Vision Hardware for a Humanoid Robot

Ho Wong s33544119

16th October 2002

ACKNOLEDGEMENTS

Gordon Wyeth for providing the thesis topic.

David Prasser for all his help and understanding of what I went through this year.
Mark Chang for his help on understanding the SH4, camera and FPGA.

All the people on the vhdl newsgroup (comp.lang.vhdl) for helping me with my vhdl problems.

And all my friends who’ve encouraged me throughout the year when things looked the bleakest.

ABSTRACT

The University of Queensland has entered a humanoid robot (GuRoo) for Robocup's Humanoid Soccer League. GuRoo was first conceived in 2001 and its design was undertaken by 12 undergraduates.

The objective of this thesis is to take what's been previously done, combine them to make a vision system for the GuRoo and then make improvements. Objects on the playing field are colour-coded hence only 8 colours are approximately needed. The on-board memory on the vision board is only 512Kbytes and is barely sufficient to store a 240x180 (approximately 8cm x 6 cm) image. It is mainly because the image is stored as raw data. This can be greatly improved by moving the Colour Lookup onto the FPGA (Field Programmable Gate Array). Unfortunately, the FPGA has only 100,000 system gates and 10 x 4Kbytes of Block Ram. This means only a few pixels can be processed on the FPGA and a new Lookup Table has to be developed because a 256 x 256 Lookup Table (UV only) will exceed the available space. Timing is also critical because each pixel has to be processed by the time a new pixel is sent from the camera.

The greatest challenge this thesis faced was setting up of the SH4 and camera. The camera board and workings of the code were poorly documented and it was not until mid-year that all the setup needed for the thesis was completed. This left little time to pursue the improvements this thesis intended.

Only the Colour Lookup has been moved onto the FPGA. The time it takes to read a pixel's colour from the FPGA is ≈ 900ns and to do a colour lookup for a 240x180 image is ≈ 19.44ms. By removing the raw image from the SH4, it is now possible to store a 640x480 image or two 360x 270 images.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Thesis Objective

The objective of this thesis is to take what's been previously done, combine them to make a vision system for the GuRoo and then make improvements.

1.2 The GuRoo[1] Project

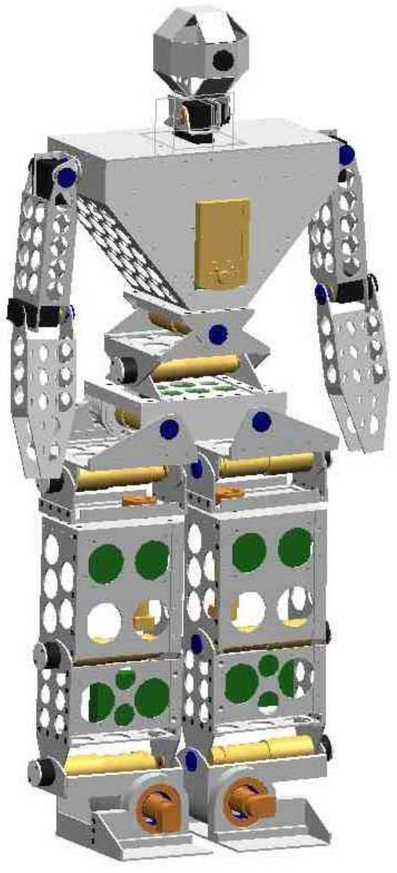
Due to the success of the RoboRoos (The University of Queensland's Small Sized Global Vision Robocup soccer team), the GuRoo was started in 2001 by a team of 12 undergraduate students to enter the Robocup2001 Humanoid Soccer League. The GuRoo is in the medium sized division (1.2m) although for Robocup2002, the size divisions were combined.

The difficulty of building a robot that can not only stand by itself, but move and decide its own actions based on what it sees and knows is unfathomable. Beyond the simple desire to play soccer with other robots, the GuRoo is a test bench for future University Of Queensland students to test new ideas on creating a humanoid that could eventually have the full capabilities of a real human being.

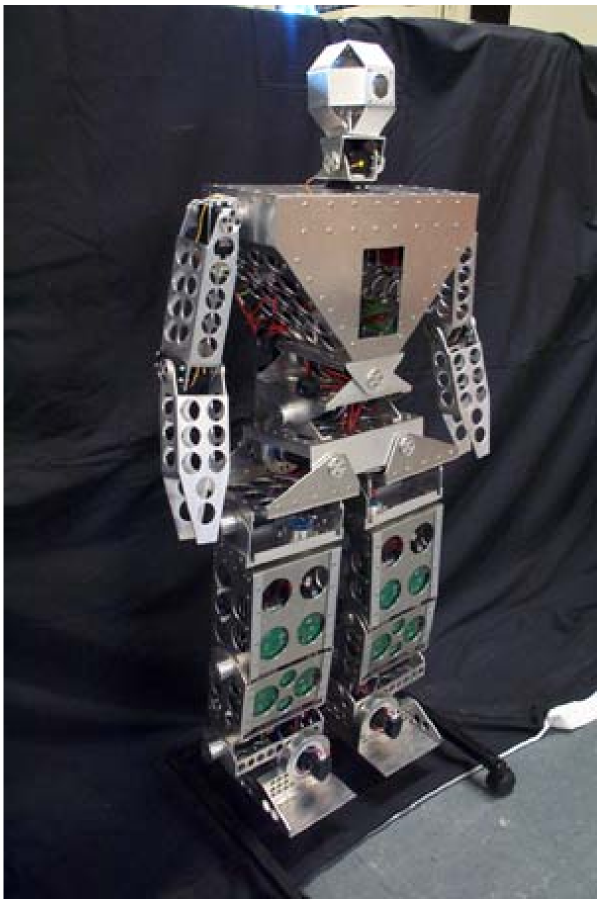
The GuRoo has 23 degrees of freedom corresponding to its 23 motors. There are 8 Hi-Tec HS705-MG Servo Motors controlling the upper body and 15 70W DC geared motors from Maxon control the lower body [1].

The 2002 GuRoo team consists of Ho Wong (Vision Hardware for a Humanoid Robot), Andrew Hood (Distributed Motion Controllers for a Humanoid Robot), Adam Drury (Gait Generation and Control Algorithms for a Humanoid Robot) and Ian Marshall (Active Balance Control for a Humanoid Robot). Supervising the project is Gordon Wyeth and Assistant supervisor is postgraduate student, Damien Kee.

[1] Grossly Underfunded Roo. The 'Roo' has been adopted as a suffix for all of the University of Queensland's robot soccer teams.

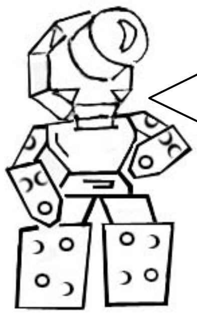


a) CAD Drawing



b) Real Picture

Figure 1-1: GuRoo



FACT: There are over 600 individual muscles in the body which account for 40% of the body's weight.

Besides the hands and toes, GuRoo has almost all the major degrees of freedom as a human.

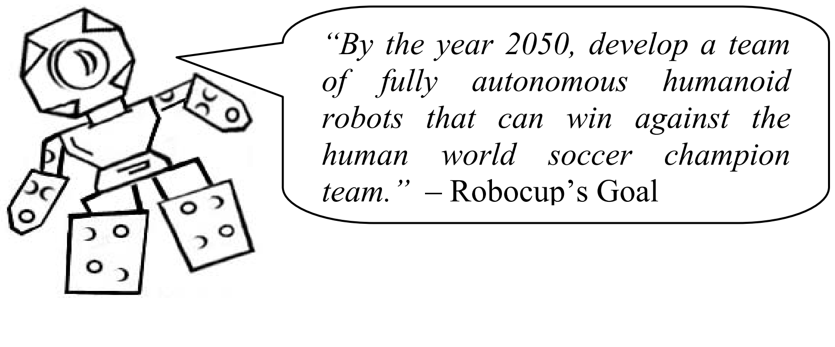
1.3 Robocup

RoboCup is an international research and education initiative. Its goal is to foster artificial intelligence and robotics research by providing a standard problem where a wide range of technologies can be examined and integrated. The concept of soccer-playing robots was first introduced in 1993. Following a two-year feasibility study, in August 1995, an announcement was made to introduce the first international conferences and soccer games. In July 1997, the first official conference and game was held in Nagoya, Japan. Followed by Paris, Stockholm, Melbourne and Seattle, the annual events attracted many participants. Following previous conferences and games, the 6th RoboCup will be held in Fukuoka, Japan in cooperation with Busan, Korea in June, 2002. This RoboCup will coincide with the"2002 World Cup Korea/Japan. [2]

There are various divisions in Robocup: Simulation, Small Sized Robot, Middle Size Robot, Sony Legged Robot and Humanoid. In the Humanoid division, there are also three height divisions: 40cm, 80cm and 120cm.

The University Of Queensland has 4 different teams: The CrocaRoo's (Simulation League); The RoboRoos (Small Sized League – Global Vision); The ViperRoos (Small Sized League – Local Vision) and the GuRoo (Humanoid League). The difference between Global Vision and Local Vision is that Global Vision uses overhead cameras whereas Local Vision uses cameras mounted on the robots themselves.

The Humanoid League competition for 2002 has many different events. These are: Humanoid Walk – humanoid is required to walk from a starting line, to and around a red marker and back again; Shoot – humanoid is required to kick a ball into the goal; Penalty Shoot Out – humanoid is required to kick a ball into a goal defended by another humanoid and Freestyle – humanoid has 10mins for freestyle actions [2]. For information on how well GuRoo performed in Robocup2002, please visit http://www.itee.uq.edu.au/~damien/_guroo .


"By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team." – Robocup's Goal

[2] Abstract from RoboCup2002 website: www.robocup2002.org

1.4 Robocup Vision

The vision requirements for GuRoo are currently determined by the events in Robocup. The requirements needed to participate effectively in the events are:

- Ability to identify the ball and its location.
- Ability to identify objects, e.g. Opponents.
- Ability to identify the goal.
- Ability to identify the boundaries of the playing field.

The playing environment has been easily colour coded. **Table 1-1** shows the colour-coded environment [3].

Object	Colour
Field	Green
Ball	Red
Edge of Field Lines	White
Opponents	Black
Goals	Blue or Yellow

Table 1-1: Colour Codes

There are many problems involved with colour detection. One of the major problems is ambient light. All the studio lighting will make everything very bright and saturate any light sensitive sensors. The solution to this is to view everything in YUV as opposed to RGB. In YUV, the visible colour spectrum is mapped onto U (Red chrominance) and V (Blue chrominance). The intensity is mapped onto Y (Yellow). **Figure A3** shows the colour spectrum mapped onto U and V. This should solve the ambient light problem.

Another problem is identifying distances to objects. With only a single camera, the size of objects has to be pre-known. For the ball, it is simple because no matter which angle it is viewed from, it is always round. However, an opponent robot would be much thinner when viewed from the side than it would be when viewed from the front. This problem; however, is beyond the scope of this thesis.

2 Previous Technology

It is silly to reinvent something that's already available. Mark Chang is a postgraduate working under Gordon Wyeth. He worked on the ViperRoos[3] for three years. Recently, he has been developing a new vision board and camera to supersede the current ones used in the ViperRoos. The ViperRoos run on SH3 microprocessors and use a PB-159 Photobit CMOS camera. The new boards use SH4 microprocessors and use a CMOS OmniVision OV7620 camera.

David Prasser and Andrew Blower were undergraduate thesis students who were part of the 2001 GuRoo team. David Prasser's thesis was on Vision Software for a Humanoid robot and Andrew Blower's thesis was on Vision Hardware for a Humanoid Robot.

2.1 Hitachi SH4

The vision system that the GuRoo uses is the new vision board developed by Mark Chang. The core processor is the Hitachi SH4. The SH4 runs at 360MIPS (Mega Instructions Per Second) whereas the SH3 ran at 104MIPS.

A general block diagram of the vision board is shown in **Figure 2-1**. The following pages are a brief documentation of some of the SH4's functionalities and the setup done by Mark Chang.

[3] University Of Queensland's Small Localised Vision, Robocup Soccer Team.

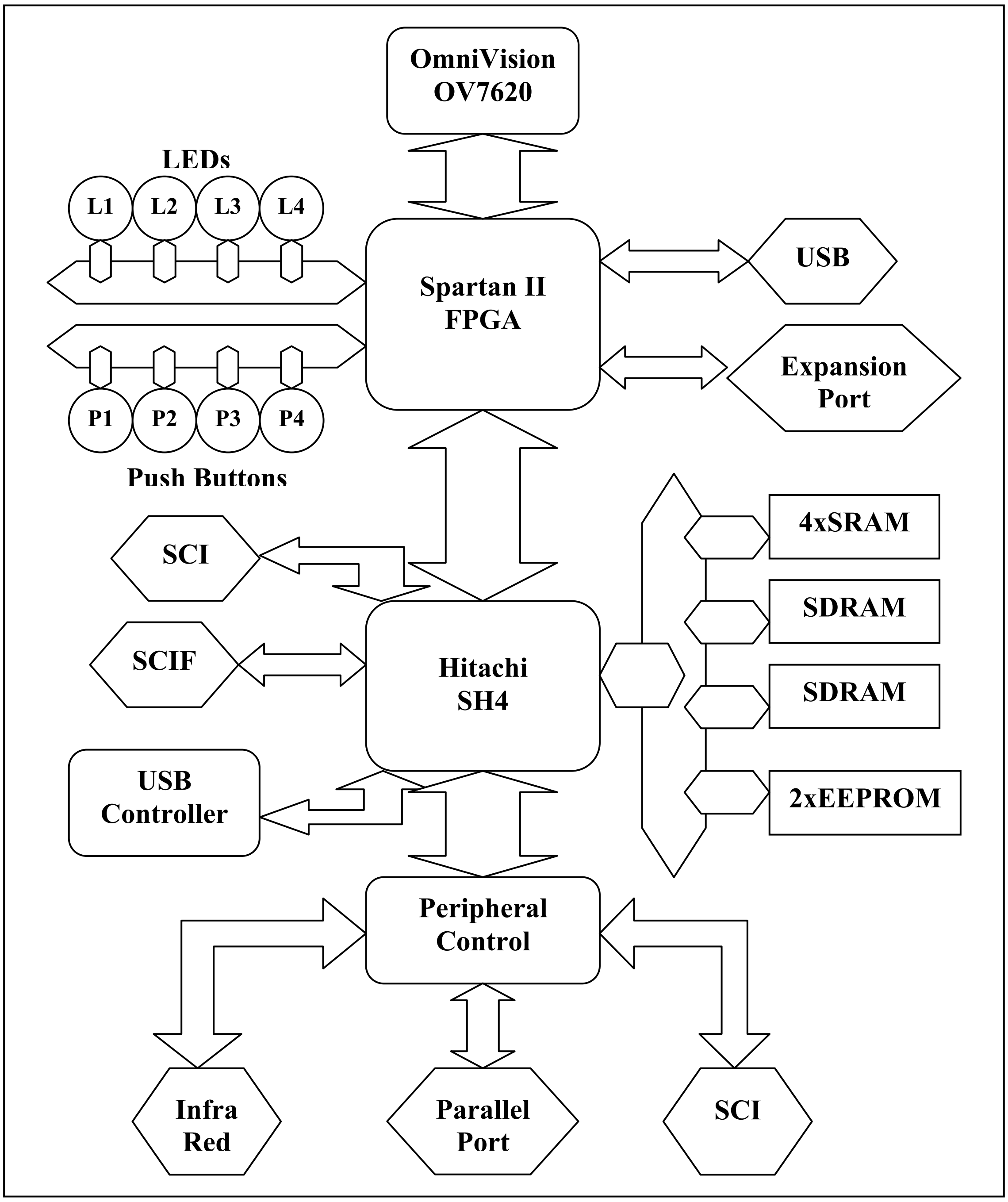
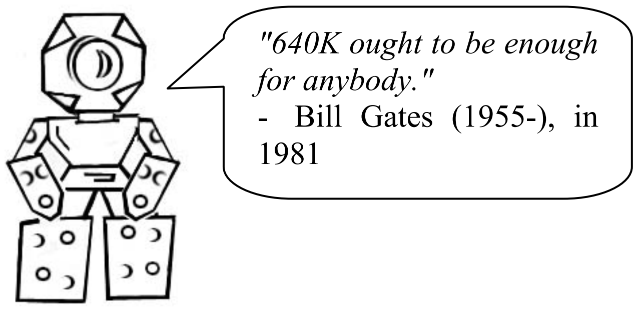


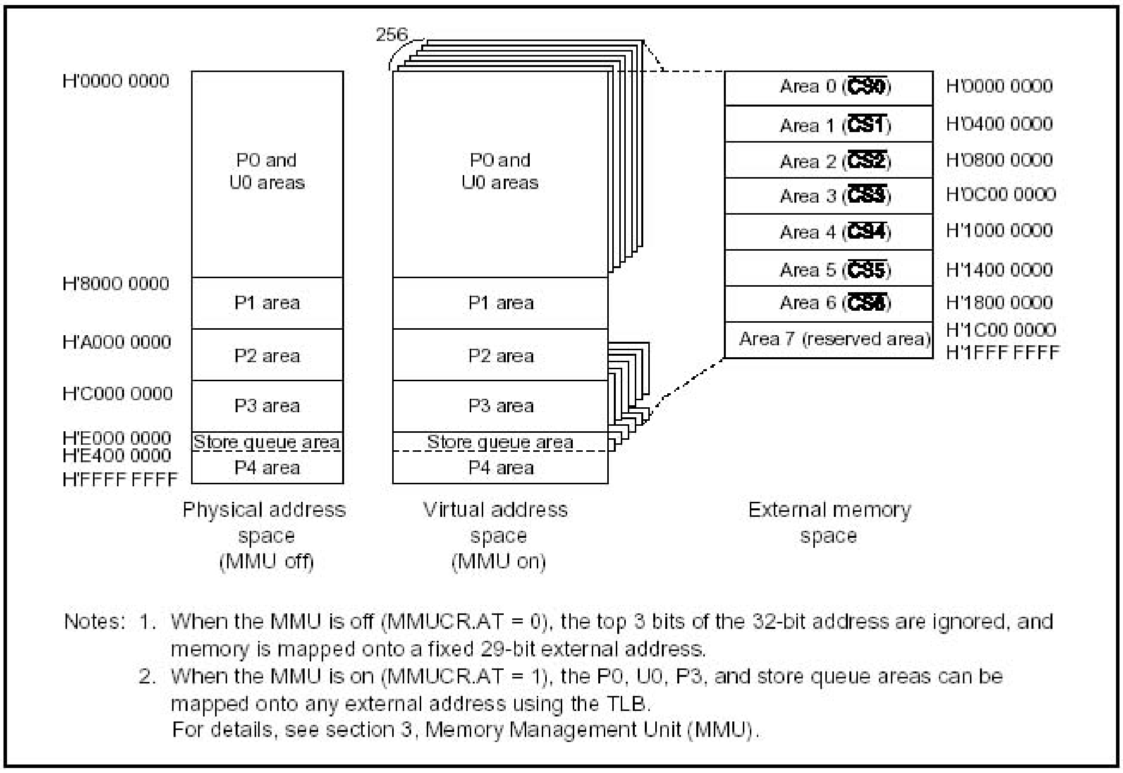
Figure 2-1: Vision Board Block Diagram

The SH4 has 3 external memory banks and 1 bank of EEPROM. There are also 2 serial ports, a USB controller and a Peripheral Controller which has an infra-red sensor, parallel port and a UART serial port. Connected to this is a SpartanII FPGA that is also the interface to the OmniVision OV7620 camera and other external devices: push buttons, LEDs, usb port and an expansion port.

2.1.1 On Board Memory

The SH4 runs on external memory. Only 8Kbyte can be used as on-chip RAM. There are four blocks of 128Kbyte SRAM stacked together to form a single 512Kbyte bank and four blocks of 2Mbyte SDRAM in each of the two 8Mbyte SDRAM banks. There is also 768Kbyte of EEPROM made up by stacking a 512Kbyte and a 256Kbyte chip. Only the SRAM is used because it performs much faster than the SDRAM but as expected, it costs much more as well. The price of the SRAM is approximately $10 per 128Kbyte chip compared to $6 for per 8Mbyte SDRAM chip. The memory addressing of the SH4 is shown in **Figure 2-2** [4]. Area's P0 to P4 are physically mapped and Area's 0-7 are external memory. Areas U0, P0 and P3 are accessed using the TLB (Translation Lookaside Buffer), while areas P1, P2 and P4 are not accessed using the TLB.


"640K ought to be enough for anybody."
- Bill Gates (1955-), in 1981



Notes: 1. When the MMU is off (MMUCR.AT = 0), the top 3 bits of the 32-bit address are ignored, and memory is mapped onto a fixed 29-bit external address.
2. When the MMU is on (MMUCR.AT = 1), the P0, U0, P3, and store queue areas can be mapped onto any external address using the TLB.
For details, see section 3, Memory Management Unit (MMU).

Figure 2-2: Memory Mapping

2.1.2 DMAC (Direct Memory Access Controller)

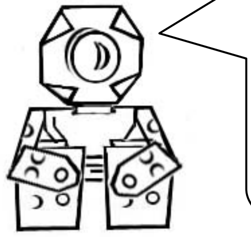
The SH4 has 4 DMA channels. In normal operation, only channels 1 and 2 can have external requests. On board peripherals can be used on all 4 channels. In DDT (Direct Data Transfer) Mode, all 4 channels can have external requests. There are two modes of DMA transfer available: Cycle Steal Mode and Burst Mode. In Cycle Steal Mode, the DMAC holds the bus until a transfer-unit (8-bit, 16-bit, 32-bit, 64- bit or 32-byte) is transferred. Burst Mode is when the DMAC holds the bus and only releases it back to the CPU after all the transfer-units in a block are transferred.

2.1.3 Interfaces

The Hitachi SH4 has a single-channel serial communication interface (SCI) and a single-channel serial communication interface with built-in FIFO registers. The SCI can handle both asynchronous and synchronous serial communication while the SCIF is a dedicated asynchronous serial communication interface with a 16-stage FIFO registers. Both are full duplex. The benefit of a FIFO register is that serial data can be continuously transmitted or received until part or all of the 16-stage registers are used up.

There are 20 general purpose I/O pins (GIO PINs). The direction for each pin can be individually specified as can the use of a pull-up resistor. Interrupt input is possible for 16 of the 20 pins.

Interfaced to the FPGA is also a 32bit data bus and 19bit address bus. This is for when memory is created on the FPGA and added onto the SH4's addressable external memory.



"If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get a million miles per gallon, and explode once a year, killing everyone inside".
– Robert X. Cringely, InfoWorld magazine

2.2 Spartan II FPGA (Field Programmable Gate Array)

An FPGA can be thought of as a software programmable hardware interface. The device that is used in the vision board is a Spartan II xc2s100. The xc2s100 has 100, 000 System Gates, 176 I/O pins, 4 Delay-Lock Loops and 40K of Block RAM. The xc2s100 has a system speed of over 200MHz [5]. Previously, the FPGA only acted as a direct connection between all the peripherals: camera, USB, Expansion Port and LEDs etc. and the SH4.

The Spartan II xcs100 does not have enough memory to store a whole picture on it. It is; therefore, only possible to store images either pixel-by-pixel or by blocks of pixels.

Currently, the USB and Expansion Port are disabled. Nearly all of the 176 I/O pins have been assigned: 32 pins for the data bus, 19 pins for the address bus, 16 pins for the camera data, 20 pins for each of the GIO, and the rest made up by control signals to and from the SH4 and the peripheral modules. Provision has been made for the use of the Block RAM but has not been implemented.

2.3 Interface to the rest of the robot

Originally, the SH4 interfaces to an IPAQ but this year, the IPAQ has been put aside as there were insufficient resources to complete its interface. The main processing code for the GuRoo is currently running from Board 6 – the motor control board for the upper torso.

Serial port SCIF can be used to interface to the SH4. Currently, serial ports SCI and SCIF are being used to interface to a PC for testing and debugging; therefore, changing SCIF to interface to Board 6 or the IPAQ is easy.

A future development for the GuRoo is to create a separate control board (Board 7).

2.4 OmniVision OV7620

The OmniVision OV760 is a CMOS digital camera able to take YCrCb or YUV (Yellow, Red chrominance, Blue chrominance) and RGB (Red, Green, Blue) images. The formulas for YUV and YCrCb are shown in **Table 2-1** [6].

	YUV Colour		YCrCb Colour
Y	0.299 x R + 0.587 x G + 0.114 x B	Y	0.299 x R + 0.587 x G + 0.114 x B
U	-0.147 x R -0.289 x G + 0.436 x B	Cb	(B - Y)/(2 – 2 x 0.114)
V	0.615 x R - 0.515 x G - 0.100 x B	Cr	(R - Y)/(2-2 x 0.299)

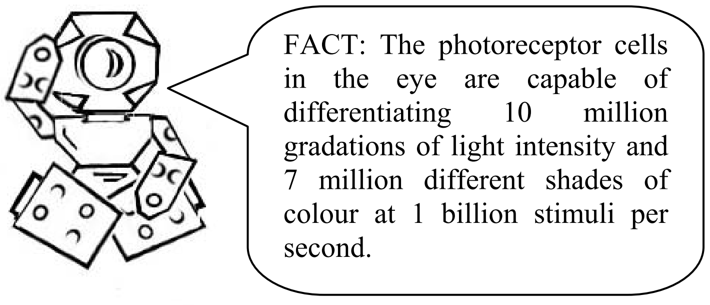
Table 2-1: Colour Formulas

The default output for the OV7620 is 640 x 480 but it has a windowing feature enabling the window size to be changed anywhere from 4x2 to 664x492. The OV7620 supports both interlaced and progressive scanning.

The OV7620 was the camera of choice by Mark Chang for the ViperRoos and so it was adopted along with his vision board for the GuRoo. One reason for its choice is that the output is digital and not analogue. This means less noise and better performance (i.e. no need for an A/D converter). The other reason is that it supports the YUV colour space because, as mentioned earlier, colour detection in YUV is the best method.

Output for YUV is either 8bit or 16bit 4:2:2. If it is 8bit, the transfer rate is twice the pixel rate. **Table 2-2** and **Table 2-3** [7] on the next page show the output timing for 16bit and 8bit transfer.

Control of the registers is done through the SCCB (Serial Camera Control Bus), which is half duplex and timing has to be carefully monitored. Refer to the datasheets for more detail.


FACT: The photoreceptor cells in the eye are capable of differentiating 10 million gradations of light intensity and 7 million different shades of colour at 1 billion stimuli per second.

Data Bus	Pixel Byte Sequence				Pixel Byte Sequence			
Y7	U7	Y7	V7	Y7	U7	Y7	V7	Y7
Y6	U6	Y6	V6	Y6	U6	Y6	V6	Y6
Y5	U5	Y5	V5	Y5	U5	Y5	V5	Y5
Y4	U4	Y4	V4	Y4	U4	Y4	V4	Y4
Y3	U3	Y3	V3	Y3	U3	Y3	V3	Y3
Y2	U2	Y2	V2	Y2	U2	Y2	V2	Y2
Y1	U1	Y1	V1	Y1	U1	Y1	V1	Y1
Y0	U0	Y0	V0	Y0	U0	Y0	V0	Y0
Y Frame	0		1		2		3	
UV Frame	0				1			

Table 2-2: 8bit 4:2:2

Data Bus	Pixel Byte Sequence			
Y7	Y7	Y7	Y7	Y7
Y6	Y6	Y6	Y6	Y6
Y5	Y5	Y5	Y5	Y5
Y4	Y4	Y4	Y4	Y4
Y3	Y3	Y3	Y3	Y3
Y2	Y2	Y2	Y2	Y2
Y1	Y1	Y1	Y1	Y1
Y0	Y0	Y0	Y0	Y0
UV7	U7	V7	U7	V7
UV6	U6	V6	U6	V6
UV5	U5	V5	U5	V5
UV4	U4	V4	U4	V4
UV3	U3	V3	U3	V3
UV2	U2	V2	U2	V2
UV1	U1	V1	U1	V1
UV0	U0	V0	U0	V0
Y Frame	0	1	2	3
UV Frame	0		1	

Table 2-3: 16bit 4:2:2

2.5 Vision Software

Vision Software had been developed by David Prasser in 2001 for the GuRoo. The image processing is described in **Figure 2-3**.

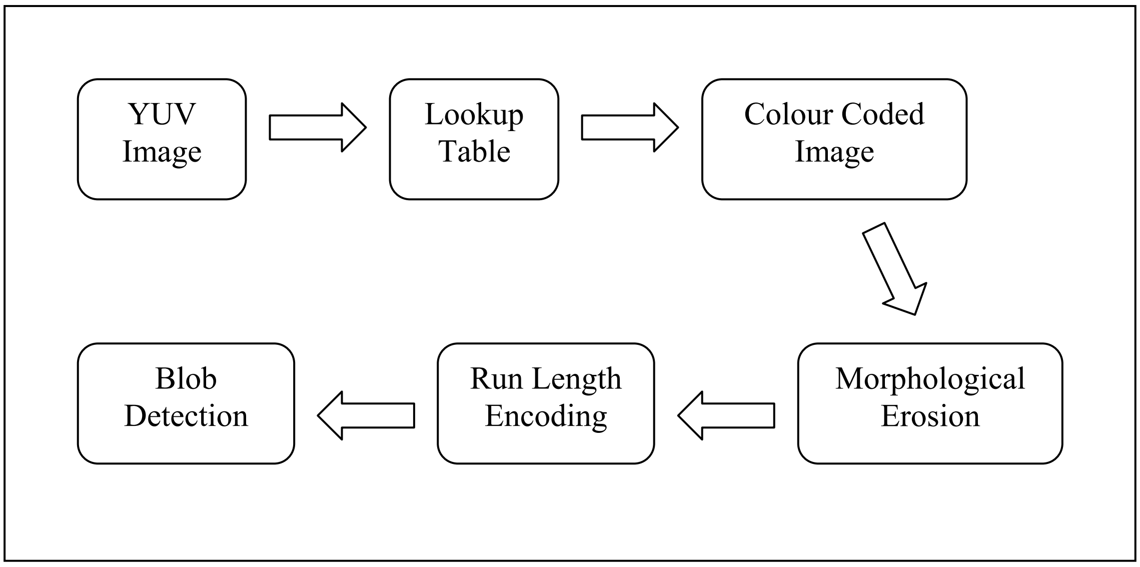


Figure 2-3: Vision Software Block Diagram

2.5.1 Colour Lookup

Colour Lookup is done through the use of a Lookup Table. There are only a preset number of colours that the GuRoo has to identify. These were described in **Table 1-1**. The code for sorting out which pixel belongs to which colour is described below [3]:

```
if Y > Max Threshold
        colour = White
elsif Y< Min Threshold
        colour = Black
else
        colour = Lookuptable(U,V)
        // Where U and V are indexes to the lookup table
```

Y measures the intensity of the colour; therefore, Max Threshold is the intensity threshold for white and Min Threshold is the intensity threshold for black. Whatever's in between is sorted through the Lookup Table. The Lookup Table is a simple table with vertices made up by the U and V components. One of the problems that occur is the identification of the colour yellow. Yellow is very close to white and hence unless the Max Threshold is calibrated properly, what is yellow will be categorised as white. **Figure 6-3** in Chapter 6: Testing and Analysis, shows a sample lookup table.

2.5.2 Morphological Erosion

After the colour lookup, each pixel is colour coded. Morphological Erosion is then used to remove noise. E.g. a black pixel surrounded by red pixels would be removed. For more information regarding Morphological Erosion, please refer to David Prasser's Undergraduate Thesis [3].

2.5.3 Run Length Encoding (RLE)

Run length encoding is where each row is examined and every run of the same colour becomes RLE elements. Each RLE element contains: the start x coordinate; end x coordinate; y coordinate; colour; run number and blobpointer. **Figure 2-4** shows an example of RLE.

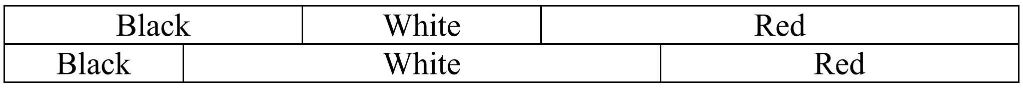


Figure 2-4: RLE

2.5.4 Blob Detection

Each run is examined one at a time and compared to the runs on the following row. Runs that have the same colour and are directly underneath each other are grouped together to form a blob. **Figure 2-5** shows how a red blob is formed.

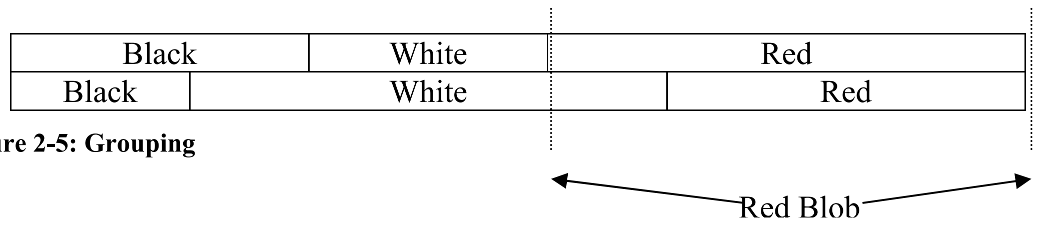


Figure 2-5: Grouping

A blob is a four connected region of pixels of the same colour. Blobs have an: area, colour and rectangular boundary co-ordinates.

These blobs are then analysed and associated with objects. E.g. the large white blobs are walls, the largest red blob is the ball and any black blob greater than a certain size is an object.

3 Project Plan

3.1 Summary of Vision System at the start of 2002

By the end of 2001, the GuRoo's Vision System was using the SH4 vision board developed by Mark Chang but did not have a working camera. David Prasser did all of his vision software using images downloaded from a PC through serial communications. Andrew Blower wrote FPGA code for a dual ram buffer [8] but could only test it in simulation because there was no working camera. The interface to the IPAQ was not fully developed either.

At the start of 2002, it was said that Mark Chang had developed a working camera board and that the board he developed should be used on the GuRoo. Unfortunately, Mark Chang went on leave for half a year and was not contactable until July, second semester.

3.2 Initial Project Plan

Robocup2002 was held from Wednesday June 19th to Tuesday June 25th in Fukuoka, Japan. At the start of the year, a project plan was drawn up for the development of the vision system of the GuRoo. This was outlined in the Progress Report and is shown in **Figure 3-1** [9].

Task	Duration	Start Date
SH4 Board Setup	1day	April 2
SH4 Compiler (example LED code)	2days	April 3
David Prassers' old code running	4days	April 5
FPGA (Camera to SH4)	5days	April 9
Camera (filling SH4 buffer with image)	7days	April 14
Image back to PC	0days	April 21
Ball/ Horizon detection	10days	May 1
Bearing/ distance to ball	10days	May 11
Main area of Interest	10days	May 21
Serial Tx of vision information	5days	May 25

Figure 3-1: Initial Plan

This plan was made based on the assumption that the camera was working. All that needed to be done was to plug it in and the setup for the SH4 would be easy.

To make the deadline for the competition, it was initially decided that the goal was to get the GuRoo being able to see a red ball or red object. To this end, all that was needed was to run David's Prasser's vision code with images taken from a camera mounted in its head. The FPGA would be a direct wire connection between the camera and the SH4. Interface from the SH4 to the rest of the Humanoid would be through serial communications.

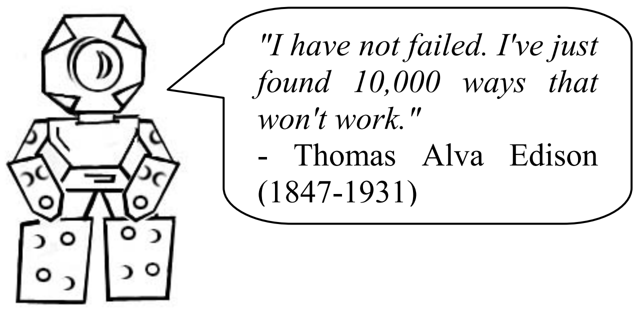
3.3 Project Progress by June

Unfortunately, problems were rife throughout the whole project. At the start, David Prasser did not leave behind the SH4 cross-compiler and a new one had to be made. Little problems like:

- Inability to find a computer that had RPM (RedHat Package Manager).
- Only newest version of cross-compiler available.

Made the progress very slow. Only source documents compressed with RPM could be found. While it may seem to be a simple matter, the fact was that all the UNIX computers available to students used SunOS on Sun Spark computers. There was no available Linux Operating System and RPM could not be installed onto the SunOS computers due to security reasons. Weeks were wasted while attempting to make the cross-compiler. The solution finally came in the form of a Backup CD found on Mark Chang's desk. This provided the cross-compiler he used as well as all the source codes.

By the time the setup was complete for the SH4's usage, the initial goals could not be met. Mark Chang returned in July and was of invaluable help in understanding the vision board but; unfortunately, there were only 11 weeks left and hence a new project plan was drafted.


"I have not failed. I've just found 10,000 ways that won't work."
- Thomas Alva Edison (1847-1931)

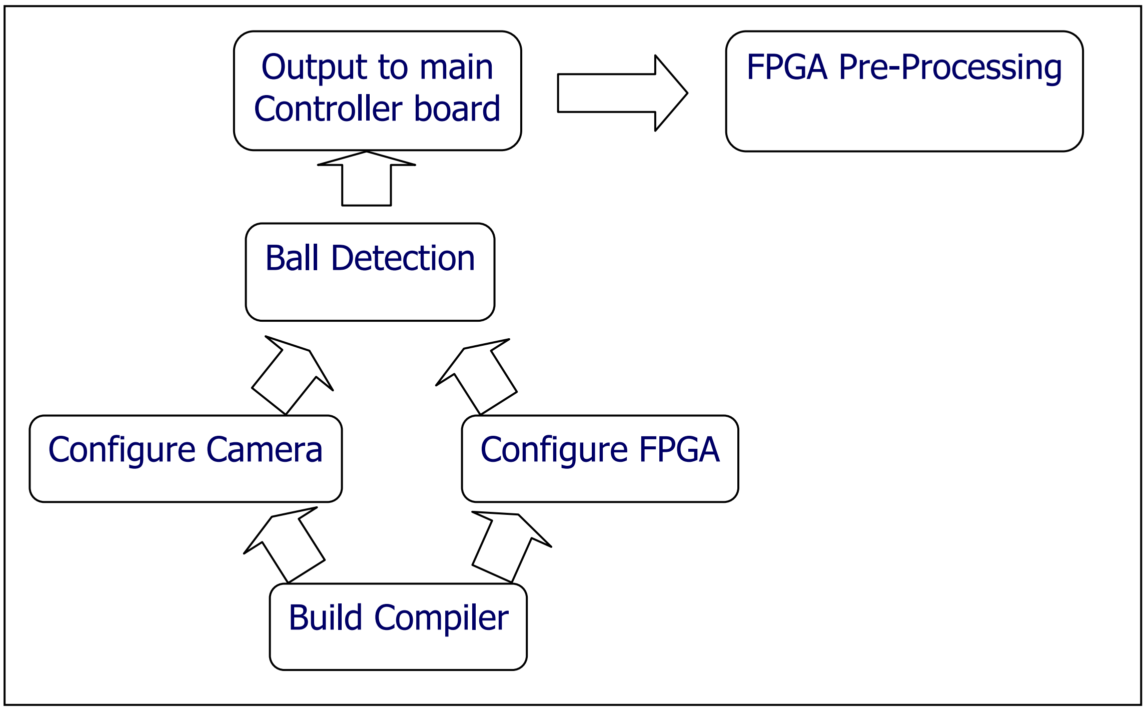


Figure 3-2: New Project Plan

Figure 3-2 shows the Block Diagram for the new plan [10]. Due to the amount of time left, the main focus of the thesis had to be shifted to the FPGA, where the pre-processing of the image will be done. Performance of image processing can be greatly improved if the pre-processing is taken off the SH4 because the FPGA runs at a much faster speed than the SH4.

By September, the camera was still not working and it was finally accepted that all the cameras available (2 built by Mark Chang and 1 built by Ho Wong) may not be operational. Configuration of the camera was put aside and the focus of the thesis moved onto the FPGA pre-processing.

The time left was judged to be sufficient for research and development on the FPGA, even though Ho Wong had no prior experience with FPGAs and the software required to program them. The programming language, VHDL, will have to be learnt from the basics.

The steps that need to be completed for the FPGA pre-processing are:

- Learn VHDL.
- Storage and retrieval of lookup-table on FPGA.
- Colour Lookup with static inputs – pixel's colour a static constant.
- Colour Lookup with dynamic inputs – pixel's colour provided by SH4.
- Colour Lookup of whole image.
- Optimisation of process.

4 Technical Design

4.1 Problem Definition

There is only 512Kbyte of SRAM available on the SH4. SRAM performs much faster than SDRAM so the usage of the SDRAM was never even been considered.

Table 4-1 shows the memory usage on the SH4 with David Prasser's vision software [3].

Buffer	Size	Memory
YUV Image	240 x 180	168.75KB
Lookup Table	256 x 256	64KB
Colour Detected Image	240 x 180	42.2KB
Eroded Image	240 x 180	42.2KB
RLE Elements	3072	72KB
Blobs	512	12KB
Objects	64	768Bytes
Edge Profile	180	720Bytes
Total		401.8KB

Table 4-1: Memory Usage

The biggest usage of memory is the storage of the raw YUV image. Each pixel has an 8-bit Y, U and V component. The size of the YUV Image (240x180) chosen by David Prasser was the largest image possible keeping the 4:3 ratio. The Colour Detected Image is only a 240 x 180 table of 8 bit data. A full 640 x 480 Colour Detected Image with 4 bit data will only take up 153.6KB of memory. Removing the YUV Image and the Lookup Table from the SH4 will free up 232.75KB of memory. If dual ram buffering is required an image 360 x 270 could be used.

Table 4-2 shows the projected memory usage if two 360x270 Colour Detected Images are used.

Buffer	Size	Memory
Colour Detected Image 1	360 x 270	97.2KB
Colour Detected Image 2	360 x 270	97.2KB
Eroded Image	360 x 270	97.2KB
RLE Elements	3072	72KB
Blobs	512	12KB
Objects	64	768Bytes
Edge Profile	180	720Bytes
Total		377.1KB

Table 4-2: Projected Memory Usage

This is assuming that the RLE Elements, Blobs and objects stay the same. Even if they increase, there is still enough space.

4.2 Initial Approach

The FPGA only has 100,000 system gates and 40KB of Block Ram. An image 240 x 180 will definitely not fit inside it. The Lookup Table will have to be stored inside the FPGA as well. Unfortunately, the standard Lookup Table takes up 64KB of memory and there is not enough Block RAM to store it. Distributed RAM could be used but the better solution is to find a different way to do the colour lookup.

4.2.1 Bruce's Lookup Table[4]

The solution to the colour lookup is what can be thought of as a compressed lookup table. A normal 2 dimensional lookup table lists, for every index of V and U, a colour. This makes a 256 x 256 8bit table if the colour is an 8bit data type. Bruce's Lookup Table only has two 256 arrays of the number of bits corresponding to the total number of available colours. For example, if the total number of colours is 8, then it will have two 256 8bit arrays.

[4] Named after James Bruce, co-author of *Fast and Inexpensive Color Image Segmentation for Interactive Robots* [11]

It uses a bitwise AND of the U and V arrays to check if it belongs in a certain colour. A small example is provided below.

If the available colour range is 8, then a normal UV Lookup Table would look similar to **Figure 4-1**. Where G, Y B & R are: Green, Yellow, Blue and Red respectively. The data types of these can be anywhere from 4 bits to 8 bits.

To find out the colour of pixel <U,V><1,2>, the row U1 would be searched until element V1 is found and the colour contained there is retrieved (reverse process for column major) In this example, it would be the colour Yellow.

U7	G	G						
U6	G	G						
U5	G				B	B		
U4								R
U3			Y				R	R
U2		Y	Y	Y		R	R	R
U1		Y	Y			R	R	R
U0	Y	Y			R	R	R	R
	V0	V1	V2	V3	V4	V5	V6	V7

Figure 4-1: Normal Lookup Table

If we assume each colour is a 4bit data type, then the total memory usage would be

8 x 8 x 4 = 256 bits.

Bruce’s Lookup Table would look similar to **Figure 4-2**. Here there are two arrays: one for U and one for V. Each element of U and V have 4 bit data with each bit representing whether or not that element is part of a colour or not.

To find the colour of pixel <U,V><1,2> element U1 [0 , 1 , 0 , 1] would be bit ANDed with element V2 [0 , 1 , 0 , 0] which would produce:

LSB[0 , 1 , 0 , 0]MSB

Which corresponds to the colour Yellow.

R	1	1	1	1	1	0	0	0
B	0	0	0	0	0	1	0	0
Y	1	1	1	1	0	0	0	0
G	0	0	0	0	0	1	1	1
	U0	U1	U2	U3	U4	U5	U6	U7

R	0	0	0	0	1	1	1	1
B	0	0	0	0	1	1	0	0
Y	1	1	1	1	0	0	0	0
G	1	1	0	0	0	0	0	0
	V0	V1	V2	V3	V4	V5	V6	V7

Figure 4-2: Bruce’s Lookup Table

The memory total memory usage would be

2 x 8 x 4 = 64 bits

As can be seen, the size of this table is a quarter the size of a normal Lookup Table. If there are 8 colours (i.e. all 4 bits in the normal lookup table correspond to a colour) then Bruce’s Lookup Table will require 8 bits per element but the total size will still be smaller than the normal Lookup Table (128bits). Bruce’s Lookup Table has the same functionality except that it uses a lot smaller space.

A normal 256 x 256 table with 8 bit data (has a capacity to represent 256 colours) would take up approximately 65.5Kbytes. Bruce’s Lookup Table would take up 16.4Kbytes when representing 256 colours.

4.2.2 Distributed RAM vs Block RAM

Distributed RAM is where RAM is created from the system gates.
The advantages of using Distributed RAM are:

- Variable RAM size – no need to stack RAM blocks.
- Maximum size dependant on available system gates
- Variable interface – i.e. buses are user defined

Disadvantages are:

- Non-dedicated hardware – would run slower than actual RAM
- A driver would have to be created.
- Timing not predefined – the timing would be determined by the coding inside the driver.

The main factor in deciding which type to use was ease of use. Time is very limited and the easiest solution would be the Block RAM. It is the easiest to implement because all the drivers already exist. The drawback is that Block RAM comes in 4KB blocks. To stack them would also require a separate driver program or the use of a core generator.

4.2.3 Spartan II Block RAM

The Spartan II series of FPGA comes with 2 columns of Block RAM located to the left and right of the CLB, as shown in **Figure 4-3**. The number of Block RAM available is dependant upon the version of Spartan II, with the minimum being 4 blocks and the maximum being 20 blocks. The xc2s100 has 10 blocks making a total of 40KB of available Block Ram. [12]

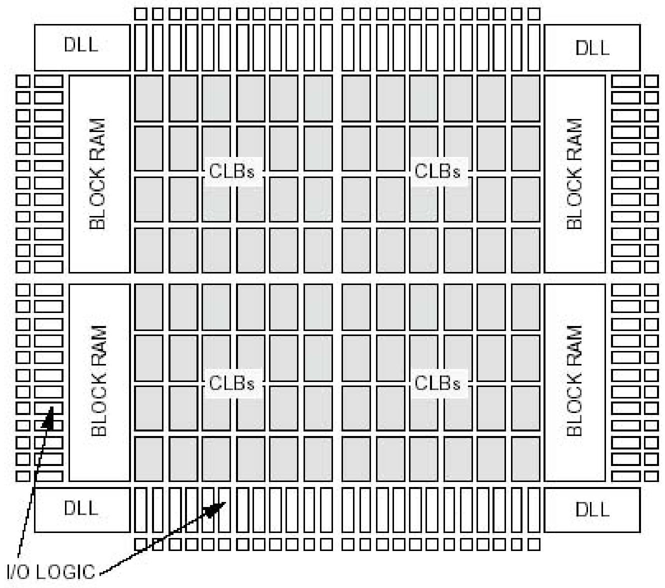
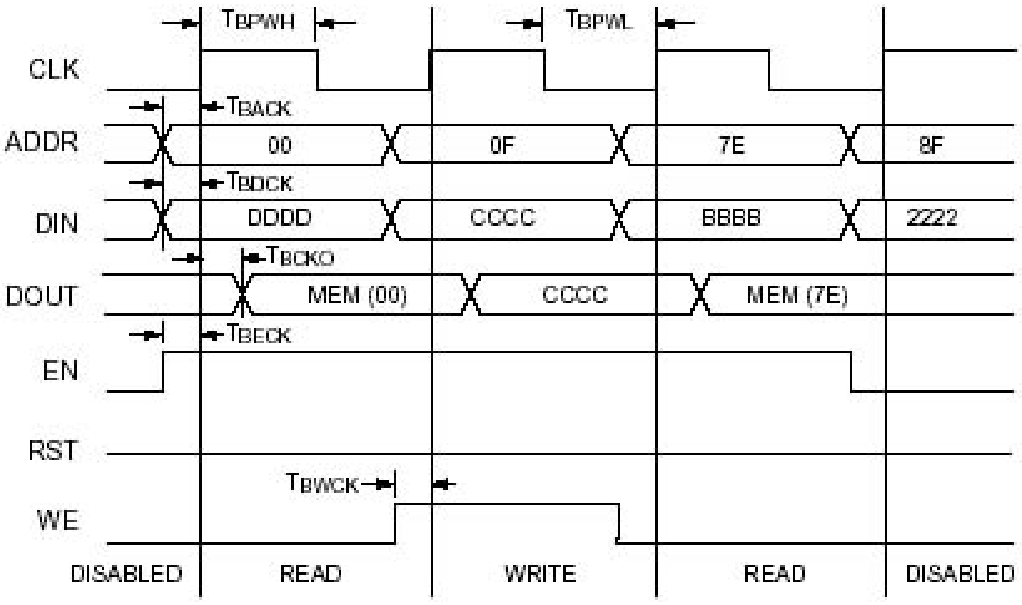


Figure 4-3: FPGA Block Diagram

The timing for the Block RAM is shown in **Figure 4-4**.[13]

On the rising edge of CLK, the WE, ADDR, DI & EN pins are sampled. If WE & EN is high (write), then DI is written into address ADDR and DO mirrors the data. If WE is low & EN is high (read), then DO mirrors the data stored in address ADDR. If EN is low, reading and writing are disabled and DO retains its last value. If RST is high, DO is latched to low.



Minimum pulse width High & Low	**1.9 ns**
ADDR inputs & DIN inputs	**1.4 ns**
DOUT output	**4.0 ns**
RST input	**3.2 ns**
EN input	**2.9 ns**
WEN input	**2.8 ns**

Figure 4-4: Block Ram Timing

As can be seen from **Figure 4-4**, the Block RAM is very fast. Mark Chang has made provision for the addition of the FPGA's Block RAM onto the SH4's BSC (Bus State Controller).

5 Design Implementation

5.1 Testing Procedure

All three cameras available were deemed inoperable; therefore, David Prasser's method of downloading an image directly to the SH4 from a PC had to be adopted [3]. The experiment procedure is:

- Take picture from normal digital camera.
- Convert picture into YUV colour space through Matlab.
- Download picture into SH4 through serial communications.
- Download Lookup Table into SH4 through serial communications.
- Upload Lookup Table into FPGA.
- Upload a pixel from SH4 into FPGA and receive back the associated colour. – Repeat for all the pixels in picture.
- Upload colour-coded image back to the PC for verification.

5.2 Test Environment

5.2.1 Test Image Size

David Prasser used a 240 x 180 image because it was the largest possible size for the given memory available. The image that will be used to test the FPGA pre-processing is 150 x 112. This will free up memory to store the new Lookup Table.

5.2.2 Colour Codes

There are only 4 colours besides white and black that will be used. These are: Red, Yellow, Green and Blue. The colour codes are displayed in **Figure 5-1**.

RED	00000001
YELLOW	00000010
GREEN	00000011
BLUE	00000100
WHITE	00000101
BLACK	00000110
OTHER	00000000

Figure 5-1: Colour Codes

5.2.3 Bruce's Lookup Table

There are only 4 colours in the Lookup Table; therefore, only 4 bits are needed for the Lookup Table. The Lookup Table will be a 256 array of 8 bit data. Each 8 bit data contains both the U and the V.

```
MSB <7  6  5  4><3  2  1  0> LSB
    <    U     ><    V     >
    <B G Y R><B G Y R>
```

The Lookup Table now only takes up 256 bytes of memory.

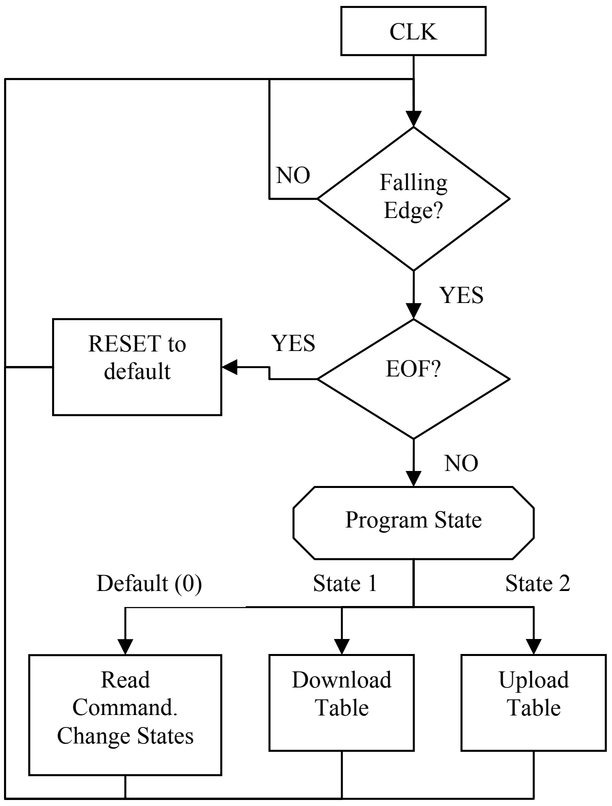
5.3 FPGA Software implementation

It was decided to use GIO pins 0 to 7 as the FPGA's data bus for simplicity. There were not enough free GIO pins for a 16bit bus because 4 are used for the LED's and another 4 are used for the camera's SCCB control. GIO pin 10 has been temporarily used as the source for an external clock. The input and output buses (DI & DO) are tri-stated to the GIO bus.

5.3.1 Storage of Bruce's Lookup Table

Bruce's Lookup Table is stored on the FPGA's Block RAM. Initial design of the code was to store each byte of the table at each state change of the external clock (rising and falling edge). This proved to be impossible to implement due to the need to clock the Block RAM twice in one clock cycle.

The current implementation is shown in **Figure 5-2**. The program continually checks the input data bus for commands. When a command is received, the state is changed and the program stays in that state till all the data is uploaded or downloaded. If more commands are required, additional states can be added easily. **Figure 5-3** shows the timing for the program.



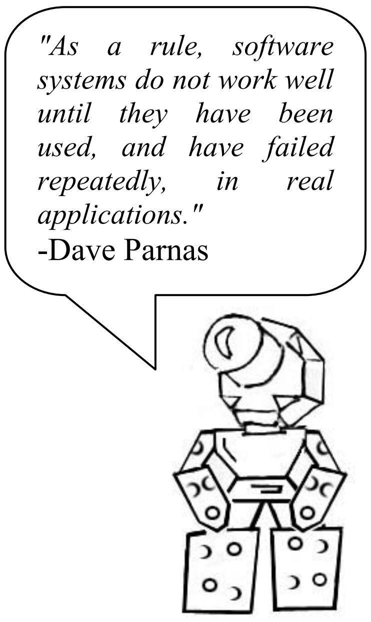

"As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications."
-Dave Parnas

Figure 5-2: FPGA Flow Chart

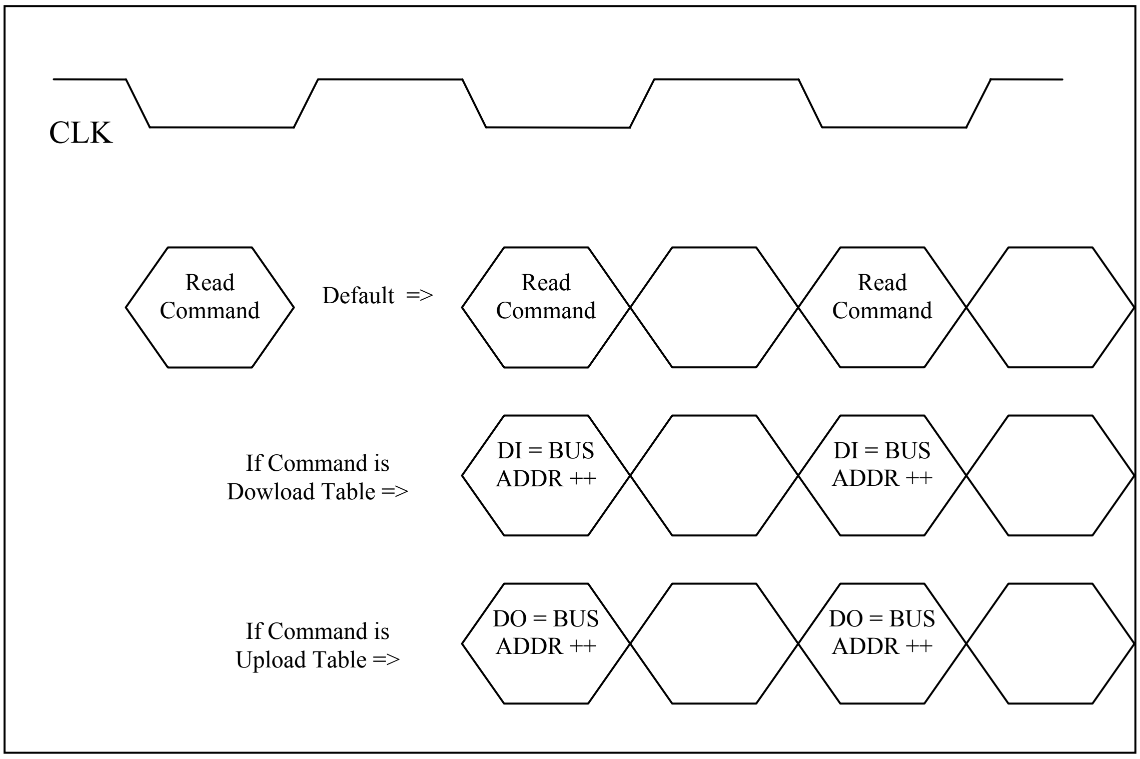


Figure 5-3: FPGA timing

As can be seen from the Block RAM timing diagram in **Figure 4-4**, the Block RAM only samples the input pins at the rising edge of the clock. The minimum hold time for the clock is 1.9ns and the largest minimum setup time for any of the inputs is 2.9ns (EN). If the clock had a period of at least 6ns, then it is possible to configure the setup on the falling edge and the Block RAM would do the read/write on the rising edge.

5.3.2 Colour Lookup

There were several options for the implementation of the colour lookup. The Lookup Table is stored in Block RAM and can only be accessed through the Block RAM's interface.

The pixel's raw YUV data and the colour for that pixel can be stored as either Block RAM or as a signal.

There are two methods to extract the lookup table from the Block RAM. One is to use the same input and output buses to retrieve the table. If the Block RAM is not used after storage of the Lookup Table, this method would be appropriate.

The other method is to use Dual Port RAM. This way, the SH4 can read from the Block RAM while the FPGA is writing into the Block RAM. Disadvantages of Dual Port RAM are that half the accessible memory is lost and a second clock signal has to be established.

The method that is currently being implemented is to only store the Y data and rout the U and V straight into the address bus. Therefore, at the next clock cycle, the indexed element is retrieved from the DO bus. The Y data and the lookup table elements for U and V are stored as signals. The pseudo code is shown in **Figure 5-4.** The timing of the code is shown in **Figure 5-5**.

Implementation is simply done by adding another state to the command process.

```
if counter = 5
        addr_bus = 0;
        counter = 0;

elsif State = ColourLookup
        switch counter
                case 0 : Y = in_bus; // First byte is Y
                case 1 : addr_bus = in_bus; // Load U to read.
                case 2 : U = out_bus; // Take element from lookup table
                         addr_bus = in_bus // Load V to read.
                case 3 : V = out_bus; // Take element from lookup table
                // This step calls a function returns the bitwise AND of
                // U and V. Signals don't update till after the process is
                // finished in VHDL.
                         temp_var = bit_and(U,out_bus)
                case 4 :  if Y > YMAX
                                pixel = WHITE;
                         elsif Y < YMIN
                                pixel = BLACK;
                        else // Do colour lookup.
                                switch temp_var
                                        case "00000001"
                                                pixel = RED;
                                        case "00000010"
                                                pixel = YELLOW;
                                        case "00000100"
                                                pixel = GREEN;
                                        case "00001000"
                                                pixel = BLUE;
                                        case others
                                                pixel = NOTHING;
                                end switch;
                        end if;
        end switch;
        counter ++; // Increment counter
```

Figure 5-4: Colour Lookup

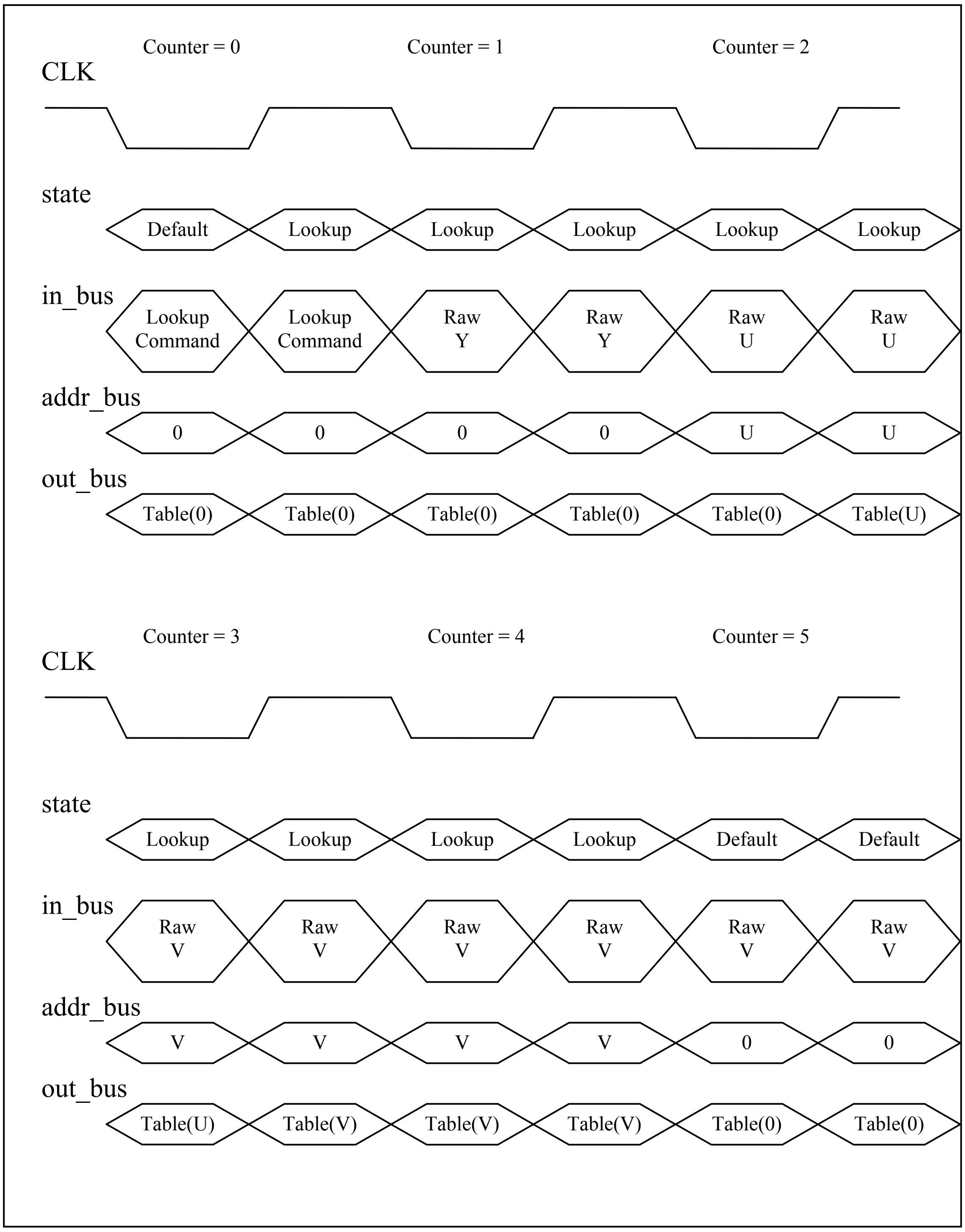


Figure 5-5: Colour Lookup Timing

6 Testing and Analysis

6.1 Test Image

Figure 6-1 shows a RGB test image. **Figure 6-2** shows the YUV equivalent.

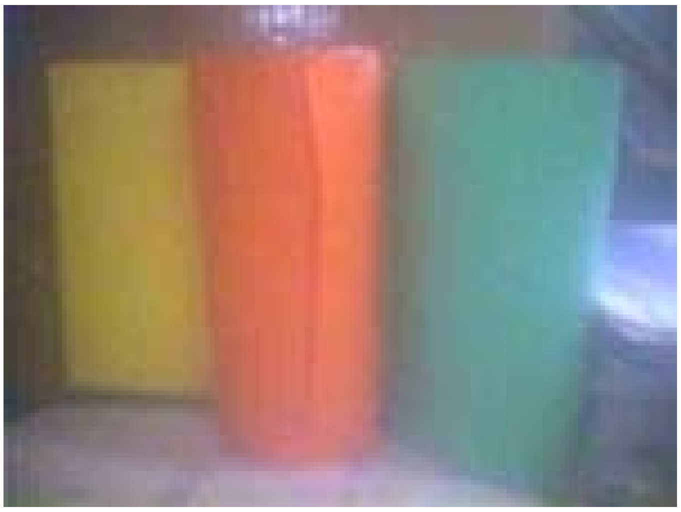


Figure 6-1: RGB Image

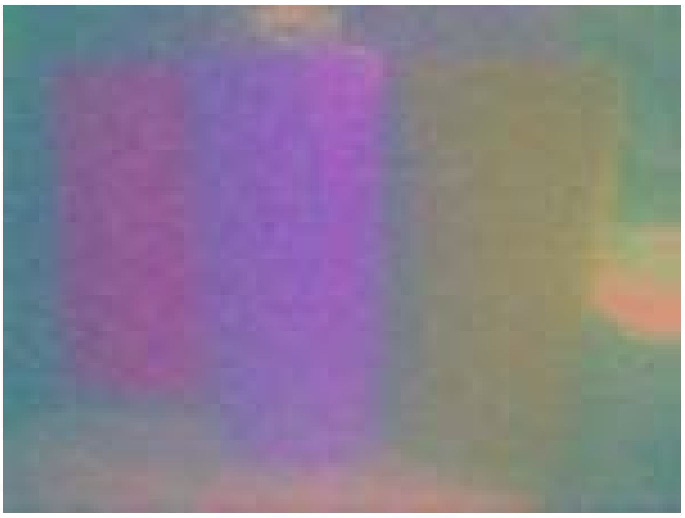


Figure 6-2: YUV image

6.2 Lookup Table

Figure 6-3 shows the test Lookup Table used. **Figure 6-4** shows the Bruce's Lookup Table equivalent. The lookup table is not an accurate representation of the real colours i.e. what is blue in the Lookup Table may not be blue in RGB space. The table was created from David Prasser's GurooView program and simplified in Matlab [14]. The orange tube is chosen as the "RED" colour, the yellow book is chosen as the "YELLOW" colour, the green tube is chosen as the "BLUE" colour and green is chosen from a random spot.

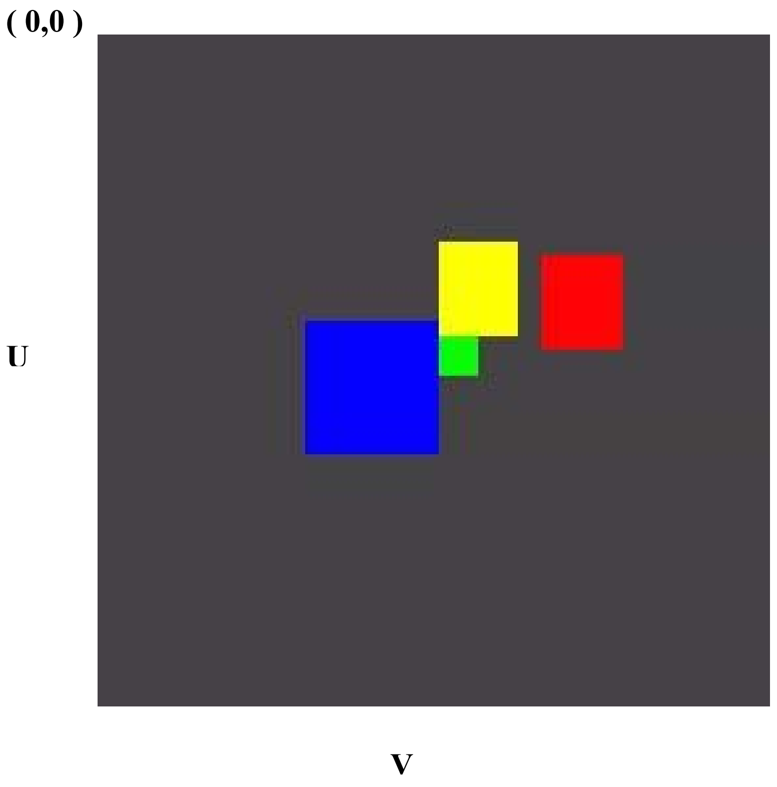


Figure 6-3: Normal Lookup Table

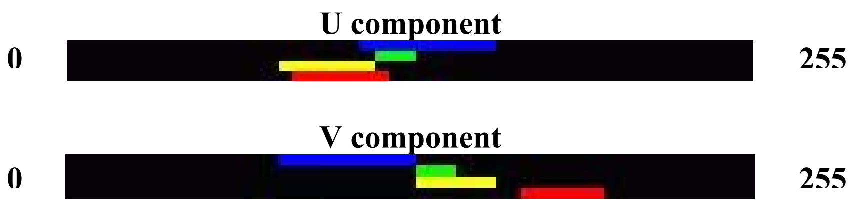


Figure 6-4: Bruce's Lookup Table

6.3 Colour Coded Image

Figure 6-5 shows the image after the colour lookup. As mentioned earlier, the Lookup Table is not accurate and only serves to test the functionality of the FPGA code. As can be seen from the outputted image, yellow, and red and blue have been accurately identified (remembering the green tube is considered blue).

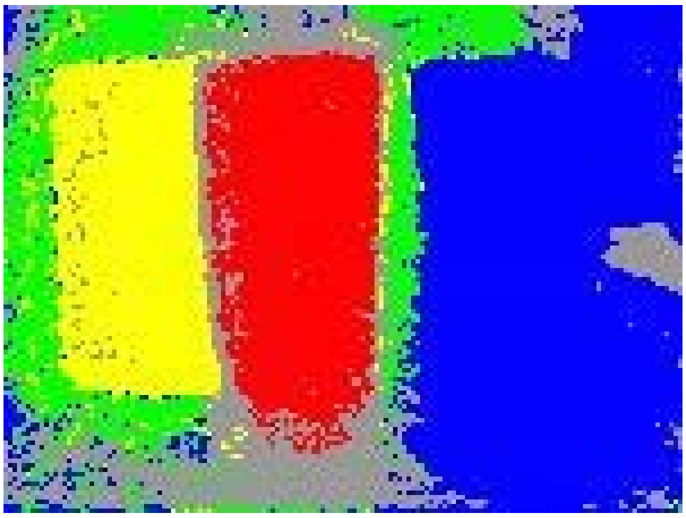


Figure 6-5: Colour Coded Image

6.4 FPGA Resource Usage and Timing

The trimmed Synthesis and Map Reports are located in Appendix C. A summary of the memory usage is outlined in **Figure 6-6**.

```
Design Summary
--------------
  Number of errors:              0
  Number of warnings:            2
  Number of Slices:              183      out of   1,200    15%
  Number of Slices containing
    unrelated logic:              0       out of   183      0%
  Number of Slice Flip Flops:    124      out of   2,400    5%
  Total Number 4 input LUTs:     292      out of   2,400    12%
    Number used as LUTs:         259
    Number used as a route-thru: 33
  Number of bonded IOBs:         161      out of   176      91%
    IOB Flip Flops:              4
  Number of Block RAMs:          1        out of   10       10%
  Number of GCLKs:               1        out of   4        25%
Total equivalent gate count for design:  19,466
Additional JTAG gate count for IOBs:  7,728
```

Figure 6-6: Memory Summary

Figure 6-7 shows the timing summary.

```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 17.686ns (Maximum Frequency: 56.542MHz)
   Minimum input arrival time before clock: 5.440ns
   Maximum output required time after clock: 12.734ns
   Maximum combinational path delay: 13.230ns
```

Figure 6-7: Timing Summary

6.5 SH4 Timing

The timing of the SH4 is far slower than the FPGA. The external clock for the FPGA is implemented through the clocking of GIO PIN 10. A toggling function has been implemented to change the state of the pin. **Figure 6-8** shows the function.

```
void toggle_clock(){
            if (low){
                        gio_set_bit_h(GIO_PORTA, GIO_PA10);
            low = 0;
            }
            else{
                        gio_set_bit_l(GIO_PORTA, GIO_PA10);
            low = 1;
            }
}
```

Figure 6-8: Toggle

The time it takes to execute this function has been measured to be ≈ 1.3µs. This is a very long and the time it takes to transmit a pixel and receive back the colour is ≈ 1.13ms. **Table 6.1** shows the execution times.

Function	Execution Time
Toggle Clock	1.3µs
Send Pixel	940µs
Get Colour	192µs
Colour Lookup of 150x112 Image	29.56s

Table 6-1: SH4 timing

6.5.1 Improving the Timing

The minimum period for the External Clock is 17.68ns. The current period is 2.6µs. The clocks period can be improved by setting the clock high and low manually. The speed that this occurs is 280ns. This is the maximum limit at which the SH4 can toggle a GIO pin. To get faster times, a dedicated clock pin will have to be used (e.g. the BSC clock). **Table 6.2** shows the timing when this new clocking is implemented.

Function	Execution Time
Toggle Clock	280ns
Send Pixel	4.4µs
Get Colour	900ns
Colour Lookup of 150x112 Image	294ms

Table 6-2: New SH4 Timing

Timing can also be improved by changing the storage of the colours. 4 bits can represent up to 16 different colours - which is plenty for the current implementation. Therefore, two pixels can be transferred in one read from the FPGA. This can cut the transfer time in half.

6.6 Analysis of Results

The minimum period for the External Clock is 17.68ns. The SH4 can only currently achieve ten times that speed using a GIO pin. The time it takes to send a pixel to the FPGA from the SH4 is not important. This is because the eventual implementation will have the OV7620 sending the pixel to the FPGA, not the SH4. The execution time that is more important is the time it takes to get the pixel's colour from the FPGA. With current implementation, that time is ≈ 900ns.

The OV7620's clock period for 16 bit data is 74ns when the system clock is 27MHz. The time it takes to transmit a pixel in 4:2:2 would be 296ns. In the time it takes to read a colour back from the FPGA (450µs using 4 bit data), approximately 1.5 pixels would have been sent.

The time it takes the OV7620 to transmit a 150x112 image is ≈ 5 ms. A 360x270 image (**Table 4-2: Projected Memory Usage**) is ≈ 28ms and a full 640x480 image is ≈ 90ms. The total processing time for David Prasser's code is shown in **Table 6-3** [3].

Task	Time (ms)
Colour Detection	28
Erosion	22
RLE	49
Grouping	Negligible
Analysis	Negligible
Edge Detection and Summation	25
Edge Analysis	Negligible
Total Time	124ms

Table 6-3: Processing Time

Currently, the Colour Detection takes 764ms for a 240x180 image. The bulk of this consists of the SH4 transmitting the image to the FPGA for processing. If it is only retrieving the colour from the FPGA, then it would take approximately 19.44ms – which, as expected, faster than processing the image on the SH4 itself. This proves that even moving the Colour Lookup onto the FPGA would speed up the processing time. The retrieval code is not even optimised because it retrieves 5 types of data: Pixel Colour; Y colour; Lookup data of U; Lookup data of V and the bit AND of U and V.

If the code is optimised a bit more, it is possible to match the pixel rate of the OV7620. The FPGA code will have to be modified first to complete in 4 clock cycles instead of the current implementation of 5. If optimisation of the SH4 code is not enough, then increasing the bus width is another solution. The current bus width is 8 bits due to the constraints on the GIO pins. If a larger bus is used, i.e. the BSC's 32 bit data bus, then it is definitely possible to match the pixel clock. The FPGA can process a pixel faster than the rate at which a pixel is transferred (provided it operates in 4 cycles). A 16 bit bus (4 pixels) is sufficient. If 12 GIO pins can be used as the data bus, then it is possible to match the pixel clock as well.

7 Evaluation of Project

It is necessary to evaluate the planning of the project and the decisions that have been made throughout the project. This will provide feedback on whether the assumptions made at the start were correct and valid as well as whether the actions and decisions made throughout the project were correct and justified. The ultimate aim of an evaluation is so future projects do not make the same mistakes.

7.1 Evaluation of Project Goals

- Were the Project's Goals achievable?

The initial project goals (Section 3-2) were achievable provided the assumption that the camera was operational and did not require much work to configure was correct. This was not the case.

The project goals made afterwards (**Figure 3-2**) were achievable and have been partly achieved. The Colour Lookup did managed to get moved onto the FPGA and the RLE could be moved as well if there was more time.

7.2 Evaluation of Project Schedule

The project schedule is fundamental to the project and if followed, should lead to the completion of the project. This, of course, is provided that it is properly constructed and is reviewed and evaluated regularly.

The initial project schedule (**Figure 3-1**) was correct based on the assumptions made at that time. Unfortunately, the setup of the compiler (scheduled for 1 day) took weeks, which pushed back the rest of the project. However, Ho Wong still thought that the project could be completed in time but this was not the case because the schedule never got past the configuration of the camera.

7.3 Evaluation of Risk Management

A risk assessment was done in the Progress Report [9] but; unfortunately, the risks were not adequately defined. A time limit should have been set such that if the camera was not working by that date, then it would be abandoned and the focus would be moved elsewhere.

The risk that the camera was not working did not fully eventuate until all possible actions were tried (e.g. direct wire connection, checking both FPGA and SH4 code for errors and checking PCB). It was very late (August) by the time it was realised that the cameras were not working after all options were exhausted. It was assumed that the problem lay elsewhere because of the initial assumption that Mark Chang had a working camera board.

The assumption made that the configuration of the camera was integral to the start of the other parts of the project was incorrect. It has turned out that a lot can be done on the FPGA itself without the need of a camera.

7.4 Recommendations for future improvement

Recommendations for future improvement are:

- Clearly define the risks and actions to be taken in the event they occur.
- Clearly define the time it takes to complete tasks so that the Critical Path methodology can be used. This will provide early warning when things start going wrong so changes to the schedule and even the goals can be made.
- Place milestones in the schedule. These can form part of the risk assessment.

8 Future Work

8.1 Data transfer between the SH4 and FPGA

The use of the SH4's BSC is advisable. Dual Port RAM will have to be implemented on the FPGA but there are drawbacks.

If Block RAM is not stacked, then only half the available memory is accessible. The best solution is to have a port width of 16 bits. This will allow 4 pixels to be sent in one read. The draw back is only 128 elements can be stored [15]. This equates to 512 pixels. A whole line can be easily stored.

A possible implementation is described by **Figure 8-1** .

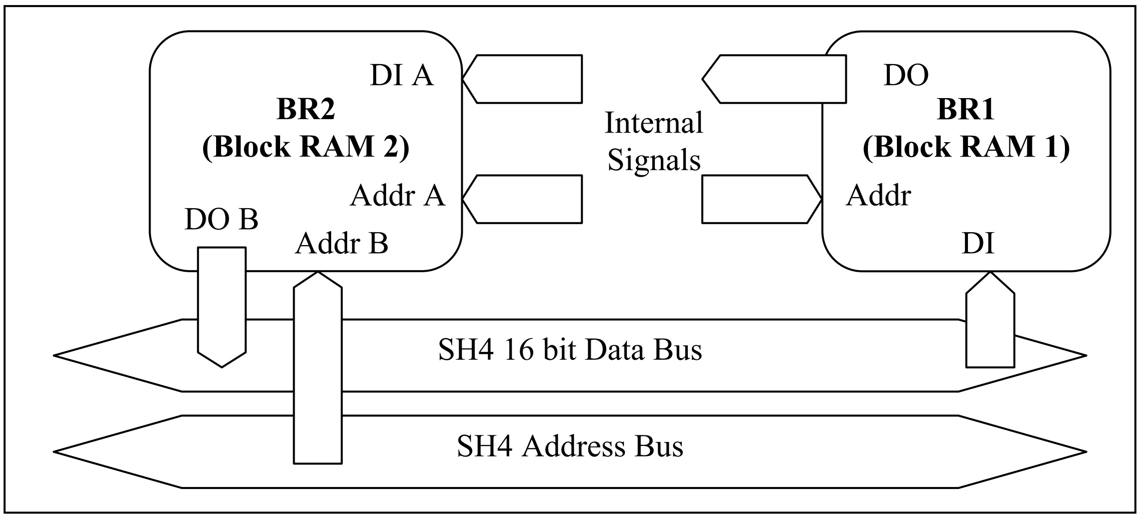


Figure 8-1: Dual Port RAM

BR1 is a Single Port RAM and BR 2 is a Dual Port RAM. The DI of BRl is from the SH4 (this is how the Lookup Table is stored). The DO of BR1 is used by internal signals to retrieve Lookup Table data. DI A of BR2 comes from internal signals storing pixel colours. DO B of BR2 is the other port that only reads stored data.

8.2 Integration with OV7620

The current design on the FPGA is through the use of only a single process. This was because it was easier to understand. The special feature of FPGAs is that a lot of things can run concurrently - unlike normal microprocessors which run sequentially.

The colour lookup can be taken out of the Command Process and made into a separate process with the OV7620's pixel clock (PCLK) as its sensitivity list. A switch will have to be added (similar to the one used to switch output buses) for the address bus and CLK.

The OV7620 timing diagram is shown in **Figure 8-2** [7].

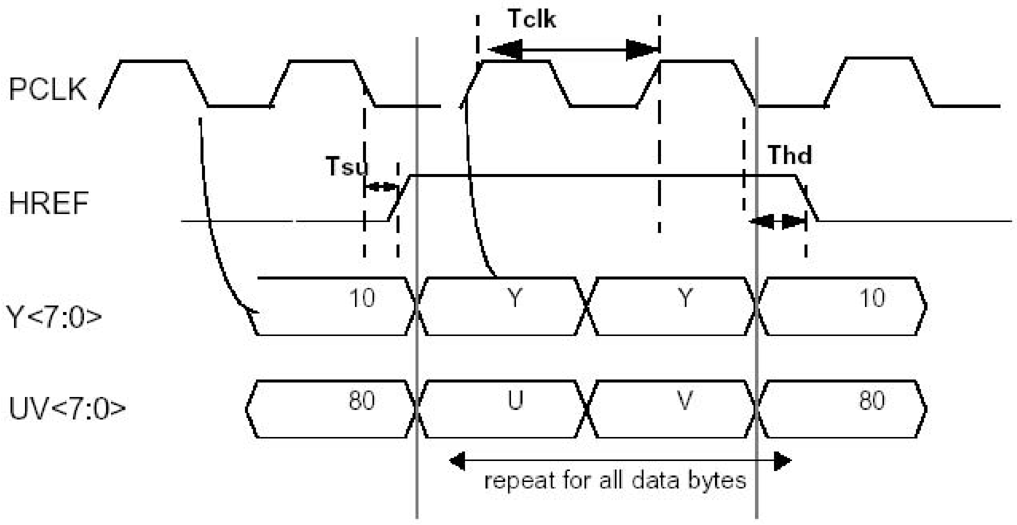


Figure 8-2: OV7620 Timing

Tclk is pixel clock period. When OV7620 system clock is 27MHz, Tclk=74ns for 16 Bit output;Tclk=37ns for 8 Bit output. Tsu is HREF set-up time, maximum is 15 ns; Thd is HREF holdtime, maximum is 15 ns.

The colour lookup process will have to be changed such that it fits inside 4 clock cycles. The comparing of the bit ANDed data can also be made into a separate process because there is no shared signal being written to.

From **Figure 8-2**, the storage of the pixel's data will have to occur on the rising edge of PCLK. The CLK signal to the Block RAM will have to be inverted such that the falling edge of PCLK will trigger the sampling of the inputs and the retrieval of the Lookup Table data.

Figure 8-3 shows a pseudo VHDL implementation.

```
-- Lookup process. It is run only when tmp_var changes state. This should happen straight after the
-- Pixel Store process has finished.

Colour Lookup : process (tmp_var)
begin
        if conv_integer(unsigned(y_colour)) > INTEN_MAX then
                pixel_colour <= WHITE;
        elsif conv_integer(unsigned(y_colour)) < INTEN_MIN then
                pixel_colour <= BLACK;
        else
                case temp_var is
                        when "00000001" => pixel_colour <= RED;
                        when "00000010" => pixel_colour <= YELLOW;
                        when "00000100" => pixel_colour <= GREEN;
                        when "00001000" => pixel_colour <= BLUE;
                        when others => pixel_colour <= OTHER;
                end case;
        end if;
end process Colour Lookup;

Pixel Store : process (PCLK)
begin
        if counter = 4 then
                counter <= 0;
                addr_bus <= 0;

        elsif rising_edge (PCLK) then
                if counter = 0 then
                        y_colour <= gio_in_bus; -- Store Y
                elsif counter = 1 then
                        gio_addr_bus <= '0'& gio_in_bus; -- Rout U into address bus
                elsif counter = 2 then
                        u_colour <= ram_out_bus; -- Retrieve Lookup Table data
                        gio_addr_bus <= '0' & gio_in_bus; -- Rout V into address bus
                elsif counter = 3 then
                        v_colour <= ram_out_bus; -- Retrieve Lookup Table data
                        tmp_var <= vectorAnd(u_colour,ram_out_bus); -- BIT AND U & V
                else;
                end if;
        else;
        end if;
end process Pixel Store;
```

Figure 8-3: OV7620 Integration

8.3 Further Pre-processing

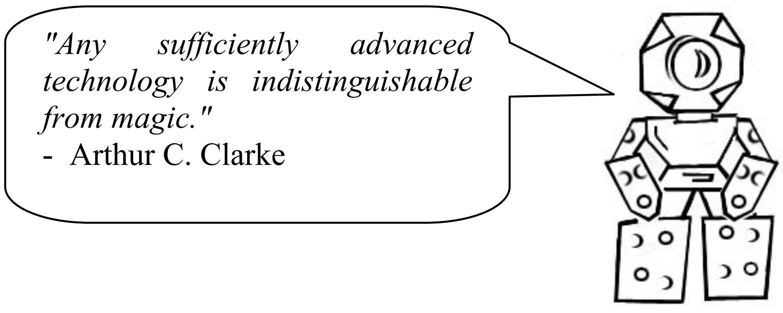
If erosion is ignored, RLE can also be moved onto the FPGA. The constraint; however, will be on the available memory. Each RLE component contains the following data members [3]:

- **begin:** The x coordinate corresponding to the beginning of a run.
- **end:** The x coordinate of the end of the run.
- **y:** The y coordinate of the run.
- **colour:** The colour of the pixels in the run.
- **tag:** The number of the run counting from the top left corner.
- **blob pointer:** A pointer to a blob structure.

In a worse case scenario, every pixel is a RLE element. If this was the case, then the memory required would be:

begin:1byte, end:1byte, y:1byte, colour:4bits, tag: 3bytes blob pointer:2bytes
= 12.5bytes. or 13 bytes (98bits) rounded up.

For a 360x270 image, to store a line of RLE's, 4.4Kbytes would be required. It is possible to store this in a single block of RAM but Single Port will have to be used and arbitration will have to be implemented. An easier way is to stack a couple of blocks and use Dual Port. A final alternative is to store it in Distributed RAM, because currently only 15% of the available memory is being used by the FPGA. The bus widths can be manually determined if a Distributed RAM driver is made [16]. It is even possible to make a 32bit Dual Port RAM with Distributed RAM.


"Any sufficiently advanced technology is indistinguishable from magic."
- Arthur C. Clarke

9 Conclusion

Although the thesis started late due to complications at the start, some progress has been achieved. The problems that occurred early in the year were unavoidable and there were no other actions that could have been taken at that time to resolve them. It is; therefore, simple bad luck that the thesis got off to a slow start.

There was only enough time to move the Colour Lookup onto the FPGA. More could've been accomplished with more time. Details on the future work needed have been documented in this thesis and the next student to work on GuRoo's vision system would be properly informed on the current status and ideas that never got implemented due to the lack of time.

This thesis has managed to improve upon the image processing done by David Prasser's vision software. The time it would take to process a 240x180 image is estimated to be about 19.44ms as opposed to 28ms with David Prasser's code. The pixel clock can be matched by optimising the code on the FPGA or using a larger data bus.

The impact of the results of this thesis is that the processing time for an image can be decreased and hence increase the frame rate. Also, larger images can be used because the raw image is not stored on the SH4.

Future work would have the RLE put onto the FPGA as well. It is not impossible that the image processing can match the rate at which images are sent from the camera. If this is the case, then a high frame rate of 35fps for a 360x270 image can be achieved.

Bibliography

[1] Damien Kee, *GuRoo Electro – Mechanical Design,* http://www.itee.uq.edu.au/~damien/_guroo/specification.htm, 23 September 2002, accessed 10 September 2002.

[2] Robocup2002, *Humanoid League 2002 Rule,* http://www.robocup.org/regulations/humanoid/rule_humanoid.htm. 2 September 2002, accessed 10 September 2002.

[3] David Prasser, *Vision Software for a Humanoid Soccer Robot*, Undergraduate Thesis, University of Queensland, 2001.

[4] Hitachi Ltd., *SH7750 SeriesSH7750, SH7750S Hardware Manual* http://www.hitachi-eu.com/hel/ecg/products/micro/32bit/sh_4.html, Revision 5.0, 19 June 2001,

[5] Xilinx Inc, *Spartan II Product Flyer*, http://www.xilinx.com/products/spartan2/

[6] David Bourgin, *Colour space FAQ*, http://www.neuro.sfc.keio.ac.jp/~aly/polygon/info/color-space-faq.html, 28 September 1994.

[7] OmiVision Technologies Inc, *OV7620 Product Specifications* http://www.ovt.com/cc7620.html, Version 2.1 July 10, 2001.

[8] Andrew Blower, *Development of a Vision System for a Humanoid Robot*, Undergraduate Thesis, University of Queensland 2001.

[9] Ho Wong, *Vision Hardware for a Humanoid Robot*, Undergraduate Thesis Progress Report, University of Queensland, 2002.

[10] Ho Wong, *Vision Hardware for a Humanoid Robot*, Undergraduate Thesis Seminar Presentation, University of Queensland, 2002.

[11] Bruce, Balch, Veloso, *Fast and Inexpensive Colour Segmentation for Interactive Robots,* Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000.

[12] Xilinx Inc, *Spartan-II 2.5V FPGA Family: Introduction and OrderingInformation,* Xilinx Datasheets,http://www.xilinx.com/partinfo/ds001.htm, v2.3 November 1, 2001.

[13] Xilinx Inc, *Spartan II 2.5V FGPA Family : Functional Description*, Xilinx Datasheets,http://www.xilinx.com/partinfo/ds001.htm, v 2.1 March 1, 2001.

[14] *Image Processing ToolBox 3,* http://www.cat.csiro.au/cmst/staff/pic/vision-tb.html, accessed 18 September 2002.

[15] Xilinx Inc, *Using Block SelectRAM+ Memory in Spartan-II FPGAs,* Xilinx Applications,http://www.xilinx.com/apps/sp2app.htm, v1.1 11 December , 2000.

[16] Xilinx Inc, *Implementing Memory*, Xilinx Applications, http://toolbox.xilinx.com/docsan/xilinx4/data/docs/sim/fpgahdl10.html, accessed 10 September 2002.

Appendix A: Colour Images

Figure A-1: Spartan II xc2s100 FPGA

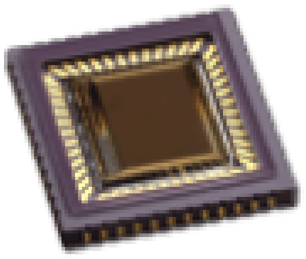


Figure A-2: OmniVision OV7620

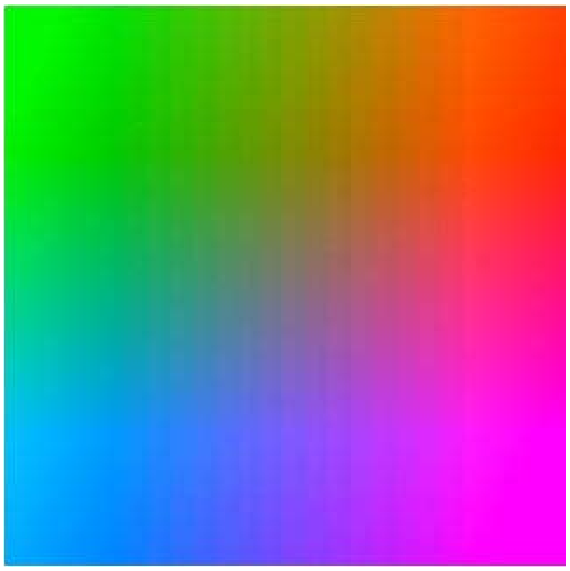


Figure A-3: Colour as a function of U and V

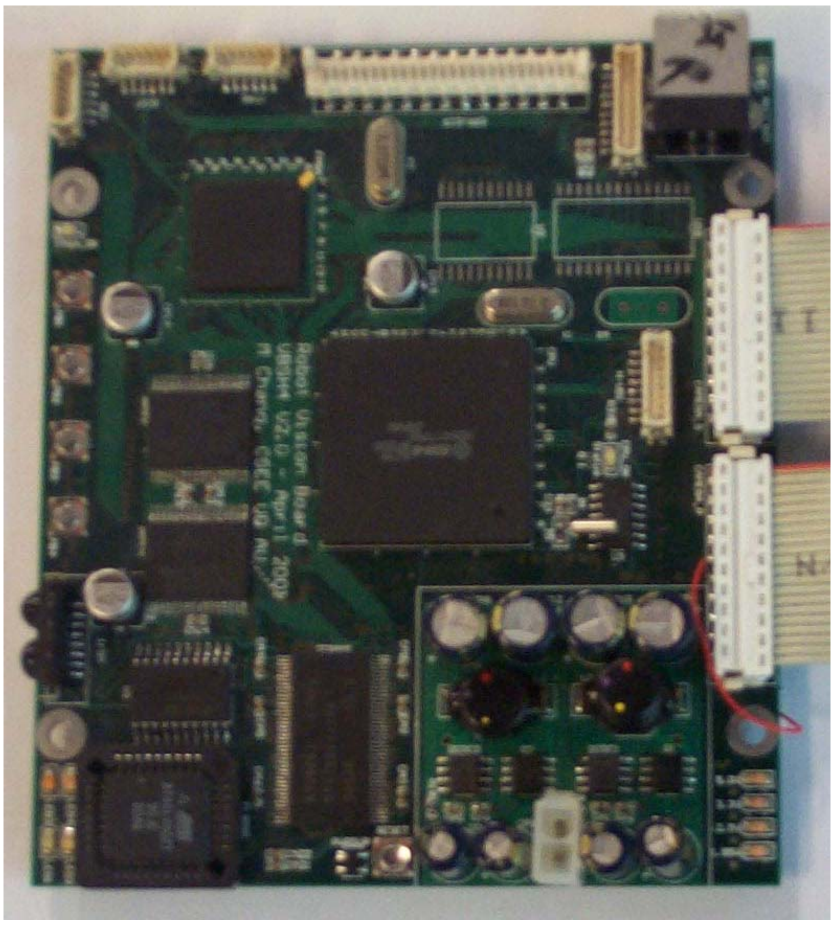


Figure A-3: Vision Board

BAppendix B: Users Manuals

B.1 Software Implementation

B.1.1 Boot Loader

The EEPROM holds the boot program for the SH4. Loading of the SH4 is done through Mark Chang's Ploader program. This has to be run under a Unix environment. However; David Prasser has a modified Ploader that runs under Windows.

B.1.2 FPGA

The FPGA program has to be first converted into a header file first then compiled with the rest of the code – refer to the User's Manual.

B.1.3 Memory

The SH4 program is stored in P2 and the SRAM is mapped to Area 4 while the 2 SDRAMs are mapped to Areas 2 and 3 respectively. Base RAM address is 0xB0000000. Currently, 4Kbyte of memory is allocated for the main program and the rest is used for storage of images.

B.1.4 GIO Ports

GIO ports 1 to 8 have not been allocated any special functions. Ports 8 to 11 are used for the camera. Ports 12 to 15 are for each of the four LEDs and ports 16 to 20 are used for the FPGA loading.

B.1.5 Interrupts

There is currently no external interrupt enabled. Only SCI receive and transmit interrupts are currently enabled. Provisions have been made for SCIF interrupts but are currently not being used.

B.1.6 DMAC

DMA has been set up for the camera’s data and also for transmission on SCI. For the camera’s data, DMA is activated by an external DMA pin. SCI DMA is activated as part of the on-board peripheral setting.

B.2 Users Manual

B.2.1 Initial Boot Up

When the SH4 is powered up initially, all the Voltage Control LEDs (5.0, 3.3, 2.5 & 1.95) should be lit and the SH4 will do a ‘LED dance’. When the boot program is finished, LED 3 will be lit. If there other LEDs lit, then there is an error so check all the connections and make sure nothing is shorted together accidentally.

B.2.2 Cross-compiler

The SH4 program is compiled using GNU GCC. Refer to Masahiro Abe’s manual on building a cross-compiler for the SH4. Mark Chang’s compiler is based on Intel architecture. A new cross-compiler will have to be built if the code is to be compiled on a different computer system. E.g. Sun’s Spark computers.

B.2.3 Loading of Program

The FPGA bit file has to be converted into a header file first. This header file is then compiled with the rest of the code. Folder /fpga in /viper has the Makefile for it. Copy the binary .rbt file into that directory and run make. This will convert the FPGA program into fpgadata.h. If a new fpga program is not required, do not include fpgadata.h in the main.c.

The makefile in /vbsh4 will compile everything and produce a vbsh4.bin binary file. After compilation, ploader will upload the program into the SH4. Use of ploader is simple. The command is: ploader <COM PORT> <File>. E.g. ‘ploader COM1 vbsh4.bin’.

After the program is loaded, the “Heart Beat” should be activated. The “Heart Beat” is the continuous blinking of LED4, hence the term “Heart Beat”. This is just the TMU inverting LED4 every time it over-runs.

B.2.4 Downloading of data

Due to the malfunction of the camera, images have to be directly loaded onto the SH4. This is done through David Prasser’s dloader. Three options currently exist for dloader: loading of an image; loading of a lookup table and loading of Bruce’s Lookup Table. The file format of all three is ‘.raw’ raw binary file. Usage is simple. The command line is ‘dloader <COM PORT> <FILE> <COMMAND>’ where COMMAND is either: ‘i’ for image; ‘t’ for a normal lookup table or ‘n’ for Bruce’s Lookup Table. E.g. ‘dloader COM1 image.raw i’.

B.2.5 Uploading of data

Data can be uploaded from the SH4 through David Prasser’s uloader. Similar to the dloader images (unprocessed and processed) can be uploaded as well as the lookup tables and blob information. Usage is also similar to dloader. ‘uloader <COM PORT> <FILE> <COMMAND>. Where COMMAND is ‘i’ for image, ‘s’ for processed image, ‘t’ for normal lookup table, ‘n’ for Bruce’s Lookup Table, ‘b’ for blob information and ‘o’ for object information. For the images, the output format is ‘.ppm’(Portable Pixel Map). For the lookup table and colour coded image, the output format is ‘.pgm’(Portable Gray Map). E.g. ‘uloader COM1 processedImage.ppm i’.

B.2.6 Error Codes

Mark Chang has made error codes for the camera. **Table 10** lists them. The error is identified by the number of pulses of LED2 to LED1. E.g. if LED2 pulses twice for every pulse of LED1, then the error is SCCB CAM.

Error Type	No. pulses
ERROR_CAM_SCCB	2
ERROR_CAM_SET	3
ERROR_CAM_DMA	4
ERROR_CAM	5

Table B-1: Camera Error Codes

Appendix C: CODE

C.1 FPGA CODE

```
-- module: vbsh4.vhd
-- Modified by Ho Wong
--          21/9/02
-- Final Version 14/10/02

-- This is the top level module that ties all sub-modules for vbsh4 together

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity vbsh4 is
            Port (
--           CLOCK IN
--          PIN_U_CLKOUT :          in std_logic;
--          PIN_TCLK :              in std_logic;
--          PIN_I_CLKOUT0 :         in std_logic;
--          PIN_I_CLKOUT1 :         in std_logic;

--          CPU
            PIN_RD :                in std_logic;
            PIN_RDWR :              in std_logic;

            PIN_BS :                in std_logic;
            PIN_RDY :               in std_logic;
            PIN_SCK :               in std_logic;
            PIN_CS1 :               in std_logic;
            PIN_CS2 :               in std_logic;
            PIN_WE0 :               in std_logic;
            PIN_WE1 :               in std_logic;
            PIN_WE2 :               in std_logic;
            PIN_WE3 :               in std_logic;

            PIN_DRAK0 :             in std_logic;
            PIN_DRAK1 :             in std_logic;
            PIN_DREQ0 :             out std_logic;
            PIN_DREQ1 :             in std_logic;
            PIN_DACK0 :                 in std_logic;
            PIN_DACK1 :                 in std_logic;

            PIN_NMI :                   in std_logic;
            PIN_IRL0 :                  out std_logic;
            PIN_IRL1 :                  in std_logic;
            PIN_IRL2 :                  in std_logic;
            PIN_IRL3 :                  in std_logic;
            PIN_MRESET :                in std_logic;

-- ADDRESS BUS
            PIN_ADDRBUS :   in std_logic_vector(18 downto 0);

-- DATA BUS
            PIN_DATABUS :   inout std_logic_vector(31 downto 0);

-- GENERAL I/O
            PIN_GIO :       inout std_logic_vector(7 downto 0); -- My databus.

                        -- sccb 4 wire - 1 clock, 2 data, 1 rd/wr
--          PIN_GIO8 :      in std_logic;       -- sccb clock
--          PIN_GIO9 :      in std_logic;       -- sccb data
--          PIN_GIO10 :     out std_logic;
--          PIN_GIO11 :     in std_logic;       -- sccb data direction control

            -- sccb 3 wire - 1 clock, 1 data, 1 rd/wr
            PIN_GIO8 :      in std_logic;       -- sccb clock
            PIN_GIO9 :      inout std_logic;    -- sccb data
            PIN_GIO10 :     in std_logic;       -- sccb data direction control// Hijack this pin temporarily.
            PIN_GIO11 :     in std_logic;       -- pclk interrupt connection

            PIN_GIO12 :     in std_logic;       -- led 1
            PIN_GIO13 :     in std_logic;       -- led 2
            PIN_GIO14 :     in std_logic;       -- led 3
            PIN_GIO15 :     in std_logic;       -- led 4

-- CAMERA
            PIN_CAM_Y :                 in std_logic_vector(7 downto 0);
            PIN_CAM_UV :                in std_logic_vector(7 downto 0);

            PIN_CAM_SIO1 :              out std_logic;
            PIN_CAM_SIO0 :              inout std_logic;

            PIN_CAM_PCLK :              in std_logic;
            PIN_CAM_HREF :              in std_logic;
            PIN_CAM_VSYNC :             in std_logic;

            PIN_CAM_PWDN :              out std_logic;
            PIN_CAM_CHYSNC :            out std_logic;

-- INTERFACE
            PIN_I_ADDR :                in std_logic_vector(2 downto 0);
            PIN_I_IOR :                 in std_logic;
            PIN_I_IOW :                 in std_logic;

            PIN_I_CS0 :                 in std_logic;
            PIN_I_CS1 :                 in std_logic;
            PIN_I_PPCS :                in std_logic;
            PIN_I_ECPCS :               in std_logic;

            PIN_I_TC :                  in std_logic;
            PIN_I_PDRQ :                in std_logic;
            PIN_I_PDACK :               in std_logic;
            PIN_I_RESET :               in std_logic;
```

```
        PIN_I_PINTR :           in std_logic;
        PIN_I_IOCHRDY :         in std_logic;

        PIN_I_TXRDY0 :          in std_logic;
        PIN_I_TXRDY1 :          in std_logic;
        PIN_I_RXRDY0 :          in std_logic;
        PIN_I_RXRDY1 :          in std_logic;
        PIN_I_INTRPT0 :         in std_logic;
        PIN_I_INTRPT1 :         in std_logic;

-- USB
        PIN_U_SUSPEND :         out std_logic;
        PIN_U_INT :             out std_logic;
        PIN_U_EOT :             out std_logic;
        PIN_U_DMACK :           out std_logic;
        PIN_U_DMREQ :           inout std_logic;

        PIN_U_CS :              out std_logic;
        PIN_U_WR :              out std_logic;
        PIN_U_RD :      out std_logic;

        PIN_U_A0 :      out std_logic;
        PIN_U_DATA :    out std_logic_vector(7 downto 0);

-- EXPANSION PORT
        PIN_EXP :       out std_logic_vector(8 downto 0);

-- SERIAL CONTROL
        PIN_SCI_C1 :    out std_logic;  -- Hijacking these 4 pins for the LEDs.
        PIN_SCI_C2 :    out std_logic;
        PIN_SCIF_C1 :   out std_logic;
        PIN_SCIF_C2 :   out std_logic;

-- PUSH BUTTONS
        PIN_PB :        in std_logic_vector(4 downto 1);

-- LEDS
        PIN_LED :       out std_logic_vector(4 downto 1);

-- AUX
        PIN_AUX :       out std_logic);

end vbsh4;

architecture vbsh4_arch of vbsh4 is

-- CONSTANT DECLARATIONS

        -- Intensity ranges for Black and White
        constant INTEN_MAX : integer := 240;
        constant INTEN_MIN : integer := 50;
        -- Colours
        constant RED:           std_logic_vector(7 downto 0) := "00000001";
        constant YELLOW :       std_logic_vector(7 downto 0) := "00000010";
        constant GREEN:         std_logic_vector(7 downto 0) := "00000011";
        constant BLUE:          std_logic_vector(7 downto 0) := "00000100";
        constant WHITE :        std_logic_vector(7 downto 0) := "00000101";
        constant BLACK :        std_logic_vector(7 downto 0) := "00000110";
        constant OTHER:         std_logic_vector(7 downto 0) := "00000000";

-- SIGNAL DECLARATIONS

        signal led_lines:        std_logic_vector(4 downto 1);
        signal pb_lines:        std_logic_vector(4 downto 1);
        -- Data buses
        signal data_in_bus:         std_logic_vector(31 downto 0);
        signal data_out_bus:        std_logic_vector(31 downto 0);

        -- Direction control for PIN_GIO
        signal gio_dir:             std_logic := '1'; -- '1' = INPUT, '0' = OUTPUT

        -- Input buses
        signal gio_in_bus:          std_logic_vector(7 downto 0):="00000000";
        signal alternate_in_bus :   std_logic_vector(7 downto 0) := "00000000";
        signal ram_in_bus :         std_logic_vector(7 downto 0):= "00000000";
        signal in_switch :          std_logic := '0';

        -- Output buses
        signal gio_out_bus:         std_logic_vector(7 downto 0):="00000000";
        signal alternate_out_bus :  std_logic_vector(7 downto 0) := "00000000";
        signal ram_out_bus :        std_logic_vector(7 downto 0):= "00000000";
        signal out_switch :         std_logic := '0';

        -- Block RAM address bus
        signal gio_addr_bus:        std_logic_vector(8 downto 0);

        -- Block RAM pins
        signal gio_clk:             std_logic := '0';
        signal gio_we:              std_logic := '0';

        -- Y U V signals.
        signal y_colour:            std_logic_vector(7 downto 0);
        signal u_colour:            std_logic_vector(7 downto 0);
        signal v_colour:            std_logic_vector(7 downto 0);

        -- Counter
        signal counter:             integer := 0;

        -- Pixel Colour
        signal pixel_colour :       std_logic_vector(7 downto 0);

        signal s_data_out  :        std_logic;-- Data bus direction control.
        signal s_n_cam_pclk :       std_logic;

        -- sccb intermediate line - for sccb 3 wire
        signal s_sccb_w  :          std_logic;-- sccb data direction control
        signal s_sccb_wn :          std_logic;-- inverted sccb data direction control
        signal s_sccb_ms :          std_logic;-- sccb master to slave
        signal s_sccb_sm :          std_logic;-- sccb slave to master

        -- Some internal variables
        signal led1_intern   :      std_logic;
        signal led2_intern   :      std_logic;
        signal led3_intern   :      std_logic;
        signal led4_intern   :      std_logic;
        signal command_state :      integer := 0;       -- Command State
        signal address:             integer := 0;       -- Address
        signal connect_clk  :       std_logic;
        signal glob_clk     :       std_logic;          -- Clock
        signal cam_buffer   :       std_logic_vector(15 downto 0);
        signal temp :               std_logic_vector(7 downto 0);-- Temp Variable

        -- These attributes are necessary for the external clock configuration.
        attribute clock_buffer : string;
        attribute clock_buffer of glob_clk : signal is "ibuf";

-- COMPONENT DECLARATION
```

```
  component bufg
            port (i: in std_logic; o: out std_logic);
  end component;

component ibuf
            port (i: in std_logic; o: out std_logic);
end component;

component pb
            port (
                        pb_in : in std_logic;
                        data : out std_logic);
end component;

component LOGIC_74x74
             port ( DATA : in std_logic;
                        CLK : in std_logic;
                         nSET : in std_logic;
                         nCLEAR : in std_logic;
                         Q : out std_logic;
                        Qn : out std_logic);
end component;

component IOBUF
             port (
                        I : in STD_LOGIC;
                        T : in STD_LOGIC;
                        O : out STD_LOGIC;
                         IO : inout STD_LOGIC);
end component;

component OBUFT
            port (
                        I : in STD_LOGIC;
                        T : in STD_LOGIC;
                        O : out STD_LOGIC);
end component;

            ----- Component RAMB4_S8 -----
component RAMB4_S8
            -- synopsys translate_off
            generic (
            INIT_00:bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
            INIT_01 bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
            INIT_02 bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
            INIT_03 bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";

            -- synopsys translate_on
                        port (DI     : in STD_LOGIC_VECTOR (7 downto 0);
            EN     : in STD_logic;
            WE     : in STD_logic;
            RST    : in STD_logic;
            CLK    : in STD_logic;
            ADDR   : in STD_LOGIC_VECTOR (8 downto 0);
            DO     : out STD_LOGIC_VECTOR (7 downto 0));
            end component;

-- FUNCTION DECLARATION

            -- This function takes 2 8bit vectors A & B and bit ANDs the upper 4 bits of A with
            -- Lower 4 bits of B.
function vectorAnd (A,B : std_logic_vector(7 downto 0)) return std_logic_vector is
            variable C : std_logic_vector(7 downto 0);
```

```
begin
        for i in 0 to 3 loop
                C(i) := A(i+4) and B(i);
        end loop;
        C(7 downto 4) := "0000";
        return C;
end vectorAnd;

                                -- BEGIN --
begin

-- LED configuration
        PIN_LED(1) <= '0' when (PIN_GIO12 = '1' or led1_intern = '1')
                                else '1';
        PIN_LED(2) <= '0' when (PIN_GIO13 = '1' or led2_intern = '1')
                                else '1';
        PIN_LED(3) <= '0' when (PIN_GIO14 = '1' or led3_intern = '1')
                                else '1';
        PIN_LED(4) <= '0' when (PIN_GIO15 = '1' or led4_intern = '1')
                                else '1';

-- Rewiring for external clock from non-global pin.
        IBUF0     : ibuf port map (i => PIN_GIO10, o => connect_clk);
        BUFG0     : bufg port map  (i => connect_clk, o => glob_clk);


-- Setup Block ram
        RAM0 : RAMB4_S8 port map (DI => ram_in_bus,
                                 EN => '1',
                                 WE => gio_we,
                                 RST => '0',
                                 CLK => glob_clk,
                                 ADDR => gio_addr_bus,
                                 DO => ram_out_bus);

-- Switches
        ram_in_bus <= gio_in_bus when in_switch = '0'
                else alternate_in_bus;
        gio_out_bus <= ram_out_bus when out_switch = '0'
                else pixel_colour;

-- Main Process
        Command : process (glob_clk)
                variable tmp : integer;
        begin

        if falling_edge (glob_clk) then
                -- End of Data check.
                if address = 256 then
                -- Reset variables
                        address <= 0;
                        gio_we <= '0';
                        command_state <= 0;
                        gio_dir <= '1';

                -- End of Counter check
                elsif counter = 5 then
                        -- Reset variables
                        counter <= 0;
                        gio_we <= '0';
                        command_state <= 0;
                        gio_dir <= '1';
                        out_switch <= '0';
```

```
-- DEFAULT STATE
elsif command_state = 0 then
        address <= 0;
        gio_we <= '0';
        led2_intern <= '0';
        led3_intern <= '0';
        led1_intern <= '1';
        if gio_in_bus = "00001011"  then

        -- Download the table from SH4
                command_state <= 1;
                gio_dir <= '1';

        elsif gio_in_bus = "00010110" then

                -- Upload table to SH4
                command_state <= 2;
                gio_dir <= '0';

        elsif gio_in_bus = "00101100" then

                -- Download YUV colours from SH4
                command_state <= 3;
                gio_dir <= '1';

        elsif gio_in_bus = "01011000" then

                -- Upload Pixel Colour to SH4
                command_state <= 4;
                gio_dir <= '0'; -- output

        else
                command_state <= 0;
        end if;

-- STATE 1: Download Lookup Table from SH4
elsif command_state = 1 then
                gio_we <= '1';
                led1_intern <= '0';
                gio_addr_bus <= conv_std_logic_vector(address,9);
                address <= address + 1;

-- STATE 2: Upload Lookup Table to SH4
elsif command_state = 2 then
                gio_we <= '0';
                led1_intern <= '0';
                gio_addr_bus <= conv_std_logic_vector(address,9);
                address <= address + 1;

-- STATE 3: Download YUV data from SH4
elsif command_state = 3 then
                gio_we <= '0';
                led1_intern <= '0';
                led2_intern <= not led2_intern;
                if counter = 0 then
                        y_colour <= gio_in_bus; -- Store Y
                elsif counter = 1 then
                        gio_addr_bus <= '0'& gio_in_bus; -- Rout U into address bus
                elsif counter = 2 then
                        u_colour <= ram_out_bus; -- Retrieve Lookup Table data
                        gio_addr_bus <= '0' & gio_in_bus; -- Rout V into address bus
                elsif counter = 3 then
                        v_colour <= ram_out_bus; -- Retrieve Lookup Table data
                        temp <= vectorAnd(u_colour,ram_out_bus); -- BIT AND U & V
```

```
                                        -- Processs signals
                                        elsif counter = 4 then
                                                if conv_integer(unsigned(y_colour)) > INTEN_MAX then
                                                        pixel_colour <= WHITE;
                                                elsif conv_integer(unsigned(y_colour)) < INTEN_MIN then
                                                        pixel_colour <= BLACK;
                                                else
                                                        led3_intern <= '1';
                                                        case temp is
                                                                when "00000001" => pixel_colour <= RED;
                                                                when "00000010" => pixel_colour <=
YELLOW;
                                                                when "00000100" => pixel_colour <= GREEN;
                                                                when "00001000" => pixel_colour <= BLUE;
                                                                when others => pixel_colour <= OTHER;
                                                        end case;
                                                end if;

                                        else
                                        end if;
                                        counter <= counter + 1; -- Increment counter

                        -- STATE 4: Upload Pixel's Colour to SH4
                        -- NB. This uploads the pixel's colour, the Y, U, V and the bit ANDed vectors
                        elsif command_state = 4 then
                                gio_we <= '0';
                                out_switch <= '1';
                                case counter is
                                        when 0 => gio_we <= '0';
                                        when 1 => pixel_colour <= y_colour;
                                        when 2 => pixel_colour <= u_colour;
                                        when 3 => pixel_colour <= v_colour;
                                        when 4 => pixel_colour <= temp;
                                        when others => pixel_colour <= "00000000";
                                end case;
                                counter <= counter + 1;
                        else
                        end if;

                else
                end if;
                end process Command;

-- USE usb data pins to check DATABUS
        PIN_EXP(0) <=   'Z';
        PIN_EXP(1) <=   'Z';

-- USE EXP to check ADDRESS BUS
        PIN_EXP(2) <= PIN_ADDRBUS(0)N or PIN_ADDRBUS(1)  or PIN_ADDRBUS(2)  or
PIN_ADDRBUS(3) orPIN_ADDRBUS(4)  or PIN_ADDRBUS(5)  or PIN_ADDRBUS(6)  or PIN_ADDRBUS(7);
        PIN_EXP(3) <= PIN_ADDRBUS(8)  or PIN_ADDRBUS(9)  or PIN_ADDRBUS(10) or
PIN_ADDRBUS(11) orPIN_ADDRBUS(12) or PIN_ADDRBUS(13) or PIN_ADDRBUS(14) or
PIN_ADDRBUS(15);
        PIN_EXP(4) <= PIN_ADDRBUS(16) or PIN_ADDRBUS(17) or PIN_ADDRBUS(18);

--      LINK UP CPU pins
        PIN_EXP(5) <= PIN_RD or PIN_RDWR or PIN_BS or PIN_RDY or PIN_SCK or PIN_CS1 or PIN_CS2;
        PIN_EXP(6) <= PIN_WE0 or PIN_WE1 or PIN_WE2 or PIN_WE3;
        PIN_EXP(7) <= PIN_DRAK0 or PIN_DRAK1  or PIN_DREQ1 or PIN_DACK0 or PIN_DACK1;
        PIN_EXP(8) <= PIN_NMI or PIN_IRL1 or PIN_IRL2 or PIN_IRL3 or PIN_MRESET;

-- LINK UP usb
        PIN_U_A0 <= 'Z';
        PIN_U_SUSPEND <= 'Z';
```

```
            PIN_U_INT <= 'Z';
            PIN_U_EOT <= 'Z';

--          PIN_U_DATA <= "1010Z1Z0";

-- check PB
            PIN_U_WR <= PIN_PB(1) and PIN_PB(2) and PIN_PB(3) and PIN_PB(4);
            PIN_U_RD <= PIN_PB(1) or  PIN_PB(2) or  PIN_PB(3) or  PIN_PB(4);

-- wire all interface pin to the usb CS
            PIN_U_CS <= PIN_I_ADDR(0) and PIN_I_ADDR(1) and PIN_I_ADDR(2) andPIN_I_IOR and
PIN_I_IOW and PIN_I_CS0 and PIN_I_CS1 and PIN_I_PPCS and PIN_I_ECPCS and PIN_I_TC and PIN_I_PDRQ
and PIN_I_PDACK and PIN_I_RESET andPIN_I_PINTR and PIN_I_IOCHRDY and PIN_I_TXRDY0 and
PIN_I_TXRDY1 and PIN_I_RXRDY0 and PIN_I_RXRDY1 and PIN_I_INTRPT0 and PIN_I_INTRPT1;

-- Serial control
            PIN_SCI_C1  <= 'Z';
            PIN_SCI_C2  <= 'Z';
            PIN_SCIF_C1 <= 'Z';
            PIN_SCIF_C2 <= 'Z';

-- camera
            -- sccb 3 wire
            s_sccb_w <= '1'; --PIN_GIO10;
            s_sccb_wn <= not (s_sccb_w);

-- SCCB master iobuf (to sh4)
            m_sccb_m : IOBUF port map(I => s_sccb_sm, T => s_sccb_wn, O => s_sccb_ms, IO => PIN_GIO9);
-- SCCB master iobuf (to sh4)
            m_sccb_s : IOBUF port map(I => s_sccb_ms, T => s_sccb_w, O => s_sccb_sm, IO => PIN_CAM_SIO0);

            PIN_CAM_SIO1 <= PIN_GIO8;

-- PCLK to IRL connection
            PIN_DREQ0 <= (PIN_CAM_PCLK and PIN_CAM_HREF) or PIN_GIO11;
            PIN_IRL0 <= (not PIN_CAM_VSYNC) or PIN_GIO11;

            cam_buffer(0)  <= PIN_CAM_Y(0);
            cam_buffer(1)  <= PIN_CAM_Y(1);
            cam_buffer(2)  <= PIN_CAM_Y(2);
            cam_buffer(3)  <= PIN_CAM_Y(3);
            cam_buffer(4)  <= PIN_CAM_Y(4);
            cam_buffer(5)  <= PIN_CAM_Y(5);
            cam_buffer(6)  <= PIN_CAM_Y(6);
            cam_buffer(7)  <= PIN_CAM_Y(7);

            cam_buffer(8)  <= PIN_CAM_UV(0);
            cam_buffer(9)  <= PIN_CAM_UV(1);
            cam_buffer(10) <= PIN_CAM_UV(2);
            cam_buffer(11) <= PIN_CAM_UV(3);
            cam_buffer(12) <= PIN_CAM_UV(4);
            cam_buffer(13) <= PIN_CAM_UV(5);
            cam_buffer(14) <= PIN_CAM_UV(6);
            cam_buffer(15) <= PIN_CAM_UV(7);
            data_out_bus(15 downto 0) <= cam_buffer;

            s_n_cam_pclk <= not (PIN_CAM_PCLK or PIN_GIO11);

            PIN_U_DATA(0) <= s_n_cam_pclk;
            PIN_U_DATA(1) <= PIN_CAM_HREF;
            PIN_U_DATA(2) <= PIN_DACK0;
            PIN_U_DATA(3) <= PIN_CAM_HREF and PIN_CAM_PCLK;
            PIN_U_DATA(4) <= (PIN_CAM_HREF and PIN_CAM_PCLK) or PIN_GIO11;
            PIN_U_DATA(5) <= (PIN_CAM_HREF and PIN_CAM_PCLK) and PIN_GIO11;
```

```
--          PIN_U_DATA(6) <= 'Z';
--          PIN_U_DATA(7) <= 'Z';

            m_cam_dreq1 : LOGIC_74x74 port map(          DATA => not(PIN_CAM_HREF),

                        CLK => s_n_cam_pclk,

                        nSET => '1',

                        nCLEAR => PIN_DACK0,

                        Q => PIN_U_DATA(7),

                        Qn => PIN_U_DMACK);

            m_cam_dreq2 : LOGIC_74x74 port map(          DATA => PIN_CAM_HREF,

                        CLK => s_n_cam_pclk,

                        nSET => '1',

                        nCLEAR => PIN_DACK0,

                        Q => PIN_U_DATA(6),

                        Qn => PIN_U_DMREQ);

            -- tie the unused pins to float
            PIN_CAM_CHYSNC <= 'Z';
            PIN_CAM_PWDN <= 'Z';

-- data bus
            s_data_out <= PIN_CS1; -- or PIN_RDWR;

            g_data0  : IOBUF port map(I => gio_out_bus(0), T => gio_dir,O => gio_in_bus(0),IO => PIN_GIO(0));
            g_data1  : IOBUF port map(I => gio_out_bus(1), T => gio_dir,O => gio_in_bus(1),IO => PIN_GIO(1));
            g_data2  : IOBUF port map(I => gio_out_bus(2), T => gio_dir,O => gio_in_bus(2),IO => PIN_GIO(2));
            g_data3  : IOBUF port map(I => gio_out_bus(3), T => gio_dir,O => gio_in_bus(3),IO => PIN_GIO(3));
            g_data4  : IOBUF port map(I => gio_out_bus(4), T => gio_dir,O => gio_in_bus(4),IO => PIN_GIO(4));
            g_data5  : IOBUF port map(I => gio_out_bus(5), T => gio_dir,O => gio_in_bus(5),IO => PIN_GIO(5));
            g_data6  : IOBUF port map(I => gio_out_bus(6), T => gio_dir,O => gio_in_bus(6),IO => PIN_GIO(6));
            g_data7  : IOBUF port map(I => gio_out_bus(7), T => gio_dir,O => gio_in_bus(7),IO => PIN_GIO(7));

m_data0  : IOBUF port map(I =>data_out_bus(0),T =>s_data_out, O => data_in_bus(0), IO =>PIN_DATABUS(0));
m_data1  : IOBUF port map(I =>data_out_bus(1),T =>s_data_out, O => data_in_bus(1), IO =>PIN_DATABUS(1));
m_data2  : IOBUF port map(I =>data_out_bus(2),T =>s_data_out, O => data_in_bus(2), IO =>PIN_DATABUS(2));
m_data3  : IOBUF port map(I =>data_out_bus(3),T =>s_data_out, O => data_in_bus(3), IO =>PIN_DATABUS(3));
m_data4  : IOBUF port map(I =>data_out_bus(4),T =>s_data_out, O => data_in_bus(4), IO =>PIN_DATABUS(4));
m_data5  : IOBUF port map(I => data_out_bus(5),T =>s_data_out, O => data_in_bus(5), IO =>PIN_DATABUS(5));
m_data6  : IOBUF port map(I => data_out_bus(6),T =>s_data_out, O => data_in_bus(6), IO =>PIN_DATABUS(6));
m_data7  : IOBUF port map(I => data_out_bus(7),T =>s_data_out, O => data_in_bus(7), IO =>PIN_DATABUS(7));

m_data8  : IOBUF port map(I => data_out_bus(8),T =>s_data_out, O => data_in_bus(8), IO =>PIN_DATABUS(8));
m_data9  : IOBUF port map(I => data_out_bus(9),T =>s_data_out, O => data_in_bus(9), IO =>PIN_DATABUS(9));
m_data10 : IOBUF port map(I=>data_out_bus(10),T=>s_data_out,O=>data_in_bus(10),IO =>PIN_DATABUS(10));
m_data11 : IOBUF port map(I=>data_out_bus(11),T=>s_data_out,O=>data_in_bus(11),IO =>PIN_DATABUS(11));
m_data12 : IOBUF port map(I=>data_out_bus(12),T=>s_data_out,O=>data_in_bus(12),IO=> PIN_DATABUS(12));
m_data13 : IOBUF port map(I=>data_out_bus(13),T =>s_data_out,O=>data_in_bus(13),IO=>PIN_DATABUS(13));
m_data14 : IOBUF port map(I=>data_out_bus(14),T=>s_data_out,O=>data_in_bus(14),IO =>PIN_DATABUS(14));
m_data15 : IOBUF port map(I=>data_out_bus(15),T=>s_data_out,O =>data_in_bus(15),IO=>PIN_DATABUS(15));

m_data15 : IOBUF port map(I=>data_out_bus(15),T=>s_data_out,O =>data_in_bus(15),IO=>PIN_DATABUS(15));
 m_data16 : IOBUF port map(I=>data_out_bus(16),T=>s_data_out,O =>data_in_bus(16),IO=>PIN_DATABUS(16));
m_data17 : IOBUF port map(I=>data_out_bus(17),T=>s_data_out,O =>data_in_bus(17),IO=>PIN_DATABUS(17));
```

```
m_data18 : IOBUF port map(I=>data_out_bus(18),T=>s_data_out,O =>data_in_bus(18),IO=>PIN_DATABUS(18));
m_data19 : IOBUF port map(I=>data_out_bus(19),T=>s_data_out,O =>data_in_bus(19),IO=>PIN_DATABUS(19));
m_data20 : IOBUF port map(I=>data_out_bus(20),T=>s_data_out,O =>data_in_bus(20),IO=>PIN_DATABUS(20));
m_data21 : IOBUF port map(I=>data_out_bus(21),T=>s_data_out,O =>data_in_bus(21),IO=>PIN_DATABUS(21));
m_data22 : IOBUF port map(I=>data_out_bus(22),T=>s_data_out,O =>data_in_bus(22),IO=>PIN_DATABUS(22));
m_data23 : IOBUF port map(I=>data_out_bus(23),T=>s_data_out,O =>data_in_bus(23),IO=>PIN_DATABUS(23));
m_data24 : IOBUF port map(I=>data_out_bus(24),T=>s_data_out,O =>data_in_bus(24),IO=>PIN_DATABUS(24));
m_data25 : IOBUF port map(I=>data_out_bus(25),T=>s_data_out,O =>data_in_bus(25),IO=>PIN_DATABUS(25));
m_data26 : IOBUF port map(I=>data_out_bus(26),T=>s_data_out,O =>data_in_bus(26),IO=>PIN_DATABUS(26));
m_data27 : IOBUF port map(I=>data_out_bus(27),T=>s_data_out,O =>data_in_bus(27),IO=>PIN_DATABUS(27));
m_data28 : IOBUF port map(I=>data_out_bus(28),T=>s_data_out,O =>data_in_bus(28),IO=>PIN_DATABUS(28));
m_data29 : IOBUF port map(I=>data_out_bus(29),T=>s_data_out,O =>data_in_bus(29),IO=>PIN_DATABUS(29));
m_data30 : IOBUF port map(I=>data_out_bus(30),T=>s_data_out,O =>data_in_bus(30),IO=>PIN_DATABUS(30));
m_data31 : IOBUF port map(I=>data_out_bus(31),T=>s_data_out,O =>data_in_bus(31),IO=>PIN_DATABUS(31));

        data_out_bus(16) <= 'Z';
        data_out_bus(17) <= 'Z';
        data_out_bus(18) <= 'Z';
        data_out_bus(19) <= 'Z';
        data_out_bus(20) <= 'Z';
        data_out_bus(21) <= 'Z';
        data_out_bus(22) <= 'Z';
        data_out_bus(23) <= 'Z';

        data_out_bus(24) <= 'Z';
        data_out_bus(25) <= 'Z';
        data_out_bus(26) <= 'Z';
        data_out_bus(27) <= 'Z';
        data_out_bus(28) <= 'Z';
        data_out_bus(29) <= 'Z';
        data_out_bus(30) <= 'Z';
        data_out_bus(31) <= 'Z';


        PIN_AUX <=      data_in_bus(0)  or data_in_bus(1)  or data_in_bus(2)  or data_in_bus(3)  or
data_in_bus(4)  or data_in_bus(5)  or data_in_bus(6)  or data_in_bus(7)  or data_in_bus(8)  or data_in_bus(9)  or
data_in_bus(10) or data_in_bus(11) or data_in_bus(12) or data_in_bus(13) or data_in_bus(14) or data_in_bus(15) or
data_in_bus(16) or data_in_bus(17) or data_in_bus(18) or data_in_bus(19) or data_in_bus(20) or data_in_bus(21) or
data_in_bus(22) or data_in_bus(23) or data_in_bus(24) or data_in_bus(25) or data_in_bus(26) or data_in_bus(27) or
data_in_bus(28) or data_in_bus(29) or data_in_bus(30) or data_in_bus(31) or PIN_GIO12 or PIN_GIO13 or
PIN_GIO14 or PIN_GIO15;
end vbsh4_arch;
```

C.2 FPGA Synthesis Report (Trimmed)

```
Release 4.2WP3.x - xst E.38
Copyright (c) 1995-2001 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to .
CPU : 0.00 / 1.49 s | Elapsed : 0.00 / 0.00 s

--> Parameter overwrite set to YES
CPU : 0.00 / 1.49 s | Elapsed : 0.00 / 0.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 1.49 s | Elapsed : 0.00 / 0.00 s

--> ===========================================================================
---- Source Parameters
Input Format                       : VHDL
Input File Name                    : vbsh4.prj

---- Target Parameters
Target Device                      : xc2s100-fg256-5
Output File Name                   : vbsh4
Output Format                      : NGC
Target Technology                  : spartan2

---- Source Options
Entity Name                        : vbsh4
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
FSM Flip-Flop Type                 : D
Mux Extraction                     : YES
Resource Sharing                   : YES
Complex Clock Enable Extraction    : YES
ROM Extraction                     : Yes
RAM Extraction                     : Yes
RAM Style                          : Auto
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
Automatic Register Balancing       : No

---- Target Options
Add IO Buffers                     : YES
Equivalent register Removal        : YES
Add Generic Clock Buffer(BUFG)     : 4
Global Maximum Fanout              : 100
Register Duplication               : YES
Move First FlipFlop Stage          : YES
Move Last FlipFlop Stage           : YES
Slice Packing                      : YES
Pack IO Registers into IOBs        : auto
Speed Grade                        : 5

---- General Options
Optimization Criterion             : Speed
Optimization Effort                : 1
Check Attribute Syntax             : YES
Keep Hierarchy                     : No
Global Optimization                : AllClockNets
Write Timing Constraints           : No

=========================================================================
Compiling vhdl file D:/Uni/Thesis/ThesisWork/vbsh4_fpga/Logic74_74.vhd in
Library work.
Entity <logic_74x74> (Architecture <logic_74x74_arch>) compiled.
Compiling vhdl file D:/Uni/Thesis/ThesisWork/vbsh4_fpga/vbsh4.vhd in Library
work.
Entity <vbsh4> analyzed. Unit <vbsh4> generated.

Analyzing Entity <logic_74x74> (Architecture <logic_74x74_arch>).
Entity <logic_74x74> analyzed. Unit <logic_74x74> generated.


Synthesizing Unit <logic_74x74>.
    Related source file is D:/Uni/Thesis/ThesisWork/vbsh4_fpga/Logic74_74.vhd.
    Found 1-bit register for signal <q>.
    Found 1-bit register for signal <qn>.
    Summary:
       inferred   2 D-type flip-flop(s).
Unit <logic_74x74> synthesized.

Synthesizing Unit <vbsh4>.
    Related source file is D:/Uni/Thesis/ThesisWork/vbsh4_fpga/vbsh4.vhd.
    Using one-hot encoding for signal <command_state>.
    Found 1-bit tristate buffer for signal <pin_cam_pwdn>.
    Found 1-bit tristate buffer for signal <pin_cam_chysnc>.
    Found 1-bit tristate buffer for signal <pin_u_suspend>.
    Found 1-bit tristate buffer for signal <pin_u_int>.
    Found 1-bit tristate buffer for signal <pin_u_eot>.
    Found 1-bit tristate buffer for signal <pin_u_a0>.
    Found 2-bit tristate buffer for signal <pin_exp<1:0>>.
    Found 1-bit tristate buffer for signal <pin_sci_c1>.
    Found 1-bit tristate buffer for signal <pin_sci_c2>.
    Found 1-bit tristate buffer for signal <pin_scif_c1>.
    Found 1-bit tristate buffer for signal <pin_scif_c2>.
    Found 8-bit comparator greater for signal <$n0099> created at line 425.
    Found 8-bit comparator less for signal <$n0100> created at line 426.
    Found 32-bit up counter for signal <address>.
    Found 5-bit register for signal <command_state>.
    Found 32-bit up counter for signal <counter>.
    Found 16-bit tristate buffer for signal <data_out_bus<31:16>>.
    Found 9-bit register for signal <gio_addr_bus>.
    Found 1-bit register for signal <gio_dir>.
    Found 1-bit register for signal <gio_we>.
    Found 1-bit register for signal <led1_intern>.
    Found 1-bit register for signal <led2_intern>.
    Found 1-bit register for signal <led3_intern>.
    Found 1-bit register for signal <out_switch>.
    Found 8-bit register for signal <pixel_colour>.
    Found 8-bit register for signal <temp>.
    Found 8-bit register for signal <u_colour>.
    Found 8-bit register for signal <v_colour>.
    Found 8-bit register for signal <y_colour>.
    Found 17 1-bit 2-to-1 multiplexers.
    Summary:
       inferred   2 Counter(s).
       inferred  60 D-type flip-flop(s).
       inferred   2 Comparator(s).
       inferred  17 Multiplexer(s).
       inferred  28 Tristate(s).
Unit <vbsh4> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
```

```
# Registers                        : 17
  5-bit register                   : 1
  1-bit register                   : 10
  9-bit register                   : 1
  8-bit register                   : 5
# Counters                         : 2
  32-bit up counter                : 2
# Multiplexers                     : 3
  2-to-1 multiplexer               : 3
# Tristates                        : 28
  1-bit tristate buffer            : 28
# Comparators                      : 2
  8-bit comparator greater         : 1
  8-bit comparator less            : 1

=========================================================================

Starting low level synthesis...
Optimizing unit <vbsh4> ...

Building and optimizing final netlist ...

=========================================================================
Final Results
Top Level Output File Name         : vbsh4
Output Format                      : NGC
Optimization Criterion             : Speed
Target Technology                  : spartan2
Keep Hierarchy                     : No
Macro Generator                    : macro+

Macro Statistics
# Registers                        : 23
  32-bit register                  : 2
  1-bit register                   : 15
  9-bit register                   : 1
  8-bit register                   : 5
# Tristates                        : 28
  1-bit tristate buffer            : 28
# Adders/Subtractors               : 2
  32-bit adder                     : 2

Design Statistics
# IOs                              : 173

Cell Usage :
# BELS                             : 454
#      GND                         : 1
#      LUT1                        : 68
#      LUT2                        : 21
#      LUT2_D                      : 2
#      LUT3                        : 84
#      LUT4                        : 148
#      LUT4_L                      : 2
#      MUXCY                       : 62
#      MUXF5                       : 3
#      VCC                         : 1
#      XORCY                       : 62
# FlipFlops/Latches                : 128
#      FDCP                        : 4
#      FDE_1                       : 52
#      FDR_1                       : 1
#      FDRE_1                      : 69
#      FDSE_1                      : 2
# RAMS                             : 1
#      RAMB4_S8                    : 1
# Tri-States                       : 16
#      BUFT                        : 16
# Clock Buffers                    : 1
#      BUFG                        : 1
# IO Buffers                       : 173
#      IBUF                        : 91
#      IOBUF                       : 42
#      OBUF                        : 28
#      OBUFT                       : 12
=========================================================================


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
I_pin_u_data_0:O                   | NONE(*)(m_cam_dreq2_q) | 5     |
pin_gio10                          | IBUF                   | 1     |
-----------------------------------+------------------------+-------+
(*) This 1 clock signal(s) are generated by combinatorial logic,
and XST is not able to identify which are the primary clock signals.
Please use the CLOCK_SIGNAL constraint to specify the clock signal(s)
generated by combinatorial logic.

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 17.686ns (Maximum Frequency: 56.542MHz)
   Minimum input arrival time before clock: 5.440ns
   Maximum output required time after clock: 12.734ns
   Maximum combinational path delay: 13.230ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

-------------------------------------------------------------------------
Timing constraint: Default period analysis for Clock 'pin_gio10'
Delay:               17.686ns (Levels of Logic = 6)
  Source:            counter_29
  Destination:       u_colour_4
  Source Clock:      pin_gio10 falling
  Destination Clock: pin_gio10 falling

  Data Path: counter_29 to u_colour_4
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    FDRE_1:C->Q            2   1.292   1.340  counter_29 (counter_29)
    LUT2:I0->O             1   0.653   1.150  I_38_LUT_7 (N1104)
    LUT4:I0->O             6   0.653   1.850  I_37_LUT_12 (N1134)
    LUT2_D:I0->LO          1   0.653   0.100  I_XXL_677 (N3765)
    LUT4:I3->O            12   0.653   2.400  I__n0042 (N1180)
    LUT2_D:I0->O           6   0.653   1.850  I__n0015 (N1406)
    LUT1:I0->O            17   0.653   2.900  I_INV__n0015_2 (I_INV__n0015_2)
    FDE_1:CE                    0.886          u_colour_4
    ----------------------------------------
    Total                     17.686ns (6.096ns logic, 11.590ns route)
                                       (34.5% logic, 65.5% route)
-------------------------------------------------------------------------
Timing constraint: Default OFFSET IN BEFORE for Clock 'I_pin_u_data_0:O'
Offset:              5.440ns (Levels of Logic = 2)
  Source:            pin_dack0
  Destination:       m_cam_dreq2_q
  Destination Clock: I_pin_u_data_0:O rising

  Data Path: pin_dack0 to m_cam_dreq2_q
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    IBUF:I->O              3   0.924   1.480  pin_dack0_IBUF
(pin_u_data_2_OBUF)
    LUT1:I0->O             4   0.653   1.600  I_INV_pin_dack0 (N642)
    FDCP:CLR                   0.783          m_cam_dreq2_q
    ----------------------------------------
    Total                      5.440ns (2.360ns logic, 3.080ns route)
                                       (43.4% logic, 56.6% route)

-------------------------------------------------------------------------
Timing constraint: Default OFFSET OUT AFTER for Clock 'pin_gio10'
Offset:              12.734ns (Levels of Logic = 2)
  Source:            ram0
  Destination:       pin_gio_3
  Source Clock:      pin_gio10 rising

  Data Path: ram0 to pin_gio_3
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    RAMB4_S8:CLK->DO3      4   3.774   1.600  ram0 (ram_out_bus_3)
    LUT3:I2->O             1   0.653   1.150  I_Mmux_gio_out_bus_I4_Result
(g_data3)
    IOBUF:I->IO                5.557          g_data3 (pin_gio_3)
    ----------------------------------------
    Total                     12.734ns (9.984ns logic, 2.750ns route)
                                       (78.4% logic, 21.6% route)

-------------------------------------------------------------------------
Timing constraint: Default path analysis
Delay:               13.230ns (Levels of Logic = 5)
  Source:            pin_gio12
  Destination:       pin_aux

  Data Path: pin_gio12 to pin_aux
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
    IBUF:I->O              2   0.924   1.340  pin_gio12_IBUF (pin_gio12_IBUF)
    LUT4:I0->O             1   0.653   1.150  I_10_LUT_10 (N729)
    LUT3:I1->O             1   0.653   1.150  I_8_LUT_9 (N741)
    LUT4:I3->O             1   0.653   1.150  I_pin_aux (pin_aux_OBUF)
    OBUF:I->O                  5.557          pin_aux_OBUF (pin_aux)
    ----------------------------------------
    Total                     13.230ns (8.440ns logic, 4.790ns route)
                                       (63.8% logic, 36.2% route)
=========================================================================
CPU : 17.90 / 19.39 s | Elapsed : 18.00 / 18.00 s

-->
```

C.3 FPGA Map Report (Trimmed)

```
Release 4.2WP3.x - Map E.38
Xilinx Mapping Report File for Design 'vbsh4'

Design Information
------------------
Command Line   : map -p xc2s100-fg256-5 -cm area -k 4 -c 100 -tx off vbsh4.ngd
Target Device  : x2s100
Target Package : fg256
Target Speed   : -5
Mapper Version : spartan2 -- $Revision: 1.58 $
Mapped Date    : Mon Oct 14 23:05:11 2002

Design Summary
--------------
   Number of errors:      0
   Number of warnings:    2
   Number of Slices:               183 out of  1,200   15%
   Number of Slices containing
      unrelated logic:               0 out of    183    0%
   Number of Slice Flip Flops:     124 out of  2,400    5%
   Total Number 4 input LUTs:      292 out of  2,400   12%
      Number used as LUTs:                       259
      Number used as a route-thru:                33
   Number of bonded IOBs:          161 out of    176   91%
      IOB Flip Flops:                              4
   Number of Block RAMs:             1 out of     10   10%
   Number of GCLKs:                  1 out of      4   25%
Total equivalent gate count for design:  19,466
Additional JTAG gate count for IOBs:  7,728

Table of Contents
-----------------
Section 1 - Errors
Section 2 - Warnings
Section 3 - Informational
Section 4 - Removed Logic Summary
Section 5 - Removed Logic
Section 6 - IOB Properties
Section 7 - RPMs
Section 8 - Guide Report
Section 9 - Area Group Summary
Section 10 - Modular Design Summary

Section 1 - Errors
------------------

Section 2 - Warnings
--------------------
WARNING:MapLib:96 - IBUF symbol "ibuf0" (output signal=connect_clk) driving
BUFG
   is LOCed to a generic IOB site. This will cause lower performance than
using
   a dedicated GCLKIOB site.
WARNING:DesignRules:372 - Netcheck: Gated clock. Clock net pin_u_data_0_OBUF
is
   sourced by a combinatorial pin. This is not good design practice. Use the
CE
   pin to control the loading of data into the flip-flop.

Section 3 - Informational
-------------------------
```

```
INFO:MapLib:62 - All of the external outputs in this design are using slew
rate
   limited output drivers. The delay on speed critical outputs can be
   dramatically reduced by designating them as fast outputs in the schematic.

Section 4 - Removed Logic Summary
---------------------------------
  42 block(s) removed
  18 block(s) optimized away
  13 signal(s) removed



Section 7 - RPMs
----------------

Section 8 - Guide Report
------------------------
Guide not run on this design.

Section 9 - Area Group Summary
------------------------------
No area groups were found in this design.

Section 10 - Modular Design Summary
-----------------------------------
Modular Design not used for this design.
```

C.4 SH4 CODE

```
/* This program handles teh interface between the fpga and the sh4.
It includes methods for loading the look-up table as well
as loading up a pixel and receiving the data back.
Ho Wong
24/9/02
*/

#include <sh/gio.h>
#include "segment.h"
#include "fpgaloader.h"
#include <sh/tmu.h>
#include <sh/scif.h>
#include <sh/sci.h>

int low;

void load_table(UCHAR* table){
		int a;
		int i;
		gio_data_out();
		gio_set_bit_h(GIO_PORTA, GIO_PA10);
		low = 0;
		gio_data(0x00);
		gio_data(DownloadTableCode);
		toggle_clock(); // FPGA reads command
		toggle_clock(); // go back high.
		toggle_clock();
		toggle_clock();
		scif_put('D');
		scif_put('\n');
		scif_put('\r');

		for (a = 0; a < 256; a++){
		gio_data(table[a]); // send data
		toggle_clock(); //clock low
		delay(5);
		toggle_clock(); // go back high
		//delay(100000);
		}
		scif_put('E');
		scif_put('\n');
		scif_put('\r');
		delay(100000);
		gio_data(0x00);
		// Clear the clock
		toggle_clock();
		toggle_clock();
		toggle_clock();
		toggle_clock();
		toggle_clock();

}
void get_table(UCHAR* table){
		int i;
		int x;
		gio_data_out();
		gio_data(0x00);
		gio_set_bit_h(GIO_PORTA, GIO_PA10);
		low = 0;
		// Send upload command
		gio_data(UploadTableCode);
		toggle_clock(); // FPGA reads code
		toggle_clock(); // goes back high
		toggle_clock();
		toggle_clock();

		gio_data_in(); // change directions
		scif_put('F');
		scif_put('\n');
		scif_put('\r');

		for (x = 0; x < 256; x++){
		table[x] = GIO_PORTA; // read data
		toggle_clock(); //clock low
		delay(5);
		toggle_clock(); // go back high

		}
		scif_put('G');
		scif_put('\n');
		scif_put('\r');
		// Clear the clock
		toggle_clock();
		toggle_clock();
		toggle_clock();
		toggle_clock();
		toggle_clock();
		toggle_clock();
		toggle_clock();
		toggle_clock();

}

void send_pixel(UCHAR Y, UCHAR U, UCHAR V){
		int i;
		int x;
		//uchar Y = 150;
		//uchar U = 120;
		//uchar V = 140;
//		scif_put(Y);
//		scif_put(U);
//		scif_put(V);
//		scif_put('\n');
//		scif_put('\r');
		gio_data_out();
		gio_data(0x00);
		//gio_set_bit_h(GIO_PORTA, GIO_PA10);
		low = 0;
		// Send upload command
		gio_data(DownloadPixelCode);
		clock_off(); //toggle_clock(); // FPGA reads code
		clock_on(); //toggle_clock(); // goes back high

		clock_off();//toggle_clock();
		clock_on(); //toggle_clock();

		gio_data(Y); // Send Y
		clock_off(); //toggle_clock(); //clock low // Counter = 0

		clock_on(); //toggle_clock(); // go back high

		gio_data(U); // Send U
		clock_off(); //toggle_clock(); //clock low // Counter = 1
		clock_on(); //toggle_clock(); // go back high
```

```
            gio_data(V); // Send V
            clock_off(); //toggle_clock(); //clock low // Counter = 2
            clock_on(); //toggle_clock(); // go back high

            gio_data(0x00);
            clock_off(); //toggle_clock(); // Counter = 3

            clock_on(); //toggle_clock();


            clock_off(); //toggle_clock(); // Counter = 4
            clock_on(); //toggle_clock();

            for ( i = 0; i < 5; i++){
            clock_off(); //toggle_clock();
            clock_on(); //toggle_clock();
            }


}

UCHAR get_pixel(){
            int i;
            uchar pixel, p2, p3, p4, p5, p6, p7;
            gio_data_out();
            gio_data(0x00);
            gio_set_bit_h(GIO_PORTA, GIO_PA10);
            low = 0;
            // Send upload command
            gio_data(UploadPixelCode);
            clock_off(); //toggle_clock(); // FPGA reads code
            clock_on(); //toggle_clock(); // goes back high
            clock_off(); //toggle_clock();
            gio_data_in(); // change directions
            clock_on(); //toggle_clock();

            p6 = GIO_PORTA; // Pixel Colour
/*          toggle_clock(); //clock low
            pixel = GIO_PORTA;
            toggle_clock(); // go back high
            p7 = GIO_PORTA;
            toggle_clock(); //clock low
            p2 = GIO_PORTA; // Y data
            toggle_clock(); // go back high


            toggle_clock(); //clock low
            p3 = GIO_PORTA; // U data
            toggle_clock(); // go back high
            toggle_clock();
            p4 = GIO_PORTA; // V data
            toggle_clock();
            toggle_clock();
            p5 = GIO_PORTA; // bit AND of U and V
            toggle_clock();
*/
            for ( i = 0; i < 10 ; i++){
                        clock_off(); //toggle_clock();
                        clock_on(); //toggle_clock();
            }
/*          scif_put('\n');
//          scif_put('\r');
            print_bin(p6);
            print_bin(pixel);
```

```
		print_bin(p7);
		print_bin(p2);
		print_bin(p3);
		print_bin(p4);
		print_bin(p5);*/

		return p6;

}

void fpga_lookup (UCHAR* intable, UCHAR* outtable, int length){
		uchar y, u, v;
		int i;
		int j = 0;
		for (i = 0; i < length; i++){
			v = intable[i++];
			u = intable[i++];
			y = intable[i++];
			send_pixel(y,u,v);
			outtable[j] = get_pixel();
			j++;

		};
}

void toggle_clock(){
		if (low){
			gio_set_bit_h(GIO_PORTA, GIO_PA10);
			low = 0;
		}
		else{
			gio_set_bit_l(GIO_PORTA, GIO_PA10);
			low = 1;
		}

}

void print_bin(UCHAR pixel){
		int i;
		for ( i = 0; i<8 ; i++){
			int temp;
			temp = 0x80 & pixel;
			if (temp)
				scif_put('1');
			else
				scif_put('0');
			pixel = pixel << 1;
		}
		scif_put('\n');
		scif_put('\r');
}
```