycle. The timing requirements for a read word transaction were more complex. After the assertion of a read word transaction on the bus, the co-processor required the full 16-bit word to be a valid output ready for sampling after two 24 MHz cycles, i.e. 83.34ns. This is known as the CAS latency. This was further complicated by the fact that the systems memory structure and data bus are 8-bit wide and therefore two bytes must be read from on-chip memory to output a 16-bit word on the external bus. This extra byte read required an extra system clock cycle.

The process used to collect and construct the 16-bit word, following the detection of the start of a read transaction, was as follows:

1. Sample read address at system clock edge from the IMPBUS. Output first memory address as a read request on the selected on-chip memory bus.
2. Collect first byte and store as least significant byte of 16-bit word and output second memory address, i.e. increment the first address by one, as a read request on the selected on-chip memory bus.
3. Collect second byte and output complete 16-bit word on IMPBUS bus ready for DQ sampling by the co-processor.

The system clock is not necessarily in synchronisation with the SDRAM data clock on the IMPBUS. A system clock frequency of 24MHz or more, may result in the sampling of the read address up to 20.83ns ( $t_{CH}$ ) after the rising edge of the SDRAM clock. This would leave only 62.51ns to output the 16-bit word on the external bus. Hence a minimum system clock period of 20.83ns, and clock frequency of 48 MHz, was required to complete the aforementioned three step process to collect and construct the 16-bit word. This clock frequency assumes the output data path for the second byte does not contain a register and that the 16-bit output is latched on the next system clock cycle until a new read transaction takes place.

#### **4.3.4 Sensor Interface**

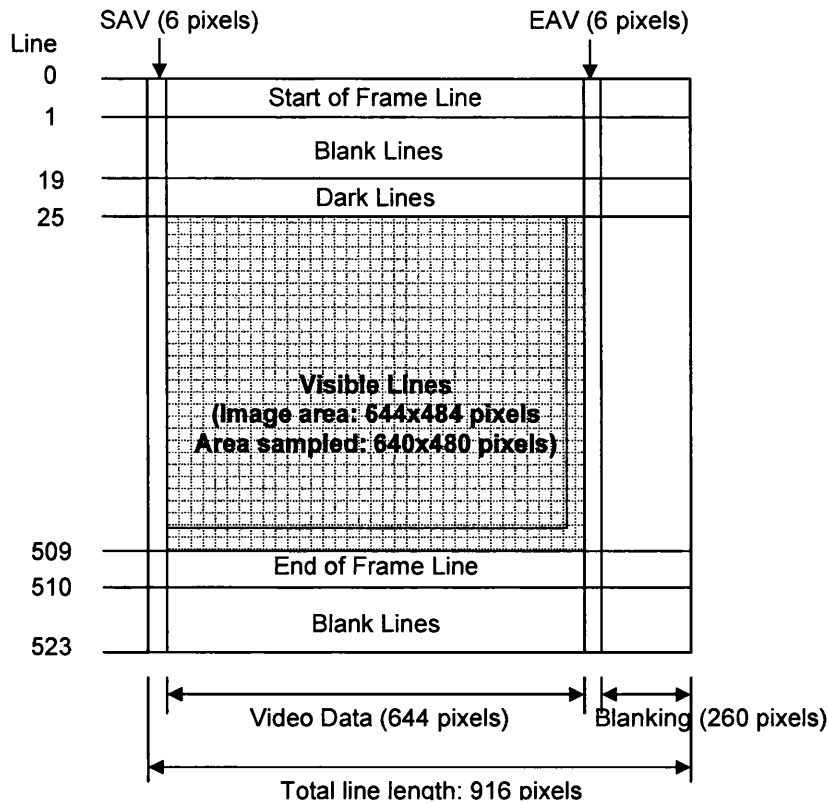
It was decided that the original method of transmitting image data to the FPGA via the co-processor would be overly complicated. A sensor interface was designed to

directly process image data from the image sensor. As STMicroelectronics had requested that their sensors were used, this required a 5-wire data interface to be implemented to connect to the sensor 5-wire bus. The incoming 5-bit VGA image data at 25fps from the sensor's five wire data bus was sampled and queued in an asynchronous FIFO to be processed by the sensor interface. The FIFO was used as a means to sample the data on the rising edge of the sensor clock, while allowing the data to be read to the sensor interface running on the system clock domain. This was important as the sensor clock varied depending on the mode of the co-processor. Although, the prototyping system was set to a specified sensor clock rate, the use of the FIFO allowed for more flexibility in the use of the system with other future unforeseen sensor modules. The only control signal passed from the FIFO to the sensor interface, was an FIFO empty signal, which was used to prevent reads from the FIFO when no data was present. Two 5-bit nibbles were concatenated to form bytes before the 8-bit embedded codes were processed. As the image sensor produced 8-bit image data value, only the top 8-bits of each nibble pair were used, with the first nibble in each pair representing the least significant bits. Therefore, two nibbles formed one pixel.

Embedded codes within the sensor data allowed complete images to be correctly interpreted, sub-sampled by a factor of 8 and outputted in the correct sequence and format, to the ping-pong unit. In addition to the image data, an active data flag indicating valid pixel data to the ping-pong unit was implemented. A vertical synchronisation pulse was also outputted to the ping-pong unit after decoding a start of frame line code. Table 4.6 shows the I/O for the sensor interface and figure 4.5 shows the format of each sensor image frame.

I/O Name	Description
sdata <4:0>	Image data from sensor via FIFO
sub_samp_x <3:0>	Horizontal sub-sampling factor (default 8)
sub_samp_y <3:0>	Vertical sub-sampling factor (default 8)
clk	System Clock
reset_n	Asynchronous System Reset (active low)
sensor_fifo_empty	FIFO empty flag
image_data <7:0>	Reconstructed 8-bit image data
env_active	Active data flag
v_sync	Vertical synchronisation flag

**Table 4.6 I/O interface for the sensor interface**

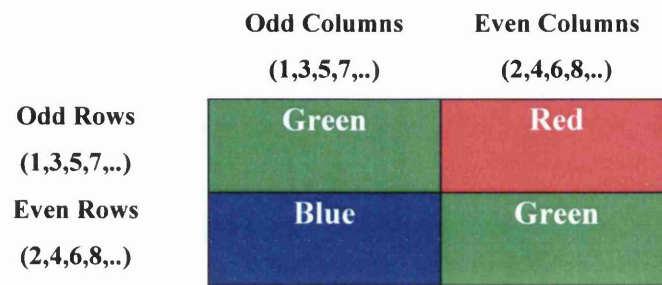


**Figure 4.5 Sensor VGA image frame and timings at 25FPS**

As can be seen from figure 4.6, each frame had a total of 916x524 pixels, equalling 479984 pixels in total. Each nibble was outputted at a default sensor clock frequency of 24 MHz resulting in a pixel clock frequency of 12 MHz. This pixel clock rate, gave a frame time of 0.040s and hence a frame rate of 25.00fps.

The SAV and EAV codes shared the same first six, 5-bit nibbles of 3FF-3FF-000, know as an escape code. Following the SAV's escape code, a line code indicating the type of line data was received. A 31C Hex code identified the line as the start of frame line and the vertical synchronisation pulse was outputted for a duration of 1 system clock cycle. On receipt of a 2D8 Hex code, the sensor interface recognised the line as a visible data line and start sub sampling. Every 8<sup>th</sup> pixel on every 8<sup>th</sup> line was transmitted to the ping-pong as an active data pixel until the complete 80 by 60 pixel image had been transmitted. The selection of pixels was important as STMicroelectronics colour image sensors possess a Bayer colourisation filter mask, over the visible pixel array. Therefore, as the silicon varies in sensitivity to different frequencies of light, only pixels with the same colour filter should be used to create

the sub-sampled image [96]. Green filter pixels were selected for sub-sampling. If a greyscale sensor was used this would not affect the sub-sampling process. Figure 4.6 displays the Bayer colourisation pattern used.

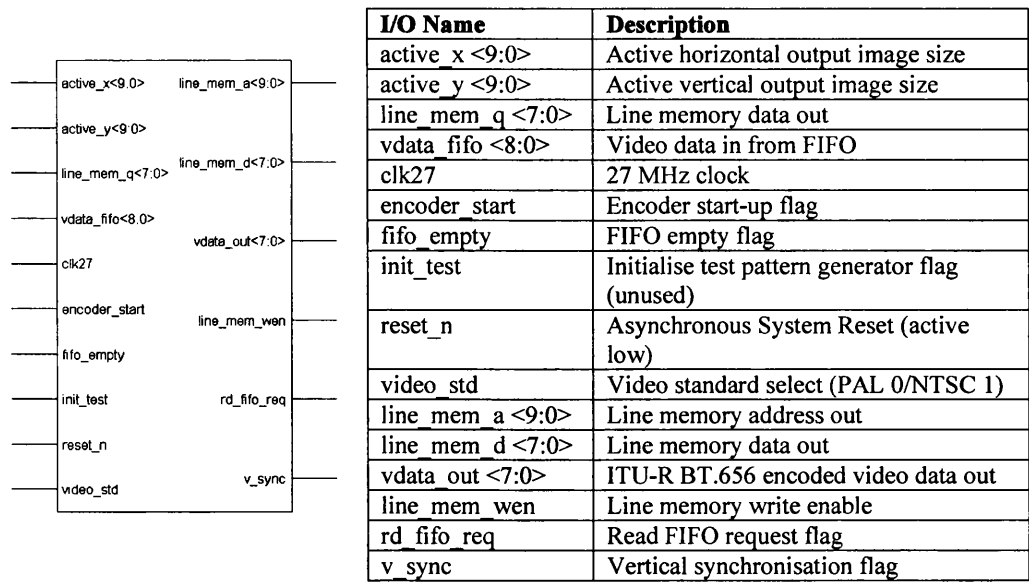


**Figure 4.6 Bayer colourisation pattern used in STMicroelectronics' colour image sensors**

The precise timing of the sensor interface was implemented using a single 15 state Mealy finite state machine (FSM) running at full system clock frequency.

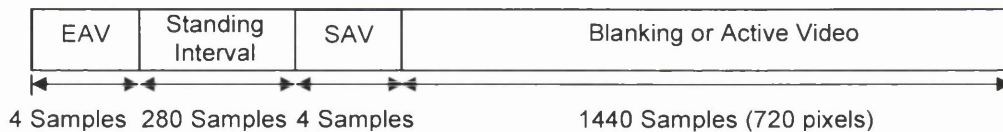
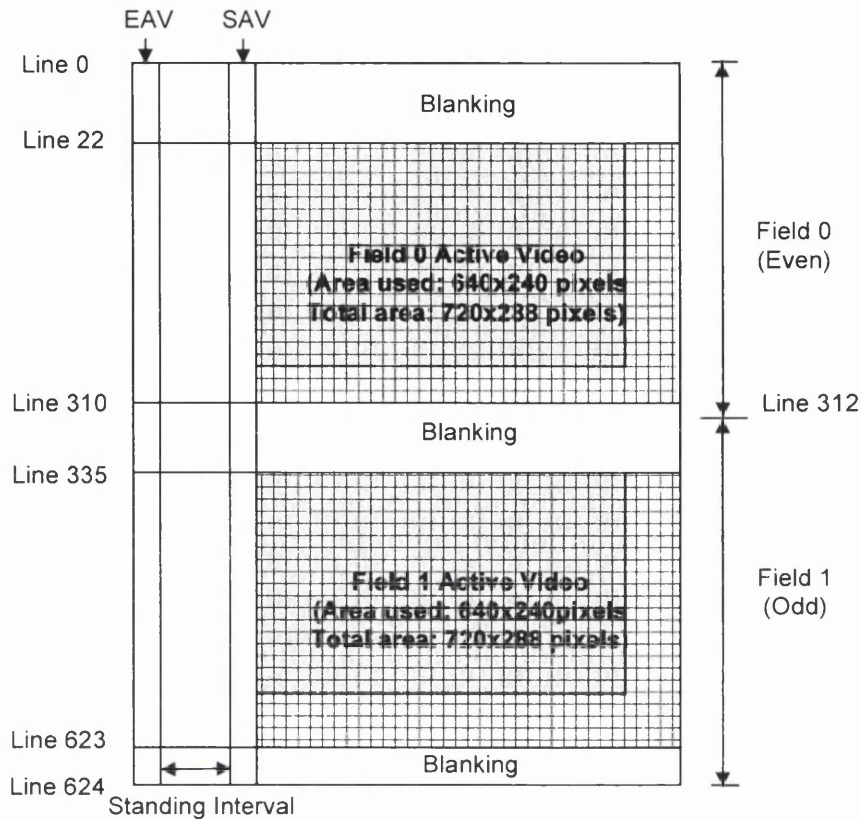
#### **4.3.5 Video Generator**

Video data to be outputted to the video encoder IC, was required to be organised into an ITU-R BT.656 compliant 8-bit 4:2:2 YCbCr video stream with embedded codes for either PAL (625 line) or NTSC (525 line) systems. The video generator module performed this task. An image stored in one of the two image banks was read by the ping-pong unit in a raster fashion into a 9-bit wide, 128 address deep asynchronous FIFO. The 9<sup>th</sup> bit was used as a flag to indicate the first pixel in an image. Keeping the size of the FIFO limited to 128 addresses, ensured that only one of the FPGA ESB memory resources was used. The video generator only read data from the FIFO when it was not empty and the encoder start flag was set to high. Table 4.7 shows the video generators I/O and its connection to the asynchronous FIFO and an associated line memory.



**Table 4.7 I/O interface for the Video generator**

Video data from the FIFO was in the form of an 80x60 pixel image which was interpolated with extra pixels to scale the image to a size dictated by inputs active\_x and active\_y, set as 640x480 by default. This was achieved by writing each 80 pixel line into a line memory at the same time it is read out. The line was then read back from the line memory a further three times. Each time a pixel was read from the FIFO or from the line memory, it was repeated a further seven times. This created a scaled image of 640x240 pixels. Unlike the sensors image format, the video generator had to produce a 25fps interlaced image. Each image frame was made up of two fields. These consisted of half the vertical video data and had to be outputted at 50 fields per second to reconstruct a 25 frames per second video stream. Hence, for every sensor image obtained, two ‘half images’ had to be outputted by the video generator in the same period of time. The combination of two 640x240 pixel images recreated the full-sized 640x480 image. The active video data in each line was 720 pixels, regardless of whether a PAL or NTSC video mode was selected. The video modes had a vertical resolution of 576 lines, for PAL, and 480 lines, for NTSC [97]. Eighty padding pixels (value 10 Hex) were added at the end of each line of the 640x480 image and also, in the case of PAL, 48 pixel lines after the 240<sup>th</sup> line. Figure 4.7 displays the interlaced vertical and horizontal timing requirements for a PAL ITU-R BT.656 digital output from the video generator.



**Figure 4.7 ITU-R BT.656 PAL video timing requirements for a 27 MHz clock**

The timing of the system was maintained by a Mealy FSM. As with the sensor data, embedded codes were required in the video data to indicate the type of line. The ITU-R BT.656 format, dictated that for an 8-bit digital video, the valid range of values a luminance or greyscale pixel should be 16 to 235 and for a chrominance (Cb or Cr) pixel, 16 to 240 [97]. Values that fell out with these values were to be clipped to the maximum or minimum value allowed. Blanking luminance pixels were assigned a value of 16 whereas the unused chrominance values, Cb and Cr, were permanently assigned a default value of 128. This is due to the fact that the system only operated in a greyscale mode. Table 4.8 shows the construction at the start and end of the video codes.

	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1 <sup>st</sup> Byte	1	1	1	1	1	1	1	1
2 <sup>nd</sup> Byte	0	0	0	0	0	0	0	0
3 <sup>rd</sup> Byte	0	0	0	0	0	0	0	0
4 <sup>th</sup> Byte	1	F	V	H	P3	P2	P1	P0

**Table 4.8 SAV and EAV code sequence**

As seen in the table above, F is the field number 0 or 1 and V is the value 1 only during vertical blanking lines. To recognise if the code sequence is a SAV or EAV, H is set to 0 for SAV and 1 for EAV. P0 to P3 are protection bits which can be used by the receiver of the video data to detect 1-bit and 2-bit errors and correct 1-bit errors. The equations for P0 to P3 are show in equation 4.1 below.

$$P0 = F \oplus V \oplus H$$

$$P1 = F \oplus V$$

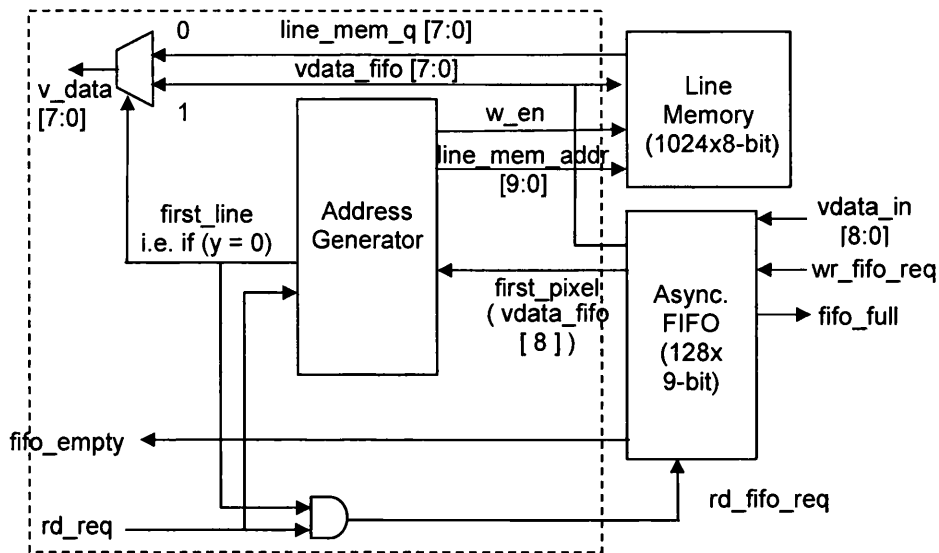
$$P2 = F \oplus H$$

$$P3 = V \oplus H$$

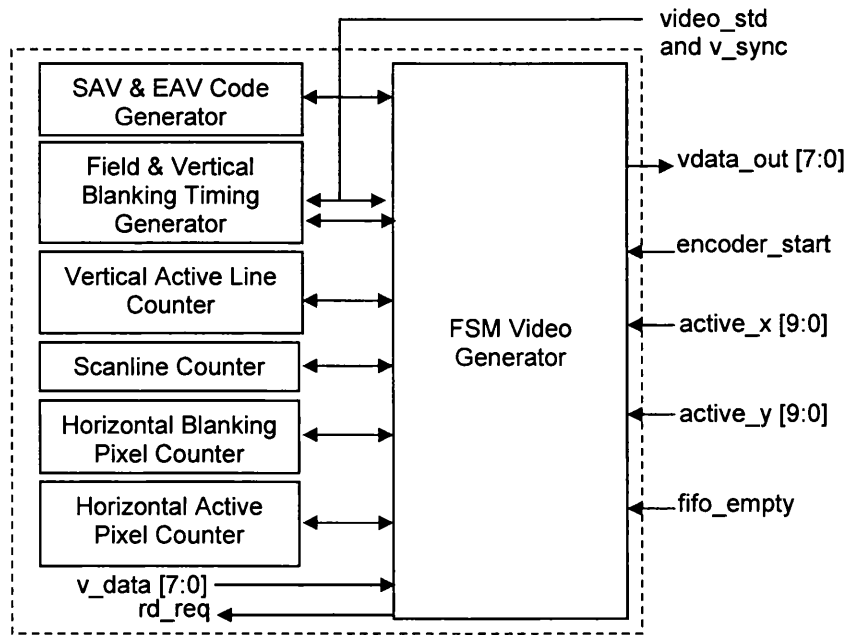
where  $\oplus$  is Exclusive – OR

**Equation 4.1 The calculation of protection bits P0 to P3 [98]**

The 4:2:2 nomenclature refers to the ratio of luminance, Y values, to chrominance, Cb and Cr values. For every four luminance values transmitted, two chrominance values were transmitted. Following an SAV code, the order of the 27MHz samples making up a pixel for a 4:2:2 is Cb-Y-Cr-Y. This sequence must be repeated until the end of the active line. The implementation of the video generator was in two parts. The first part was that of the FIFO and line memory control to re-produce a 640x480 image from an 80x60 pixel image. The second part is the generation of the ITU-R BT.656 video stream with the correct timings by a Mealy FSM. The control mechanism and top-level diagram for the FSM structure, without the system clock and reset, are shown respectively in figures 4.8 and 4.9.



**Figure 4.8 Control mechanism for Video generator's line memory and FIFO**



**Figure 4.9 Top-level block diagram of Video generators FSM**

### 4.3.6 Ping-pong Unit

The purpose of the ping-pong unit was to read an image frame from the sensor interface and write it into an image bank whilst simultaneously reading an image from the other image bank and writing it into the asynchronous FIFO attached to the video generator. The ping-pong unit had to ensure that both the video generator and sensor interface were synchronised. Once a frame had been written to one memory



and a frame read twice from the other memory, the ping-pong unit swapped memory interfaces between the read side and the write side. This enabled the system to maintain a default frame rate of 25fps while only using single-port memories. Table 4.9 shows the ping-pong unit's I/O.

I/O Name	Description
mem_q_1 <7:0>	Memory bank 1 data output
mem_q_2 <7:0>	Memory bank 2 data output
ping_pong_ctrl <1:0>	Indicates accessibility of both memory banks
roi_x <9:0>	Horizontal size of image to read and write
roi_y <9:0>	Vertical size of image to read and write
clk	System clock
encoder_bank_sel	Selects intra-bank offset address for reading
env_active	Active data signal from sensor
fifo_full	Video generator FIFO full flag
pingpong_dsp_ctrl	Freeze frame select (active high)
reset_n	Asynchronous System Reset (active low)
sensor_bank_sel	Selects intra-bank offset address for reading
v_sync	Vertical synchronisation signal from sensor
image_data_out <8:0>	Video data out to video generator
mem_a_1 <13:0>	Memory bank 1 address select
mem_a_2 <13:0>	Memory bank 1 address select
end_of_frame	End of frame flag
mem_w_en_1	Memory bank 1 write enable
mem_w_en_2	Memory bank 2 write enable
odd_even	Indicates which bank to write to (high: bank 1 and low: bank 2)
pingpong_rd_bank_ctrl	Flag to indicate a read is taking place
wr_fifo_req	Write to FIFO request

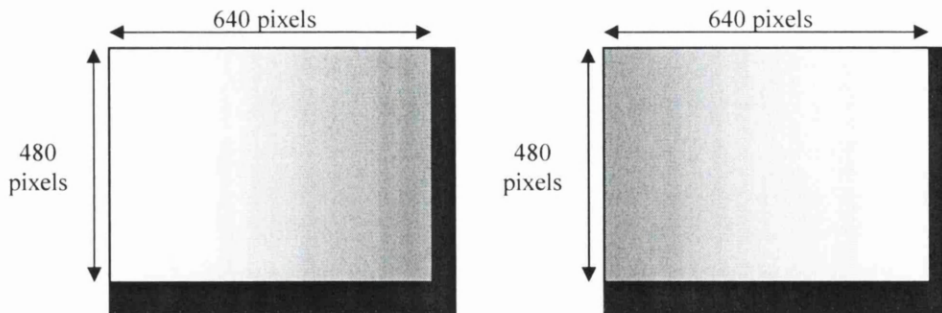
**Table 4.9 I/O interface for the Ping-pong unit**

The ping-pong unit consisted of four main sub-units; the image bank controller, the read unit, the write unit and an odd/even signal generator. The odd/even signal generator unit controlled all of the other sub-units. Its function was to provide a signal to toggle read/write functions between the two image banks when the rising edge of the v\_sync pulse was detected. The odd/even signal could be overridden by an active high pingpong\_dsp\_ctrl input, which would set the system to freeze frame mode, i.e. continuously read from one image bank without ever updating the image. The ping\_pong\_ctrl input could also override the ping-pong unit by forcing reads

from a specified bank or prevent access to all banks. For example, value 1 would set reads from bank 1 and conversely, value 2 would set reads from bank 2. Value 3 would block reads from all banks and force the video generator to output a blank screen. The image bank controller used the odd/even signal to toggle the signals from the read unit and the write unit between the two external memory interfaces. The design of the image bank controller was such that it allowed the images to be locked into either being read or written into the top or bottom half of each image store. Setting `sensor_bank_sel` bit-line high from the register bank forced an image to be written into the top half of a memory store and thus preserve any data in the bottom half.

The write unit performed a single image frame write, during cycles when `env_active` was high and `pingpong_dsp_ctrl` was low. The write mechanism generated the necessary addresses for all of the image data writes and reset the addressing generator when a `v_sync` signal was detected. Unlike the write unit, the read unit performed two reads of an image frame to maintain the data rate and achieve 50 fields per second. The read data was written to the video generator's FIFO, at system clock speed, whenever it was not full. This had to be performed at a frequency not lower than 13.5 MHz so as to prevent the video generator from being starved of data and blank pixels being incorporated into the active video data stream. The FIFO provided the opportunity to complete a pre-fetch of 128 bytes of data at the start of every image frame read, whilst the video generator was outputting blanking lines. As with the write unit, addressing was generated and finally reset by the `v_sync` pulse from the sensor interface.

A test generator was designed for the ping-pong unit to provide a test pattern in place of the sensor data. Setting either bit of `test_sel[1:0]` high in the register bank would activate the test generator. Once activated the test generator 'highjacked' the interface between the sensor and the ping-pong unit. The test pattern outputted to the ping-pong unit was that of a greyscale ramp. This was selected as the STV0119a Video Encoder IC's test mode was a set of test bars and it was deemed sensible to be able to distinguish between the two. The test patterns visible on a PAL monitor are shown in figure 4.10.



**Figure 4.10 Test patterns visible for test\_sel=1 (left) and test\_sel=2 (right)**

### 4.3.7 System Control Unit

As a design decision was made to not incorporate a microprocessor into the FPGA, a system control unit was provided as a mechanism for controlling the different sub-systems within the FPGA architecture. It is unlike modern control units found in microprocessors, microcontrollers and digital signal processors for several reasons. These are as follows:

1. Most instructions found in these other architectures are typically 12-bit to 64-bit in length. The system controller uses separate 8-bit wide instructions and literal values, as a result of the system-wide decision to only use byte-wide memories.
2. Most instructions configure and execute co-processor operations rather than performing typical ALU-type operations.
3. Minimal or no processing is performed on large sets of data by the system controller.
4. The instruction store (I-store) can only be written to, via the SDRAM decoder using an external device, such as the co-processor.

Table 4.10 illustrates the input and output interface of the system control unit. Figure 4.11 shows how these interface to the instruction store, FIFO and general-purpose memory interface.

I/O Name	Description
dsp_ctrl_fifo_wr_used <3:0>	Pre-fetch FIFO used words
instruction <7:0>	Instruction data path
reg_q <7:0>	System memory data in
clk	System clock
config_complete_n	I-store and system configuration complete signal
ctrl_rst	Control unit reset (synchronous)
dsp_ctrl_fifo_empty	Pre-fetch FIFO empty
end_of_frame	End of frame flag
ip_busy	DSP IP block active flag
reset_n	Asynchronous System Reset (active low)
system_start	Control unit activate signal
pc <7:0>	Program counter and I-store address
reg_a <15:0>	System memory address
reg_d <7:0>	Data out to System memory
dsp_ctrl_fifo_rd	Pre-fetch FIFO read request
dsp_ctrl_fifo_reset	Pre-fetch FIFO reset
dsp_ctrl_fifo_wr	Pre-fetch FIFO write request
reg_w_en	System memory write strobe

Table 4.10 I/O interface for the system control unit

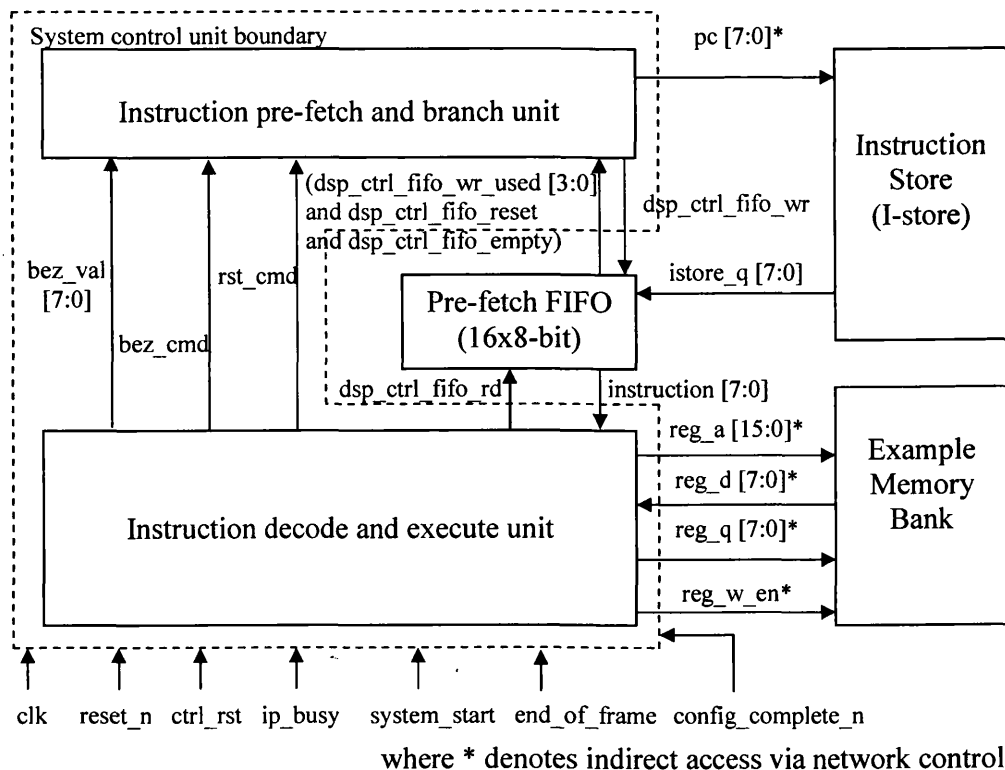


Figure 4.11 Top-level interfaces of system control unit

The system controller supported a total of 35 different operations. Twenty six of these operations internally controlled the system control unit, performed loads or moves to and from memory. The remaining nine operations controlled other external sub-systems or image processing co-processing blocks via the register bank. Given the instruction opcodes currently allocated, up to a further 191 operations could be assigned opcodes. The 35 operations are listed below in table 4.11.

<b>Instruction Name</b>	<b>Opcod</b>	<b>Description</b>
NOP	0x00	No operation
WAIT_EOF	0x01	Wait for end of frame flag to go active
WAIT_BUSY	0x02	Wait for DSP IP block busy flag to go inactive
LOOP	0x05	Unconditional loop to I-store start address
MOVE_MEMMEM	0x0F	Copy memory byte to new address
MOVE_MEMREG_GPR0	0x10	Copy memory byte to internal general-purpose register 0
MOVE_MEMREG_GPR1	0x11	Copy memory byte to internal general-purpose register 1
MOVE_REGMEM_GPR0	0x12	Copy memory byte from general-purpose register 0 to new memory address
MOVE_REGMEM_GPR1	0x13	Copy memory byte from general-purpose register 1 to new memory address
LOAD_MEM	0x14	Load memory address with literal data byte
ADD_REG_GPR0	0x15	Add literal value to general-purpose register 0
ADD_REG_GPR1	0x16	Add literal value to general-purpose register 1
ADD_REG_ADDR_RD	0x17	Add literal value to read address register
ADD_REG_ADDR_WR	0x18	Add literal value to write address register
SUB_REG_GPR0	0x19	Subtract literal value from general-purpose register 0
SUB_REG_GPR1	0x1A	Subtract literal value from general-purpose register 1
SUB_REG_ADDR_RD	0x1B	Subtract literal value from read address register
SUB_REG_ADDR_WR	0x1C	Subtract literal value from write address register
LOAD_REG_GPR0	0x1D	Load general-purpose register 0 with literal value
LOAD_REG_GPR1	0x1E	Load general-purpose register 1 with literal value
LOAD_REG_ADDR_RD_LO	0x1F	Load literal value to lower byte of read address register
LOAD_REG_ADDR_RD_HI	0x20	Load literal value to upper byte of read address register
LOAD_REG_ADDR_WR_LO	0x21	Load literal value to lower byte of write address register
LOAD_REG_ADDR_WR_HI	0x22	Load literal value to upper byte of write address register
BEZ	0x30 to 0x3F	Branch if equal to zero. First 4 bits of instruction indicated which general-purpose register to evaluate
BNEZ	0x40 to 0x4F	Branch if not equal to zero. First 4 bits of instruction indicated which general-purpose register to evaluate
STOP_PINGPONG	0x03	Disable Ping-pong unit
START_PINGPONG	0x04	Enable Ping-pong unit
THRESHOLD	0x06	Perform threshold operation on ROI using a value stored in the register bank
RECTANGLE	0x07	Draw rectangle
GETCOORDS	0x08	Get max and min coordinates cover of all active pixels
ABSDIFF	0x09	Perform absolute difference
COPY	0x0A	Copy image or ROI to new memory address
GETOBS	0x0B	Get all active objects parameters and build object database in Scratch pad memory
FILTER3X3	0x0C	Perform 3x3 neighbourhood filter using weights stored in the register bank

**Table 4.11 Instruction list supported by prototyping system**

By default, the system controller was inactive at power-on and the instruction store was empty. The external co-processor was given full access to the system to allow the automatic programming of the instruction store. Once the instruction store had been programmed, the `config_complete_n` signal went active low. This signal was followed by the `system_start` signal which initiated the instruction pre-fetch mechanism, instruction decode and execute unit.

The instruction pre-fetch and branch unit operated in a relatively simple manner. This ensured the pre-fetch FIFO was always full by writing instructions to it from the instruction store by incrementing the program counter. This process continued until either a reset, loop or branch signal was received. The complete system control unit and FIFO, could be externally reset at any time by setting the reset register, which in turn, set the `ctrl_rst` signal active high. In the case of a loop command, the internal `rst_cmd` signal from the decode and execute unit pulsed high, causing the program counter to be reset to 0 and the FIFO to be flushed of data. Branch operations set the `bez_cmd` high and asserted the 8-bit value to be decremented from the program counter on `bez_val`. This operation also flushed the FIFO and started filling the FIFO with data from the instruction store, using the new program counter value as an address.

The instruction decode and execute unit was also reset by the `rst_cmd` and was also stalled by an empty FIFO. The instructions decoded by the decode unit could be divided into two groups; those that required a literal value to execute an operation and those that did not. Instructions without a literal value usually changed the system control units internal state, activated a DSP co-processor block or triggered a MOVE operation. External memory operations, such as a MOVE, had to use a read or write address register. These two registers were 16-bit wide and provided access to the full system memory mapped address range. The use of these registers was a result of using not allowing literal values to be used as addresses. This was deemed acceptable as adding a target address to the instruction, would either limit the possible address range to a small number of addresses, or add the burden of requiring addresses to be stored with the instructions. It was also anticipated that few external memory moves would be used in a typical application developed with the prototyping system and the time required for a move was still several magnitudes

smaller than the time intensive image processing operations. The two 16-bit registers could be incremented, decremented or loaded using operation literal values. Loading the registers could be achieved by loading either the upper or lower byte per clock cycle. Again, this was not seen as a hindrance, as it was expected that external memory accesses would take place in the same data address localities, for example within the scratch pad memory or 8-bit addressable register bank.

The group of instructions requiring operands, all used the same process for decoding and executing operations. Once an instruction had been decoded, the literal value was obtained on the next clock cycle, as it was stored in the subsequent address in the instruction store. The instruction was then executed. This allowed for a reduction in the size of the mealy FSM used to 39 states. In the case of branch instructions, the first four bits of the opcode were used to determine which of the internal general purpose registers were to be evaluated. Although the branch mechanism decremented the program counter, it was possible to branch forward due to the wrap-a-around characteristics of registers. For example, branching from program decimal address 45 to address 177, would require the literal to be the value 123. The implementation of a 256 byte program memory made this branching possible.

Two of the most important instructions were WAIT\_EOF and WAIT\_BUSY as both would be required in all but a few applications developed. WAIT\_EOF could be executed to wait for a new image to be written from the sensor into memory, before a set of image processing operations could be executed. Typically, each image processing instruction would be followed by a WAIT\_BUSY instruction, preventing the system controller from executing any more instructions, until the co-processor had finished. In some cases however, the time during image processing operations would be used for updating other memories or registers.

#### **4.3.8 Network Control**

Following the specification that point-to-point buses be used, rather than a traditional shared bus, a method was required to ensure that the buses could be controlled and that a mutual exclusion mechanism would be in place to prevent conflicts to shared

resources, such as memory banks. The end result of this requirement was the implementation of a network control unit integrated into the bus network. This control unit provided low latency control of the point-to-point data bus structures within the complete system architecture. These point-to-point data bus structures were interconnected between a sub-system or DSP block and a memory, as most inter-sub-system communication occurred via the register bank. The latency in the bus control and associated network was kept to a minimum by not using clocked registers and using multiplexers and priority select structures. Given the large number of signals involved and that the network control unit was integrated into the bus network, an I/O diagram has not been provided.

Five different memory interfaces existed within the FPGA system, with only one sub-system or IP block granted access during any clock cycle. As discussed previously, the instruction store was only accessible by the SDRAM decoder when the `config_complete_n` signal was high and the system control unit signal was low. The memory interface access controls for the register bank, image banks and scratch pad memory, worked on the same four level priority system, with priority one being the highest level and resulting in granted access to the memory. These levels are as follows;

1. The SDRAM controller always has the greatest access priority and enables access to all memory banks irrespective of the system's status. The SDRAM controller is responsible for controlling the programming of the system.
2. DSP blocks have the second level of priority. As many DSP blocks do not require access to the register bank, they typically do not take access control of the register bank or release full control to the system controller after configuring themselves in the first few clock cycles of their initialisation. The network control unit ensures that DSP block has default access to the most recently obtained image, unless instructed otherwise.
3. The System controller generally has access to the register bank and data memory by default and typically does not require access to the image banks.
4. The ping-pong unit has the lowest level of priority but it only required access to the image banks. This allowed it to generally co-exist with the system controller as their access to memory banks rarely conflict.



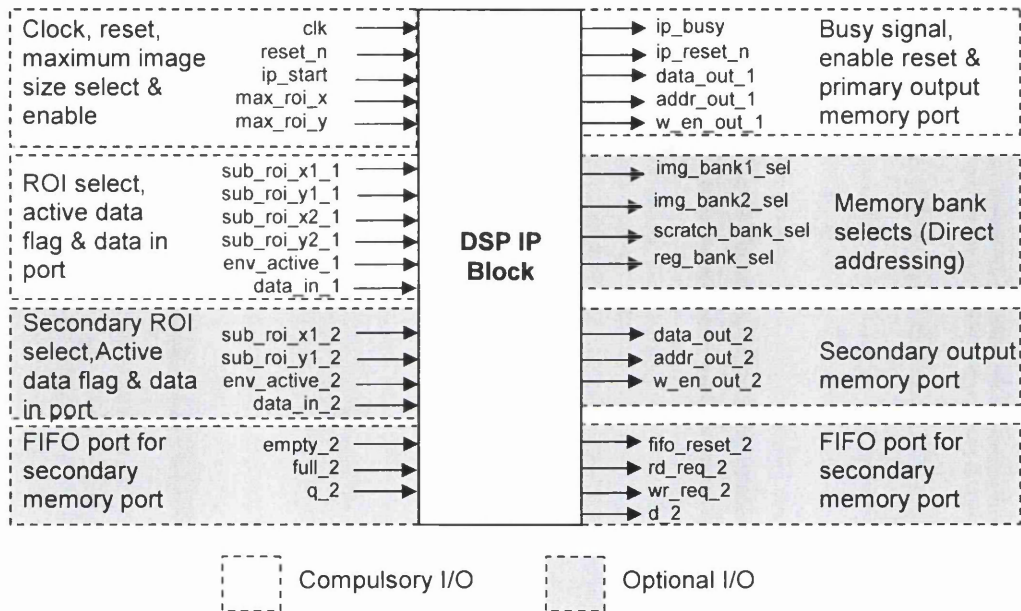
The ping-pong unit was given a lower priority than the DSP block as in some circumstances only a partial image may need to be written from the image sensor for processing to be performed. The WAIT\_BUSY function could also be used to ensure that the DSP block only took control for processing in a memory bank at the end of a complete image write to memory. Also, as the video generator included a FIFO and line memory, the frequency of reads from a memory bank was low. Hence it was possible for a DSP block to temporally override access to the image bank currently being read by the ping-pong unit. This would normally have no noticeable effect on the output video but allow a DSP block to execute concurrently with the ping-pong read mechanism. In instances where frequent access was required by a DSP block, the pingpong\_rd\_bank\_ctrl flag from the ping-pong block, could be monitored by the DSP block, to prevent it accessing a memory bank in the same clock cycle as the ping-pong read unit.

The network control unit offered two methods for address translation mapping to the correct memory. The first method was the direct method, which involved a memory bank select line being activated and a memory address passed unaltered to the selected memory. The second method was the indirect method. The indirect method took a full 16-bit address and translated the address into a memory select signal and an offset memory address. No significant difference was observed between the two schemes except that the direct method reduced the need for slightly more complicated addressing within the network control unit. This reduction in complexity slightly improved the ability of the design synthesis process to meet timing requirements for greater system clock frequencies. The in-direct method was more appealing however given that the DSP blocks interface was simpler.

#### **4.4 DSP IP Block Library**

The DSP IP block library consisted of seven co-processor IP blocks that performed image processing operations over a whole image or a region of interest (ROI). It was expected that almost all image processing within the prototyping system would be performed by these co-processors. Each co-processor generated memory addressing

schemes to access the register bank and image banks. As STMicroelectronics had requested that the DSP IP blocks should be reusable, the IP block's interfaces needed to be standardised, yet remain flexible. As mentioned in section 4.3, the interface of each co-processor to the bus network and memories could be either direct or indirect. These co-processors could also have either a single memory interface port or a dual port depending on the designer's choice and requirements when designing a DSP block. A diagram encompassing the standard I/O interfaces is shown in figure 4.12.



**Figure 4.12 I/O interface for DSP IP co-processor blocks**

The design decision to only use two image banks, with one bank being written into by the sensor interface and the another concurrently reading out to the video generator, had two effects on the design and control of the DSP IP blocks. These were:

1. Processing of the current image was only easily possible at the end of each sensor image frame, when the sensor had stopped writing into the memory bank. This meant that the addressing and control of the DSP IP blocks had to be optimized for their execution at the end of each sensor frame.
2. Image processing operations using two image banks for processing had to be designed to avoid memory read conflicts with the ping-pong unit, when the ping-pong unit was supplying pixel data to the video generator.

The ip\_start signal originating from the register bank, activated the DSP block. On the next clock cycle an active low acknowledgement signal was sent directly back to the register bank, to reset the ip\_start signal. During the same clock cycle, the ip\_busy flag was activated and remained so, until the system was reset or the DSP operation was completed. The parameters of the maximum image size, were given by max\_roi\_x and max\_roi\_y, and were sampled in addition to the region of interest given by the sub\_roi inputs. At this point in the co-processor operation's execution further configuration data could be read from the register bank or from one of the memory banks.

When env\_active\_1 was high this was indicative that the ping-pong interface was writing data to an image bank. The optional env\_active\_2 input indicated if the ping-pong unit was reading data from the other image bank. These signals helped the DSP block prevent memory access conflicts when requiring the reading or writing of data to and from an image bank. The secondary memory port and associated FIFO were only used in those instances where two streams of data were required for concurrent processing, such as an absolute difference operation.

Three different types of addressing schemes were supported by the DSP block architectures. These were;

1. **Raster scheme** – Image data was read and written to and from a memory from the top line to the bottom line by reading left to right on each line. Images were written and read from the image banks in this way also. Typically, the read and write addresses only needed to be incremented by one for every new read or write.
2. **Structured scheme** – Image data was read and written in a set predictable pattern following a simple rule set. For example, drawing a rectangle on an image required the drawing of 2 vertical and 2 horizontal interconnected lines. As an image was stored in a raster format, drawing the vertical line required the address to be incremented by the width of the image to write the next pixel below the current pixel. The horizontal lines however only required the address to be incremented by one for each new pixel write.

3. **Random scheme** – Image data is read and written in a pattern which is not easily predicted or is greatly affected by the contents of the input image.

Most common image processing functions use predictable addressing pattern, such as the raster and structured types listed above [99]. These addresses were generated using mealy FSM. The FSMs were coded to enable their re-use as a template for new algorithms, as a part of the requirement for the architecture to allow new DSP IP blocks to be developed in the future. Seven DSP IP blocks were developed to demonstrate the aforementioned different types of addressing and the use of the secondary memory port (dual port). These blocks are listed in table 4.12.

DSP Block Name	Dual Port	Operation Type	Addressing Scheme	Read / Write
THRESHOLD	No	Point	Raster	Yes / Yes
COPY	No	Point	Raster	Yes / Yes
GETCOORDS	No	Point	Raster	Yes / No
RECTANGLE	No	Point	Structured	No / Yes
ABSDIFF	Yes	Point	Raster	Yes / Yes
GETOBS	No	Neighbourhood	Random	Yes / Yes
FILTER3X3	No	Neighbourhood	Raster	Yes / Yes

**Table 4.12 Supported operations by DSP block library**

The first five DSP blocks in table 4.12 are simple operations and can be defined as;

1. A threshold DSP block, reads pixels and compares them to a value retrieved from the register bank. If the pixel value is greater to this threshold value, the pixel is written back as a decimal value 255 otherwise it is written back as a value 0. This can be explained mathematically for image  $u$  and threshold value  $t$  by the following equation;

$$u = \begin{cases} 1 & \text{if } u(x, y) > t \\ 0 & \text{if } u(x, y) \leq t \end{cases}$$

**Equation 4.2 Threshold equation**

2. A copy DSP block, performs a translation of an image or ROI from one location to another address location stored in the register bank. This IP block can be defined mathematically using displacements  $x_d$  and  $y_d$ , as;

$$u(x_{new}, y_{new}) = u((x + x_d), (y + y_d))$$

**Equation 4.3 Copy equation**

3. A getcoords DSP block, reads an image and stores the minimum and maximum x and y values in four pre-defined locations, R<sub>1</sub> to R<sub>4</sub>, in the register bank.

$$\begin{aligned} R_1 &= u(\min(x)) \text{ where } u(x, y) = 255 \\ R_2 &= u(\min(y)) \text{ where } u(x, y) = 255 \\ R_3 &= u(\max(x)) \text{ where } u(x, y) = 255 \\ R_4 &= u(\max(y)) \text{ where } u(x, y) = 255 \end{aligned}$$

**Equation 4.4 Getcoords Equations**

4. The rectangle operation, draws from an address location, a rectangle with the dimension specified by the sub\_roi inputs. Pixels are written using the value stored in the register bank, which as default is the mid-value 128. The order of line drawing is horizontally from top left to top right, vertically to bottom right, horizontally to bottom left and back to top left.
5. The absdiff DSP block, uses two synchronised image data streams, each from a different image banks and calculates the magnitude of the difference of pixel values between the identical pixel locations within the two images, u and v. The equation for this operation is shown in equation 4.5.

$$\begin{aligned} u(x, y) &= u(x, y) - v(x, y) & \text{if } u(x, y) > v(x, y) \\ &v(x, y) - u(x, y) & \text{if } u(x, y) < v(x, y) \\ &0 & \text{if } u(x, y) = v(x, y) \end{aligned}$$

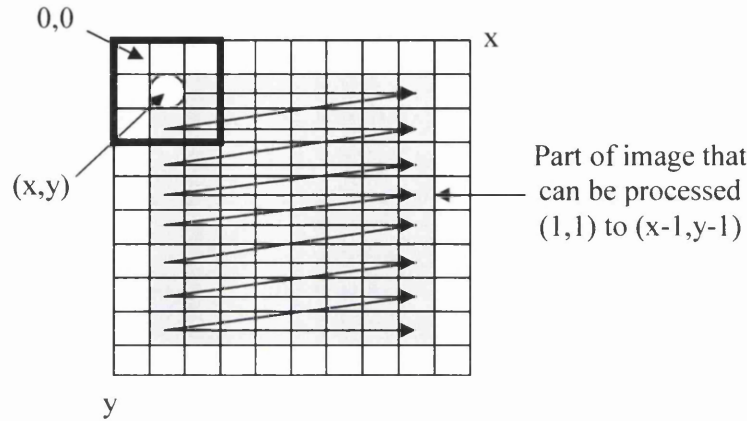
**Equation 4.5 Absdiff equation**

The last two of the image processing blocks, the getobjs operation and 3x3filter were more computationally complex than the other DSP blocks. The 3x3 filter is a spatial domain neighbourhood filter operating on the pixels that form an image. This is typically expressed as;

$$v(x, y) = T[u(x, y)]$$

**Equation 4.6 Equation for the spatial domain neighbourhood operator 3x3filter**

Where  $v(x,y)$  is the output image and  $T$  is the operator on image input image  $u$ , over some neighbourhood of  $(x,y)$ . Filter $3 \times 3$  uses a  $3 \times 3$  pixel mask over each pixel in an image and adds together all nine multiplications of the value of  $u(x,y)$  with the corresponding mask value (weights) to form pixel  $v(x,y)$ . As no data exists outside the image only the inner image of the dimensions 1 to  $x-1$  and 1 to  $y-1$  can be correctly processed. This can be show diagrammatically as in figure 4.13.



**Figure 4.13 3x3 neighbourhood filter operation on an image**

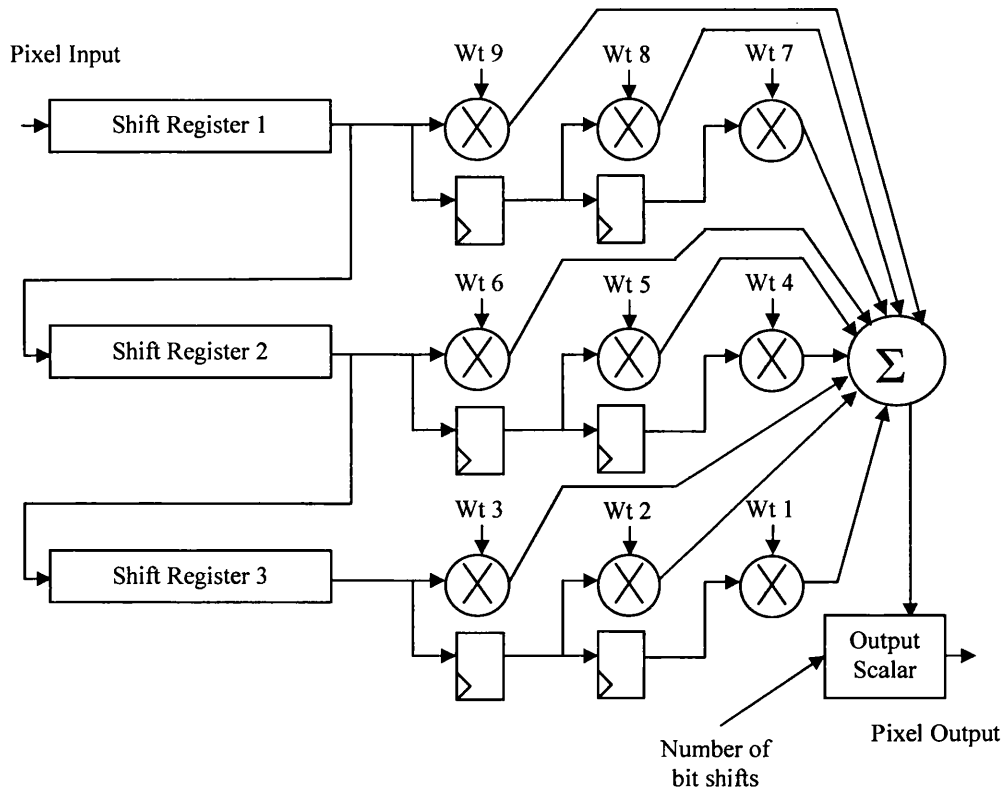
The development of a  $3 \times 3$  filter for the IP library was particularly important as it formed the basis of many image enhancements, such as smoothing (low pass filter), sharpening (high pass filter) and edge enhancement. An example of one of these is the Sobel vertical edge enhancement mask, shown in figure 4.14 with the commonly used Lenna 512x512 pixel image and resultant image processed using PC based software.



**Figure 4.14 Sobel vertical mask (left), input Lenna image (middle) and processed image (right)**  
[100]

As can be seen from the right image in figure 4.14, vertical lines from the original image have been enhanced to a far greater extent than the horizontal lines.

In the implementation of the filter DSP block, the nine weights for the mask were obtained from a pre-defined set of memory addresses in the register bank. Each weight was in a two's complement format using 4-bits and providing a range of -7 to +7. Image data was then read from an image bank to fill three shift registers, each with the capacity to hold one image line, i.e. 80 pixels. The output of each shift register was fed to a bank of three multipliers and the first shift register's output connected to the input of the second shift register. The output of the second shift register was also connected to the input of the third shift register, as shown in figure 4.15. From all three shift registers, data was read into the 3x3 array of 12-bit output multipliers and summed to form an output raster image data stream. Data continued to be read from the image bank into the input of the first shift register at the same rate as the output data, i.e. one pixel per clock cycle. This ensured that the process was not starved of data. As the system only used 8-bit positive integers, any negative output pixel values were clipped at 0. An 8-bit shift operation on the output also allowed downscaling of the output pixel values by factors of two.



**Figure 4.15 Representation of the 3x3 filter's datapath**

The getobjs operation is a useful DSP IP block as it provided the ability to segment different objects out of a thresholded image and store each object's specifications in a database within scratch pad memory. This operation forms the basis of many object recognition and object tracking algorithms. Typically segmentation operations perform multiple passes over an image. The initial pass labels active object pixels and subsequent passes perform connectivity checks between pixels and labels interconnect pixels with the same label. Once each interconnected pixel group has a single distinct label, it is recognised as an object. As a result of the design decision to implement an 8-bit data architecture, only 254 different labels could be used, when 0 was used as a non-active pixel and 255 an active pixel. Using a typical labelling algorithm could possibly result in an insufficient number of labels to complete the first pass. To prevent this outcome, a new more complex algorithm was devised and implemented which was not based on a previous reference algorithm.

The devised algorithm performed a single raster scan pass over an image. Upon finding an active pixel, it was labelled with the current object number along with any

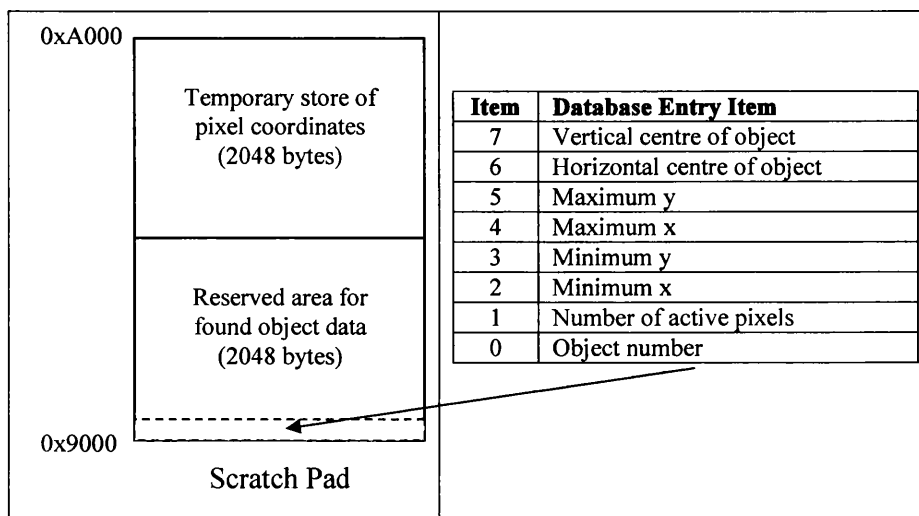


active pixels in the surrounding 9 pixels. The locations of the newly labelled surrounding pixels were stored in the upper 2048 bytes of the scratch pad memory. Instead of reading the image in a raster scan fashion the algorithm tracked around the inside of the object until no more active pixels were found. The tracking path's direction inside the object was determined by figure 4.16, with lower numbers representing the highest priority. This gave a preference to interconnected pixels on the vertical and horizontal plane.



**Figure 4.16 Preference of connected pixels in Getobjs algorithm**

The algorithm then used the stored locations of active pixels as starting points to find any remaining active pixels that had not been found on the first track around the inside of the object. Once all the store pixels had been read, the algorithm stored the objects information in the scratch pad in the format shown in 4.17 and returned to the original raster scan path over the image to find new active pixels. A more detailed algorithm description can be found in flow chart form in figure 4.18.



**Figure 4.17 Object database item format**

Unlike the previous operations, this DSP IP block required a detailed algorithm to create a suitable memory addressing scheme. Rather than implement this block in

Verilog, it was decided to first prototype the algorithm using the Matlab software package. The algorithm was implemented in a single Matlab code file using the standard set of commands with no additional libraries. This enabled the prototyping of the algorithm in a shorter space of time than using a hardware description language and electronics simulation.

As Matlab is a high-level language for algorithm development, data visualization and numerical computation it does not map its written code onto electronic components. Therefore, as with other programming languages such as C or C++, Matlab does not take signal timings or registers operations into account. Also, Matlab code is processed sequentially as there is no concept for parallelism as with languages such as Verilog or VHDL. This meant that when the Matlab code was converted into a Verilog IP block, parts of the algorithm were optimised to be performed in parallel. Delays as a result of register operations and accessing memory banks were scrutinised to prevent any unnecessary latency from being introduced. An interface wrapper was designed using the standard DSP IP block I/O interface and the necessary logic introduced to control timings for external I/O transactions. This process of conversion was done manually and formed a 50 state Mealy FSM.

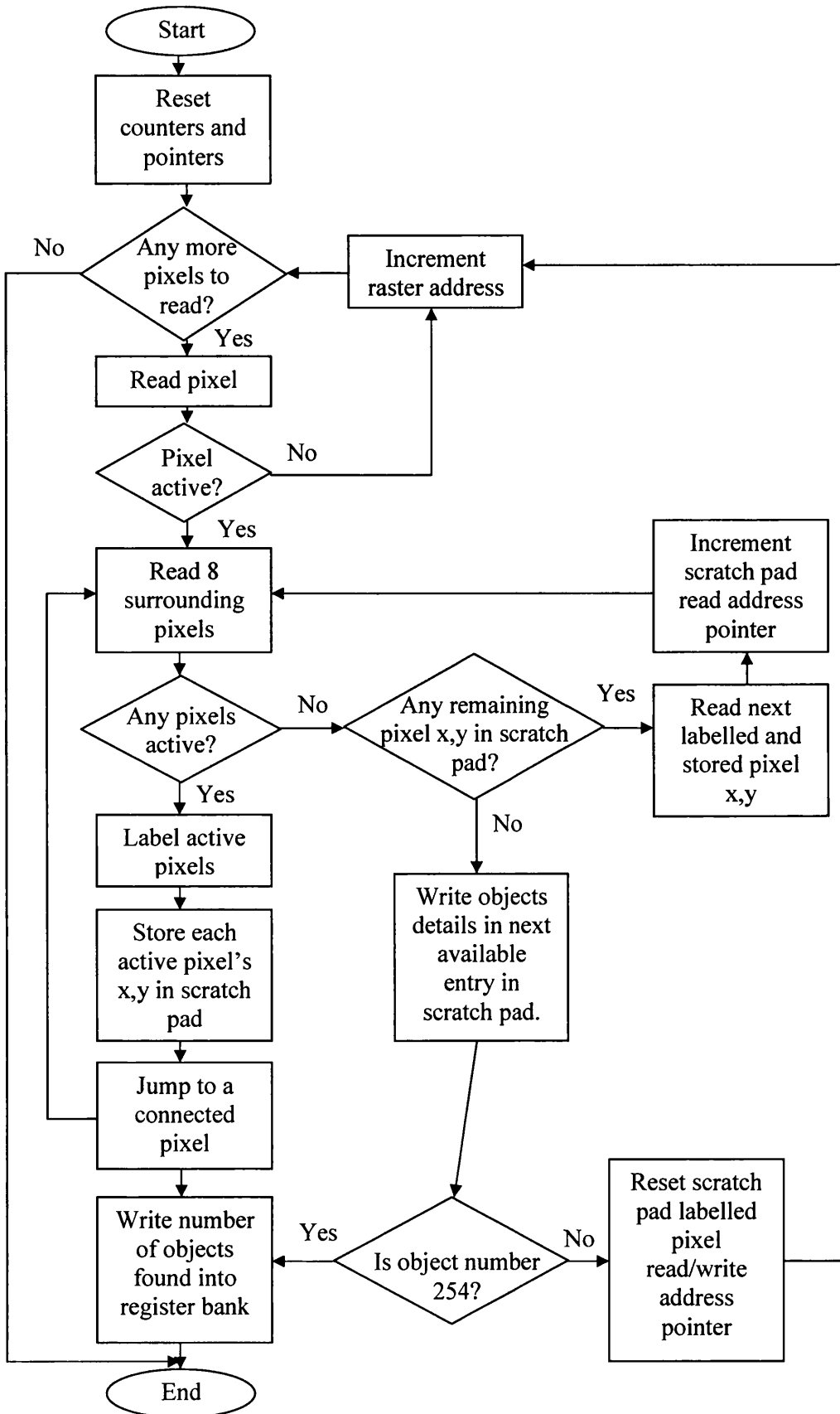
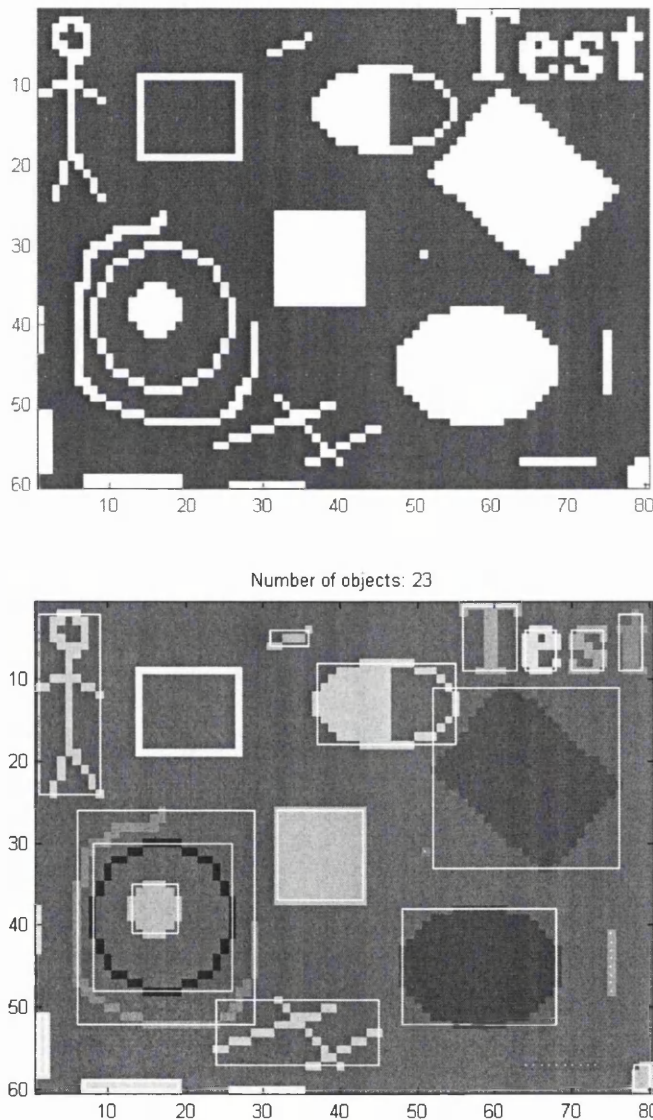


Figure 4.18 Getsobjs algorithm flow chart

Figure 4.19 show a binary 80x60 pixel test image and the associated output image from the prototyped object segmentation script `getobjjs`. The script was extended to draw a rectangle around each object using the stored values in the scratch pad memory and output the number of objects on screen. Please note: the greyscale palette of the output image has been changed post-processing, in order that the highlighted objects are clearly distinguishable. Also, when drawing a rectangle for an object that only consisted of a line of pixels, Matlab highlighted the object using dots rather than a continuous white line or rectangle.



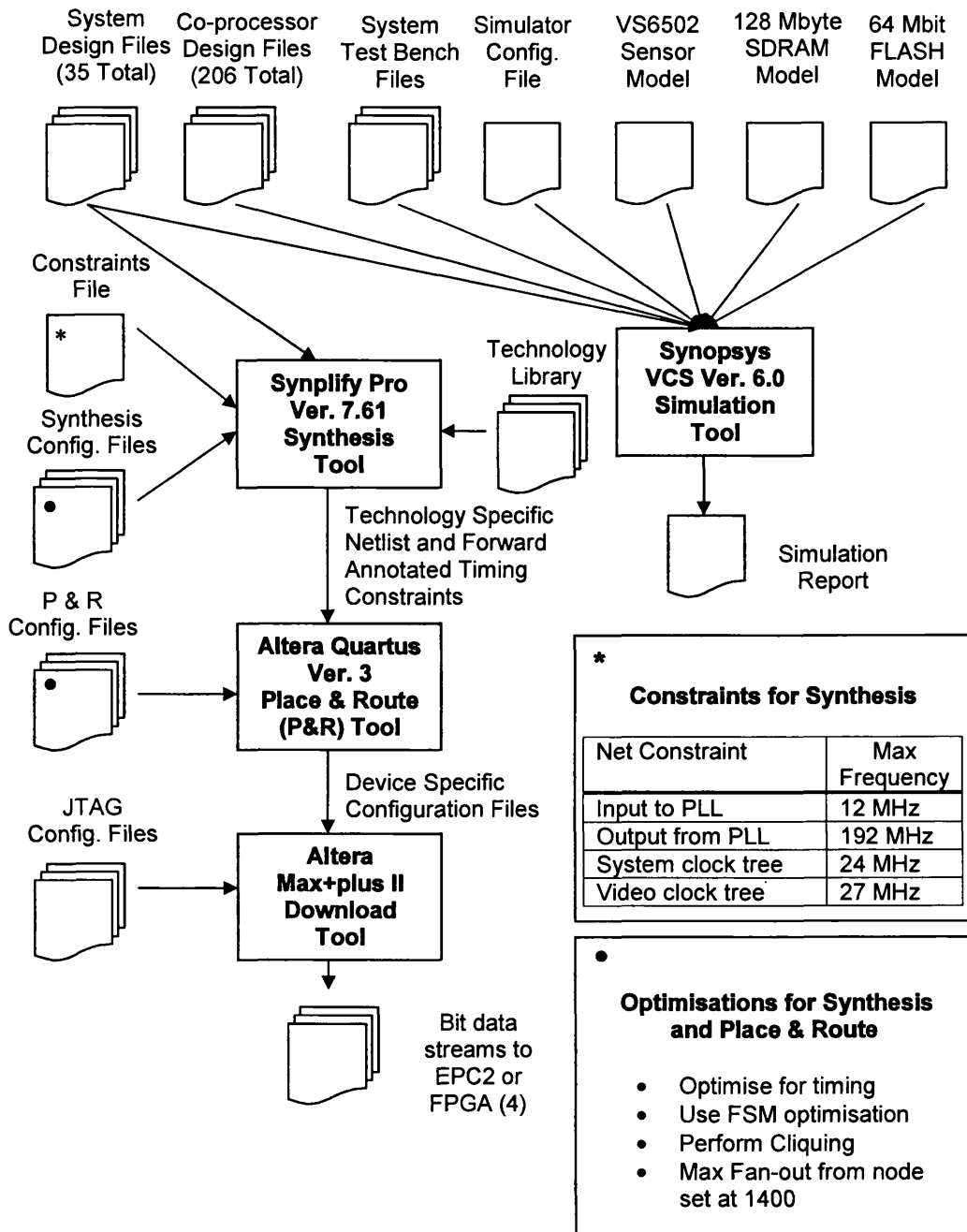
**Figure 4.19 Binary input image (top) and processed image using `Getobjjs` Matlab script (bottom)**

## **4.5 Simulation and Functional Verification**

Simulation and functional verification of the FPGA architecture was performed throughout several development stages. These stages included modification of the core system architecture for new DSP IP blocks (before and after integration) and for benchmarking new demonstration applications. Synopsys VCS version 6.0 was used as a simulation environment. Numerous Verilog simulation scripts and test benches were written to check the functionality of the various parts of the architecture.

Functional verification typically encompassed some of the same tests but was executed in real-time at full system clock frequency on the prototyping system.

Figures 4.20 shows the process flow used for simulation and functional verification and the constraints and optimisations.



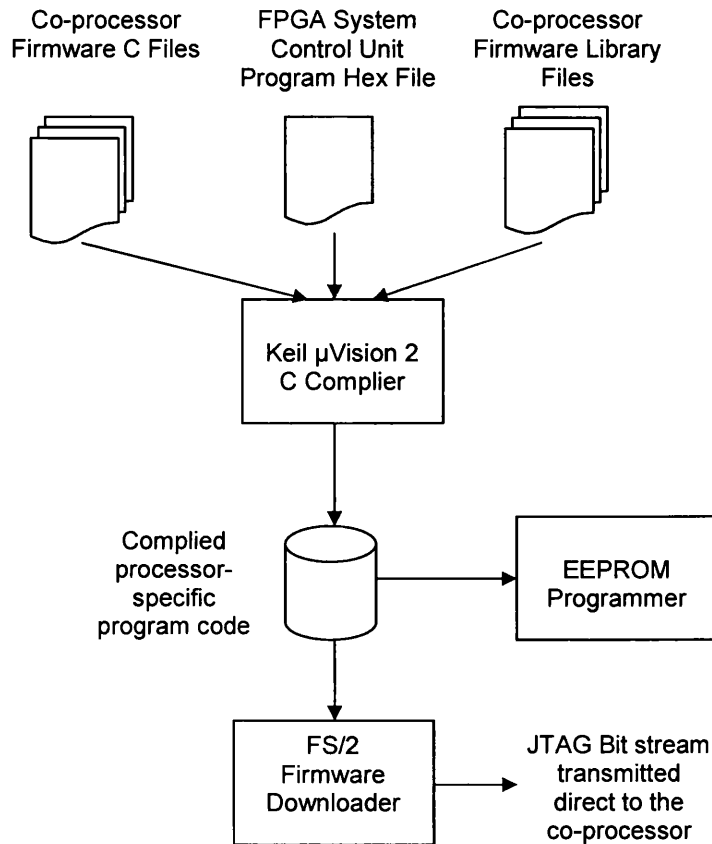
**Figure 4.20 Functional verification and simulation flow**

## 4.6 System Programming

Two programmable elements were present within the prototyping platform. These were the STV0674 sensor co-processor and the system control unit located within the FPGA architecture. Two methods could be used to control the FPGA architecture, namely programming the I-store and running the control unit or via function calls from the co-processor. Although the use of function calls to execute the different DSP operations may seem appealing, the latency involved with the function call mechanism is greater than using the I-store and system control unit. It is for this reason that it is recommended that users of the prototyping system program the I-store. Programming the I-store was typically performed by the co-processor at power-on by performing SDRAM writes across the IMPBUS to the FPGA. Another option was to fix the operation of the system control unit during the FPGA synthesis by replacing the I-store RAM Verilog module with a ROM module that had been configured with the desired program. Using both methods, the 8-bit addressable I-store was always programmed from the start address 0x00. If no program was detected by the control unit at start-up, the system defaulted to outputting unprocessed video to the video encoder.

To enable the co-processor to program the I-store, the co-processor had to be patched with the correct firmware, either by the EEPROM or by using an in-system programming and debugging device via the JTAG port on the daughterboard. This firmware also correctly configured the sensor and the video encoder via the I<sup>2</sup>C bus before downloading the program code to the I-store. The sensor was configured to run at 25fps in a VGA mode, with all auto-exposure and gain controls turned off. Without configuring the sensor, it would default to auto-exposure and gain mode. This would result in the image being continually altered, depending on the amount of illumination within the scene. The auto-exposure and gain mode would prevent inter-frame differencing functioning correctly. The video encoder was configured to a PAL mode, receiving data as a slave device and inverting DAC codes to compensate for an inverting video output stage on the daughterboard. Figure 4.21 shows the programming method used for the co-processor.

It must be noted that future users may wish to add new DSP IP blocks to the FPGA architecture to increase the systems versatility. This could be achieved by swapping-out one or more of the seven blocks. The remaining interfaces could be connected to the new blocks or by extending the network control interface, register bank, instruction set and system control unit, by replication of the current Verilog code.



**Figure 4.21 Programming method for the STV0674 co-processor**

## 4.7 FPGA Conversion to SoC

The conversion of a FPGA design to a SoC design is complex. In the case of the prototyping system, not only the FPGA based design needs to be integrated into a single chip but also the sensor co-processor, video encoder and sensor. It is expected that when the user of the prototyping system has a suitable design for integration, an in-house STMicroelectronics design team would perform the system-level integration to create a single chip. Given that STMicroelectronics owns the



intellectual property rights for most of the board-level components, the licensing issues may not be as complex when compared to agreeing licensing terms with a third party. It is likely that the integration of these components is eased by the design files residing in-house. Due to the commercially sensitive nature of STMicroelectronics design flows, no details can be provided.

## 4.8 Results

This section describes the results gained from analysis of the FPGA architecture. The results relating to the instruction decoding and execution are provided, followed by the number of cycles to complete the execution of each of the DSP IP blocks. The resources required for each part of the FPGA system will also be stated. The effect that processing time has on the system frame rate will be outlined.

The time taken to decode each instruction was one system clock cycle. Table 4.13 lists the number of clock cycle to execute each instruction. Instructions that require literal values to execute have been marked with an asterisk.

Instruction Name	Number of cycles	Instruction Name	Number of cycles
NOP	1	LOAD MEM*	2
WAIT_EOF	1	ADD REG GPR0*	2
WAIT_IPBUSY	1	ADD REG GPR1*	2
STOP_PINGPONG	1	ADD REG ADDR RD*	2
START_PINGPONG	1	ADD REG ADDR WR*	2
LOOP	4	SUB REG GPR0*	2
THRESHOLD	1	SUB REG GPR1*	2
FILTER 3X3	1	SUB REG ADDR RD*	2
RECTANGLE	1	SUB REG ADDR WR*	2
GETSCORDS	1	LOAD REG GPR0*	2
ABSDIFF	1	LOAD REG GPR1*	2
COPY	1	LOAD REG ADDR RD LO*	2
GETOBS	1	LOAD REG ADDR RD HI*	2
MOVE MEMMEM	3	LOAD REG ADDR WR LO*	2
MOVE MEMREG GPR0	3	LOAD REG ADDR WR HI*	2
MOVE MEMREG GPR1	3	BEZ*	5
MOVE REGMEM GPR0	1	BNEZ*	5
MOVE REGMEM GPR1	1		

**Table 4.13 Instruction execution time**

The simulated DSP IP blocks maximum processing times were calculated using whole 80x60 pixel images and in the case of the getobjs operation, using an image with every pixel interconnected and set to the decimal value 255, i.e. one object filling the whole image. The processing times were measured from the rising edge of the ip\_busy signal to the falling edge of the ip\_busy signal. These processing times can be seen in table 4.14. All operations read and/or write data back to the primary image bank.

DSP IP Block	Simulated number of cycles at 80x60 pixels	Cycles per pixel (to four decimal places)
THRESHOLD	9601	2.0002
FILTER 3X3	9528	1.9850
RECTANGLE	277	1.0036
GETSCOODS	4805	1.0010
ABSDIFF	14404	3.0008
COPY	14401	3.0002
GETOBS	79216	16.5033

**Table 4.14 Processing times and cycles per pixel for the DSP IP block library**

Equations 4.7 to 4.12 provide the calculation of the maximum number of cycles required for the completion of each DSP IP block. The letter  $x$  represents the horizontal ROI size and  $y$  represents the vertical ROI size. These equations were developed using their Verilog descriptions and tested in simulation.

$$Cycle_{MAX} = 2xy + 1$$

**Equation 4.7 Maximum number of cycles to perform Threshold DSP operation**

$$Cycle_{MAX} = 2((x(y - 4)) + (2(x - 1))) + (3x + 1) + 11$$

**Equation 4.8 Maximum number of cycles to perform Filter3x3 DSP operation**

$$Cycle_{MAX} = 2(x - 1) + 2(y - 1) + 1$$

**Equation 4.9 Maximum number of cycles to perform Rectangle operation**

$$Cycle_{MAX} = xy + 4 + 1$$

**Equation 4.10 Maximum number of cycles to perform Getcoords operation**

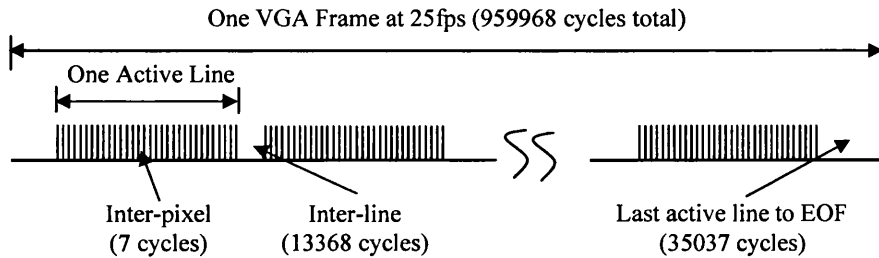
$$Cycle_{MAX} = 3xy + 3 + 1$$

**Equation 4.11 Maximum number of cycles to perform Absdiff operation**

$$Cycle_{MAX} = 3xy + 1$$

**Equation 4.12 Maximum number of cycles to perform Copy operation**

The available time slots to conduct uninterrupted processing on an incoming image are shown in figure 4.22.



**Figure 4.22 Three types of available time slot for uninterrupted image processing**

The number of measured cycles displayed in figure 4.22 takes into account that only a 640x480 pixel image is sampled from the incoming 644x484 pixel sensor image. Typically, most or all of the processing is performed at the end of the frame when the complete image is available for processing. In situations where a processing loop cannot be completed within the 35037 cycles at the end of each frame, a STOP\_PING instruction can be executed at the start of each processing loop followed by a START\_PINGPONG instruction at the end of each loop. This ensures that no incoming sensor images are stored and that the video generator continues to output the same image until the processing loop is completed and the next v\_sync signal is asserted. This method allows the system to automatically degrade the output frame rate from 25fps to 12.5fps, 8.33fps, 6.25fps and downwards, providing the processing loop takes approximately the same time to execute for each new image.

Table 4.15 shows the FPGA resources used by each system component, optimised for timing, at a system clock frequency of 24 MHz. In addition to providing figures for the number of registers, the number of each type of logic cell (column 3-5) making the total number of logic cells (column 2) has been provided. The total design uses 37.6% of the 24320 available logic cells within the FPGA.

	<b>Registers</b>	<b>Logic Cells (LC)</b>	<b>Register Only LC</b>	<b>LUT Only LC</b>	<b>Register/LUT LC</b>
<b>Top-Level Pins Structure</b>	0	43	0	43	0
<b>Top-level Architecture</b>	118	599	91	481	27
<b>192 MHz PLL</b>	3	3	1	0	2
<b>3x3Filter Block</b>	2150	2946	1966	796	184
<b>Absdiff Block</b>	98	290	8	192	90
<b>Copy Block</b>	41	53	2	12	39
<b>Getcoords Block</b>	31	80	3	49	28
<b>Getobjs Block</b>	330	2207	29	1877	301
<b>Rectangle Block</b>	50	167	12	117	38
<b>Threshold Block</b>	27	76	3	49	24
<b>Video Test Pattern Generator</b>	42	64	0	22	42
<b>System Control Unit</b>	117	549	0	432	117
<b>Ping-Pong Unit</b>	32	51	1	19	31
<b>Video Generator</b>	94	183	9	89	85
<b>Sensor Interface</b>	49	93	20	44	29
<b>SDRAM Decoder</b>	105	127	76	22	29
<b>Sensor Video Async. FIFO</b>	117	228	100	111	17
<b>Video Generator Async. FIFO</b>	64	113	14	49	50
<b>System Control Unit FIFO</b>	150	264	120	114	30
<b>Absdiff FIFO</b>	150	264	128	114	22
<b>Register Bank</b>	276	600	233	324	43
<b>I-Store</b>	0	8	0	8	0
<b>Line memory</b>	0	0	0	0	0
<b>Scratch Pad</b>	1	11	1	10	0
<b>Image Bank 1</b>	3	65	3	62	0
<b>Image Bank 2</b>	3	65	3	62	0
<b>Total</b>	<b>4051</b>	<b>9149</b>	<b>2823</b>	<b>5098</b>	<b>1228</b>

**Table 4.15 Resource usage by FPGA system component**

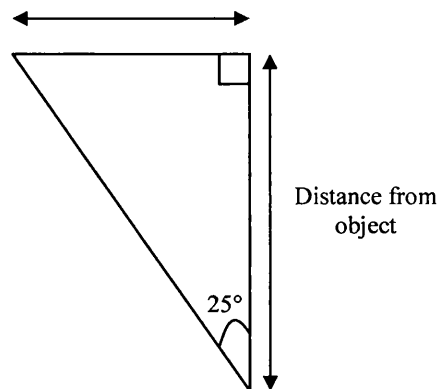
Table 4.16 shows the memory resources used in the FPGA for each memory device and the percentage of the 152 ESB used in the Altera Apex 20K600E FPGA.

	Address Depth	Data Bit Width	Total Size in Bits	Used ESBs
<b>Image Bank 1</b>	16384	8	131072	64
<b>Image Bank 2</b>	16384	8	131072	64
<b>Scratch Pad</b>	4096	8	32768	16
<b>I-Store</b>	256	8	2048	1
<b>Line Memory</b>	1024	8	8192	4
<b>Async. Video FIFO</b>	128	9	1152	1
<b>Total</b>			306304	150
<b>% of Available</b>			98.40	98.68

**Table 4.16 FPGA memory utilisation**

As many machine vision applications involve tracking objects, it is useful to gauge the maximum speed an object can travel to still be detected in at least one frame. An assumption is made that an object is large enough to be detected by one of the 4800 sub-sampled pixels and that it can escape detection by passing through the sensors field of view within the period of a sensor frame. A lens with a 50° field of view has been assumed. Using the model in figure 4.23 and basic trigonometric functions, half the distance the object is required to travel can be calculated. Table 4.17 contains the maximum velocity an object can travel at to still be detected by the system.

Half the distance the object must travel in the period of a frame to escape detection



**Figure 4.23 Distance an object must travel across the lens field of view**

Distance (m)	<b>25 FPS</b>	<b>12.5 FPS</b>	<b>8.33 FPS</b>	<b>6.25 FPS</b>	<b>5.00 FPS</b>	<b>4.17 FPS</b>
<b>1</b>	27.58	13.79	9.19	6.90	5.52	4.60
<b>2</b>	55.17	27.58	18.39	13.79	11.03	9.19
<b>5</b>	137.92	68.96	45.97	34.48	27.58	22.99
<b>10</b>	275.84	137.92	91.95	68.96	55.17	45.97

**Table 4.17 Maximum object velocity in m/s to still guarantee detection**

Sub-sampling of the sensor images affects the systems ability to reliably detect small objects. As only one pixel in every 8x8 pixel block is sampled, the object must be at least the size of an 8x8 pixel block to guarantee detection. Using the model in figure 4.23 the minimum object size has been calculated for four distances from the lens.

Table 4.18 shows these values.

<b>Distance (m)</b>	<b>Horizontal Sensor Pixel Size (mm)</b>	<b>Vertical Sensor Pixel Size (mm)</b>	<b>Minimum Horizontal Object Size (mm)</b>	<b>Minimum Vertical Object Size (mm)</b>	<b>Minimum Object Area (mm<sup>2</sup>)</b>
<b>1</b>	3.45	4.60	27.58	36.78	1014.54
<b>2</b>	6.90	9.19	55.17	73.56	4058.14
<b>5</b>	17.24	22.99	137.92	183.90	25363.39
<b>10</b>	34.48	45.97	275.84	367.79	101453.57

**Table 4.18 Minimum detectable object size**

Three technical challenges were encountered during the development of the FPGA. The first was that the video outputted to the monitor began to slowly shift vertically and horizontally as a result of an error in the timing produced by the video generator. The second was an intermittent fault that became apparent through data transmission errors across the six inter-board connector during the testing of the SDRAM decoder. This was resolved by replacing the old connectors with a set that connected more firmly to the socket on the daughterboard and the pins on the backplane. The third challenge was the limited size of the data scratch pad memory, which served to have an effect on the functioning of the Getobj IP block. It was noted that when one large object with a size of 80x60 pixels was used in tests, up to 6833 bytes of scratch pad memory was required to store all of the temporary data. Unfortunately, due to the restrictions on the amount of FPGA-embedded memory, it was not possible to run tests on the bench with a larger scratch pad memory. As such, the figure for the maximum number of cycles required for the Getobj operation was generated using a double sized scratch pad in simulation. The use of standard 4KByte scratch pad memory yield an execution time of 53757 cycles, compared to 79216 cycles for a 8KByte scratch pad. The execution time of the 4KByte memory was lower due to the getobj IP block ceasing to write further temporary data to the scratch pad and hence prevent a memory overflow by reducing the size of the search path for that

object. This resulted in the system detecting two objects instead of one when a visible object was close to the size of the complete screen.

Details of an example application prototyped on the system have been included in Appendix C.

## **4.9 Summary**

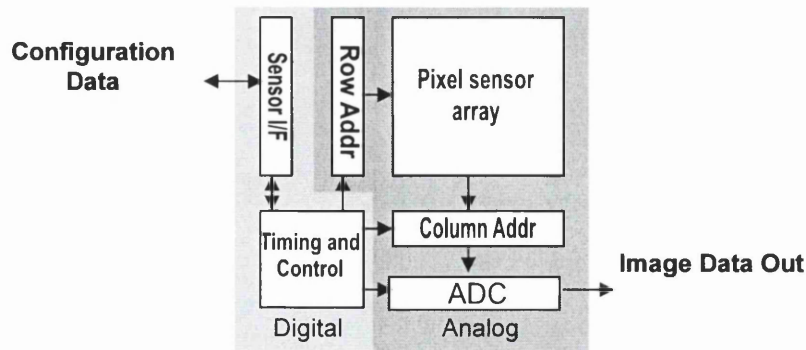
This chapter has covered the development of the FPGA-based architecture for the prototyping system. It has detailed the functionality of the load/store based system control unit and associated instruction store. The video data flow has been described through the sub-systems, with particular emphasis on the interleaving of data between the two image banks. Seven DSP IP blocks from the IP library have been detailed, their performance given and the equations used to calculate the number of cycles required for their execution provided. The simulation, functional verification and system programming mechanisms have been explained from the stand-point of the files required and their process flow.

## 5 IoC Manufacturing Cost Modelling

As the prototyping system was aimed at low-cost mass-market applications, it was thought beneficial to develop a cost model that could provide some indication as to the manufacturing cost of a single chip IoC. Several design and system-level cost models are described in available literature [101, 102, 103]. These models are limited in that they only address general SoC designs and do not take into account additional issues such as, a result of the inclusion of an array of photo-sensitive elements and associated support structures. The ability to model manufacturing cost is further exacerbated by the commercially sensitive nature of the data required for cost calculations. As a result, a manufacturing model was developed using publicly available data to provide an indication of the costs involved.

### 5.1 CMOS Imager Area Issues

CMOS imagers contain analog and digital components, see figure 5.1. In terms of area, the most area-dominant component is the analog pixel sensor array.



**Figure 5.1 Typical CMOS imager architecture**

The pixel array consists of numerous photo-sensitive elements that transduce photons into electrons. The size of each element determines its sensitivity to light, with larger elements being more sensitive. Most image sensors that have arrays of 640x480 pixels (VGA) typically have pixel element dimensions of 5.6 $\mu\text{m}$  by 5.6 $\mu\text{m}$  [104, 105, 106]. Below these dimensions, the optical diffraction limits of lenses start



to become apparent and result in the degradation of the overall output image quality. Although beyond the scope of this manufacturing model, the design of each pixel element is crucial to the collection of light. The more transistors used in a pixel element, the less space that is available to implement the photodiode. Typically, CMOS image sensors have 3 to 4 transistors per element and a photodiode with a fill factor of 40%. To improve the amount of light collected over the complete pixel element, micro lenses are overlaid on each element. This increases the amount of light collected, to provide a fill factor greater than 90%.

Another optically-related limiting factor, is the complete area of the pixel array. As the array is shrunk in size, it becomes increasingly difficult to accurately focus the image onto the pixel array. To achieve optimum accuracy as the pixel array decreases, the lens assembly and mechanical placement process result in an increased lens tooling and assembly cost.

The silicon process technology used to manufacture the image sensor has a substantial financial impact on the final IC cost. The use of the latest technology typically allows the digital parts of the sensor to be designed or scaled to a smaller area but analog parts usual cannot be easily designed or scaled below a 0.25 $\mu\text{m}$  process [101]. This can be explained by the decreased supply voltages used by smaller processes. Analog components ideally require a large voltage range to operate effectively. To circumvent this limitation, smaller processes are still used for the digital and analog components but the gate oxide thickness for the analog components is increased. This approach is called a dual oxide process and enables the analog components to be fabricated on the same die as the digital logic. It allows a higher operation voltage for the analog components while benefiting from a smaller process size for the digital components.

These area related issues may result in the imager becoming the dominate part within an IoC. A potential risk is that the IoC design becomes no longer financially feasible due to a large cost associated with a large die size and the cost sensitive nature of the target market. This places most of the design emphasis on the optimisation of the pixel and array dimensions. Careful selection of the silicon process technology used is also required to ensure a feasible product.

## 5.2 Cost Model

The system cost model for an IoC is given in equation 5.1.  $C_{system}$  is the system cost,  $C_{die}$  is the cost of a yielded die,  $C_{test}$  is the test cost,  $C_{packaging}$  is the cost of packaging,  $C_{NRE}$  is the non-recurring expense and  $n$  is the number of parts produced. It must be noted that this model does not account for packaging yield.

$$C_{system} = C_{die} + C_{test} + C_{packaging} + \frac{C_{NRE}}{n}$$

**Equation 5.1 IoC system cost model**

The die cost can be calculated using equation 5.2, where  $C_{die}$  is the cost of a yielded die,  $C_{waf}$  is the cost of a wafer,  $Y_{die}$  is the die yield,  $A_{waf}$  is the area of the wafer,  $A_{die}$  is the area of the die and  $U_{waf}$  is the utilization of the wafers area for die.

$$C_{die} = \frac{C_{waf}}{\frac{(U_{waf} A_{waf})}{A_{die}} Y_{die}}$$

**Equation 5.2 Die cost calculation**

To calculate the overall die yield,  $Y_{die}$ , the commonly used ITRS die yield model was applied. See equation 5.3.

$$Y_{die} = Y_S Y_R = Y_S \left( \frac{1}{1 + \frac{A_{die} D_0}{\alpha}} \right)^\alpha$$

**Equation 5.3 Die yield model [107]**

In equation 5.3,  $Y_{die}$  is the overall die yield,  $Y_S$  is the gross limited yield and  $Y_R$  is the random-defect limited yield.  $\alpha$  is the cluster factor which models the defect distribution amongst most fabrication facilities.  $A_{die}$  is the area of the device die and  $D_0$  is the electrical fault density.

Optical and electrical test costs,  $C_{test}$ , were approximated using a value of \$0.02 per pin. Package cost variable,  $C_{packaging}$ , was approximated at \$0.01 per pin.  $C_{NRE}$  was taken as the mask set costs, as this is the dominant non-recurring expense during manufacture.

To calculate the area of the die, a summation of each system component was performed. An assumption has been made that the area occupied by interconnects between each component is negligible. Equation 5.4 shows an example equation for a typical IoC.

$$A_{die} = A_{DSP} + A_{sensor} + A_{video} + A_{\mu C} + A_{RAM} + A_{ROM} + A_{PLL}$$

**Equation 5.4 Area of die**

Where  $A_{die}$  is the area of the device die,  $A_{DSP}$  is the area of the DSP,  $A_{sensor}$  is the area of the sensor,  $A_{video}$  is the area of the video encoder including 8-bit DAC,  $A_{\mu C}$  is the area of the 8-bit micro-controller,  $A_{ROM}$  is the area of the ROM,  $A_{PLL}$  is the area of the PLL and  $A_{SRAM}$  is the area of the SRAMs.

A model for producing approximate values for  $A_{sensor}$  was devised based on a theoretical VGA sensor design in a 0.25 $\mu$ m process. Equations 5.5 to 5.10 form this model.

$$A_{sensor} = A_{array} + A_{ctrl} + A_{addr} + A_m$$

**Equation 5.5 Area of sensor**

$$A_{array} = x_{new} y_{new} A_{pixel}$$

**Equation 5.6 Area of pixel array**

$$A_{ctrl} = P_{scale} A_{cnorm}$$

**Equation 5.7 Area of digital logic**

$$A_{addr} = \left( \frac{y_{new}}{y_{norm}} \right) P_{scale} A_{row} + \left( \frac{x_{new}}{x_{norm}} \right) P_{scale} A_{col}$$

**Equation 5.8 Area of analog address units**

$$A_m = P_{scale} A_{mnorm}$$

**Equation 5.9 Area of miscellaneous analog components**

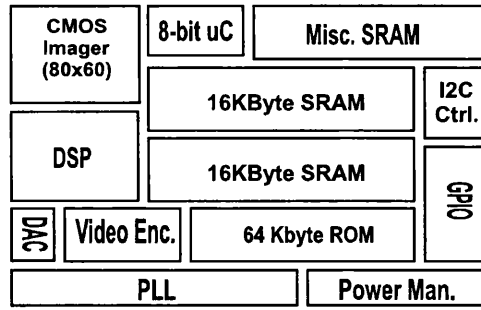
$$P_{scale} = \left( \frac{P_{new}^2}{P_{norm}^2} \right)$$

**Equation 5.10 Inter-process scaling factor**

Where  $A_{array}$  is the area of the pixel,  $A_{ctrl}$  is the area of the digital control, timing and interface logic,  $A_{addr}$  is the area of the analog address units, line memory and ADCs and  $A_m$  is the miscellaneous analog components such as power management.  $P_{scale}$  is the scaling factor for area from one process to another and for equations 7.8 and 7.9, if  $P_{new} \leq 0.25$  then  $P_{scale} = 0.9$  for  $0.18\mu\text{m}$  and  $P_{scale} = 0.9^2$  for  $0.13\mu\text{m}$ .  $P_{norm}$ ,  $x_{norm}$  and  $y_{norm}$  are constants in the model i.e. silicon process, 0.25, pixel array row size, 640, and pixel array column size, 480, respectively. The remaining constants for a VGA imager are; digital logic ( $A_{cnorm}$ ) at  $5\text{mm}^2$ , row address unit ( $A_{row}$ ) at  $1\text{mm}^2$ , column address/decode unit ( $A_{col}$ ) at  $4\text{mm}^2$  and miscellaneous analog components ( $A_{mnorm}$ ) at  $3\text{mm}^2$ .  $P_{new}$  is the silicon process at which to calculate the die area.  $X_{new}$  and  $Y_{new}$  are the required pixel array row and column size.

### 5.3 Application of Model to Example IoC

As a demonstration of the use of the model, an IoC that was suitable to support the application discussed in appendix C was created. Figure 5.2 shows the integrated sub-systems that would be required to support the example application. In addition to the FPGA's sub-systems (marked as DSP), the 8-bit microcontroller has been added with a 64 KByte metal programmable ROM, video encoder, 8-bit DAC, simple power management, general-purpose I/O and an I<sup>2</sup>C bus controller.



**Figure 5.2 An example IoC to support the demonstration application**

The Altera 20K gate counting method was used to produce the equivalent number of gates from the number of logic elements (LE) used in the FPGA architecture. This was calculated by multiplying the number of LEs by a factor of 12 [108]. The area was then obtained by dividing the gate count by the gate density for a given process. The die areas of the ROM and RAMs,  $A_{ROM}$  and  $A_{RAM}$ , were obtained using Dolphin Integration's memory generator for TSMC processes [109].  $A_{uC}$  and  $A_{video}$  were approximated at 10K gates and 4K gates plus the area of the DAC, respectively. The I<sup>2</sup>C controller was estimated as 3600 gates and the power management at 300 gates. The pixel area,  $A_{pixel}$ , of the CMOS imager was set to  $5.6\mu\text{m}$  by  $5.6\mu\text{m}$  for the example IoC with an array size of 80 ( $x_{new}$ ) by 60 ( $y_{new}$ ) pixels. The other values used in the calculations are summarised in table 5.1 for an 8 inch wafer production.

	<b>0.35<math>\mu\text{m}</math></b>	<b>0.25<math>\mu\text{m}</math></b>	<b>0.18<math>\mu\text{m}</math></b>	<b>0.13<math>\mu\text{m}</math></b>
<b>Mask set costs (US\$) [110]</b>	50000	85000	250000	600000
<b><math>C_{waf}</math> (US\$)</b>	1000	1500	2000	2500
<b><math>D_0</math> (mm<sup>2</sup>)</b>	0.003	0.004	0.006	0.01
<b>Gate Density (gates/mm<sup>2</sup>) [111]</b>	25000	50000	100000	200000
<b><math>\alpha</math> [107]</b>	2			
<b><math>U_{waf}</math></b>	0.97			
<b><math>Y_s</math></b>	0.8			
<b><math>A_{waf}</math> (mm<sup>2</sup>)</b>	31416			
<b>Number of pins required by IoC</b>	44			

**Table 5.1 Values used for cost model**

The total area for the system and the imager at different processes, are shown in figure 5.2. It can clearly be seen, that as the process decreases in size, the imager die size becomes more dominant in the complete die area.

Process ( $\mu\text{m}$ )	Imager Size ( $\text{mm}^2$ )	Total System Die Size ( $\text{mm}^2$ )	% of Total Die as Imager
0.35	17.10	44.20	38.69
0.25	8.78	20.10	43.68
0.18	6.01	11.22	53.57
0.13	4.44	7.08	62.71

Table 5.2 Area size for imager and complete IoC

The values obtained for the example IoC unit cost are tabulated in table 5.3. These have been used to create the graph in figure 5.3. For comparison, the fixed cost of producing the two-board prototyping system was approximated at \$1400 US and indicated on the graph for a single unit quantity.

Units	Silicon Technology Process			
	0.35	0.25	0.18	0.13
0	50000.0000	85000.0000	250000.0000	600000.0000
1	50003.3813	85002.6579	250002.3037	600002.0989
5	10003.3813	17002.6579	50002.3037	120002.0989
10	5003.3813	8502.6579	25002.3037	60002.0989
50	1003.3813	1702.6579	5002.3037	12002.0989
100	503.3813	852.6579	2502.3037	6002.0989
500	103.3813	172.6579	502.3037	1202.0989
1000	53.3813	87.6579	252.3037	602.0989
5000	13.3813	19.6579	52.3037	122.0989
10000	8.3813	11.1579	27.3037	62.0989
50000	4.3813	4.3579	7.3037	14.0989
100000	3.8813	3.5079	4.8037	8.0989
500000	3.4813	2.8279	2.8037	3.2989
1000000	3.4313	2.7429	2.5537	2.6989
5000000	3.3913	2.6749	2.3537	2.2189
10000000	3.3863	2.6664	2.3287	2.1589

Table 5.3 Cost of the IoC at a given process and unit quantity

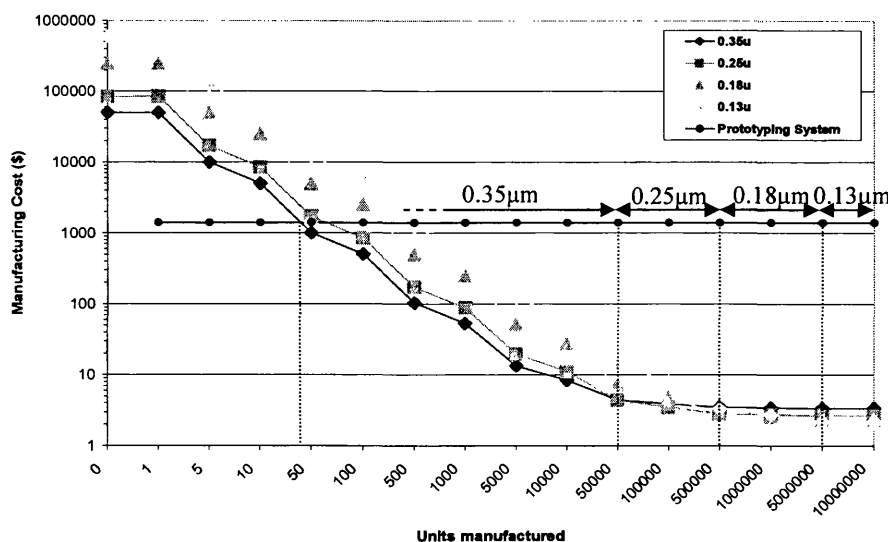


Figure 5.3 Manufacturing cost graph for example IoC

The silicon processes attributed with the lowest unit cost for a given number of units manufactured are shown on figure 5.3. Examining the graph it can be seen that after 40 units there is clearly a benefit of fully integrating the prototyping system into an IoC.

## **5.4 Summary**

Using publicly available data, a high-level IoC manufacturing cost model has been presented. Optically-related CMOS imager issues which may affect the end unit cost of an IoC have also been highlighted. An IoC has been specified which would support the application detailed in appendix C. The unit cost of this example IoC has been calculated using the cost model for four technology processes at different unit quantities. The model highlighted the importance of carefully selecting the technology process used, the size of the imager array implemented and the quantity of units produced to meet a required die unit cost.

## **6 Discussion and Conclusions**

This thesis has described the development of an IoC application prototyping system. Chapter two reviewed the current literature for vision systems and directed the research towards the implementation of a frame-based architecture using FPGA technology. Chapter three detailed the system's requirements and specifications with particular emphasis on the board-level bus structures. Chapter four and five described the board-level and FPGA-level architecture implemented and the results obtained during testing. An IoC manufacturing cost model was presented in chapter six to calculate the cost of implementing single chip IoC applications. In this chapter results from the previous chapters are discussed and a range of improvements to the prototyping system and possible areas of future research presented.

### **6.1 Discussion**

The aim of this research was to investigate and develop a new prototyping platform for low-cost mass-market IoC applications. A frame processing based system architecture was devised. This consisted of a re-usable FPGA backplane and daughter card containing the necessary components to support a range of image processing and machine vision applications. A technology independent on-chip architecture was implemented on the FPGA. Given the technology-independent design and the STMicroelectronics components selected, the integration of the system into a single IoC should be feasible. An example object counting and highlighting real-time application was implemented to demonstrate that the system was suitable for the implementation of machine vision applications. Using the manufacturing cost model, the costs of implementing the application as an IoC at different TSMC silicon processes has been provided. In this chapter, the results of the implementation of PCB, FPGA architecture and complete prototyping system are discussed. The implications of the cost model will also be discussed and recommendations for future work detailed.



### **6.1.1 PCB Work**

As no suitable daughter board for connection to an STMicroelectronics backplane was available, development of a new daughter board was necessary.

STMicroelectronics requested that wherever possible its components should be used. The combination of the daughter board and FPGA worked sufficiently to meet the project requirements. At an approximate unit cost of US\$1400 it compares favourably with similar systems, such as the AteMEK6414 at approximately US\$6000 per unit.

An advantage of the complete two board system was the opportunity to upgrade the FPGA by replacing the backplane. The decision to mount the daughter board on top of the FPGA backplane allowed the whole system to be easily handled while operational. The ability to easily handle the complete system was particularly useful during the testing of the object segmentation applications, as the system could be aimed at a target area or an object with relative ease. Once the minor board modifications had been made, the performance of the system met specification. Four different CMOS image sensors were tested with the system and all functioned correctly.

A disadvantage of the two board system was that no direct connection between the sensor and FPGA was implemented. If the daughter PCB was to be redesigned this board-level architectural issue should be addressed. Whilst the decision to provide access to only SDRAM and FLASH memory devices was driven to provide low-cost high-capacity memory devices, two SRAM devices may have been more suitable. The addition of two SRAM ICs would have provided the opportunity to develop IoC applications requiring image sizes greater than 80x60pixels without the control overheads of SDRAM. It may also be possible to use or modify the 54-pin SDRAM IC TSOP(II) footprints to support SRAM devices. The ability to easily move the complete system while operational was partly attributable to the tight coupling of the six inter-board connectors. As a result of the thickness of connector pins it is feasible that at frequencies above 24 MHz the transmission delays may become unacceptable, signals within the same bus may become skewed with each other or cross-talk effects between wires may be noticed. At the low frequencies that data

was transmitted across these connectors, there were no issues regarding signal timing or signal integrity issues as a result of capacitance effects. To assess these issues further, electrical testing of the complete system is required. The size of the PCB could be minimised by reducing the empty spaces on the left side and bottom edge of the board by a more compact placement of components.

The key bottleneck in the prototyping system's architecture was that of the IMPBUS and STV0674 co-processor. As the SDRAM controller was specified for use when communicating with the FPGA, the maximum data transmission speed across the IMPBUS was at 384Mbit/s at 24MHz. This was as a result of the STV0674 clock domain structure and the design of the co-processor's SDRAM controller. This operational bus frequency supports 25fps video streams of a maximum resolution of 1600x1200 with an 8-bit pixel depth. This resolution does not take into account the overhead of non-burst SDRAM transactions, which are three cycles for every byte-read and byte-write cycle. As only one cycle in every four cycles read or wrote data, the overhead was 75%. This lowered the effective data transmission speed across the IMPBUS to a maximum speed of 92Mbit/s at 24MHz. This would support a maximum resolution of 800x600 8-bit pixels at 25fps. As the maximum resolution of the sensors to be used with the prototyping system was 640x480, this bottleneck was not deemed to be problematic, unless an image stream had to be read from and written to the FPGA concurrently.

To reduce the transmission of data over the IMPBUS, the sensor's data bus and clock line were fed into the FPGA using a ribbon cable rather than via the co-processor and IMPBUS. A potential solution to the IMPBUS bottleneck would have been to have implemented the co-processor's 8052 compatible microcontroller in the FPGA. Due to licensing issues this was not possible. An alternative solution would have been to implement a microprocessor with USB port and fast GPIO, at board-level. This would have removed the requirement of SDRAM transactions for communication across the IMPBUS. The disadvantage of this approach would be the requirement to obtain a suitable development suite and modify the original 8052 code to execute on the new processor. It is also very likely that the Altera 20KE FPGA GPIO pins would limit the maximum bus speed to less than 100MHz.

## 6.1.2 FPGA Work

The FPGA architecture developed was based on 8-bit data paths and a load-store system control unit with an associated instruction store. A library of hardware DSP algorithms was implemented to perform the image processing operations within the architecture.

### 6.1.2.1 The Instruction Set

The first set of results obtained for the FPGA system was the number of cycles each instruction needed to execute. The time taken to decode each of these instructions was one system clock cycle. It can be seen in table 4.13 that approximately half of all the instructions were required to obtain their literal values on a separate clock cycle. The execution time was also affected for memory operations by the set up of an address in the 16-bit read or write address registers. The requirement to set up these addresses could result in the worst-case values show in table 6.1. It must be noted that the instruction decode time has also been added to each value.

Instruction Name	Number of cycles	Instruction Name	Number of cycles
NOP	2 (1)	LOAD MEM*	3 + 6 (2-3)
WAIT EOF	2 (n/a)	ADD REG GPR0*	3 (1)
WAIT IPBUSY	2 (n/a)	ADD REG GPR1*	3 (1)
STOP PINGPONG	2 (n/a)	ADD REG ADDR RD*	3 (1)
START PINGPONG	2 (n/a)	ADD REG ADDR WR*	3 (1)
LOOP (Jump absolute)	5 (1-3)	SUB REG GPR0*	3 (1)
THRESHOLD	2 (n/a)	SUB REG GPR1*	3 (1)
FILTER 3X3	2 (n/a)	SUB REG ADDR RD*	3 (1)
RECTANGLE	2 (n/a)	SUB REG ADDR WR*	3 (1)
GETSCOODS	2 (n/a)	LOAD REG GPR0*	3 (1)
ABSDIFF	2 (n/a)	LOAD REG GPR1*	3 (1)
COPY	2 (n/a)	LOAD REG ADDR RD LO*	3 (1)
GETOBS	2 (n/a)	LOAD REG ADDR RD HI*	3 (1)
MOVE MEMMEM	4 + 6 +6 (n/a)	LOAD REG ADDR WR LO*	3 (1)
MOVE MEMREG GPR0	4 + 6 (2-3)	LOAD REG ADDR WR HI*	3 (1)
MOVE MEMREG GPR1	4 + 6 (2-3)	BEZ* (Jump relative)	6 (2-3)
MOVE REGMEM GPR0	2 + 6 (2-3)	BNEZ* (Jump relative)	6 (2-3)
MOVE REGMEM GPR1	2 + 6 (2-3)		

**Table 6.1 Worst case instruction execution time**

The typical values for common 16-bit and 32-bit instruction processors have been added to table 6.1 in brackets. The figures for the developed architecture compare

poorly with most microcontrollers and microprocessors. Four explanations are provided in relation to these values.

1. The instruction non-pipelined decode mechanism developed for the system control unit is sub-optimal as a result of adding an extra cycle onto the execution of every instruction.
2. The use of an intermediate mode by storing data required for an operation following the instruction in program memory increases the execution time of almost half the instructions by one cycle.
3. The specification of the 16-bit read and write addresses in registers for indirect memory addressing, adds an extra 3 cycles for every byte of an address register changed during a MOVE operation.
4. The absolute and relative branching mechanism has a performance below most other processors due to FIFO latencies.

It could be argued that as a result of the use of the DSP IP blocks, very little of the complete application time is taken up processing instructions. Several modifications could be made to the system control unit without impacting negatively on the remainder of the system. The easiest of these to implement are listed below.

1. Pipeline the decode mechanism to remove the unnecessary extra clock cycle.
2. Implement an optional direct memory addressing scheme by storing a 16-bit memory address in the two subsequent bytes after an instruction. In the case of MOVE\_MEMMEM it would require four subsequent bytes after an instruction.
3. Add more general-purpose registers to hold frequently used values. i.e. reduce the frequency of memory transactions. The system architecture can currently support up to 16 of these registers in total.
4. Add an increment by one and decrement by one instruction for register operations and post-execution increment or decrement addressing for memory operations.

The second set of potential modifications is more complex than the first set. These are:

5. Change the instruction length to 16-bits by concatenating a literal value as the least significant byte with the instruction. This would save an additional execution cycle for almost half of all operations and reduce the number of FSM states by one. This would require the instruction stores word width to be doubled, along with the FIFO (unless removed) and all intra-system control unit data-paths. A mechanism would have to be implemented to convert the I-stores 16-bit data into 8-bit words for the purpose of writing and reading to and from the SDRAM decoder.
6. Reorganise the instructions opcodes to make more effective use of the 16-bit long instructions. This could be based on the instruction's frequency of use. The instruction set could also use instruction compression techniques as used by ARM and Atmel's Thumb based microcontrollers.
7. As the FIFO currently pre-fetches only one instruction, it could be removed. This would remove the latency of the FIFO reset and re-filling process.

The implementation of the first four modifications would probably require approximately 300 FPGA logic cells. Modifications 5-7 would result in the requirement of an extra ESB memory block, if the I-store depth was maintained, and a modest reduction of up to 200 logic cells as a result of the removal of the pre-fetch FIFO.

Table 6.2 shows the effect of implementing one, two, five and seven from the list of potential modifications.

Instruction Name	Number of cycles	Instruction Name	Number of cycles
NOP	1 (1)	LOAD MEM*	1 + 1 (2-3)
WAIT_EOF	1 (n/a)	ADD_REG_GPR0*	1 (1)
WAIT_IPBUSY	1 (n/a)	ADD_REG_GPR1*	1 (1)
STOP_PINGPONG	1 (n/a)	ADD_REG_ADDR_RD*	1 (1)
START_PINGPONG	1 (n/a)	ADD_REG_ADDR_WR*	1 (1)
LOOP (Jump absolute)	3 (1-3)	SUB_REG_GPR0*	1 (1)
THRESHOLD	1 (n/a)	SUB_REG_GPR1*	1 (1)
FILTER_3X3	1 (n/a)	SUB_REG_ADDR_RD*	1 (1)
RECTANGLE	1 (n/a)	SUB_REG_ADDR_WR*	1 (1)
GETSCOOORDS	1 (n/a)	LOAD_REG_GPR0*	1 (1)
ABSDIFF	1 (n/a)	LOAD_REG_GPR1*	1 (1)
COPY	1 (n/a)	LOAD_REG_ADDR_RD_LO*	1 (1)
GETOBSJS	1 (n/a)	LOAD_REG_ADDR_RD_HI*	1 (1)
MOVE_MEMMEM	3 + 2 (n/a)	LOAD_REG_ADDR_WR_LO*	1 (1)
MOVE_MEMREG_GPR0	3 + 1 (2-3)	LOAD_REG_ADDR_WR_HI*	1 (1)
MOVE_MEMREG_GPR1	3 + 1 (2-3)	BEZ* (Jump relative)	3 (2-3)
MOVE_REGMEM_GPR0	1 + 1 (2-3)	BNEZ* (Jump relative)	3 (2-3)
MOVE_REGMEM_GPR1	1 + 1 (2-3)		

**Table 6.2 Improved instruction worst-case execution time**

It is evident from table 6.2 that MOVE\_MEM operations are still outside an acceptable range. If only 8-bits were required to access the relevant memory read address, they could be added as an operand to the instruction. The MOVE\_MEMREG operations would then be reduced to three clock cycles. This partial address mode could allow the other MOVE instructions to have their execution times decreased by one cycle. In the case of the MOVE\_MEMMEM the memory write address could be fetched during the memory read latency and hence reduce its execution time to three clock cycles also. This would allow an 8-bit read address to be provided with the instruction and a full 16-bit write address retrieved on the next clock cycle ready for the following clock cycle in which a write to memory would be performed.

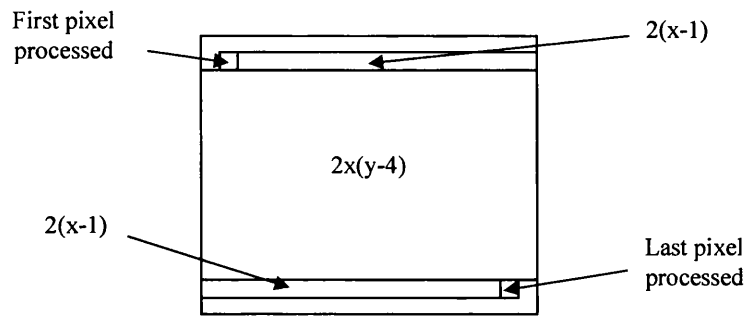
Several methods exist for improving branch performance but many of them would significantly increase the size of the system control unit. These include branch prediction based on opcode or instruction history, pre-fetch of a branch target instruction and caching of recently fetched instructions.

### 6.1.2.2 The DSP IP Block Library

Table 4.13 and equations 4.7 to 4.12 demonstrate the number of cycles required to execute the DSP IP block on a complete 80x60 pixels image. All DSP IP blocks require an initial clock cycle to start their address generator and sample the ROI values at their input. Rectangle and getcoords IP blocks, both have cycles per pixel in the region of one. This is explained by both blocks performing only writes or reads on the image data. In the case of the rectangle IP block, the number of cycles is equal to the number of pixels written after the one cycle set-up. The getcoords IP block performs a read of every pixel in an image and then writes the minimum and maximum values for x and y into the register bank, in the last four clock cycles.

The threshold IP block performs a read-modify-write operation. It would be expected that the absdiff IP block and copy IP block would have a very similar value for the cycles per pixel. This is not the case, as the design of the threshold IP block was different to the other two IP blocks. The threshold block was optimised to perform reads and writes on consecutive cycles. This was achieved by not using a register in the output data path, i.e. the data input port was wired directly to the data output port with a compare function built into the datapath. This optimisation would be possible with the copy and absdiff IP blocks, without causing timing closure problems during the synthesis process. As can be seen by equation 4.11, the absdiff IP block had an extra three cycles compared to the copy IP block. These three cycles were required to set-up the FIFO on the secondary memory port and start filling the FIFO with image data.

The filter3x3 IP block had an unusual cycle per pixel count as a result of its internal architecture. The constant value of 11 in equation 4.8 is the number of cycles required to obtain the nine weights. The  $(3x+1)$  term, is the time taken to fill the three shift registers with image data. Two cycles were required to read-modify-write each pixel as a result of the pipelined structure. The  $2((x(y-4))+(2(x-1)))$  part of the equation, relates to the number of cycles for processing each pixel. This last part of the equation is clarified in figure 6.1.



**Figure 6.1 Clock cycles required to process each pixel using the 3x3 filter DSP IP block**

There are two optimisations which could be performed on the filter3x3 block. The first is to pipeline the reads performed for filling the shift registers and obtaining the weights. This would reduce the cycles count by two. The second optimisation is to remove the first shift register and start feeding pixel data into the multipliers as soon as the first pixel from the third line has been obtained. This would reduce the time required to fill the registers to  $2x+1$  and reduce the register count of the 3x3 filter IP block by approximately 600 register logic cells.

The getobjs IP block required 16.5 cycles per pixel to obtain information on each object within an image. An equation was not presented, as the equation execution time is dependant upon the number of objects in the scene and their size and shape. The number of cycles required during the initial reading of the pixels however can be calculated. Every pixel is read at least once. Only four pixels can be obtained when at the corner of an image and six pixels while on the edges of an image. This provides a figure of 42364 cycles for an 80x60 pixel image, during which only reads are taking place. This figure does not include pixel coordinates reads from the scratch pad or the pixel values read at these coordinates. This would indicate that 53% of clock cycles are spent obtaining the pixels in the first instance. It is always the case that once the algorithm cannot find any more pixels to label it uses the coordinates of a previous object pixel to continue its search. For large objects that appear as a filled shape, already processed neighbourhoods of pixels are re-read. This is inefficient and therefore could be optimised. This could be achieved by removing addresses of pixel coordinates from the scratch pad where all the pixel's neighbouring pixels had been labelled. This could save an extra 8 cycles per stored set of pixel coordinates that are no longer required, as the labelled neighbouring



pixels would not have to be re-read. Another optimisation could be the caching of recently read pixels or the storage of neighbourhoods that did not need reading as they had already been labelled.

A further optimisation for the getobjs IP block is the better handling of the scratch pad memory. In the situation of a single object filling the image, only eight bytes would be required from the 2048 bytes in lower half of the scratch pad memory to store the objects parameters. This could be improved by writing temporary data starting from the highest address, i.e. 4095, down until the first set of object parameters. This would provide up to a further 2040 bytes for temporary data. Once all of the temporary data had been processed, it was no longer needed and could be overwritten by object parameters or new temporary data relating to a new object.

The threshold, 3x3filter, and copy DSP IP blocks could have their cycles per pixel reduced to one, if they were used in a situation where data was read from one image bank and output to another image bank. This assumes that either the video generator was not used or the IP blocks could interleave data safely between reads by the ping-pong unit. This is not applicable for absdiff, as this DSP IP block currently has to read images from both image banks. The getobjs DSP IP block could also benefit from the use of a dual-port configuration to allow concurrent reads and writes between two image banks and the scratch pad.

### 6.1.2.3 Available Processing Time

Figure 4.22 showed the available time slots for uninterrupted image processing. During the project only the last 35037 cycles of each frame was used for processing. Applications that required more time to execute, caused the system to automatically decrease the output frame rate of the video generator. This mechanism worked well as demonstrated by the example application. It is possible to increase the amount of time at the end of each frame window by changing the output of the vertical synchronisation pulse, `v_sync`, from the sensor interface. This could be achieved by outputting the `v_sync` signal at the start of the active video lines, instead of at the start of each sensor frame. This shifts 25 line times of the next frame to the end of

the current frame, effectively extending the time between the last active line and the end of frame. An extra 45800 cycles would be gained from this shift, a total of 4.77% of the total frame time. The total time available for processing at the end of each frame would be increased to 80837 cycles (3.37ms) representing 8.42% of the total frame time.

To gain any further processing time using 80x60 pixel images, the inter-pixel and inter-line time slots would be required to be utilised. Using the DSP IP interface could prove to be a problem when trying to process within the inter-pixel time slots if the horizontal resolution of the image was increased. For example, an increase of horizontal image size from 80 pixels to 320 pixels would reduce the inter-pixel time to one cycle. With one cycle it would not be possible to perform the necessary pixel processing functions. Ideally the inter-pixel time slots could be used to perform point operations on each pixel as they arrived. Instead of using the current DSP IP interface a configurable pipeline could be introduced between the sensor interface and ping-pong unit. This could be used to implement various functions including statistics gathering, such as image histogram or filter operations to remove noise. In addition to the possible pipeline processing of data from the sensor interface, a processing pipeline could also be added to the output to the video encoder. This could perform post-processing functions, such as graphics or text overlay on images.

Using the inter-line time slots provides more time to perform processing and would allow the use of line processing algorithms. This could easily be achieved using the current DSP IP interface. Even if the vertical resolution was increased by a factor of four to 240 lines, 2376 cycles would still be available for processing use. With an image size of 320x240 pixels, three full line two-cycle line operations could be executed. To facilitate the use of line based algorithms, the addition of an inter-line strobe could be provided to indicate when an inter-line time slot is available.

#### 6.1.2.4 FPGA Resource Usage

As stated in the results chapter, the FPGA used 37.6% of logic cells and 98.7% of the available embedded system blocks, each containing 2048bits of memory.

Examination of the types of logic cells used by the system's sub-system and DSP IP blocks indicated the following:

- The top-level architecture is heavily biased to using only the look-up table (LUT) parts of the logic cells. This is explained by the fact that the usage figure provided in table 4.16 also includes the network controller and point-to-point bus network. These two components require very few registers, as most of the required circuitry is made from multiplexers and simple priority structures. These can be implemented using the 4-input LUTs embedded in each of the Altera FPGA's logic cells [112].
- Instantiated memory devices require very few or no registers as each ESB block has registered inputs and output, if required.
- The PLL used is one of four PLLs embedded into the FPGA fabric. The three registers used, generate three clock domains, 96, 48 and 24 MHz.
- The FIFOs used a balance of registers and LUT. This is understandable given that the FPGA synthesizer and logic fitter can make more efficient use of registers, as the storage elements in each FIFO, rather than ESBs. It is interesting to note that the registers only LCs and LUT only LCs have not been combined to increase the use of register/LUT combined configurations. This points to the high-level of control structure required per register element used, forcing use of an extra LC per register. This is also evident with the two asynchronous FIFOs which require careful read and write control between two clock domains. In the case of asynchronous FIFOs, the balance is slightly more skewed to LUT only LCs when compared to the synchronous FIFOs. The number of registers used by the asynchronous video FIFO was much lower than the other FIFOs as it also used an ESB. The use of the ESB was probably as a result of a logic mapping optimisation, given the FIFOs depth and data width of 9-bits rather than a multiple of two.
- Most of the other system IP blocks had a majority of LUT only LCs. An obvious exception to this was the 3x3 filter DSP IP block. This exception can be explained by the use of three 80 register long shift registers. These shift registers accounted for 1848 of the total 1966 registers only LCs and no LUT or Register/LUT combination LCs. Another exception was the SDRAM decoder IP block. This is explained by the registers required for the

conversion of the 16-bit data and 23-bit addresses from the IMPBUS clock domain to the FPGA's clock domain.

The percentage of the FPGA's total LCs for each of the system blocks has been provided in table 6.3. IP Blocks with associated FIFOs have had their LC count amalgamated.

<b>IP Block Name</b>	<b>Number of Logic Cells (LC)</b>	<b>% of Total LC used</b>
3x3Filter Block	2946	32.20
Getobjjs Block	2207	24.12
System Control Unit + FIFO	813	8.89
Register Bank	600	6.56
Core-level Architecture	599	6.55
Absdiff Block + FIFO	554	6.06
Sensor Interface + FIFO	321	3.51
Video Generator + FIFO	296	3.24
Rectangle Block	167	1.83
SDRAM Decoder	127	1.39
Getcoords Block	80	0.87
Threshold Block	76	0.83
Image Bank 1	65	0.71
Image Bank 1	65	0.71
Video Test Pattern Generator	64	0.70
Copy Block	53	0.58
Ping-Pong Unit	51	0.56
Top-Level Pins Structure	43	0.47
Scratch Pad	11	0.12
I-Store	8	0.09
192 MHz PLL	3	0.03
Line memory	0	0.00
<b>Total</b>	<b>9149</b>	<b>100</b>

**Table 6.3 LC usage and percentage of total FPGA LC by IP block**

As is stated in table 6.3, the 3x3Filter and getobjjs DSP IP blocks occupy the vast majority of the LC used, with a combined FPGA LC usage of 56.32%. Using the recommended optimisation of removing a shift register from the 3x3filter would bring its LC usage to approximately 2330 LCs. Given the size of the 3x3filter and the getobjjs IP block, it may be argued that they should not be included unless absolutely necessary. In the case of the getobjjs block, a small general-purpose processor core may be more area efficient but at the cost of increased processing time. This option is less applicable to the 3x3filter as it can perform nine multiplications, a summation and a shift per clock cycle and hence process a pixel

per clock cycle. The implementation of this function on a small general-purpose processor core would generally require in excess of nine clock cycles per pixel given the computationally expensive nature of multiplier operations.

Table 6.4 shows the number of LCs used and their gate equivalent for three types of sub-systems within the FPGA architecture.

<b>Sub-System Type</b>	<b>% Total LCs</b>	<b>Gates Equivalent (12 Gates per LC)</b>
DSP IP Blocks	66.48814078	72996
Core System Architecture	26.03563231	28584
Memory Elements	7.47622691	8208
<b>Total</b>	100	109788

**Table 6.4 Number of LCs used by each sub-system type of the FPGA architecture**

As calculated in table 6.4, the DSP IP block occupies two-thirds of the total LCs used. The value given for the gate count, for the memory elements, excludes the ESB used in the FPGA. Including the ESB in the gate equivalence calculation, would add a further 1225216 gates to the total, for the 306304 bits used and using a four gates per bit conversion factor. It is interesting to note that the size of the complete system, excluding ESBs is relatively small in comparison to modern SoC designs. In fact, many individual IP blocks currently designed have gate counts in excess of 100Kgates. It must be noted that 12 gates per LC is only an estimation. This figure has been obtained from the Altera 20K gate count methodology used in the IoC manufacturing cost model. The disadvantage of the gate count methodology is that it is based on an average LCs to gates value obtained by the synthesis of over 100 designs using only one technology library and one synthesis tool for comparison.

#### 6.1.2.5 Minimum Object Size and Maximum Object Velocity

The minimum size and maximum velocity of an object detectable by the system using a lens assembly with 50° field of view indicates the prototyping system's suitability for certain applications. As the object's distance from the lens increases, the maximum velocity an object can be detected linearly increases. The size of the

object must also increase linearly. As the resolution support for the current architecture is 80x60 it would probably be suitable within the following situations:

- Medium complexity object inspection or object recognition at less than one metre from the lens and very simple object recognition past five metres. Typically this kind of operation would be based on the objects shape unless consistent lighting was used, which would allow the luminance of the object to be assessed.
- Measuring approximate dimensions of objects within one metre from the lens.
- Find position and orientation of a known object within one metre from the lens.
- Detect simple faults on an object. For example, a missing label off a plastic drinks bottle.
- Object tracking of moving objects at a distance of five metres and above.

An interesting use of the prototyping system, is in the detection of speeding vehicles. Current Gatso speed cameras can detect and measure car speeds of up to 160mph or  $71.52\text{ms}^{-1}$ . Gatso cameras take two photographs after measuring the speed of the vehicle using radar. These photographs can then later be used to verify the speed of the vehicle by measuring the distance travelled between the two photographs.

Hypothetically, the prototyping system could be used to replace the radar element of the Gatso. This could be achieved by determining if a vehicle has broken the speed limit and then instructing two photographs to be taken at close to road level for later speed analysis. Assuming the prototyping system's CMOS imager was directed vertically down onto a single lane of a road, two images could be taken and the approximate speed of the vehicle obtained. Using the same model as in figure 4.23 and assuming the system was attached to a 10 metre high lamppost, the maximum speed at which a vehicle could be detected is half of the values in table 4.18. This also assumes that measurements are performed from the leading edge of the vehicle. Therefore, at 10 metres and a frame rate of 12.5fps, a vehicle would escape speed assessment if it travelled at a speed greater than 154.27mph or  $68.96\text{ms}^{-1}$ . This is a

similar figure to that of the Gatso cameras. At 10 metres above the road, a car with the real-world dimensions of 4m by 2m would appear as a 29x10 pixel object.

The technical issue of the minimum size of objects detectable could be reduced by replacing the sub-sampling mechanism with a pixel averaging mechanism. The current configuration of the sensor interface could, in an unlikely event, allow an object with a very precise shape and containing 4800 holes, to be up to 302400 pixels in size and not be detected. This is because none of the 4800 sub-sampled pixels would detect a pixel from the object. Pixel averaging would prevent this happening and reduce noise but at a disadvantage of very small objects of interest not being detected.

#### 6.1.2.6 Processing Architecture Comparison with Other Systems

Undertaking a detailed comparison with other similar frame-based integrated vision systems is difficult for two reasons. Of the three frame-based architectures described in the literature review, Fang's vision chip and Neuricam's SmartPupilla are theoretical architectures. Neuricam's VISoC vision chip is the only system in current production. A comparison is further compounded by the fact that all three provide little or no performance-related information. Ideally models or samples of these systems would be required for benchmarking the developed architecture, unfortunately these are not available. Despite these issues it is still possible to perform a limited comparison between the developed system architecture and the Neuricam's VISoC architecture.

Both the developed architecture's system control unit and VISoC RISC processor have the ability to process a maximum of one million instructions per MHz. This assumes that the instruction decoder in the system control unit has been pipelined. This figure is comparable to popular 32-bit microprocessor IP cores, such as the Altera NIOS II at 1.16 MIPS/MHz, Xilinx Microblaze at 0.8 MIPS/MHz and Tensilica's Xtensa V 1.2 MIPS/MHz [113][114][115]. Both the developed architecture and VISoC benefit from embedded DSP elements. As VISoC uses a neural network and the developed architecture uses DSP IP blocks, it is difficult to

perform a suitable performance comparison. Both architectures share one distinct advantage over many of the non-frame-based architectures, which is their ability to be programmed using a higher-level language, such as C. This is only applicable when the developed architecture's DSP blocks are called using function calls from the STV0674 co-processor. As discussed previously, the function call mechanism adds latency to the execution of a DSP IP block as a result of the use of the SDRAM decoder.

It is possible that the microcontroller from the STV0674 could be integrated directly into the on-chip bus structure and hence remove some latency for DSP IP block execution. Several IP processor cores, such as ARC700, Leon 2 and Xtensa V, approach this issue by allowing their instruction set to be augmented with new instructions. These new instructions execute hardware accelerator circuitry to improve execution times for computational expensive calculations. Unfortunately, as with the developed system, this is a far more complicated task than writing an algorithm in software. Typically, the developers using the prototyping system would only design new DSP IP blocks when the gain in processing performance was required, as a result of part of an algorithm lying within the processing critical path. A further advantage of using optimised hardware DSP IP blocks, it that they typically offer more processing performance per clock cycle than most general purpose DSP or microprocessors. This could potentially lead to the processing part of an IoC having lower power requirements than a traditional DSP or microprocessor.

As is the case with most Harvard architectures, it is difficult to write self modifying programs. In fact, the current configuration of the architecture does not allow this process to take place unless initiated by the co-processor. Typically in vision application, the ability to write self modifying programs is not required.

### **6.1.3 The Complete Prototyping System**

As has been demonstrated, the prototyping system is suitable for the implementation of vision applications. It has also been shown how an application can be coded in a



small number of instructions, for example the object counting and highlight application only used 56 instructions. It must be noted that the average power consumption of 3.8W would be dramatically reduced to the region of less than 500mW if the system was integrated into an IoC.

Several limitations were found with the FPGA component of the prototyping system. The primary limitation was that of meeting timing constraints. Using the design flow optimised for timing, the configuration of the FPGA architecture used for example application had a maximum operational system clock frequency of 27.1 MHz. This was acceptable for the functional verification of the application but prevented read transactions across the IMPBUS completing correctly. This was due to insufficient time to obtain the two bytes required to construct the requested 16-bit words. Generally, the lack of read functionality was not a problem as the programming of the I-store and configuration of the registers only required IMPBUS write transactions. The addition of further DSP IP blocks could further reduce its maximum operation frequency. A possible solution to this problem could be the replacement of the Altera 20K600 FPGA to the 20K1000 or 20K1500 part. Using a FPGA with more resources may offer the opportunity to better group interconnected logic cells in IP blocks, with an aim of increasing the maximum clock frequency. The use of a larger FPGA does not necessarily guarantee an improvement in the designs maximum operation frequency, as the distance travelled by signal could increase and lead to further signal delays. This was evident during the development of the prototyping system as even the mid-size FPGA used had direct pin-to-pin delays of up to 20ns.

The FPGA design flow used, also supported Altera's LogicLock methodology. This allowed specific IP blocks to be placed in fixed locations on the FPGA floorplan. It may be possible using the methodology to obtain higher operation clock frequency but this would further complicate the design flow of application designers. It is also possible that the system architecture developed could be slightly re-designed for better FPGA-based results. Again this is not an ideal solution, as it would be better for the system design to be process independent, especially if end applications were to be integrated as an IoC.

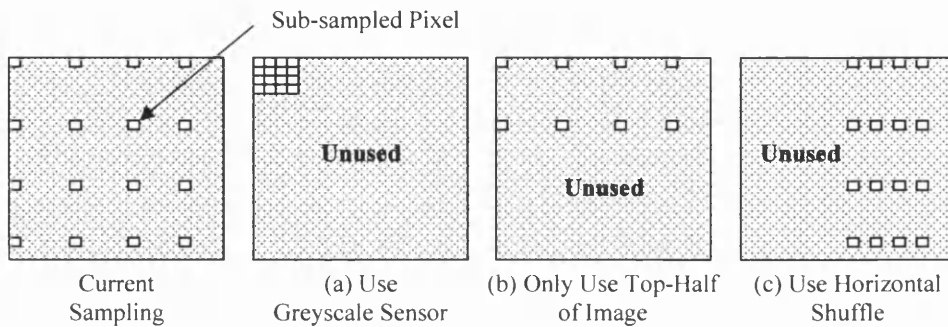
Another limitation was the lack of 50 Hz flicker detection and cancelling. As a result intermittent flickering pixels and pulsing light sources were seen during the execution of the demonstration application. Ideally, a pipelined IP block could be placed within the data path from the sensor interface to the ping-pong unit to detect flickering and aid in the cancelling-out of these effects.

The stipulation for the use of a STMicroelectronics VGA sensor during the research had an important impact on the maximum possible frame rate. The VGA sensor operated between 25 and 30fps. This provided rigid timing requirements for the sampling of images. As a result of application processing times, the system, in the majority of cases, reverted to 12.5fps because the application execution time extended beyond the available time slot at the end of each frame's sampling period. Many integrated vision chips have imagers with controllable image sampling rates which allow the applications to decide when to grab an image frame. For example, the VISoc can sample images up to 180fps, depending on scene illumination. This ability would have greatly enhanced the prototyping system, by enabling a higher frame rate in instances when an application only requires a very small amount of extra processing time at the end of each frame's sampling period. Using a sensor with a faster frame rate would have also been useful in extending the prototyping system's ability for high-speed applications. A faster frame rate would result in more time available for image processing, as the image sampling at the sensor interface would be completed in a shorter period of time.

Other methods for improving the performance of the system by providing more usable processing time slots are as follows:

- Use a greyscale image sensor, i.e. without a Bayer colour filter mask. This would enable the top left pixels in the image to be grabbed for use as the sub-sampled image. See figure 6.2a.
- Only perform processing on the top half of the images. This would allow the remaining half of the image to be discarded and the remaining frame time used for processing. See figure 6.2b.

- Use horizontal shuffle setting on a STMicroelectronics image sensor. This forces the sensor to read out all the pixels on each even column first, followed by the odd column's pixels for that line. This would result in longer inter-line times, which can be used for processing, at the expense of reducing the inter-pixel times by half. See figure 6.2c.



**Figure 6.2 Methods of reordered pixel sub-sampling for improve performance**

The comparison of the prototyping system with other similar systems is difficult. There are no known prototyping systems that cater for the specific development of IoCs. This is typically due to the fact that only a few semiconductor companies produce CMOS Imager sensors and do not publicly offer prototyping systems of this nature. These companies also do not openly offer CMOS sensors as IP blocks to third parties, probably due to the design complexities and fabrication requirements. Typically, the only method of producing an IoC is by partnering with, or contracting a semiconductor company to perform the design and fabrication processes. Any available IoC development platforms are likely to be developed in-house and their details not published due to their commercially sensitive nature.

General conclusions can be drawn about the advantages of the developed prototyping system when compared to general prototyping system. Firstly, the developed system has been optimised for vision tasks. It is relatively small and the unit cost is low. Secondly, an IP hardware and software architecture has been provided in addition to the two PCBs used. A prescribed and verified design flow has also been provided. The intellectual property rights to most of the selected board-level and all of the FPGA-level components are owned by STMicroelectronics. This will aid the integration of designs to IoC process from a business and technical standpoint.

#### 6.1.4 IoC Manufacturing Cost Modelling

The imager component of the IoC manufacturing model was validated internally using several STMicroelectronics sensor designs from the last five years. Due to the commercially sensitive nature of the figures obtained, these cannot be disclosed. The process of externally validating the model was made difficult. This was as a result of the lack of CMOS image sensors within the literature that provide information on die area and the technology process size used. Details of two image sensors were obtained. The first was the Burns and Hornsey's 80x80 CMOS image sensor in a 0.18 $\mu\text{m}$  process [121]. Using the cost model, a value was obtained within +13.6% of the correct value. The second image sensor was the Fillfactory LUPA 1300 1280 x 1024 CMOS image sensor in a 0.5 $\mu\text{m}$  process [122]. This required an extension of the cost model to take into account the scaling of the analog and digital components to a larger process. The model calculated a value within +5.9% of the correct value. As a high-level model, these figures were acceptable. Ideally, if the relevant information regarding other IoC-type systems, such as Neuricam's VISoC, could be obtained, then the whole cost model could be verified to a greater extent. It is also unlikely that the actual costs involved with the manufacture of the IoC would be provided, unless fabricated through a vendor, such as Europractice. If more time had been available, then the example system in Appendix C could have been synthesised with a suitable CMOS imager and the area results compared to the values calculated by the model.

As was previously shown, for a small IoC based on the developed FPGA architecture with an integrated 80x60 imager, integration using a 0.35 $\mu\text{m}$  process becomes financially beneficial at approximately 40 units. Retargeting the design to a 0.25 $\mu\text{m}$  process proved to only be financially viable at 50K units and for a 0.18 $\mu\text{m}$  process at approximately 500K unit. Moving to a 0.13 $\mu\text{m}$  process would require a production run of five million units. The salient reason for the very high production requirements at 0.13 $\mu\text{m}$  is the high mask set cost, the inability to easily scale or design the analog component below 0.25 $\mu\text{m}$  and the higher defect density for newer processes. At 10 millions units, the lowest unit cost is in a 0.13 $\mu\text{m}$  process. In this process, the total die area is 7.08mm<sup>2</sup>, of which 4.44mm<sup>2</sup> is the imager, and the unit cost is \$2.16, of which \$1.32 is fixed costs. As the imager's pixel array area is only

0.15mm<sup>2</sup> or 2.1% of the total die area, it may be advantageous it increase the pixel size. This increase would potentially improve the pixel sensitivity to light as a result of an increased photodiode and reduce optical related issues, such a lens assembly placement.

It may be argued that the use of a low cost 90nm-based FPGA does not justify the financial risk of IoC integration. As a result of the cost of using two external SRAMs, an FPGA with enough embedded memory would be required.

Unfortunately, embedded memory within a die is expensive in terms of area. Typically, FPGA manufacturers would rather use this die area for logic cells or embedded DSP blocks. This results in a requirement to select a FPGA that has more logic cells than required in order to obtain enough memory to implement the image banks and other memories.

The most low-cost, suitable FPGA from Xilinx, is the 90nm Spartan 3E XC3S500E with up to 360kbits of memory and from Altera it is the 90nm Cyclone II EP2C35 with up to 483kbits of memory [116][117]. Both of these FPGAs have approximately 400-500k usable gates. The Xilinx Spartan 3E FPGA costs approximately \$9 for a device upto 1.2 million gates and \$2 for a device with 100kgates in quantities of 500k units [118]. The Altera Cyclone II EP2C35 FPGA costs \$22 in quantities of 250k units [119]. As both of these FPGAs have volatile configuration storage, they require configuration devices which further add to the system cost. The requirement for a configuration device could be removed by using a non-volatile FPGA based on FLASH memory, for example the Lattice XP range of FPGAs. Despite this, separate video encoder ICs and sensor would still be required, further increasing the overall system cost. At such high volumes, the low-cost FPGAs are not a cost effective solution when compared to a fully integrated IoC. As a result, it is likely that low-cost FPGAs would only be suitable when producing up to approximately 1000 units. This figure has taken into account that FPGAs also cost more in small volumes.

Mid-ground between the fully integrated IoC and the FPGA-based multi-chip solution is that of a structured ASIC. A structured ASIC uses an array of fixed elements, similar to an FPGA, which can be used to implement multiple designs by

changing the metal layers above the silicon. This reduces the mask set costs as the manufacturing company can apportion their costs over several customers' projects. This design methodology is still in its infancy as many companies are currently trying to form viable and sustainable business models.

The use the cost model has indicated, in the case of the example, that low-cost mass-market vision systems benefit to a great extent by full integration to a single IoC. It is likely that as pixel array sizes increase and digital processes decrease in size it will become more appealing to implement more DSP functions on the same die as the imager. This has now been realised with the launch of the STMicroelectronics VS6524 highly-integrated camera module. The VS6524 integrates a VGA CMOS imager with a video processor to perform various image processing functions, such as colour space conversion, gamma correction and defect pixel correction. It has been manufactured in a 0.18 $\mu$ m ST CMOS imaging process and has been aimed at the mobile phone and PDA market. The VS6524 has a unit cost of \$6, in large volumes, for the chip, flexible connector and lens assembly [120].

## **6.2 Recommendations for Future work**

There are several recommendations regarding future improvements that could be made to the prototyping system. These are listed and separated into three categories; PCB, FPGA and System-wide related recommendations.

### PCB Recommendations

1. Redesign the daughter board PCB to include the modifications discussed in Appendix B.2 on page 209 and 6.1.1 on pages 121-122.
2. Add the option of two low-latency SRAMs to allow an upgrade path from the FPGA's embedded memory resources.
3. Add the five wire sensor data bus and clock line from the sensor to the backplane connectors. The FPGA can then be configured to accept data across this bus.

4. Perform signal timing and integrity checks to ascertain the maximum operational frequency for IMPBUS and memory bus communications.

#### FPGA Architecture Recommendations

1. Implement the seven recommendations in chapter 6.1.2.1 on pages 124-126 for improving the system controller. This will require an extra 100-200 logic cells and an ESB.
2. Add inter-general-purpose register operations. Depending on the number of operations implemented, this change would probably require in the region of 100 logic cells.
3. Extend the choice of DSP IP blocks, using the current DSP IP block addressing structures and interfaces for point and neighbourhood operations. It is particularly important to implement arithmetic and morphology DSP IP blocks. Point operations are likely to require 500-2000 logic cells per IP block and neighbourhood operations are likely to require 2000-3000 logic cells per IP block.
4. Enable selectable sub-sampling schemes as described in Chapter 6.1.3 on pages 138-139. This would make better use of available inter-line processing slots and require less than 100 extra logic cells.
5. Integrate the co-processor's 8052 microcontroller core, I<sup>2</sup>C controller and sensor configuration unit into the FPGA to remove the need for the sensor co-processor. A total of 1200-1800 logic cells would be used to implement this recommendation.
6. Implement an anti-flicker detection and cancellation unit in the FPGA architecture.

#### Recommendations for the Complete Prototyping Platform

1. Develop further applications for the prototyping system within the domain of object tracking and object recognition.
2. Synthesise the example IoC application as outlined in Chapter 6, to target a multi-project wafer process and compare results to that of the IoC manufacturing cost model's values for die area. Once fabrication is

performed, functional verification should be conducted with the IoC, using the simulations and the FPGA-based system for comparison.

3. Implement the entire DSP IP block library in a software language, such as C. This will allow functions that do not reside in the application execution's critical path, to be implemented in software on the embedded 8052 microcontroller.
4. Scale the architecture to support 25fps 640x480 pixel video processing.

To implement the fourth recommendation would only require the FPGA's architecture to be modified. The image banks would have to be scaled to support 640x480 8-bit images, dramatically increasing the number of ESBs used for each image stored from 19 to 1200. The consequences of increasing the size of both image banks are several-fold. The bus network and addressing scheme used in all DSP blocks, the network control unit, ping-pong unit and system control unit, would require extensions to support a minimum of 20-bit addressing. The number of extensions could be reduced by using the direct addressing method as described in Chapter 4.3.8 on page 89. The structure of the instruction set and SDRAM decoder would also have to be augmented to support the larger memory map. The sampling mechanism for the sensor interface and video encoder would require minor modifications. Most of the IP blocks could remain unchanged with the exception of the 3x3 filter which would require larger shift registers to accommodate lines of 640 pixels. In order to support the correct functioning of the getobjs DSP IP block, the scratch pad memory would also have to be increased significantly.

Two areas of recommended further research are as follows:

1. Investigate the possibility of prototyping parallel processing architectures, using multiple video streams or create deep multi-operation pipelines within DSP IP blocks to improve the pixel processing rate.
2. Further investigate the architectural implications of using colour images for processing. The use of colour images may be useful for skin tone detection or object recognition and segmentation by colour histogram profiling.



### **6.3 Example Re-Design of Prototyping Platform**

Using selected recommendations in sub-section 6.2, this sub-section implements a re-design of the prototyping platform. Initially, the current prototyping platform is assessed for its ability to provide support for applications using higher image resolutions. During the assessment, areas which need to be modified to support higher resolutions are highlighted and the required changes stated. The modified version of the prototyping platform is then used as a base upon which to re-design the platform to provide higher processing performance.

#### **6.3.1 Scalability of Current Platform to Higher Image Resolutions**

The CMOS image sensors provided by STMicroelectronics were all limited to a 640x480 pixels (VGA) resolution at up to 30 frames per second. With this in mind, the majority of the original prototyping platform was designed to support this resolution and frame rate. Implementing a platform to expressly support higher resolutions would have been complicated given that there would have been no simple process by which to test the platform. It was also postulated that low-cost mass-market machine vision and image processing applications would generally not require any resolution higher than 640x480 pixels, as this is approximately the effective resolution of a standard home television. Although the original platform was designed for VGA resolutions, the memory resources in the FPGA limited the useable image size to 80x60 8-bit pixels. In this sub-section, each of the FPGA-based sub-systems is analyzed to assess their suitability for use if the platform was required to support a greater image size.

Two assumptions have been made during the following analysis. The first assumption is that the system clock frequency is kept at the current frequency of 24MHz. The second assumption is that the image size required to be support is 640x480 pixels at 25fps and that the sensor used in the platform is a STMicroelectronic 8-bit greyscale VGA CMOS image sensor, i.e. without a Bayer colourisation filter mask. The lack of a Bayer filter allows every pixel to be read

from the image sensor without the need to adjust the silicon's absorption characteristics for different frequencies of light.

### 6.3.1.1 Daughter Board and FPGA Backplane

The daughter board and FPGA backplane have no issues with supporting a VGA resolution at board-level. This assumes that the sensor data is fed directly into the FPGA rather than via the sensor co-processor.

### 6.3.1.2 FPGA Memory Sub-Systems

The issue of memory in the original version of the prototyping platform limited the usable resolution to 80x60 pixels. Assuming that the same configuration of memory banks exists when larger image sizes are used and that each image bank can contain 2 image, the memory requirement (in bytes) for common resolutions are listed in table 6.5.

	<b>Image Size Supported</b>				
	<b>80x60</b>	<b>320x240</b>	<b>640x480</b>	<b>1280x1024</b>	<b>1600x1200</b>
<b>Image Bank 1</b>	9600	153600	614400	2621440	3840000
<b>Image Bank 2</b>	9600	153600	614400	2621440	3840000
<b>Scratch Pad</b>	6833	109328	437312	1749248	2733200
<b>Video Line Memory</b>	80	320	640	1280	1600
<b>I-Store</b>	256	256	256	256	256
<b>Register Bank</b>	256	256	256	256	256
<b>Async. Video FIFO</b>	144	144	144	144	144
<b>Total</b>	26753	417360	1667264	6993920	10415312

**Table 6.5 Platform's Memory Requirements (in Bytes) to Support Higher Image Sizes**

As can be seen in table 6.5 increasing the current image size of 80x60 pixels to the next common image size of 320x240 pixels, requires almost 16 times more memory. The values for the scratch pad entries have been estimated assuming that 6833 bytes

are required for each block of 80x60 pixels. The values for I-store and the register bank remain fixed as these would not be affected by scaling. The asynchronous Video FIFO is listed as 144 bytes, as its implementation is 9-bits wide and 128 elements long.

As memory addressing works on powers of two, it is complicated to implement memories with exactly the correct number of bytes require for each bank. This unfortunately leads to image banks being scaled in size to meet the requirement of the address scheme. Table 6.6 shows the actual number of bytes required for each memory bank to allow standard memory addresses to be used.

	<b>Image Size Supported</b>				
	<b>80x60</b>	<b>320x240</b>	<b>640x480</b>	<b>1280x1024</b>	<b>1600x1200</b>
<b>Image Bank 1</b>	16384	262144	1048576	4194304	4194304
<b>Image Bank 2</b>	16384	262144	1048576	4194304	4194304
<b>Scratch Pad</b>	8192	131072	524288	2097152	4194304
<b>Video Line Memory</b>	128	512	1024	2048	2048
<b>I-Store</b>	256	256	256	256	256
<b>Register Bank</b>	256	256	256	256	256
<b>Async. Video FIFO</b>	144	144	144	144	144
<b>Total</b>	41728	656512	2623104	10488448	12585600

**Table 6.6 Platform's Memory Requirements to Support Higher Image Sizes**

The total memory resources of the Altera Apex 20K600E FPGA in the prototyping platform is 38912 bytes. This would indicate that not even the 80x60 image resolution could be supported. The disparity between the number of bytes for the 80x60 pixel image resolution in table 6.6 and the maximum available for the FPGA is due to two implementation differences. Firstly, it must be noted that the video generator's line memory was implemented on the original platform using 1024 bytes to support up to 640x480 pixel images and not 128 bytes. This design decision was taken to allow the video generation sub-system to be re-used in other STMicroelectronics products. Secondly, the scratch pad memory in the original platform was implemented with 4096 bytes which was an insufficient amount of

memory for the getobjs operation, as discovered during the testing of the object counting and highlighting application in appendix C. As can be seen from table 6.6 none of the other image sizes can be supported due to a lack of FPGA memory resources, i.e. the platform's FPGA architecture is not scalable to support image sizes greater than 80x60 8-bit pixels with a half sized (4096bytes) scratch pad.

Two options are available to correct the scalability of the platform. The first option is to use an FPGA with larger memory resources. The second option is to use the daughter boards SDRAM ICs, as originally intended for applications using larger image sizes.

Two of the current FPGAs with the largest memory resources are the Xilinx Virtex 4 FX (XC4VFX140) and the Altera Stratix II (EP2S180) with 1242000 bytes and 1172880 bytes respectively [123][124]. Both FPGAs do not have sufficient memory resources for an implementation of two image banks storing two 640x480 8-bit images each. Another issue with the implementation of the larger FPGAs is that their footprints and pin configurations are different than the Apex 20K FPGA currently used. As the current FPGA foot print is a 672-pin fine-line BGA and the Xilinx Virtex 4 part is only available in 1517 or 1760 pin configurations and the Altera Stratix II in 1020 or 1508 pin configurations, the FPGA backplane would need to be completely redesigned to support either of the larger FPGAs. Using the Xilinx FPGA also compounds the problem of its implementation as its tool chain is different to the Altera part after the initial Verilog synthesis process.

The second option of implementing an embedded SDRAM controller and using the prototyping platforms SDRAM ICs for the two image banks is far more attractive from a system cost or ease of implementation standpoint when compared to the use of a larger FPGA. The specific advantages of this approach are:

- Each SDRAM IC provides 128 Mbits of memory, equating to a storage potential of up to 54 8-bit VGA images.
- Embedded FPGA memory can be re-used for a larger scratch pad memory bank or new DSP IP blocks.

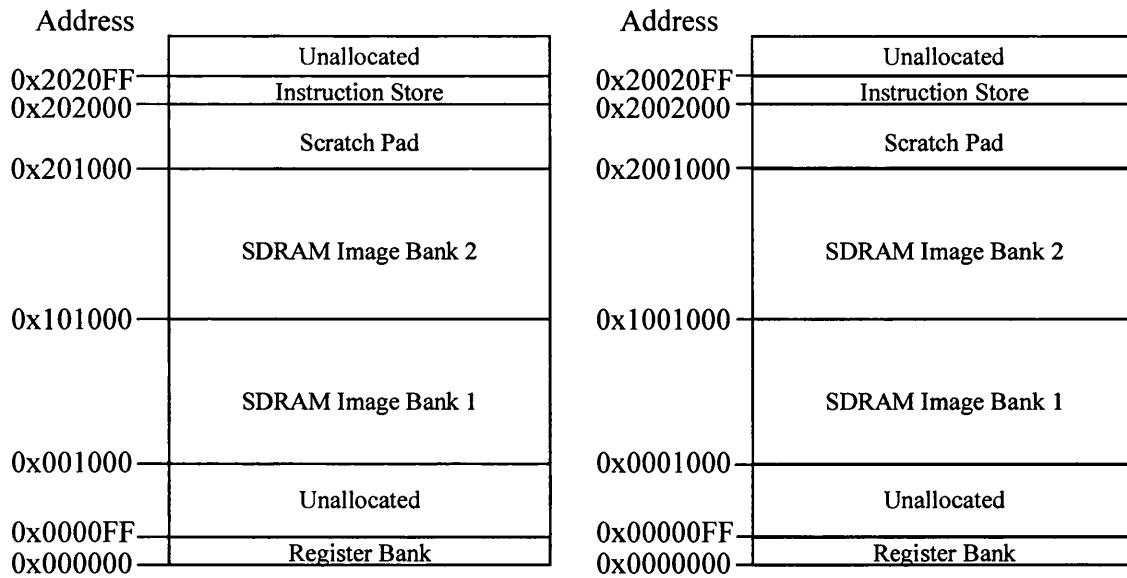
- No re-design of the FPGA backplane or daughterboard is required.
- The data ports on the SDRAM ICs are 16-bit wide and hence allow 2 pixels per cycle to be written or read.

The disadvantages of SDRAM usage are:

- The need to design and implement an SDRAM control unit to support two concurrent transactions to the two SDRAMs. This requires additional FPGA logic block resources.
- The added latency of setting-up addresses and reading data from the SDRAMs. This issue can be reduced by using burst modes which perform reads or writes on subsequent memory addresses after each memory clock cycle.
- The requirement to refresh the SDRAMs after a fixed period of time to retain the stored information correctly. During a refresh no data reads or writes can take place.

The SDRAMs have an advantage that if a prototyped design was integrated into a single IoC, embedded DRAM provides higher densities hence lower costs per Mbit, lower power consumption and lower soft error rate (SER) than embedded SRAM technologies [125].

Using the SDRAM approach with VGA image sizes affect the systems memory map. Two possible memory maps are presented in figure 6.3.



(a) Limiting SDRAM Use to Two VGA Images

(b) Full SDRAM Use

**Figure 6.3 Two different FPGA memory maps taking into account the use of off-chip SDRAM**

As the SDRAM memory configuration stores data in 16-bit words, only 24-bits are required to access their full 16777216 bytes but as the data architecture is 8-bit, an extra bit is required to select which byte of the 16-bit word should be read or written. This would require an upper/lower byte select mechanism to be implemented in the SDRAM control unit for each of the two SDRAMs.

As can be seen in figure 6.3 (column b), making the whole of each SDRAM byte addressable requires 26-bits, whereas limiting the SDRAM address space to the storage requirement of two VGA images per SDRAM, requires 22-bits. The scratch pad memory has been set as 4096 bytes. This is because implementing the required scratch pad for the getobjjs algorithm would require 524288 bytes, which is more than all the memory resources available in the Altera Apex 20K FPGA. The use of the SDRAM memories raises the issue of extending the addressing interfaces on the DSP IP block and sub-system within the FPGA. It is still more attractive to allow the full use of the SDRAM (figure 6.3 column b) as it allows the system to be scalable for use with larger image sizes in the future. Rearrangement of the memory map can yield a decrease in the bits required for addressing to 25 bits, as shown in figure 6.4.

Address	
0x2FFFFFFF	SDRAM Image Bank 2
0x2000000	
0x1000000	SDRAM Image Bank 1
0x0002FFF	
0x0002000	Unallocated
0x00010FF	Scratch Pad
0x0001000	Unallocated
0x0000FF	Instruction Store
0x00000FF	Unallocated
0x0000000	Register Bank

**Figure 6.4 Reordered FPGA memory map**

The memory map in figure 6.4 is used for the remainder of the scalability analysis of the prototyping platform.

### 6.3.1.3 Sensor Interface, Video Generator and Ping-pong Unit

Minimal changes would be required to the sensor interface as it was originally designed to support VGA image capture at 25fps. The only change required would be to remove or bypass the image sub-sampling mechanism that is used to create 80x60 pixel images. Higher resolutions than VGA probably require a faster clock rate and/or the sensor image data interface and bus width extension from 5 wires to 8 wires and thus prevent the need of two sensor clock cycle to construct each 8-bit pixel. Unfortunately, the latter would only possible by redesigning the daughter board or by direct wiring between the sensor module and FPGA input pins.

The video generator in the FPGA could also be simplified, if only VGA sized image were used, by the removal of the requirement for an image scaling mechanism from 80x60 pixels to 640x480 pixels, as the images would arrive at the video generator at the correct size, i.e. 640x480 pixels. The rate of 13.5 MHz currently used in the video generator would be sufficient to maintain a suitable pixel data rate. If images

larger than 640x480 were used by the prototyping system, a cropping or scaling-down mechanism would need to be implemented.

The effective pixel data rate from the sensor interface to the ping-pong unit and from the ping-pong to the video generator would increase by a factor of 64. As a result, at a system clock frequency of 24MHz, the ping-pong will write a pixel every two clock cycles from the sensor interface to an image bank and read a pixel every clock cycle from an image bank to the video generators async fifo to retain the correct pixel flow. This would limit the inter-pixel processing time available to one cycle, which is unlikely to be sufficient for most operations, especially read-modify-write operations over the whole VGA image. The memory access for processing from the image bank also used by the video generator would be limited to the start and end of the image fields, as the inter-pixel time available for processing would be less than one cycle. This indicates that while the ping-pong unit supports the use of VGA images at 25fps, the processing performance capable by the FPGA architecture is particularly limited. The issue of improving performance will be covered in section 6.3.2.

As a result of implementing an extended memory map, the ping-pong unit would need to be extended to support 25-bit addresses for image memory bank access. The use of the off-chip SDRAM would also necessitate the use of a delay mechanism in the ping-pong unit, also known as wait-states, to allow the ping-pong unit to take into account the latency between a read request and the return of valid data from the SDRAMs.

#### 6.3.1.4 System and Network Control Units and DSP IP Blocks

To support the extended memory map, the system control unit, network control unit and the point-to-point network structure would require the address buses and addressing interfaces to be extended to support 25-bit addressing. For the system control unit, its 16-bit external read and write address registers would also require extending to 25-bits. This would have a resultant effect of requiring the



implementation of four new instructions to load the additional 9-bits of the extended read and write address registers.

The use of the extended memory map would also affect the DSP IP blocks. Their address interfaces would have to be extended as with the other sub-systems and their counter-based addressing mechanisms extended to 25-bits. The FSMs used to generate memory addresses would also require the insertion of wait states to take into account the latency of data from the off-chip SDRAMs.

The DSP IP block specific modifications would include elongating the 3x3 filter's shift registers from 80 elements to 640 elements to hold a complete line from a VGA image. The getobjs DSP IP block would also need to be modified to use SDRAM rather than the on-chip scratch pad, or only operate on small areas of each VGA image, given that the required memory to build the object database for a VGA image is in excess of the available FPGA memory resources.

#### 6.3.1.5 Available Processing Time

The processing time available per frame helps to indicate the level of performance capable by the prototyping platform. As stated in section 6.3.1.3 image processing is limited to the timeslot after of each new frame has been written to memory. Using the recommendation in section 6.1.2.3, the frame synchronisation pulse could be adjusted to gain the 25 lines at the start of the next image frame for processing. This would increase the number of cycles available for processing per image from 35037 to 80837 cycles. As 480 lines are now sampled to construct a 640x480 image, the previously unsampled lines of 474 to 480 are now used and hence reduce the available time for processing by 7 lines. Therefore actual time available for processing is 68013 cycles, i.e. 7.08% of the frame period may be used for contiguous image processing.

Assuming a DSP IP block processes each pixel at a rate of 1 pixel every 2 cycles and only one pixel may be read or written to the SDRAM every cycle, only 11% of a

whole VGA image may be processed in any frame period to maintain a video rate of 25fps. This is insufficient performance for all but the most basic applications.

#### 6.3.1.6 Summary

The analysis of the current prototyping platform indicates that the changes required to support 640x480 pixel (VGA) images at 25fps are mainly related to the use of SDRAM, inclusion of a dual SDRAM control unit and the extension of the architecture to support a 25-bit address memory map.

The amount of available processing time per image frame has drastically reduced, due to a 64-fold increase in data throughput requirements. This effectively confines most of the image processing to the end of each image frame write from the sensor interface via the ping-pong unit. Compounding the reduction in available processing time, the amount of data to process in each VGA image has been increased by 64 times. Given that it was difficult to maintain a video rate of 25fps in applications using 80x60 pixels, it is very unlikely, except in very simple applications, that the rate will be maintainable at 25fps when processing VGA images. Therefore, although the suggested architectural changes would support VGA image based applications, the performance is likely to be a limiting factor for these applications. The issue of processing performance needs to be addressed to ensure that the prototyping system would still provide a viable platform for prototyping marketable VGA image processing based products.

#### **6.3.2 Re-designed Architecture**

As shown in the previous section, the modifications required to support 640x480 images at 25fps are relatively easy to perform. The key issue is that the image processing performance of the system would be insufficient for most applications. This section details a new improved version of the architecture based on the modifications in section 6.3.1 and the recommendations for improvements in section 6.2. Initially the top-level of the architecture is outlined and this is then followed by

sub-sections detailing the required architectural changes to each sub-system in the FPGA.

The specific aim of the new architecture is to be able to process 640x480 8-bit pixel images at a suitable level of performance to allow a wide range of developed application to maintain a frame rate of 25fps. The current STMicroelectronics CMOS VGA image sensor will be used in the new architecture.

#### 6.3.2.1 Overview of New Architecture

Although many of the recommendations in section 6.2 would improve performance, the level of performance gain would be insufficient for processing VGA images at 25fps. This is due to the limited time available for processing in each frame and that the FPGA architecture's DSP IP blocks can only perform 1 DSP operation per pixel per cycle. This limits the maximum pixel processing performance to 24 million operations per second (MOPS) at a clock frequency of 24MHz. During the synthesis of the FPGA architecture, the maximum attainable clock frequency for the complete system was found to be approximately 60MHz. Even at this maximum frequency, the FPGA architecture still does not achieve a suitable processing rate and therefore a more complex architecture solution is required.

Two possible groups of FPGA architecture recommendations could be implemented as part of the new improved FPGA architecture. The first is to improve the performance of the system controller by implementing changes 1 and 2 from the list on page 143. The second is to implement an 8051 microcontroller core and I<sup>2</sup>C control unit on the FPGA as suggested in recommendation 5. Improving the system control unit would be relatively simple and only require a low number of FPGA resources but would only yield relatively modest overall system performance increase. The best option is to implement the microcontroller for the following reasons;

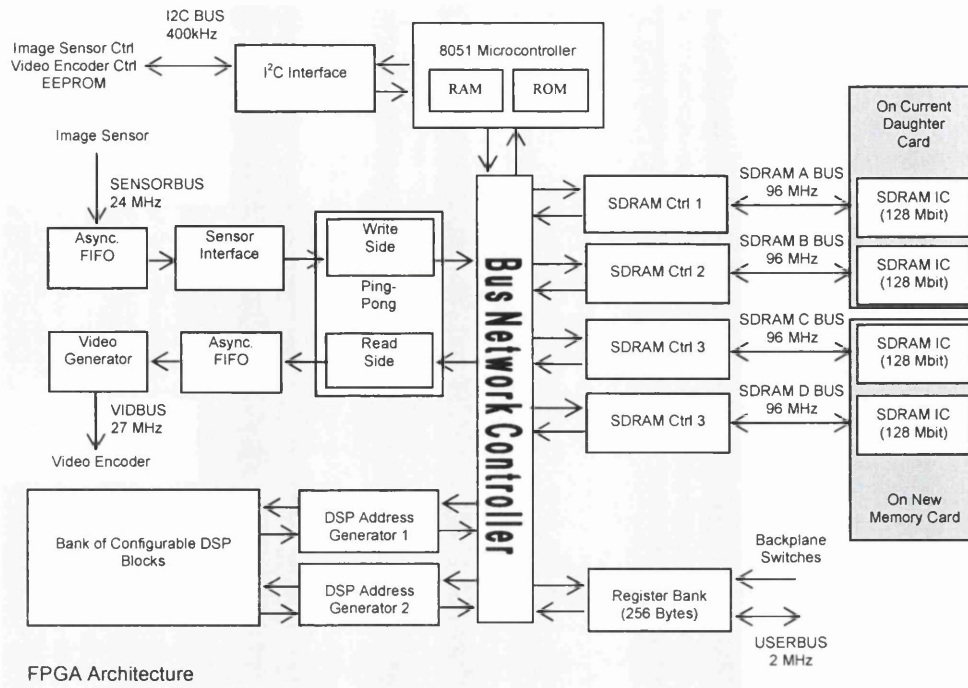
- It can perform the same functions as the system controller and hence the system controller could be removed, freeing resources for use elsewhere.

- The inclusion of the microcontroller and an I<sup>2</sup>C control unit on the FPGA also removes the need for the sensor co-processor IC, as the sensor and video encoder can be controlled by using I<sup>2</sup>C bus. This allows the removal of the SDRAM decoder unit as communications are no longer required to and from the sensor co-processor.
- It is more flexible than the system controller due to the capability to perform some image processing. This is particularly useful for implementing DSP operations that are more effectively executed in software than as a hardware IP block.
- Unlike the system control unit it can be easily programmed in C, using the current design and tool flows.

The disadvantage of the inclusion of the microcontroller is that a large number of resources are required and that there may be licensing issues.

The 4<sup>th</sup> recommendation of using configurable sub-sampling schemes to improve performance is not feasible as almost all the pixel in the pixel array are being read out to form a 640x480 pixel image rather than a small 80x60 pixel image. The 6<sup>th</sup> recommendation of implementing an anti-flicker detection and cancellation unit will not be implemented in the new architecture as it does not directly improve processing performance.

To vastly improve performance of the FPGA architecture, the recommendation for parallel processing on page 144 is used. A combination of the use of multi-operation pipelines and multiple video stream is probably the most effective method for improving performance. Figure 6.5 shows the new proposed FPGA architecture for the prototyping system.



**Figure 6.5 New proposed FPGA architecture**

Comparing the new proposed architecture in figure 6.5 to the original architecture in figure 3.4, four main differences can be noted. These are:

1. The SDRAM decoder, system controller and instruction store have been replaced with an 8051 microcontroller with a program and data memory and an I<sup>2</sup>C interface.
2. The two 16Kbyte SRAM image banks and 4KByte scratch pad have been replaced by 4 SDRAM controllers and off-chip 128Mbit SDRAMs. Two of the SDRAMs used are on the daughter board and two are on a new PCB attached to the unused pin banks on the FPGA backplane.
3. The individual DSP IP blocks have been replaced by two DSP address generators and a configurable matrix of DSP operators.
4. The bus network controller is shown to be more integrated into bus network. This is not as a result of a large change in the functionality of the bus network but as a clarification that it is connected to all the systems point-to-point buses rather than a single two-way connection as in figure 3.4.

This new system configuration is described in the next sub-sections in terms of the microcontroller sub-system, ping-pong sub-system, memory sub-system, DSP component and the bus network controller.

### 6.3.2.2 Memory Sub-system

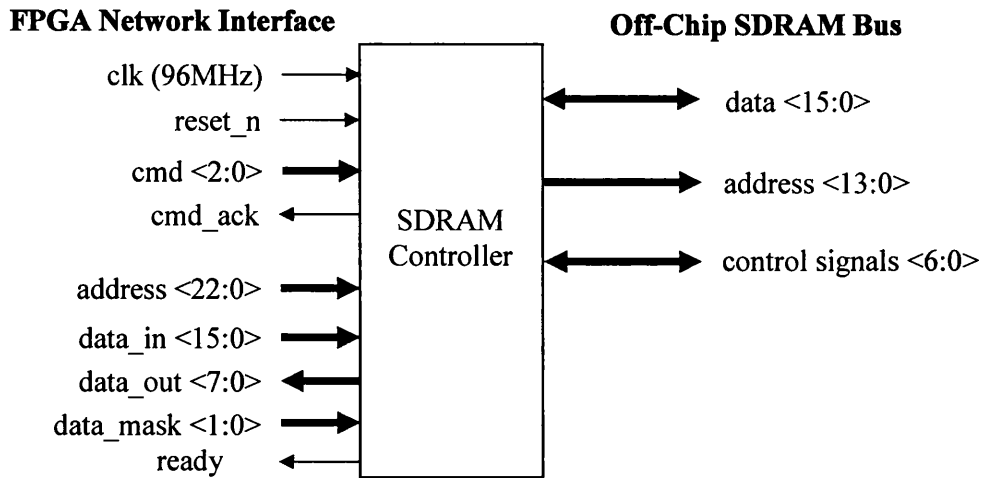
The new proposed architecture uses 4 SDRAM ICs for image banks and general storage. Two of the SDRAM ICs currently exist on the daughter board and two would need to be incorporated onto a new memory expansion board attached to the unused pin banks on the FPGA backplane. As the Apex 20K FPGAs support the use of SDRAM up to a clock frequency of 133MHz, a SDRAM clock frequency of 96MHz was chosen for the new architecture [126]. The reason this frequency was selected, was that it is a multiple of the 24MHz system clock. Using the FPGA's PLL, it is easy to output a 96MHz clock and derived in-phase clocks of 48MHz and 24MHz. At 96MHz this would provide up to a four fold increase in memory transaction speed per bank when compared to the original architecture. The use of the additional two banks also increases the memory band by an additional factor of 2. A further performance increase is gained from the ability to fetch upto 16-bits, i.e. 2 pixels, per clock cycle. The cumulative effect of these improvements results in a bandwidth increase of 16 times the original FPGA memory architecture. The most important aspect of the inclusion of two extra memory banks is that while one bank is being written by the sensor interface and another read from by the video generator, two banks may be solely used for image processing. Unfortunately, unlike the internal SRAM memories used in the original architecture, SDRAM IC require a time overhead for demultiplexing and setting up address, reading data and performing a refresh process to retain data. The effect that these overheads have on performance will be calculated in section 6.3.3.

In order to control the 4 SDRAM ICs, the new architecture requires 4 SDRAM controllers to be used. These SDRAM controllers are identical to each other and operate at 96MHz. The SDRAM controllers will be the only IP blocks in the architecture to run solely at this frequency and therefore it should be possible through careful design and floorplanning to meet the target clock frequency. It is suggested

that the Altera SDR SDRAM controller or a similar IP block be used for the new architecture [127]. The SDRAM controller should have the following characteristics and commands to obtain the most optimal performance;

1. CAS latency of 2 clock cycles.
2. 16-bit data path widths.
3. Burst length of 8 data words for DSP operations only.
4. Single data word transactions for the 8051.
5. Data masking for write operations executed from the 8-bit 8051 microcontroller.
6. Auto refresh of SDRAM to retain the integrity of stored data.
7. Auto configuration of the SDRAM's mode register for CAS latency, burst length, burst type and write burst mode.
8. NOP command (0) – No operation command.
9. READ command (1) – Starts single word read operation from 8052.
10. WRITE command (2) – Starts single word write operation from 8052.
11. BURST\_READ command (3) – Start 8 word read operation from a DSP address generator or ping-pong unit.
12. BURST\_WRITE command (4) - Start 8 word write operation from a DSP address generator or ping-pong unit.

Unlike the majority of typical SDRAM controllers, for ease the user does not have to have direct control over the sub-commands used to construct read and write memory transactions, for example refresh and precharge. The specific sub-commands, timings and the signal required to be transmitted to the SDRAM ICs may be found in the SDRAM manufacturers datasheets, for example [92]. The interface for the SDRAM controller is shown in figure 6.6.



**Figure 6.6 I/O interface for the SDRAM controllers**

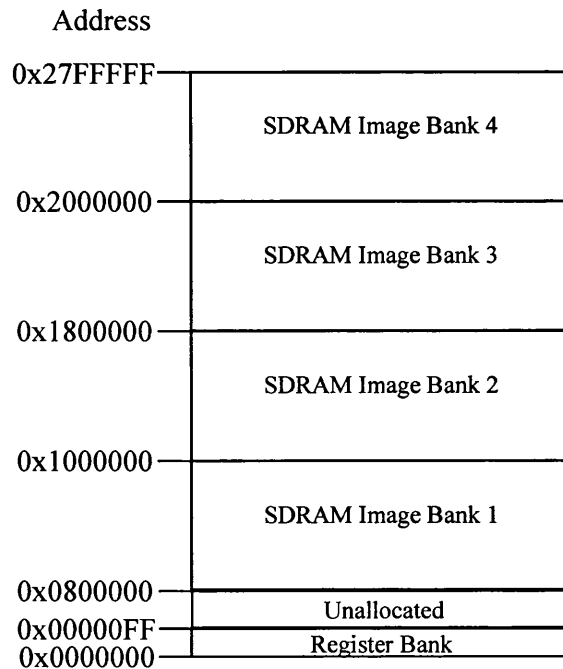
The SDRAM controller allows two types of transactions; single word or burst 8 word read and writes. When no operation is required, the cmd <2:0> is set to zero by the network bus controller. When data is set-up on the data\_in bus, an address on the address bus and a write command is issued to the controller, the controller outputs a command acknowledgement and starts the memory transactions. In the case of single word write transactions, the data set-up on the data\_in bus is sampled and outputted to the SDRAM on the next cycle after the command acknowledgement. During burst write transactions, the SDRAM controller expects 8 data words to be set-up on the bus on consecutive clock cycles. Both transactions require a further clock cycle for the automatic precharge of the SDRAM bank. Following a read command, the controller also outputs a command acknowledge signal on the following clock cycle and starts the read transaction with the address provided. Three cycles following the command acknowledgment, valid data is asserted onto the data\_out bus. In a similar manner to burst write transactions, burst read transactions output 8 words of data on consecutive clock cycles followed by an automatic precharge cycle. It must be noted that during the automatic refresh of the SDRAM the controller will not send a command acknowledgment following any command request until the refresh process is completed. The ready signal from the SDRAM controller remains low until a write transaction completes or valid read data is available, when the ready signals pulses active high. This signal is used to enable



the microcontroller to function with off-chip memories that have latency due to multiplexed addressing and read latency.

SDRAM memory transactions requested by the 8051 microcontroller only deal with 8-bit words. When an 8-bit word is outputted from the microcontroller the network bus controller duplicates the 8-bits to form a 16-bit word. This 16-bit word is then set-up on the data\_in input and a mask (data\_mask <1:0>) used to select the lower or upper byte to be written. When a read request is made, the data mask is not used by the SDRAM controller but by the network controller to select which of the upper and lower byte, read from the SDRAM, should be sent onto the microcontroller.

The memory map for the new proposed architecture is shown in figure 6.7. The reason for not including the scratch pad is to provide the most memory resources possible to the DSP IP block designer. As the microcontroller has on-chip RAM for its exclusive use and the DSP IP blocks now have access to fast SDRAM, it was no longer necessary to include additional on-chip memory.



**Figure 6.7 New proposed FPGA memory map**

The microcontroller's RAM and ROM are not represented in figure 6.7, as they are not accessible by the rest of the system. These memory blocks are covered in section

6.3.2.3. Only the microcontroller will require access to the register bank via the memory interface. The DSP IP blocks using the register bank will have direct connection to and from specific registers and thus bypass the memory interface. This will reduce the address complexity for the DSP IP blocks and associated address generators.

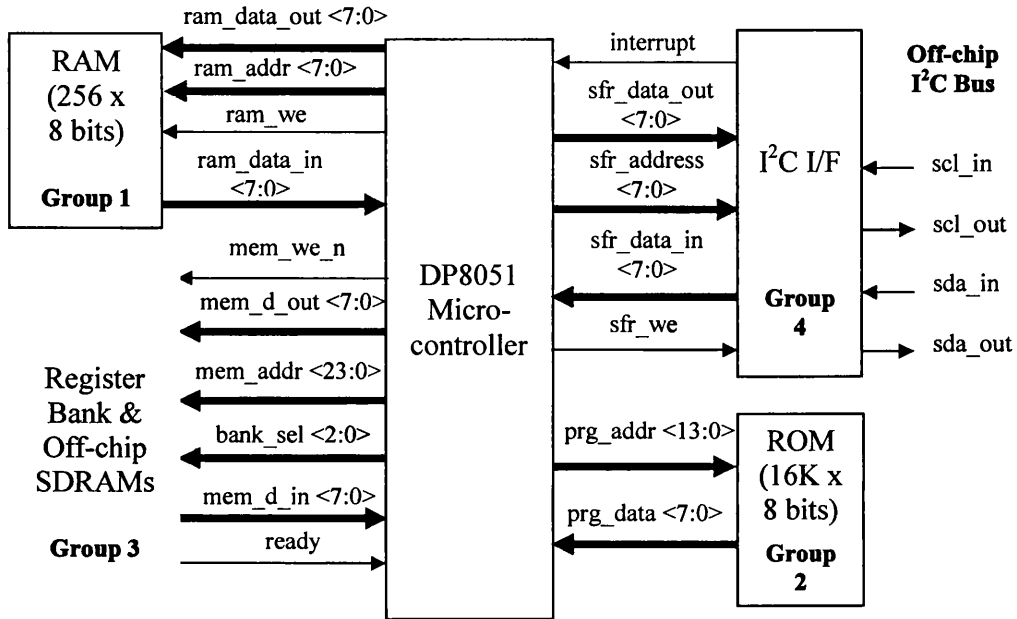
#### 6.3.2.3 Microcontroller & I<sup>2</sup>C Interface

An 8051 microcontroller has been specified for proposed architecture. The advantage of using this microcontroller core is that it is code compatible with the microcontroller in the sensor co-processor. This allows re-use of the majority of the co-processor's program code and the re-use of the current software tool chain. Other possible 8-bit processor cores that could have been included were the Z80 and Microchip PIC C165X. The 16-bit processors, such as Motorola 68000 and Intel 80186, were deemed to require too many FPGA resources.

The original Intel 8051 microcontroller, requires 12 or 24 cycles per instruction but there are many recent versions of the microcontroller that process instructions in 1-2 cycles, for example the Dolphin Integration's Flip 8051 Cyclone [128], Cast R8051 [129] and Digital Core Design's DP8051 [130]. The DP8051 was selected for the new architecture, as it provides sufficient performance for the control of the FPGAs sub-systems. Compared to the other 8051 cores available, the DP8051 has the distinct advantage of possessing three dedicated memory interfaces for up to a 64KB program ROM, a 256byte data RAM and up to 16Mbytes of addressable external memory. The ability to address 16Mbytes of external memory would allow a whole 128Mbit SDRAM bank addressable space to be accessed at any one time. Other 8051 microcontrollers, such as the Cast 8051, only allow up to 64kbytes of addressable external memory.

The IP vendor Digital Core Design specifies that the DP8051 can be implemented in an Altera APEX 20KE FPGA (as on the backplane) with clock speeds up to 68 MHz, although it will be implemented in the new architecture with a 48MHz clock [131]. The lower clock frequency should make it easier to meeting timing requirement

during place and route of the microcontroller. The DP8051 also has a compatible 400kHz I<sup>2</sup>C master interface, also available from Digital Core Design, which is shown in the proposed microcontroller sub-system in figure 6.8. Please note that only the I/O ports used within the system are shown in the diagram. The 48MHz system clock and reset have been excluded from each block component on the diagram to improve readability.

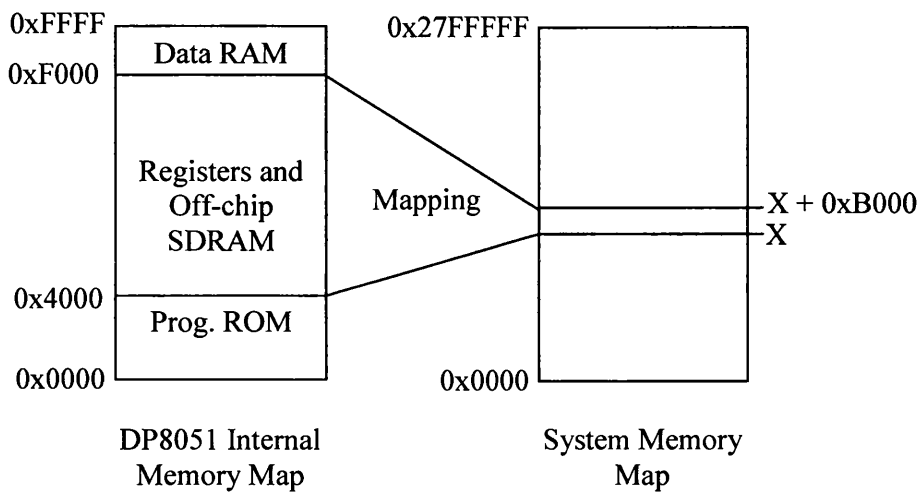


**Figure 6.8 DP8051 microcontroller sub-system**

The components and signals in figure 6.8 can be separated into 4 groups. The first grouping is the 256 byte data memory, marked RAM, and its associated memory buses. The data memory is used for storing temporary data during the execution of the microcontroller program code.

The second group involves the 16Kbyte ROM. It is expected that the ROM will be used to store program code to be executed on the microcontroller. Although the DP8051 supports up to 64Kbytes ROM, 16Kbytes was deemed sufficient for most programs, especially since the original architecture had an instruction store of only 256bytes.

The third group consists of the interface to the register bank and four off-chip SDRAMs. As the DP8051 can only access up to 128Mbits, a bank select mechanism (`bank_sel`) was implemented using a 3-bit register fed by a user controllable output port. In the microcontroller's default state, the register bank's address space is accessible. Setting `bank_sel` to 1 to 4 selected the appropriate off-chip SDRAM. Internally, the DP8051 only supports 64Kbytes and therefore despite the external memory interfacing supporting the full system memory map, a memory windowing scheme is required. The memory windowing automatically allows 45056 bytes of the external memory map to be internally addressable at any one time, in addition to the 16kbyte ROM and 256 byte RAM. The memory map for the DP8051 is shown in figure 6.9.



**Figure 6.9 DP8051 Memory Map**

As the microcontroller only reads and writes 8-bit data and not 16-bit, the LSB bit of the `mem_addr` bus is used to select the upper or lower bytes by masking using the `data_mask` signal on the SDRAM controllers during read or write transactions. Also, the read transactions from the SDRAMs involve several cycles of latency. The DP8051 accounts for this by using wait-states and reading the ready signal from the SDRAM controller in-use, to indicate when valid data is available.

The fourth group involves the I<sup>2</sup>C interface unit. DP8051's special function registers (SFR) are used to drive 8-bit data, addresses and a write enable signal (`sfr_we`) to the I<sup>2</sup>C interface unit to initialise read and write transactions on the off-chip I<sup>2</sup>C bus. The SFRs are also used to receive data from the I<sup>2</sup>C interface unit. An interrupt wire

has been included between the DP8051 and I<sup>2</sup>C interface to trigger a hardware interrupt if an off-chip device transmits unrequested data to the FPGA. This is an unlikely event as the current configuration of the EEPROM, CMOS image sensor and video encoder only allow them to function as slave devices. The advantage of using the special function registers for this purpose, is that the registers are directly addressable in the microcontroller's memory map and are therefore serves as a simple mechanism for reading and writing data to and from the I<sup>2</sup>C interface unit. For further information on the I<sup>2</sup>C bus specification, consult [132].

#### 6.3.2.4 Ping-Pong Unit

As mentioned in section 6.3.1.3, only minor modifications are required to the sensor interface and video generator. Assuming images are only read in at 640x480 pixels, the sub-sampling in the sensor interface could be removed. Assuming only full 640x480 frames images were to be transmitted to the monitor rather than scaled-up sub-VGA images, the video generators line memory could be removed as no image scaling would be required. The video generator's address generator, shown in figure 4.8, could then be simplified as it would only be required to read 320 alternate image lines per field, to create the final interlaced 640 line VGA image consisting of two fields. Both asynchronous FIFO would have to remain to allow data to be transmitted between different clock domains.

The ping-pong unit would require changes to both its read mechanism and write mechanism. These changes would be in the form of modified address generator units to support burst memory transactions. In the original architecture, the ping-pong unit had the highest level of priority over image bank, with the exception of the SDRAM decoder. This meant that under normal operation, it always had exclusive use of the image banks when required. In the new architecture, the larger image size results in the requirement for a pixel to be written every other 24MHz clock cycle and read every 1-2 clock cycles. As the time between memory access is so small, it is difficult for other system components, such as the microcontroller, to gain temporary access to the memory being used. This is generally not an issue as the other system

components are likely to be accessing the 2 (out of 4) SDRAM banks not currently being used by the ping-pong unit.

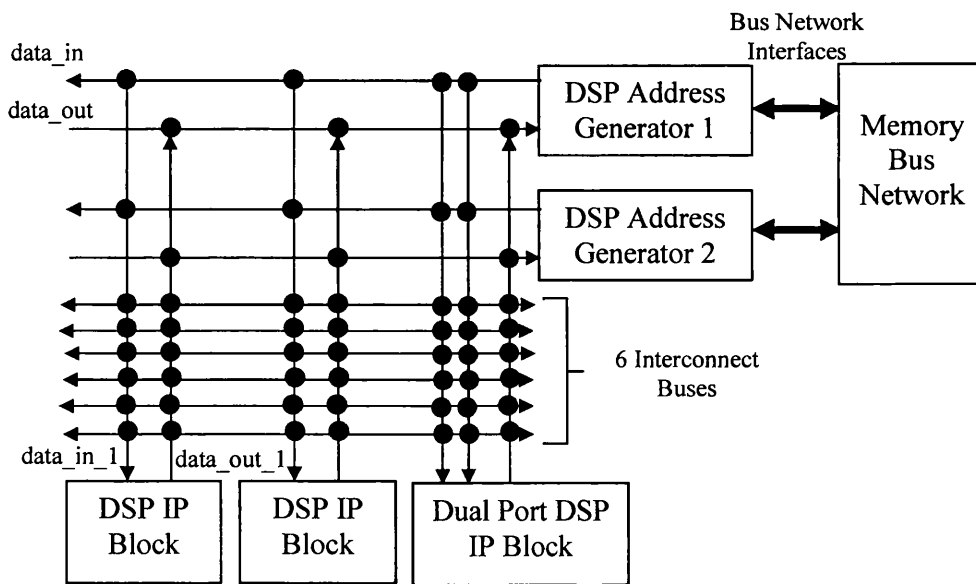
The new architecture ping-pong write unit stores pixels to be written into an image bank in a 16-bit wide 64 word deep asynchronous FIFO. The first pixel and second pixel entering the FIFO are concatenated to form a 16-bit word, with the first pixel occupying the lowest significant bits. When the FIFO is filled with at least 16 pixels, i.e. eight 16-bit words, a write request is made to the bus network controller. If access is not granted immediately because another sub-system is accessing the memory bank, the ping-pong write unit continues to write pixels into the FIFO. As all the DSP address generators and microprocessor can only read or write up to 8 words to memory in any transaction, the ping-pong's write unit is granted access quickly. Overflow is prevented from occurring because once the FIFO contains 48 words, i.e. nearly full, its priority is upgraded from lowest to the highest priority in the system allowing it to write pixel data out once any current bus network access has finished. The data written out of the write FIFO to the memory is performed at 96MHz, i.e. 8 times the pixel rate, which is also twice as fast as the FIFO can be filled.

The ping-pong's read unit also incorporates a 16-bit wide 64 elements deep asynchronous FIFO. When the FIFO is not full, a read request is made to the required memory bank via the bus network controller. All other sub-systems have a higher priority over shared memory banks than the read unit. This is unless the read unit's FIFO contains less than 16 words, in which case the read unit has the 2nd highest priority in the system. Writes into the FIFO from memory occur at 96MHz, while read from the FIFO occur at 24MHz to guarantee the required pixel rate for the video generator of 13.5MHz. In the unlikely situation when a single memory bank is used by both the write and read ping-pong units, the write unit is granted priority by the bus network controller. If the write unit was not granted priority, valuable information could be lost before being processed, causing important events to go undetected. If the FIFO for the video generator is empty, the video generator will output black pixels. This only causes a problem for the person viewing the monitor, as the processing has already occurred. It must be noted that the ping-pong read unit must re-submit its read request with an incremented address to take into account the

number of pixels that have been ‘lost’. Failing to adjust for ‘lost pixels’ would result in too many pixels being outputted in the current frame and hence skewing the image.

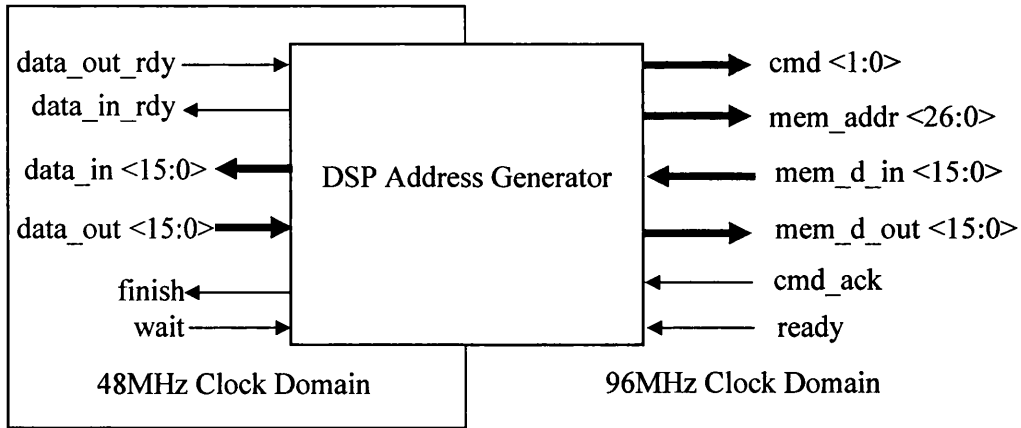
### 6.3.2.5 Configurable DSP block

In the original FPGA architecture, only one DSP operation could be performed in any given cycle. To improve the image processing performance, the DSP sub-system has been redesigned to incorporate two separate video streams and a programmable bus matrix to support the construction of pipelines of up to 7 DSP operation stages. Figure 6.10 shows the top-level of the DSP sub-system containing 3 example DSP blocks with only the datapaths shown for clarity.



**Figure 6.10 New DSP sub-system**

As can be seen in figure 6.10, the address generators have been separated from the DSP IP blocks to reduce the duplication of addressing logic. Figure 6.11 shows the interface for the address generators.



**Figure 6.11 DSP address generator I/O interface**

The address generators are activated from the register bank by setting the least significant bit high of the 8-bit `addr_gen` register to activate the first address generator or the second least significant bit high for the 2<sup>nd</sup> generator. Upon initialisation the DSP generators sample the minimum and maximum x and y coordinates from the register bank to select the region of interest (ROI) for processing and the output x and y coordinates to specify where to write the processed image data to. The first address to be read is output on the `mem_addr` port with a value of 1 on the `cmd` bus to request a read transaction. The address generator waits for an acknowledgement signal from the bus network controller to indicate that the memory is available. When the `ready` signal goes high, the address generator samples two pixels from the 16-bit incoming data bus and writes them at 96MHz to a 128 element asynchronous FIFO. The seven remaining data words of the burst read transaction are also stored in the FIFO. The next address in the ROI is then calculated. It must be noted that both address generators only generate Raster addressing as this addressing scheme is suitable for most image processing operations. The read transaction process is repeated while the FIFO contains not more than 120 words. The read process finally terminates once all the pixels from the ROI have been read.

When the FIFO is not empty and the `wait` signal not high, a data word consisting of two pixels is read from the FIFO at 48MHz outputted on the `data_in` bus in conjunction with a logic high `data_in_rdy` signal. This indicates to any attached DSP



IP block, that valid data is available for processing. While data remains in the FIFO the process repeats.

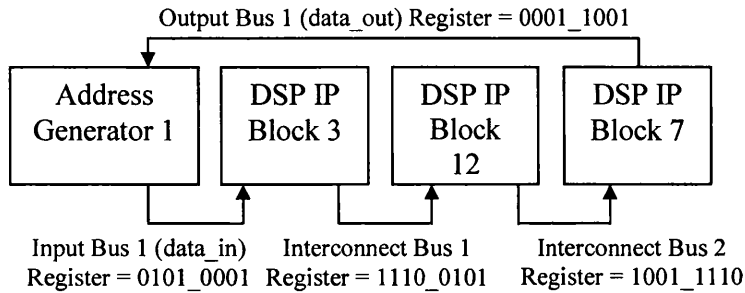
If the data\_out\_rdy is set high, the DSP address generator samples 16-bit data from the data\_out bus and writes it into a second 128 element deep asynchronous FIFO at 48MHz. When 8 or more words of data are stored in the FIFO, a burst write transaction is requested by setting the cmd signal to a value of 2 and outputting the write address. Following the receipt of a logic high cmd\_ack, the first 16-bit word is output onto mem\_d\_out on the next cycle. The remaining seven data words are outputted on subsequent 96MHz cycles. Once the last write transaction has been performed, the finish signal is pulsed to reset the appropriate bit (i.e. 1<sup>st</sup> or 2<sup>nd</sup> LSB) of the addr\_reg register in the register bank.

Between the address generators and DSP IP blocks exists a collection of 6 interconnect busses and an input and output bus per address generator. These busses have nodes that are configurable to enable connections to be made between interconnect busses and the data input (data\_in\_1) and data output (data\_out\_1) of each of the DSP IP blocks. Node configuration is performed by setting a register in the register bank for each of the 6 interconnects and the 2 input and 2 output buses. Each of the ten registers contains two 4-bit addresses. The least significant 4 bits sets the incoming connection to the associated bus and the remaining 4 bits set the outgoing connection from the bus. This mechanism provides the facility to connect up to 13 different DSP IP blocks and/or 2 DSP address generators input and output buses. Table 6.7 shows the addresses to be stored in the register bank to set-up connections between DSP IP blocks and DSP address generators using the programmable nodes.

Address	Connection
15	DSP IP Block 13
4	DSP IP Block 2
3	DSP IP Block 1
2	DSP Address Generator 2 Input or Output
1	DSP Address Generator 1 Input or Output
0	None

**Table 6.7 Interconnect Address Scheme**

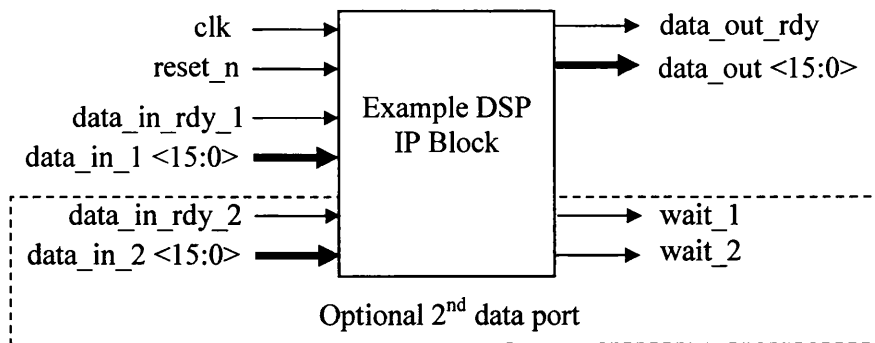
For example, if a three stage pipeline was set-up using DSP IP block 3,12,7 and address generator 1, the registers would be set as in figure 6.12.



**Figure 6.12 Example DSP pipeline configuration**

As can be seen in figure 6.12, only 4 out of the 10 node configuration registers need to be set. It must be noted that the user is responsible for ensuring that only one bus feeds each DSP IP block to prevent bus contention.

The inclusion of the separate address generators and pipeline mechanism, removes the need to incorporate an address generator into each DSP IP block. This has the effect of simplifying the I/O interfaces of DSP IP blocks in addition to reducing the resources required for their implementation. Figure 6.13 shows the I/O interface for the new DSP IP blocks.



**Figure 6.13 I/O interface for new DSP IP blocks**

The clk signal in figure 6.13 is a 48MHz clock signal and the reset\_n signal is an active low system-wide reset signal. Upon receiving an active high pulse on the data\_in\_rdy\_1 wire, the IP block samples 16-bits of data from the data\_in\_1 bus. Each DSP operation is pipelined and operates on two pixels simultaneously to

provide two pixels on the data\_out bus every 48MHz clock cycle. When each data word is outputted on data\_out, it is accompanied by a logic high pulse on data\_out\_rdy. The data\_out\_rdy signal is routed along with data\_out and also controlled by the nodes to direct it into the data\_in\_rdy input of another DSP IP block or back to an address generator.

Operations requiring two input video streams can be implemented using an optional 2<sup>nd</sup> data port. The dual port IP blocks function in a similar manner to the single port IP blocks. In a dual port configuration both data\_in\_1 and data\_in\_2 have 16 element deep synchronous FIFO mechanisms. When data arrives at either data\_in ports, it is written into the relevant FIFO. Only when at least one 16-bit data word exists in each input's FIFO, will the DSP operation take place. If either FIFO contains 8 or more data words, an active high wait signal corresponding to the correct input attached to the FIFO is sent to the address generator. This stops the address generator writing further data onto the configurable bus structure. The FIFO are 16 elements deep to allow any data currently being processing earlier on in a DSP IP block pipeline to be collected without overflowing. This mechanism ensures that two video streams entering a dual port DSP IP block remain synchronised.

The architecture adopted for the DSP sub-system has four advantages;

1. DSP operation pipelines can be constructed in real-time up to a length of 7 stages to allow 7 operations to be simultaneously executed on a video stream from memory.
2. Two video streams can be processed in parallel and hence this improves the processing rate.
3. The interconnect structure could be easily scaled by adding more interconnect buses to support even longer pipelines.
4. The number of address generators could be increased to allow more concurrent video streams to be processed, i.e. the architecture is scalable.

The disadvantages of the approach are;

1. The current configuration using two address generators only allows one dual port DSP IP block to be implemented at any given time. This reduces the flexibility of the pipeline.
2. The ROI horizontal size must be a multiple of 8 pixels as a result of the burst read mechanism used to obtain data from memory or some data received by the address generator may need to be discarded. This will reduce the performance of the system when processing many ROI with horizontal sizes of less than 8.
3. The mechanism required to set-up DSP operations is more complicated than the original architecture, due to the requirement to configure the node register in the register bank.

The third disadvantage could be minimised for the end user by writing microcontroller sub-routines to automatically connect the required DSP operations together to form pipelines followed by initialising the address generator.

#### 6.3.2.6 Bus Network Controller

The new bus network controller performs a similar function to the controller in the original architecture. All data transactions between the microcontroller, ping-pong sub-system, DSP sub-system and memories are controlled by the unit. The network controller selects which sub-system has access to a shared memory resource depending on a set of priorities. Table 6.8 shows the priorities for each of the sub-systems.

<b>Priority (1 is highest)</b>	<b>Sub-System</b>
1	Ping-Pong Write Unit (FIFO near full)
2	Ping-Pong Read Unit (FIFO near empty)
3	Microcontroller
4	Address Generator 1
4	Address Generator 2
5	Ping-Pong Write Unit
5	Ping-Pong Read Unit

**Table 6.8 Shared memory access priorities allocation**

As there are 4 SDRAM banks and a maximum of 5 possible sub-systems requiring memory access at any given time, careful algorithm design will minimise the number of memory access request conflicts. It should be noted that only the DP8051 microcontroller accesses the register bank via its memory interface and that the majority of the time the microcontroller should not need access to the 4 SDRAM banks. As the register bank does not have a controller generating command acknowledgement or valid data ready signals these are generated by the bus network controller.

In addition to granting priorities, the network controller ensures that when an address and a transaction request is received, the address is decoded and sent to the correct SDRAM controller or the register bank. The network controller also connects the datapaths between a memory and the sub-system that requested it. As the microcontroller only uses 8-bit data, read and write transactions convert 8-bit data to and from 16-bit data and use masking to ensure that the correct byte is read or written to the SDRAM.

Signals are required to be transmitted at up to 96MHz through the network controller and therefore the implementation of the bus network controller mainly consists of simple priority based multiplexing structures. Unlike the original bus network, the new network controller uses a register to pipeline all address, write data and control signals to increase the likelihood of meeting the timing requirements. Also, as the microcontroller's memory interface only operates at 48MHz, a conversion process is required to cross to and from the 96MHz memory clock domain. This is relatively simple as all transactions originating from the microcontroller only read or write a byte per transaction and not 8 data words as with the other sub-systems.

### **6.3.3 Analysis of Suggested New Architecture**

As stated in section 6.3.1 although the original architecture could be relatively easily scaled to support VGA images, the resultant DSP performance was very limited. This sub-section details the comparison of various metrics between the original

architecture and the proposed architecture in order to highlight the improvements that would result if the new architecture was implemented.

#### 6.3.3.1 DSP Performance

The original architecture only allowed the execution of one DSP IP blocks at a time. As it performed a maximum of one operation on one pixel every 24MHz clock cycle, the maximum pixel processing rate was 24Mpixels/s. The new architecture has the ability to process two video streams of 2 pixels per stream per 48MHz clock cycle. It is also possible to configure the DSP reconfigurable bus network to create a pipeline of 7 DSP IP block attached to address generator 1 and a single DSP IP block attached to address generator 2. Therefore, in this configuration pipeline 1 can process 14 pixels per clock cycle and pipeline 2 can process 2 pixels per cycle. Therefore at 48MHz the maximum DSP performance is 768Mpixel/s, an increase of 32 times of the original architecture's performance.

#### 6.3.3.2 Cycles Available for DSP

Despite the original architecture being capable of processing up to 24Mpixels/s, the major limitation of the system was as a result of the lack of time available for processing. This was caused by the requirement to use both memory banks for the ping-pong mechanism. This was addressed in the new architecture in three ways, namely by increasing the number of memory banks and the clock frequency of the SDRAM, while increasing bandwidth of communications across the bus network and through the network controller. This provided two whole memory banks dedicated to DSP functions and increased the speed at which data could be transmitted to or received from the DSP sub-system. This ensured that the DSP operation could take place at a sustainable rate of 48MHz. The communications bandwidth was improved by increasing the clock frequency of the memory sub-systems and bus network controller from 24MHz to 96MHz, while extending the datapath widths from 8-bit to 16-bits. Hence, the bandwidth was increased by up to 8 times more than the original architecture.

If the original architecture was modified as in section 6.3.1 to support VGA the number of cycles available for contiguous image processing would be 68013 per frame, as calculated in section 6.3.1.5. Using the new architecture would supply two whole SDRAM banks for processing in addition to the 68013. Therefore, as the bottleneck of the processing system is the DSP sub-system running at 48MHz, the maximum number of cycles available for processing per image frame at 25fps is calculated as in equation 6.1.

$$Cycles \ per \ frame = 68013 + 2 \left( \frac{48MHz}{25 \ fps} \right) = 3908013 \ cycles$$

**Equation 6.1 Maximum cycles available for processing per image frame**

As shown by equation 6.1, the number of cycles available for processing VGA images is increased by 57 times.

Combining the maximum pixels processed per cycle and the cycles available for processing, provides an indication of the amount of image data that can be processed. The original architecture could achieve a maximum of 68013 pixels processed every frame for single cycle read or write DSP operations and therefore a total of 1700325 pixels/s. However the new architecture could perform 8 operations on 2 pixels in each of the 3908013 cycles per frame using pipelining and hence a total of 62528208 pixels/s. Given that most processing operations require one cycle to read the pixel and one cycle to write the pixel back, read-modify-write (2 cycles) operations using the original architecture would only allow a maximum of 850162 pixels/s to be processed. To find the number of pixels that can be processed by read-modify-write operations with the new architecture, the number of cycles for a 8 word burst read and a burst write SDRAM transaction were calculated, as shown in table 6.9.

Clock Cycles Required	SDRAM Read Transaction	SDRAM Write Transaction
Network Bus Controller Latency	1	1
SDRAM Controller Latency	1	1
SDRAM Row Addressing	1	1
CAS Latency	2	n/a
Data In/Out	8	8
Total	13	11

**Table 6.9 Number of 96MHz clock cycles to perform an SDRAM Read and Write**

As shown in figure 4.4, a Precharge of the SDRAM bank is required for both reads and writes followed by a NOP command. This has not been shown in table 6.9, as these would occur during the same clock cycles as the network bus controller and SDRAM controller latency of the sub-sequent SDRAM transaction. The total number of 96MHz clock cycles for a combined read and write or 8 data words (i.e. 16 pixels) is 24 but this equates to a period equal to twelve 48MHz clock cycles. Hence, the number of 48MHz clock cycles to obtain 1 data word is  $12/8 = 1.5$  cycles. This reduces the number of pixels processed for read-modify-write operations to  $62528208/1.5 = 41685472$  pixels/s.

With an increase of pixels processable per second of over 49 times the original architecture, it is now possible to process over 135 full 8-bit 640x480 pixel (VGA) images per second. This drastically improves the system architecture for handling VGA sized images, especially as the original architecture could only handle less than 3 full VGA images per second.

### 6.3.3.3 Microcontroller Instruction Execution Speed and Code Size.

The recommendation to include a microcontroller in the new architecture removes the need for the off-chip sensor co-processor on the prototyping platform. Although in final single IoC solutions, the off-chip co-processor core would have been included automatically on the IoC, its inclusion in the new architecture will reduce the time required to integrate IoC products. Also, as the DP8051 microcontroller core is directly linked into the network bus controller, there is a decrease in the latency of data transfers to/from memory. For example, the use of the off-chip 8051 and the original architecture required a minimum of 8 cycles to read a byte from the



register bank and write a byte back. This latency was due to the use of SDRAM transactions across the IMPBUS and the SDRAM decoder. The new architecture with the embedded DP8051 only requires 5 cycles, 3 read and 2 write.

This performance increase is particularly useful as the system controller's functions have been delegated to the embedded microcontroller. The removal of the system controller simplifies the software development flow as program code can be programmed in C on the microcontroller. Indeed, many simple image processing operations may be more efficiently implemented in software as their performance in software may be very similar to their hardware IP block implementation. It could also be a possibility that the performance increase offered by DSP IP block implementation over software implementation may not be required, as the operation may not reside in the applications processing critical path. For example, the draw rectangle algorithm is better implemented in software, as it is a very simple algorithm. As the getobj DSP IP block requires 24% of the total logic cell usage in the original architecture, it is more cost effective to redesign and implement the algorithm in software. It was not possible to implement these DSP algorithms on the system controller due to its limited range of instructions.

As a replacement for the system controller, it is unlikely that the microcontroller will be any faster at decoding and executing DSP instructions. This is explained by the fact that the system controller was designed, optimised and hard coded specifically for DSP instruction execution. The microcontroller ability and handling of conditional looping, ALU operations, function calls and a clock speed of 48MHz instead of 24MHz, provides better general-purpose performance and easier coding of applications compared to when using the system controller. At 48MHz, the instruction processing speed should be up to 48MIPs, dependant on the mixture of instructions used.

In terms of code size, both the DP8051 and original system controller use byte wide instructions and literal values. Therefore the code sizes should be similar. The main difference will be seen when executing a DSP instruction. Unlike the system controller, the microcontroller will have to set the address of the correct register to activate the DSP function before executing the DSP function by toggling the register.

This will require an overhead of up to several instructions and literals rather than just one instruction used by the system controller. Fortunately, the code size of an application should not be greatly affected, given that DSP functions are usually only called several times per frame.

#### 6.3.3.4 Pin Count, FPGA Logic Cell and Memory Usage

The use of two extra SDRAM devices and an I<sup>2</sup>C interface in the new architecture affects the total number of I/O pins required. The addition of these to the architecture and the removal of the IMPBUS increases the I/O pin count to 176 I/O pins. This is an overall increase of 28.5%.

Table 6.10 shows a summary of the size of each of the architectures components

<b>Component Name</b>	<b>Number of Logic Elements</b>	<b>Number of ESB</b>
3x3Filter Block	1108	8
Register Bank	600	0
Core-level Architecture + Network control	599	0
Absdiff Block	210	0
Video Generator	183	0
Ping-Pong Unit	100	0
Sensor Interface	93	0
Video Test Pattern Generator	64	0
Top-Level Pins Structure	55	0
Threshold Block	36	0
192 MHz PLL	3	0
DP8051 Microcontroller	1750 [130]	0
I2C Controller	290 [133]	0
DSP Address Generator 1	250	0
DSP Address Generator 2	250	0
DSP Reconfigurable Node Network	200	0
SDRAM Controller 1	250	0
SDRAM Controller 2	250	0
SDRAM Controller 3	250	0
SDRAM Controller 4	250	0
DP8051 ROM (16KBytes)	16	8
DP8051 RAM (256 Bytes)	8	1
Sensor Interface FIFO	228	0
Absdiff FIFO	150	1
Ping-Pong Read Async. FIFO (128 x 16bit)	150	1
Ping-Pong Write Async. FIFO (128 x 16bit)	150	1
DSP Address Generator 1 Read Async. FIFO (64 x 16-bit)	150	1
DSP Address Generator 1 Write Async. FIFO (64 x 16-bit)	150	1
DSP Address Generator 2 Read Async. FIFO (64 x 16-bit)	150	1
DSP Address Generator 2 Write Async. FIFO (64 x 16-bit)	150	1
Video FIFO	113	1
<b>Total</b>	<b>8206</b>	<b>25</b>

**Table 6.10 Logic Element and ESB usage in the new architecture**

As can be seen in table 6.10, the total number of logic elements required has reduced from 9149 by 10.3%, to 8206 and the number of ESB required has dropped from 150 by 83.3% to 25. The decrease in logic elements is primarily as a result of the removal of the getobj DSP IP block from the architecture. The removal of the two 16KByte on-chip memory banks has resulted in a dramatic decrease in the number of

ESB required. The reduction in the logic element and ESB usage may increase the likelihood of meeting the timing requirement of the system.

The usage figures in table 6.10 for the implementation of the new architecture do not include the getobjs, getcoords, copy and rectangle DSP IP blocks for several reasons. The getobjs IP block was very large and hence very costly to implement in silicon and therefore it is suggested it should be run in software on the DP8051 microcontroller. Both the getcoords and rectangle DSP IP blocks were very simple and are more efficiently implemented in software. As the address generator could read data and write it back to a different memory location, the copy DSP IP block is redundant. Although the 3x3filter IP block uses more resources than any DSP IP block or sub-systems, it should not be ported to run in software. The reason for this decision is that it performs nine simultaneous multiplications which would not be possible on the microcontroller. This IP block is also used in a large number of applications and often may preside in its processing critical path. The values for the remaining DSP IP blocks (3x3filter, Absdiff, Threshold) have had their LE count reduced by 40LE per input data port, as an approximation of the reduction in LE, due to the removal of their address generators and a simplification of their interfaces. An approximate value of 40 LE was chosen as the original DSP IP copy block required 53 LE and it was estimated that the majority of LE were used for addressing.

The 3x3 filter block is the only DSP IP block requiring modification to support 640x480 images sizes. As 1848 logic cells are used for the implementation of three 80 element shift registers, the scaling of the architecture to VGA resolutions would increase each shift register length to 640 elements. An increase to 640 elements would require approximately 5120 logic cells per shift register, if one logic cell was used per bit [134]. If three 640 element shift registers were implemented in the FPGA, 15360 logic cells would be required. This is unacceptable given that this would represent 63.2% of the total logic cells available in the Altera 20K600 FPGA! In the current design of the 3x3filter IP block the first shift register in the chain is redundant and can be removed, feeding pixels directing into the 2 shift register and the first row of the filter multipliers. This reduces the resources used by the 2 shift registers to 42.1% of the total FPGA resources.

An alternative approach would be to implement two 1024 byte memories and address incrementing control logic. This may be far more efficient in terms of logic cells resource usage, while providing the same performance. It is possible to implement a 1024 element 8-bit line memory only using 4 ESB resources and no logic cells, as shown by the line memory requirements in tables 4.15 and 4.16. As the addresses generated for both line memories would be the same, a single address generator could be implemented. A disadvantage of this approach would be the requirement to use a memory clock speed of double the system clock speed, i.e. 96MHz. This clock frequency is required to allow both a read and a write operation to each of the line memory to take place in the period of a system clock cycle and hence maintain a pixel data rate of 48MHz. This is the approach adopted for the new architecture and results in a saving of approximately 1798 logic cells, assuming the new line memory address generator would require 50 logic cells. The implementation of the two line memories would require a total of 8 ESB blocks consisting of 2KBytes of RAM. The inclusion of the reduction in LEs as a result of the removal of the address generator would bring the total resources usage to 1108 LE and 8 ESB, as shown in table 6.10.

The register bank, core-level architecture and bus network controller are unlikely to change greatly from their original LE and ESB values as their complexity has increased at the same time as the support structures for 4 DSP IP blocks have been removed. Therefore their values have been left unchanged.

The remaining changes in LE and ESB usage were:

- The LE usage for the video generator remains the same but the scaling to 640x480 pixel images removes the need for the associated line memory.
- The ping-pong unit's LE usage has been estimated at 100 LE, almost a doubling in LE count from the original architecture, as a result of implementing a control structure for its FIFOs. The FIFOs themselves have been approximated at 150 LE and 1 ESB, each using the video generator's FIFO as a base for calculation. These figures have also been used to estimate the LE and ESB usage of the remaining FIFOs in the system.

- Using the percentage increase in the number of I/O pins as a guide, the total number of LE used by the top-level pin structure has been estimated to increase by 28% to 55LE.
- The recommendation to implement a microcontroller in the FPGA has resulted in a cost of 1750LE [130] and a further 24 LE and 9 ESB for implementing of the data RAM and program ROM. The inclusion of the I<sup>2</sup>C interface also adds a further 290 LE to the systems total usage [133].
- The addition of 4 SDRAM controllers added a further 250 LE each. This figure was obtained by using Altera's reference SDRAM controller usage figures as a base [127]. As Altera's design did not include a ready/wait-state signal, the count was increased for 218 to 250 LE to compensate for the extra circuitry required. The address generators for the DSP operations were also estimated at 250 LE.

#### 6.3.3.5 Summary

The new architecture incorporating a microcontroller and the use of two parallel processing pipelines has increased the DSP capability from 24Mpixel/s to 768Mpixel/s. The overall system performance has increased by 49 times. This has required the inclusion of an extra two SDRAM banks to provide 4 banks totalling 64Mbytes of RAM. The resultant resource usage cost of increasing the performance of the system has not been realised due to the removal of some of the DSP IP block operations and the recommendation of executing some DSP operation in software on the microcontroller.

The changes required at PCB level are minor and it is still possible to re-use the original daughter board and FPGA backplane. It is expected that the power requirements of the prototyping system will only increase as a result of the use of SDRAM.

#### 6.3.4 Colour Processing

An extension to the new architecture could be its adaptation for colour image processing. Typically, traditional machine vision applications only use the

luminance pixel data, as this is more useful for detecting edges, boundaries and other discontinuity. Using colour is difficult, as the colour of an observed object depends on the environmental lighting and reflectivity of that object. An example application where colour information is particularly useful is in skin tone detection, for example the tracking of a person's hand. It has been shown that despite human colour perception of the skin colour of different races, once intensity has been factored out the distribution of skin colour is clustered in a small area of colour-space [135]. This allows the application algorithms to detect pixels that may represent an area of skin.

The effects of implementing colour processing on the new architecture are several fold. The image data received from the sensor is formatted as luminance and two sub sampled colour difference signals (YCbCr). Every 4 bytes contains two pixels, i.e. 16bits per pixels. Storing the unprocessed data would require twice the memory requirement of luminance alone which is used in the new system architecture. If the pixel data were demultiplexed, each pixel would consist of 24bits, i.e. requiring 3 times the storage requirement of the current architecture's pixels. As the architecture has 64Mbytes of SDRAM memory available the increase in storage requirement is a relatively minor issue.

The use of colour pixels significantly affects the processing rate of DSP operation pipelines due to a three-fold increase in the size of pixel data. Its effect would be realised as a reduction in the pixel processing rate of any DSP pipeline containing at least one DSP operations using both luminance and chrominance pixel data. As a result, the maximum processing rate of 768Mpixels/s for luminance only DSP operations would fall to 256Mpixels/s.

If the pixel data was demultiplexed to 24-bits per pixel in the ping-pong unit, the frequency required to write and read pixel data to memory would increase by three times, if the widths of the data bus remained unchanged. The increase flow of data on the bus network would also reduce the slack time available for the microcontroller to access the SDRAM banks. As the microcontroller only operates on byte wide data, its performance with software algorithms for 24-bit colour processing would be further reduced.

In some instances algorithms may not use a luminance/chrominance (YCbCr) colour space and instead use Red/Green/Blue (RGB) or Hue/Saturation/Luminance (HSL) colour spaces. This could add extra latency in a DSP pipeline to perform a colour space conversion before and after performing a DSP operation, as well as increasing the number of logic cells required to implement a conversion IP block.

Overall the use of colour pixel data is likely to reduce the DSP rate of the FPGA architecture by a factor of at least three. Therefore, if possible, it is desirable to implement DSP algorithms which only require luminance pixel data.

## **6.4 Conclusions**

A new two-board prototyping platform for the development of low-cost mass-market IoC applications has been presented. Its key characteristic is its relatively low-cost, modular and flexible structure, which enables a potentially wide range of imaging and machine vision applications to be constructed from a common base. These applications could include intelligent lighting control systems and children's toys. An example application was successfully implemented to demonstrate the system's suitability as a prototyping system. The developed hardware and software core architecture and an image processing IP block library, allows application designers to concentrate on application issues. The image processing IP blocks created were also designed for use in other future STMicroelectronics products. The careful selection of board-level components should reduce the potential cost-related and time-related issues of intellectual property licensing. To calculate the cost of manufacturing a prototyped application as a single integrated circuit, a high-level IoC cost model was formulated. A total system cost was calculated for the example application, if integrated as an IoC. This revealed that the IoC could be manufactured for less than a \$10 unit cost for a quantity of 10000 units. This model highlights the need for analysis of applications to find the optimum sized embedded image sensor to meet resolution and cost requirements. A new architecture has also been proposed using recommendations from the completion of the old architecture to create a prototyping system capable of supporting image processing and machine vision applications requiring 640x480 pixel images (VGA) at a frame rate of 25fps. The new



architecture has shown that the inclusion of a microcontroller, two extra SDRAM banks, two new DSP pipelines and a reallocation of DSP operations to software can result in up to a 32 times increase in DSP performance to 768Mpixels/s. The increase in the time available for DSP processing has increased the overall system performance by up to 49 times.

This research has resulted in not only providing STMicroelectronics with a re-usable IoC prototyping platform, avoiding the potential cost of developing prototyping architecture for specific projects, but also demonstrated the suitability of frame-based architectures for use in IoCs.

## References

- [1] Machine Vision Handbook. UK Industrial Vision Association, 2001.
- [2] Braggins D. The European Machine Vision Market – a 20-year perspective. Proceedings of the Industrial Vision Day; 2004 June 2; Lyngby, Denmark.
- [3] The Machine Vision Market: 2003 Results and Forecasts to 2008. Automated Imaging Association; 2004.
- [4] ATI Radeon X800 3D Architecture White Paper. Canada: ATI Technologies Inc.; 2004.
- [5] Bohr M. High Performance Logic, Technology and Reliability Challenges [powerpoint presentation]. International Reliability Physics Symposium; 2003 Apr 1.
- [6] Xilinx Extends High-End FPGA Leadership By Shipping World's Highest Capacity & Capability FPGA. San Jose, California: Xilinx Inc.; 2003 Sept 16.
- [7] VISoc Intelligent Camera Short-Form Data Sheet. Trento, Italy: Neuricam S.p.A; 2002.
- [8] Claasen TACM. Platform design: the next paradigm shift to deal with complexity. Proceedings of the 2003 International Symposium on VLSI Technology, Systems, and Applications; 2003 Oct 6-8; Hsinchu, Taiwan. p. 8-12
- [9] Keating M, Bricaud P, Rickford RJ. Reuse Methodology Manual for System-On-A-Chip Designs. 3rd ed. Norwell, Massachusetts: Kluwer Academic Publishers; 2002.
- [10] Altizer B. Platform-Based Design: The Next Reuse Frontier. Proceedings of the Embedded Systems Conference; 2002 Mar 12-16; San Francisco, California.

- [11] Martin G. Design Flows for IP Integration: A Tutorial. Medea+ Design Automation Conference; 2003 Nov 4-6; Stuttgart, Germany.
- [12] Altera Stratix II Device Handbook. ver.1.0; Altera Corp.; 2004.
- [13] Xilinx Virtex-4 Revolutionizes Platform FPGAs White paper. San Jose, California: Xilinx Inc.; 2004.
- [14] General MPW schedule and prices 2004, Leuven, Belgium: Europractice IC Service; 2004.
- [15] Ejiri M. Robotics and Machine Vision for the Future. Proceeding of the IEEE/ASME. International Conference on Advanced Intelligent Mechatronics; 2001 July 8-12; Como, Italy. 2:917-922.
- [16] McGarry JE. An Outlook for Machine Vision. Automated Imaging Association [online]. Available at:  
<http://www.machinevisiononline.org/public/articles/articlesdetails.cfm?id=1134>,  
Accessed May 3, 2004.
- [17] Soini AJ. Machine Vision technology take-up in industrial applications. Proceeding of the 2nd IEEE International Symposium on Image and Signal Processing and Analysis; 2001 June 19-21; Pula, Croatia. p. 332–338.
- [18] Collins G., Silver B. Low-cost vision sensors. IEE Manufacturing Engineer 2001 Apr; 80(2):89-92.
- [19] The Machine Vision Market: 2002 Results and Forecasts to 2007. Automated Imaging Association, 2004.
- [20] North American Vision Market Grows 4% in 2003, Reversing 2002 Declines. Automated Imaging Association [online] 2004 Mar 5. Available at:

<http://www.machinevisiononline.org/public/articles/archive.cfm?cat=20>, Accessed May 3, 2004.

[21] Richtmyer R. Shifting fundamentals sink traditional industry forecasting methods. CNN Money 2002 Jan 8.

[22] LaPedus M. Analysts forecast IC market slowdown in '05, '06. Electronics Supply & Manufacturing [online] 2004 June 7. Available at: <http://www.my-esm.com/showArticle?articleID=21401818>. Accessed Dec 20, 2004.

[23] Clarke P. SIA sees 28.4% 2004 chip boom then two-year bust. EE Times [online] 2004 June 10. Available at: <http://www.eetuk.com/mr/news?mmyyyy=06/2004>. Accessed Dec 20, 2004.

[24] Boom-to-bust cycle looms for semiconductor industry. Silicon Strategies [online] 2003 Oct 11. Available at: <http://www.siliconstrategies.com/article/showArticle.jhtml?articleId=16100200>. Accessed Dec 20, 2004.

[25] Shelton J. Fabless Vision. Future Fab International, 2003 Feb 11; vol. 14.

[26] Roelandts W. Creating Fabless Dynasties [powerpoint presentation], Fabless Semiconductor Association, 2002 Mar 18.

[27] Fuller B. Fab costs, capacity glut seen pointing to consolidation. EE Times [online] 2003 Mar 17. Available at: <http://www.eetimes.com/article/showArticle.jhtml?articleId=18308168>. Accessed Dec 20, 2004.

[28] Chasey AD, Merchant, S. Construction Challenges for the 300 mm Fab. 10th ed. Semiconductor Fabtech; 1999. p.145-153.

[29] Fab Update: Capital Spending to Increase by 55%. Semiconductor International [online] 2004 July 1. Available at: <http://www.reed-electronics.com/semiconductor/article/CA433426>. Accessed Dec 20, 2004.

[30] Wilson A. CCD/CMOS sensors spot niche applications, *Vision Systems Design* 2003 June.

[31] SMaL Camera Technologies [online] Available at: <http://www.smalcamera.com>. Accessed Dec 20, 2004.

[32] Storm GG, Hurwitz JED, Renshaw D, Findlater KM, Henderson RK, Purcel MD. Combined Linear-Logarithmic CMOS Image Sensor. *Proceedings of the IEEE International Solid-State Circuits Conference*; 2004 Feb 15-19; San Francisco. 1:116-117.

[33] Hoefflinger B. Horizons in Vision - Vision Imaging Moves Outdoors. *Vision Systems Design* [online] 2002 Dec. Available from: [http://vsd.pennwellnet.com/Articles/Article\\_Display.cfm?Section=Archives&Subsection=Display&ARTICLE\\_ID=163617](http://vsd.pennwellnet.com/Articles/Article_Display.cfm?Section=Archives&Subsection=Display&ARTICLE_ID=163617). Accessed Sep 4, 2004.

[34] Foresti GL, Regazzoni CS. New Trends in Video Communications, Processing and Understanding in Surveillance Applications [Tutorial]. *IEEE International Conference on Image Processing*, 2001 Oct 7-10; Thessaloniki, Greece.

[35] *Observation Camera User Guide, Issue 2*, Nokia, 2003.

[36] Vellacott O. CMOS in Camera. *IEE Review* 1994 May 19; 40(3):111-114.

[37] Stevenson I, *Investigation of Imputer Architecture* [dissertation]. Edinburgh: Univ. Edinburgh; 1997.

[38] Galbraith I, Cairngorm Weather Station [online]. Available from: <http://www.phy.hw.ac.uk/resrev/aws/riming.htm>. Accessed Sep 5, 2004.

[39] Overview of Legend 500 Series SmartImage Sensors [video presentation], DVT Corporation, April 24, 2004.

[40] Product Information VCSBC11-VCSBC38. Germany: Vision Components GmbH, April 30, 2004.

[41] Product Information VCXX cameras. Germany: Vision Components GmbH, May 3, 2004.

[42] Product Information VC20XX cameras. Germany: Vision Components GmbH, May 3, 2004.

[43] Impact T10 Intelligent Camera. Eden Prairie, Minnesota: PPT Vision Inc., June 7, 2004.

[44] ZiCAM Datasheet, Rev. 2. Pulnix Inc., May 2004.

[45] OPSIS 5150ALC Vision Processing Camera System. Wintriss Engineering Corporation.

[46] Lyon RF. The optical mouse, and an architectural methodology for smart digital sensors. In proceeding of CMU Conference on VLSI Structures and Computations; Oct 1981. Rockville, Maryland; Computer Science Press; 1981.

[47] Mahowald M. Analog VLSI chip for stereo-correspondence. Proceedings of the IEEE International Symposium on Circuits and Systems; 1994 May 30-Jun 2; London, England. 6:347-350.

[48] Mead C. Adaptive retina . In: Mead C, Ismail M, editors. Analog VLSI implementation of neural systems. Boston, Massachusetts: Kluwer Academic Publishers; 1989. p. 239-246.

- [49] Bair W, Koch C. Real-time motion detection using an analog VLSI zero-crossing chip. Proceeding of the SPIE, Visual Information Processing: From Neurons to Chips; 1991. 1473:59-65.
- [50] Kobayashi H, Matsumoto T, Yagi T, Tanaka K. Light-adaptive architectures for regularization vision chips. Neural Networks; 1995. 8(1): 87-101.
- [51] Mahowald M, Delbrück T. Cooperative stereo matching using static and dynamic image features. In Mead C, Ismail M, editors. Analog VLSI implementation of neural systems. Boston, Massachusetts: Kluwer Academic Publishers; 1989. p. 213-238.
- [52] Mahowald M, Delbrück T. Cooperative stereo matching using static and dynamic image features. In Mead C, Ismail M, editors. Analog VLSI implementation of neural systems. Boston, Massachusetts: Kluwer Academic Publishers; 1989. p.171-188.
- [53] Standley DL. An object position and orientation IC with embedded imager. IEEE Journal of Solid State Circuits 1991 Dec; 26(12): 1853-1859.
- [54] Venier P, Landolt O, Debergh P, Arreguit X, Analog CMOS photosensitive array for solar illumination monitoring. Proceeding of the 43rd IEEE International Solid State Circuits Conference; 1996 Feb 8-10; San Francisco. p. 96-97.
- [55] Andreou AG, Boahen KA. A 590,000 transistor 48,000 pixel, contrast sensitive, edge enhancing, CMOS imager-silicon retina. Proceedings of the 16th Conference on Advanced Research in VLSI; 1995 Mar 27-29; Chapel Hill, Carolina. p225-240.
- [56] Andreou AG, Boahen KA. A 48,000 pixel, 590,000 transistor silicon retina in current-mode subthreshold CMOS. Proceedings of the 37th Midwest Symposium on Circuits and Systems; 1994 Aug 3-5; Lafayette, Louisiana.1:97-102.

- [57] Meitzler RC, Andreou AG, Strohhahn K, Jenkins RE. A sampled-data motion chip. Proceedings of the 36th Midwest Symposium on Circuits and Systems; 1993 Aug 6-18; Detroit, Michigan. 1:288-291.
- [58] Meitzler RC, Strohhahn K, Andreou AG. A silicon retina for 2-D position and motion computation. Proceedings of the IEEE International Symposium on Circuits and Systems; 1995 Apr 28-May 3; Seattle, Washington. 3:2096-2099.
- [59] Andreou AG. Towards “Eyes” for Sensor Network Systems [presentation]. 2004 Telluride Workshop on Neuromorphic Engineering; 2004 Jun 27-Jul 17; Telluride, Columbia.
- [60] Van der Spiegel J, Etienne-Cummings R, Nishimura M. Biologically inspired vision sensors. 23rd International Conference on Microelectronics; 2002 May 12-15; Philadelphia, Pennsylvania. 1:125-131.
- [61] Wodnicki R, Roberts GW, Levine MD. Design and Evaluation of a Log-Polar Image Sensor Fabricated Using a Standard 1.2  $\mu\text{m}$  ASIC CMOS Process. IEEE Journal of Solid-State Circuits 1997 Aug; 32(8):1274-1277.
- [62] Etienne-Cummings R, Van der Spiegel J, Mueller P, Zhang MZ. A Foveated Silicon Retina for Two-Dimensional Tracking. IEEE Transactions Circuits and Systems II: Analog and Digital Signal Processing 2000 June; 47(6):504-517.
- [63] Pardo F, Dierickx B, Scheffer D. CMOS foveated image sensor: signal scaling and small geometry effects. IEEE Transactions on Electron Devices 1997 Oct; 44(10):1731-1737.
- [64] Debusschere I, Bronckaers E, Claeys C, Kreider G, Van der Spiegel J, Sandini G et al. A retinal CCD sensor for fast 2-D shape recognition and tracking. Sensors and Actuators 1990. p. 456–460.



- [65] Lulé T, Benthien S, Keller H, Mütze F, Rieve P, Seibel K et al. Sensitivity of CMOS based Imagers and Scaling Perspectives. IEEE Transactions on Electron Devices 2000 Nov; 47(11): 2110-2122.
- [66] Anderson S, Bruce WH, Denyer PB, Renshaw D and Wang G. A Single Chip Sensor & Image Processor for Fingerprint Verification. Proceedings of the IEEE Custom Integrated Circuits Conference; 1991 May 12-15; San Diego, California. p. 12.1/1-12.1/4.
- [67] Skribanowitz J, Knobloch T, Schreiter J, Konig A. VLSI implementation of an application-specific vision chip for overtake monitoring, real time eye tracking, and automated visual inspection. Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems; 1999 Apr 7-9; Granada, Spain. p. 45-52.
- [68] Ishikawa M. 1ms VLSI vision chip system and its applications. Proceedings of the 3rd IEEE International Conference on Automatic Face and Gesture Recognition; 1998 Apr 14-16; Nara, Japan. p. 214-219
- [69] Komuro T, Ishii I, Ishikawa M. Vision chip architecture using general-purpose processing elements for 1ms vision system. Proceeding of the 4th IEEE International Workshop on Computer on Computer Architectures for Machine Perception; 1997 Oct 20-22; Como, Italy. p. 276-279.
- [70] Eklund JE, Svensson C, Astrom A. VLSI implementation of a focal plane image processor-a realization of the near-sensor image processing concept. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 1996 Sept; 4(3):322-335.
- [71] Kagami S, Komuro T, Ishii I, Ishikawa M. A real-time visual processing system using a general-purpose vision chip. Proceedings of the IEEE International Conference on Robotics and Automation; 2002 May 11-15; Washington D.C. 2:1229-1234.

[72] Ishikawa M, Komuro T. Digital vision chips and high-speed vision systems. Proceedings of the 2001 Symposium on VLSI Circuits; 2001 June 14-16; Kyoto, Japan. p.1-4.

[73] Komuro T, Kagami S, Ishikawa M. A Dynamically Reconfigurable SIMD Processor for a Vision Chip. IEEE Journal of Solid-States Circuits 2004 Jan 1; 39(1):265-268.

[74] Dudek P, Hicks PJ. A General-Purpose Vision Chip with a Processor-Per-Pixel SIMD Array. Proceedings of the European Solid State Circuits Conference; 2001 Sept 18-20; Villach, Austria. p.228-231.

[75] Dudek P. SCAMP [online]. Available from:  
<http://personalpages.umist.ac.uk/staff/p.dudek/projects/scamp/>. Accessed: December 20, 2004.

[76] Dudek P. A Processing Element for an Analogue SIMD Vision Chip. Proceedings of the European Conference on Circuit Theory and Design; 2003 Sept 1-4; Krakow, Poland. 3:221-224.

[77] Dudek P. A 39x48 General-Purpose Focal-Plane Processor Array Integrated Circuit. Proceedings of the IEEE International Symposium on Circuits and Systems, 2004 May 23-26; Vancouver, Canada. 5:449-452.

[78] Jonker PP. Why linear arrays are better image processors. Proceedings of the 12th IAPR International Conference on Pattern Recognition; 1994 Oct 9-13; Jerusalem, Israel.  
IEEE Computer Society Press, Los Alamitos, California; 1994. (3)334-338.

[79] Gokstorp M, Forchheimer R. Smart vision sensors. Proceedings of the 1998 International Conference on Image Processing, 1998 Oct 4-7; Chicago, Illinois. 1:479-482

- [80] Chen K, Danielsson PE, Astrom A. PASIC A sensor/processor array for computer vision. Proceedings of the International Conference on Application Specific Array Processors; 1990 Sept 5-7; Princeton, New Jersey. p. 352-366.
- [81] Kleihorst RP, Abbo AA, van der Avoird A, Op de Beek MJR, Sevat L, Wielage P et al. Xetal: a low-power high-performance smart camera processor. Proceedings of the 2001 IEEE International Symposium on Circuits and Systems; 2001 May 6-9; Sydney, Australia. 5:215-218.
- [82] Kleihorst R, Lee M-S, Abbo A, Cohen-Solal E. Real time skin-region detection with a single-chip digital camera. Proceedings of the IEEE International Conference Image Processing; 2001 Oct 7-10; Thessaloniki, Greece. 3:306-309.
- [83] Ni Y, Guan J. A 256×256 pixel smart CMOS image sensor for line-based stereo vision applications. IEEE Journal of Solid-State Circuits 2000 July; 35(7):1055-1061.
- [84] Fang W-C. A system-on-chip design of a low-power smart vision system. Proceedings of the 1998 IEEE Workshop on Signal Processing Systems; 1998 Oct 8-10; Cambridge, Massachusetts. p. 63-72.
- [85] Fang W-C. A smart vision system-on-a-chip design based on programmable neural processor integrated with active pixel sensor. Proceeding of the 2000 IEEE International Symposium on Circuits and Systems; 2000 May 28-31; Geneva, Switzerland. 2:128-131.
- [86] Albani L, Chiesa P, Covi D, Pedegani G, Sartori A, Vatteroni M. VISoc: a Smart Camera SoC. Proceedings of the 28th European Solid-State Circuits Conference; 2002 Sept 24-26; Florence, Italy. p. 367-370.
- [87] McBader S. On the Feasibility of Miniaturised Vision Systems. European Commission's Annals of the Marie Curie Fellowships; vol 3; 2004.

[88] Horowitz P, Hill W. The Art of Electronics. 2nd ed. Cambridge, England: Cambridge Press; 1989. p. 970-971.

[89] Aptix System Explorer Product Brief. Aptix Corp.; 2000.

[90] DMEK 6414 Datasheet. ATEME SA, 2004.

[91] STV0674 Tri-mode CMOS digital camera co-processor Datasheet. Edinburgh: STMicroelectronics Ltd.; July 2003.

[92] K4S281632D 128Mbit 2M x 16Bit x 4 Bank CMOS SDRAM Datasheet. Rev 0.1; Samsung Electronics; Sept. 2001.

[93] ULN2064B 50V – 1.5A Quad Darlington Switch Datasheet. STMicroelectronics; Sept 2003.

[94] Gonzalez RC, Woods RE. Digital Image Processing. Reading, Massachusetts: Addison-Wesley Publishing; 1993.

[95] Wolf WH. Computers as Components: Principals of Embedded Computing System Design. San Francisco, US: Morgan Kaufmann Publishers; 2001. p. 58-59.

[96] Lyon RF, Hubel PM. Eyeing the Camera: Into the Next Century. Proceedings of the 10<sup>th</sup> Color Imaging Conference; 2002 Nov 12; Scottsdale, Arizona. p. 349-355.

[97] ITU-R BT.656-5, Interfaces for Digital Component Video Signals in 525-line and 625-line Television Systems Operating at the 4:2:2 Level of Recommendation ITU-R BT.601; 1995.

[98] Jack K. Video Demystified – A Handbook for the Digital Engineer. 3<sup>rd</sup> ed. Eagle Rock, Virginia: LLH Technology Publishing; 2001. p. 94.

[99] Panda P, Dutt N, Nicolau A. Memory issues in embedded systems on chip: optimizations and exploration. Norwell, Massachusetts: Kluwer Academic Publishers; Dec 1999.

[100] The Lenna Story [online]. Available at: <http://www.lenna.org>. Accessed Feb 23, 2005.

[101] Khare J, Heineken HT, d'Abreu M. Cost Trade-Offs in System On Chip Designs. Proceedings of the 13<sup>th</sup> IEEE International Conference on VLSI Design; 2000 Jan 4-7; Calcutta, India. p. 178-184.

[102] Kahng A, Smith G. A New Design Cost Model for the 2001 ITRS. Proceeding of the IEEE International Symposium on Quality Electronic Design; 2002 March 18-21; San Jose, California. p. 190-193.

[103] 2005 IC Cost Model. Rev. 1; Georgetown, Massachusetts: IC Knowledge LLC.; 2005.

[104] OV7640 Color CMOS VGA (640x480) CAMERACHIP. Ver 2.0; OmniVision Technologies, Inc.; Feb 2004.

[105] MT9V011 - 1/4-Inch VGA Digital Image Sensor Datasheet. Micron Technology, Inc.; 2004.

[106] VS6502 VGA Color CMOS Image Sensor Module Datasheet. Edinburgh: STMicroelectronics Ltd.; July 2004.

[107] International Technology Roadmap for Semiconductors, 2003.

[108] Gate Counting Methodology for APEX 20K Devices. Application Note 110, ver.1.01; Altera Corp.; Sept 1999.

[109] Embedded Memory Generator. Dolphin Integration [online]. Available at: <http://www.dolphin.fr>. Accessed Dec 20, 2004.

[110] Mahoney J. Sticker shock for photomasks. Electronic Business Online [online]. 2003 Jan 5. Available at: <http://www.reed-electronics.com/eb-mag/index.asp?layout=article&stt=000&articleid=CA294465>, Accessed May 3, 2004.

[111] TSMC technology portfolio. Hsinchu, Taiwan: TSMC; April 2003.

[112] Integrating Product-Term Logic in APEX 20K Devices. Application Note 112, ver.1.0; Altera Corp.; April 1999.

[113] Nios II Processor Reference Handbook. ver. 1.2; Altera Corp.; Jan 2005.

[114] MicroBlaze – The Low-Cost and Flexible Processing Solution. Product brief, San Jose, California: Xilinx Inc.; 2005.

[115] Xtensa V Performance and Benchmarks [online]. Tensilica Inc., 2005. Available at: <http://www.tensilica.com/html/performance.html>. Accessed Feb 2, 2005.

[116] Spartan-3E Data Sheet. ver. 1.0; Xilinx Inc.; Mar 1, 2005.

[117] Cyclone II Device Handbook. ver. 1.2; Altera Corp.; Feb 2005.

[118] Xilinx Shatters Price/Density Barrier For Low Cost Fpgas With New Spartan-3e Family Starting At Less Than \$2.00. Press Release #0531. Monterey, California: Xilinx Inc.; Mar 1, 2005.

[119] Altera Demonstrates 90-nm Leadership as New Low-Cost Cyclone II FPGAs Begin Shipping Early. Press Release. San Jose, California: Altera Corp.; Jan 21, 2005.

[120] STMicroelectronics Delivers Best-in-Class Single-Chip VGA Camera in Tiny Module for High-Volume Mobile Applications. Press Release. Geneva, Switzerland: STMicroelectronics; Feb 9, 2005.

[121] Burns R, Hornsey R. CMOS Image Sensor With Cumulative Cross Section Readout. IEEE Workshop on CCDs and Advanced Image Sensors; 2003 May 15-17; Elmau, Germany.

[122] Lupa 1300 Datasheet. ver. 3.0. Mechelen, Belgium: Fillfactory NV; July 10, 2004.

[123] Product Selector Xilinx Virtex-4 Series FPGAs. San Jose. Xilinx Corp; Spring 2005.

[124] Altera Stratix II Device Handbook. Datasheet, ver. 3.0. San Jose: Altera Corp.; May 2005.

[125] Horikawa H. Merged-logic-type embedded DRAM suits high-performance SoCs [online]. Eetimes. 2003 Mar 17. Available from:  
[http://www.eetimes.com/in\\_focus/silicon\\_engineering/OEG20030317S0054](http://www.eetimes.com/in_focus/silicon_engineering/OEG20030317S0054).  
Accessed Aug 1, 2005.

[126] APEX 20K Device Family: External Memory [online]. Altera Corp. Available from: <http://www.altera.com/products/devices/apex/features/apx-memory.html>.  
Accessed Aug 1, 2005.

[127] SDR SDRAM Controller. White Paper, ver. 1.1. San Jose: Altera Corp.; Aug 2002.

[128] FLIP8051 Microcontroller family. Datasheet, Ver2.0, Dolphin Integration; 2002.

[129] CAST R8051 Altera Datasheet. Woodcliff Lake, NJ: CAST Inc; June 2004.

[130] DP8051CPU Pipelined High Performance 8-bit Microcontroller. Datasheet, ver 4.0, Digital Core Design; 2005.

[131] DP8051CPU Pipelined High Performance Microcontroller [online]. Digital Core Design, Jul 20 2005. Available at: <http://www.dcd.pl/acore.php?idcore=43>. Accessed Aug 1, 2005.

[132] The I<sup>2</sup>C-BUS Specification. ver 2.1, Philips; Jan 2000.

[133] DI2CM I2C Bus Interface – Master. Datasheet, ver. 3.07, Digital Core Design; 2004.

[134] lpm\_shiftreg Megafunction. Datasheet, ver 1.0. San Jose: Altera Corp.; Mar 2005.

[135] Yang J, Waibel A. A real-time face tracker. Proceedings of the 3rd IEEE Workshop on Applications of Computer Vision; 1996 Dec 2-4; Sarasota, FL. p.142-147.



## Appendix A

Port Name	SFP	Pin	Special	Bus Wire	Multiplexed signals on IMPBUS		
					MODE 1 FLASH Access	MODE 2 SDRAM Access	MODE 3 Inter-chip Communication
LCD1	3	91	CS NAND~				
LCD_3	16	65	CS_SMC~				
LCD_3	18	63	CD_SMC~				
LCD_3	19	62	WPO~				
LCD_3	21	46		IMPBUS38			IMPCOMM_MUTEX
NV_P0	23	44		IMPBUS1		DQ1	IMPCOMM_CLK
NV_P0	24	33		IMPBUS3		DQ3	IMPCOMM_PBIT
P1	25	5		IMPBUS5	I/O0	DQ5	
P1	26	4		IMPBUS7	I/O1	DQ7	
P1	27	3		IMPBUS8	I/O2	DQ8	
P1	28	2		IMPBUS10	I/O3	DQ10	
P1	29	1		IMPBUS12	I/O4	DQ12	
P1	30	100		IMPBUS14	I/O5	DQ14	
P1	31	99		IMPBUS33	I/O6	DQML	
P1	32	98		IMPBUS0	I/O7	DQ0	
P2	33	97		IMPBUS2		DQ2	IMPCOMM14
P2	34	96		IMPBUS4		DQ4	IMPCOMM15
P2	35	79		IMPBUS6	WE	DQ6	
P2	36	78		IMPBUS9	ALE	DQ9	
P2	37	77		IMPBUS11	CLE	DQ11	
P2	38	76		IMPBUS13	RB	DQ13	
P2	39	75		IMPBUS15	RE	DQ15	
P2	40	74		IMPBUS16		A0	IMPCOMM0
P3	41	73		IMPBUS17		A1	IMPCOMM1
P3	42	72		IMPBUS18		A2	IMPCOMM2
P3	43	71		IMPBUS19		A3	IMPCOMM3
P3	44	54		IMPBUS20		A4	IMPCOMM4
P3	45	53		IMPBUS21		A5	IMPCOMM5
P3	46	52		IMPBUS22		A6	IMPCOMM6
P3	47	51		IMPBUS23		A7	IMPCOMM7
P3	48	50		IMPBUS24		A8	IMPCOMM8
P4	49	49		IMPBUS25		A9	IMPCOMM9
P4	50	48		IMPBUS26		A10	IMPCOMM10
P4	51	47		IMPBUS27		A11	IMPCOMM11
P4	52	30		IMPBUS28		A12	IMPCOMM12
P4	53	29		IMPBUS29		A13	IMPCOMM13
P4	54	28		IMPBUS30		CLK	
P4	55	27		IMPBUS31		CKE	
P4	56	26		IMPBUS32		DQMH	
P5	57	25		IMPBUS34		RAS	
P5	58	24		IMPBUS35		CAS	
P5	59	23		IMPBUS36		WE	
P5	60	22	CS_SDRAM~	IMPBUS37		CS	CS

Where: ~ denotes inverted signal (i.e. active low)

CS\_ denotes Chip Select signal for FLASH, SDRAM or Smartmedia cards

CD\_ denotes Chip Detect signal for Smartmedia cards

WPO denotes Write Protect signal for Smartmedia cards

**Table A.11 IMPBUS pin mappings for the STV0674**

## **Appendix B**

This appendix describes the methodology used in the design and fabrication of the daughter board PCB. The first section outlines the complete design and fabrication flow. The following sub-sections cover the daughter board PCB's performance and any associated PCB problems.

### **B.1 Design and Fabrication Process Overview**

The process flow used in the design and fabrication of the daughter board PCB was based on the STMicroelectronics Hardware groups PCB methodology. This consisted of 4 processes; schematic entry, layout and routing, PCB manufacture and component population. These are described in the next 4 sub-sections followed by the functional test plan.

#### **B.1.1 Schematic Entry**

The first process involved the creation of the necessary design information required by an external contractor to fabricate the PCB. To generate this information, a PCB design package called Viewlogic's Viewdraw was utilised, as it was the only PCB design package available at the sponsoring company. Several design procedures were performed using this package. These stages are listed below in chronological order.

1. Creation of symbols that are not in the present component libraries. This included the clock driver, video encoder and Darlington Switch IC and the video phono connector for the video monitor.
2. Drawing of a system schematic with symbols from the symbol library.
3. Indication of PCB components or areas for text and logo labelling.
4. Production of PCB netlist containing the interconnections of all components.
5. Manual check for full component interconnection.
6. Generation of Bill of Materials (BOM), i.e. component list.

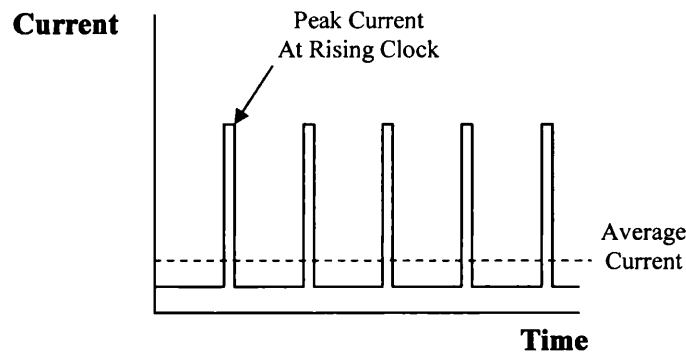
7. Outline drawing of component placement.
8. Statement of electrical specification for PCB, including power and ground planes.
9. Creation of mechanical specification for PCB

During these procedures special consideration was taken to reduce future possible problems. The two important design considerations were locality of densely interconnected board components and the separation of analog and digital ICs. These two constraints typically conflict with each other, for example, the primarily digital STV0674 sensor co-processor needs to be located next to the mainly analog sensor it supports. Power regulation and clock generation were also separated from each other. The careful floorplanning scheme used reduced the chance of crosstalk and thermal noise effects due to the heating of the power regulation IC.

A particular concern relating to the PCB floorplan was that of the length of track routing between the PCB and the FPGA backplane. Ideally the length of these tracks should be as short as possible to reduce signal transmission delays caused by resistor-capacitor loads. The three most important buses for delay minimisation were the IMPBUS and SDRAM A/B buses. An outline drawing of the PCB was produced which laid out the bus-associated component as close to the backplane connector as possible to reduce bus wire lengths and hence delays. The track delays for the SDRAM buses dictate the maximum frequency the SDRAM IC can reliably operate at. Also, if the FPGA was used to forward signals from the co-processor directly to the SDRAM and back, the number of passes through the backplane connectors could be up to 4, i.e. from 674 to FPGA, FPGA to SDRAM, SDRAM to FPGA and FPGA to 674. This could seriously hamper the system's ability to meet the co-processor SDRAM transaction timing requirements given the potentially large total transmission delay i.e. for a read delay.

The electric specification of the PCB included the separation of the signals on the top and bottom surface of the PCB and the power and ground planes on two split layers within the PCB. The main aim of this distribution was to reduce the possibility of electrical short occurring between power and ground planes which could damage the prototyping system.

Clocked digital logic consumes most of its power during the rising edge of each clock pulse, when the logic is transitioning from one state to another, see figure B.1. It is at this transition point the current supplied to the digital logic must rapidly increase in a very short period of time. To supply enough current to ensure the correct transition, a large power supply would typically be required. To reduce this problem for the daughter card PCB, low impedance decoupling capacitors were connected close to the power pins of the digital ICs. These capacitors can supply the high currents required over a short time period and the lower current requires after the transition in each clock cycle. The capacitors are recharged by the power supply during the times between the current peaks.



**Figure B.1 Peak current at rising clock edge**

As only 2 clock domains are used to drive 12 digital ICs on the daughter board and making the assumption that ICs on the same clock domain are synchronised, two very large current requirement peaks could arise. Therefore, the information regarding the correct number and capacity of capacitors required for each of these ICs were obtained from the relevant datasheet.

The mechanism specification included the description of the position for holes on the PCB and the precise locations of the 6 large backplane connections on the underside of the PCB. Information regarding the attachment of heatsinks was also included, to ensure the necessary distance from other components could be taken into account.

### **B.1.2 Layout and Routing**

The second procedure was layout and routing of the PCB and was performed by a sub-contracted company with extensive expertise in the area. An engineer from this company was assigned to the project and provided status updates during the process which included confirmation of the final component placement. The sub-contracted company used their own priority automated design rule checks (DRC) on the finished design set, followed by checks at STMicroelectronics before the sign-off process was complete. The detailed in-house checks are listed below. Unfortunately the sub-contracted company was unwilling to provide precise details on their own DRC. The final post-layout and routing design files were provided in an industry-standard Gerbers format.

Pads PowerPCB software package was used for DRC to ascertain if any errors were present in the design. Three automated checks were performed, these were;

- Clearance – Checks all items to ensure that the clearance between objects follows the clearance rules specified for the project.
- Connectivity – Details any nets that are not fully connected.
- Plane – Checks pad connections on the plane layers.

The subsequent checks on the Gerber files were visual and performed manually.

These were;

- Power and ground planes – Check for no shorts and that all used star-points.
- Components – Check for corrected component type, orientation and labelled value.
- Connector – Check for correct positioning and type
- Track width – Check for suitable track width, especially power carrying tracks.
- USB track length – Check that tracks used for the differential signal were equal in length or with USB 1.1 tolerances and met signal delay requirements.

- Bus track lengths – Check that all bus lines, in the same bus, were approximately the same length. This was required to reduce time skew between signals to ensure each data bit would be in synchronisation with the other associated data bits.
- Check correct labels and logos for the PCB, i.e. ID number, version, ST Logo and text describing the use or configuration of components.

### **B.1.3 PCB Manufacture**

Once the set of checks had been performed on the Gerber files, three quotes were obtained for the manufacture of the PCB, with the most cost favourable offer accepted. A total of 8 PCBs were fabricated as it was thought that more than one may be required in the future by the sponsoring company. The other reason for selecting 8 PCB to be manufactured was that at a size of 200mm by 165mm the PCB could only be produced 4 at a time, in what is know as a panel. The production of less than 4 PCBs per frame would not reduce the manufacturing cost to a great extent, due to wasted material and fixed cost tooling charges. The PCB was fabricated with an industrial standard hot air solder levelled (HASL) process. The PCB was specified to have 4 layers (1 power, 1 ground, 2 surface routing) using 2 lithographic silk-screens to add labelling to the top and underside of the PCB. A set of 2 resists were also used to create the top and underside tracks on the PCB.

After manufacture, all of the PCBs were checked for electrical shorts from power plans to ground and checked for shorts between component pads. A visual inspection was also made to find any faults in the PCB fabric, such as cracks, track breaks or surface grooves.

### **B.1.4 Component Population**

The fourth procedure was the automated population of the PCBs with components by a 3<sup>rd</sup> party company. Components had already been sourced from in-house or external suppliers but some were fitted in-house at a later date due to unexpected limited availability. Once the two PCBs had been received, a set of checks were performed without power being supplied to the PCB. These checks were;

1. Check that the daughter card connects correctly to the backplane.
2. Look for short circuits on the PCB between tracks or components.
3. Check all power and ground rails with multi-meter for power to ground shorts.
4. Check for broken tracks.
5. Ensure that no components are missing.
6. Check that components are not damaged, misaligned with solder pads or have the wrong orientation.
7. Check that the components populating the PCB are the correct type and value.
8. Check that all PCB connectors connect correctly to the appropriate cable connectors.

### **B.1.5 Functional Testing**

Nine function tests were created to check basic functionality of the system. There were seven prerequisites before the tests could be carried out, these were;

1. Place system in a uniformly illuminated room, if possible.
2. Install STV0674 device drivers, the G2 Video application, V2W tool, Ipatch and Camdebug on the host PC.
3. Fit an ST CMOS video sensor to board. For full functionality use a 6500 series sensor module attached to the horseshoe shaped connector, J8.
4. Attach daughter card to FPGA backplane.
5. Wire power supply to a current meter to allow the current to be measured for excessive levels that may indicated shorts or wire contentions.
6. Connect speakers to the PC that is connected to the prototyping system.
7. Attach power supply to backplane and set voltage to 7.5V with a current limit of 1A.

The nine tests, listed in the order in which they were performed, were;

1. USB enumeration test – see if the sensor and sensor co-processor is recognised by the host PC.
2. Video acquisition from sensor – check video can be streamed from the sensor via USB to a host PC and saved as a video file using G2 Video application.
3. Audio acquisition from microphone – check that audio with video can be streamed from the sensor via USB to a host PC and saved as a synchronised audio-video file.
4. Reset recovery – ensure that the system recovers correctly after a hardware reset.
5. NAND FLASH read & write – check that data is written to and read from the FLASH non-volatile memory and that the memory holds data after power-down. This test was executed using the Camdebug application.
6. Smartmedia read & write – check that data is written to and read from the Smartmedia card and that the memory holds data after power-down.
7. EEPROM code patching – ensure that new firmware patches can be downloaded to the STV0674 sensor co-processor using Ipatch.
8. Video out – ascertain if colour bars can be outputted from the video encoder by configuring its registers via the I<sup>2</sup>C bus using the V2W tool.
9. Darlington switch toggle – check that the Darlington switch ICs 4 outputs be toggled on and off.

## **B.2 Results**

The PCB was successfully completed and passed all 9 tests with only minor modifications. The system drew an approximate current of 0.5W at 7.5V, the majority of which was due to the power consumption requirements of the un-configured FPGA backplane. During the PCB design and population process, several mistakes were made by the subcontracted companies. Incorrect labelling was added to the Gerber file set, indicating that the crystal oscillators operated at Terahertz rather than the correct frequency range in Megahertz. At the time of population, the microphone and JTAG test switches on one PCB were damaged and were replaced. It was also at this stage that the 6 inter-board connectors were

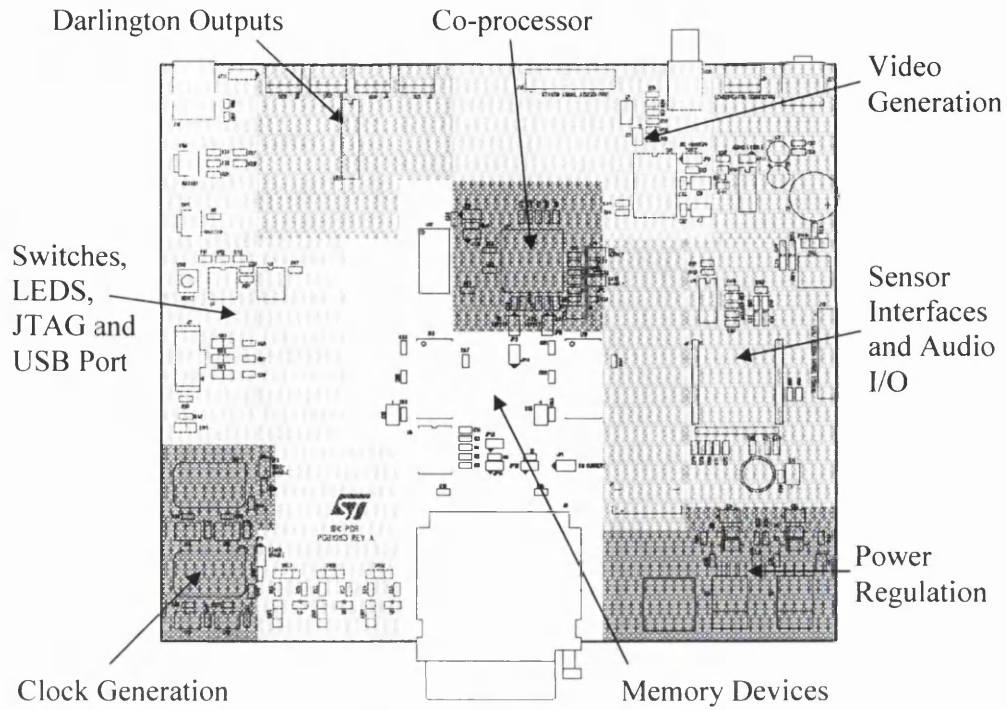


soldered onto the upper side of the PCB rather than on its underside, indicated by 6 connector footprints.

The minor post-production modifications that had to be made the prototyping system to pass the tests were. These were;

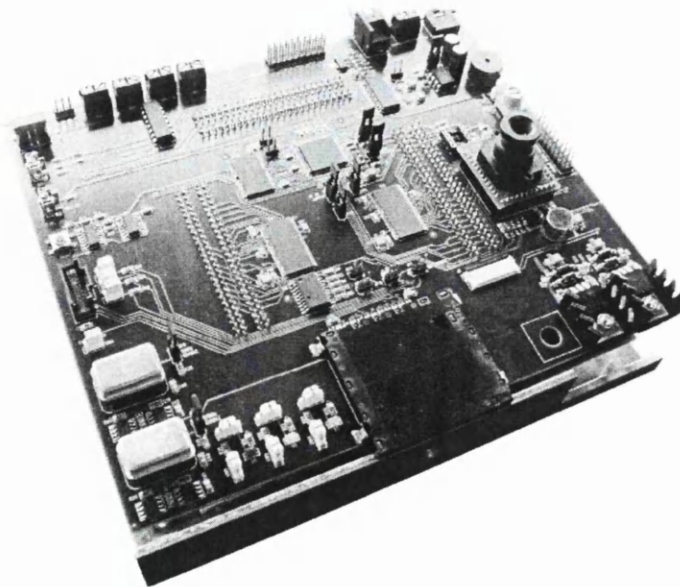
- Replacing a surface-mount pull-up  $1\text{M}\Omega$  resistor for a  $47\text{k}\Omega$  resistor, to allow the STV0674 co-processor to sense it was in a 100-pin package configuration. This change was required as a result of a change in the STV0674 co-processor's silicon i.e. at the time the PCB was being designed the co-processor was only in an early design stage.
- Changing a surface-mount  $1\text{M}\Omega$  resistor for a  $10\text{k}\Omega$  resistor to prevent the CMOS image sensor going into a permanent state of suspension.
- Adding a discrete pull-up  $10\text{k}\Omega$  resistor to the ready/busy line of the FLASH to ensure that when FLASH operations had completed the ready/busy line would go to a high logic level signally the device was ready for a new command from the co-processor.
- Redirecting the 12 MHz and 27 MHz clock signal to the output pads of the unpopulated crystal oscillator sockets on the FPGA backplane to feed the clock signals directly into the FPGA clock inputs. This was required because although the 'FAST' input pins selected had been indicated in the datasheet as suitable for clock signals, the FPGA's on-chip PLL would not output any frequency higher than the 'FAST' inputs frequency.
- Spacer connectors were connected to the inter-board connector to raise the height of the PCB above the FPGA backplane to prevent backplane components touch the underside of the daughter board PCB.

All user-definable SPF pin were utilised on the STV0674 sensor co-processor to provide the maximum flexibility for the system. The PCB sub-systems were mostly separated from each other, reducing possible crosstalk effects. Figure B.2 shows the areas occupied by different sub systems.



**Figure B.2 Sub system locations on topside of PCB**

Figure B.3 is a photograph of the completed PCB, populated with all the components, and connected to the FPGA backplane using the 6 inter-board connectors.



**Figure B.3 A photograph of the complete PCB attached to the FPGA backplane**

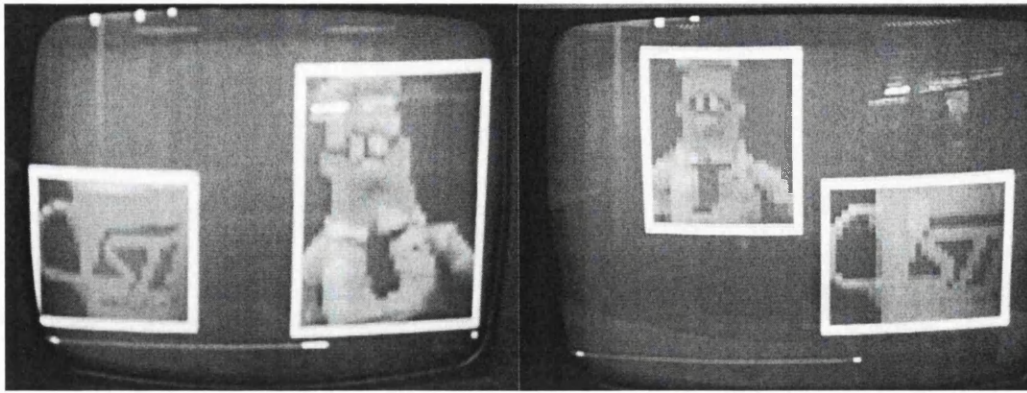
### **B.3 Summary**

This appendix has presented on the first part of the project's practical work. The method for the development of the board-level architecture of the prototyping system has been described. The results for the system have been given in the form of the problems encountered, modifications required and the set of tests successfully completed. A photograph of the complete PCB has also been provided to give a better indication of the completed construction of the prototyping system.

## Appendix C

An example application was implemented on the prototyping system to count the number of objects in a scene, display the number of objects on the FPGA backplane's LEDs and highlight them on a video monitor using superimposed rectangles. To achieve this, five of the seven DSP IP blocks were used in conjunction with application code written for the system control unit and the default system configuration code executed by the microcontroller in the sensor co-processor. The application code initially grabbed a reference image which it absolute differenced with subsequent images, to find inter-frame pixel value changes. The differenced image was then thresholded, to remove small changes due to changes in scene illumination and create a binary image. A search was performed on the binary image to find the collection of interconnected pixels making up each object in the scene. The parameters of each object were stored in the Scratch pad memory and then used to draw a rectangle around each object. A very simple wire assignment change to the Verilog design also allowed the number of objects found to be displayed on the bank of 10 LEDs on the backplane in a binary format.

The application was simulated and then implemented on the prototyping system with a system clock frequency of 24 MHz. The complete two board prototyping system had an average power consumption of 3.8W at a sustained frame rate of 12.5FPS. A total of 74 pins were used on the FPGA. The application code size for the system controller was 87bytes, comprising of 56 instructions and 31 associated literal value. The minimum execution time of the application was measured at 2.018ms (48442 clock cycles). The maximum execution time was measured at 3.858ms (92592 clock cycles) with the standard 4Kbyte sized scratch pad memory and 4.918ms (118051 clock cycles) with a simulated 8KByte scratch pad. Figure C.1 shows two photographs of the video monitor displaying two objects that had been identified by the system.



**Figure C.1 Screenshots of a coffee mug and Dilbert toy detected and highlighted by the system**

As a result of office lighting and the reflectivity of the monitor screen's surface, some strong reflections are visible. The 50 Hz refresh rate of the office fluorescent strip lighting caused visible flickering pixels when pointing the system sensor at ceiling light fittings or highly reflective surfaces. Dilbert's black trousers, which were within 10 pixel values of the background in the reference image, were thresholded out and therefore not recognised by the system. The system control unit's code for the demonstration application is shown in table C.1.

	Instructions		Comments
0	LOAD_REG_ADDR_WR_HI	0x00	
2	LOAD_REG_ADDR_WR_LO	0x1b	Set draw rectangle to memory 1
4	LOAD_MEM	0x01	
6	LOAD_REG_ADDR_WR_LO	0x15	Set absdiff to read memory 1
8	LOAD_MEM	0x01	
10	LOAD_REG_ADDR_WR_LO	0x0a	Set threshold value
12	LOAD_MEM	0x0d	
14	WAIT_EOF		Grab reference image
15	WAIT_EOF		(*)
16	START_PINGPONG		Start video ping-pong (p-p)
17	LOAD_REG_ADDR_WR_LO	0x07	Set p-p ctrl input/outputs
19	LOAD_MEM	0x0c	
21	WAIT_EOF		
22	STOP_PINGPONG		Stop ping-pong
23	ABSDIFF		Absolute Difference reference and current image
24	WAIT_BUSY		
25	THRESHOLD		Threshold
26	WAIT_BUSY		
27	LOAD_REG_ADDR_RD_LO	0x1a	
29	NOOP		
30	LOAD_REG_ADDR_RD_HI	0x00	
32	NOOP		
33	LOAD_REG_ADDR_WR_LO	0x0e	
35	GETOBS		Find objects in image
36	WAIT_BUSY		
37	MOVE_MEMREG_GPRO		Copy # of objects found to reg_0
38	BEZ	0x19	If # of object in reg_0 == 0 branch
40	LOAD_REG_ADDR_RD_LO	0x02	Set read address register to first object in scratch pad database
42	NOOP		
43	LOAD_REG_ADDR_RD_HI	0x50	
45	MOVE_MEMMEM		Copy min X of object to regs
46	ADD_REG_ADDR_RD	0x01	
48	ADD_REG_ADDR_WR	0x01	
50	MOVE_MEMMEM		Copy min Y of object to regs
51	ADD_REG_ADDR_RD	0x01	
53	ADD_REG_ADDR_WR	0x01	
55	MOVE_MEMMEM		Copy max X of object to regs
56	ADD_REG_ADDR_RD	0x01	
58	ADD_REG_ADDR_WR	0x01	
60	MOVE_MEMMEM		Copy max Y of object to regs
61	RECTANGLE		Draw Rectangle
62	WAIT_BUSY		
63	WAIT_BUSY		
64	ADD_REG_ADDR_RD	0x05	Inc. read address register by 5
66	SUB_REG_ADDR_WR	0x03	Dec. write address register by 3
68	SUB_REG_GPRO	0x01	Dec. number of objects by 1
70	LOAD_REG_ADDR_WR_LO	0x0e	
72	BNEZ	0x1f	Branch if no more object to highlight
74	LOAD_REG_GPRO	0x00	Reset reg_0 to zero
76	LOAD_REG_ADDR_WR_HI	0x00	Configure copy function
78	NOOP		
79	LOAD_REG_ADDR_WR_LO	0x17	
81	LOAD_MEM	0x04	
83	COPY		Copy reference image to current image
84	WAIT_BUSY		
85	BEZ	0x48	Unconditional branch to (*)

**Table C.1 Application code for object count and highlight demonstration**

