A Transputer Based Implementation of a Quick Look Processor for an Airborne SAR System

P. J. Archer

26 January 1997

Thesis submitted to the
Department of Electrical Engineering
at the University of Cape Town in fulfillment of
the requirements for the degree of
Master of Science in Engineering

Abstract

The purpose of this thesis is to describe the development of a transputer based real-time Synthetic Aperture Radar (SAR) processor called the Quick Look Processor (QLP). The QLP is required to produce medium resolution, real-time images at a wavelength of 2.5m for the airborne South African SAR (SASAR) system which is under development at the University of Cape Town (UCT). The required azimuth resolution for the QLP is 30m and system is required to process 2048 range bins at a rate of 39 range lines per second.

The algorithm used was developed at UCT, and works on the principle of dividing up the synthetic aperture into subapertures with appropriate phase corrections. This method is used in order to reduce computational loading (for real-time processing), but at the same time achieve medium resolution processing. One of the fundamental issues concerning this algorithm is its efficiency as the required azimuth resolution is increased.

The system is designed around a host PC and a network of nine transputers. The host PC communicates with the other SASAR subsystems via an Ethernet network. It is responsible for displaying and saving the SAR image, recieving and displaying geocoding information and configuring the transputer network. The transputer network is responsible for processing the SAR data. The network is connected in a pipeline configuration with a master transputer controlling the other eight slave transputers. Each slave transputer concurrently processes a section of the swath width. This method allows for easy scalability.

Once implemented, it was found that the system operated at 25% of its expected performance. This expected performance was based on a set of assumptions which were initially made about the transputers' performance with the QLP algorithm. After further investigation, it was found that the time taken to address data in memory had been neglected, as had the delay associated with calculating the address of a particular byte of data in an array. For example instead of taking $0.35\mu s$[1] to do an addition, it takes $3.1\mu s$ to do

[1]This figure is supplied in the transputer data book and was used in the initial assumptions.

an addition of two numbers stored in a 2-dimensional array while returning the result to another 2-dimensional array. Once this was considered, it was understood why the transputer performance was poor. The performance of the host PC hardware was however found to be sufficient.

In order to process the data in real-time according to the given specifications, four times as many transputers would be required, or a different hardware platform to replace the transputer technology.

Acknowledgments

I would like to thank my supervisor, Professor Michael Inggs, for his support and guidance throughout the project. All the members of the UCT Radar Remote Sensing Group are due thanks, especially Pete Golda for his proof-reading, Jasper Horrell for his help in SAR theory and Peter Koeppen for his help with the hardware and system integration. Finally, I would like to thank my wife, Karen, for her support throughout the writeup.

Contents

List of Figures

List of Tables

Glossary

ADC	- Analog to Digital Converter
Azimuth direction	- Flight direction
CONSOLE	- Console subsystem
ECB	- Event Control Block
GPS	- Global Positioning System
HBR	- High Bitrate Recorder
IMU	- Inertial Measurement Unit
I/O	- Input/Output
IPX	- Internetwork Packet Exchange
IQ	- In phase and Quadrature vector components
LBR	- Low Bitrate Recorder
LBR/CONSOLE	- The LBR and Console subsystems seen a one physical unit.
PC	- Personal Computer
PRF	- Pulse Repetition Frequency
QFA	- Quasi Focused Algorithm
QL	- Quick Look
QLP	- Quick Look Processor
Range Direction	- Direction orthogonal to flight direction
RGB	- Red, Green, Blue
RRSG	- Radar Remote Sensing Group (at UCT)
SAR	- Synthetic Aperture Radar
TPU	- Transputer Processing Unit
UCT	- University of Cape Town

List of Symbols

Symbol	Explanation
V	aircraft velocity measured in m/s
S	swath width measured in m
h	height of aircraft above ground measured in m
BW	band width of a pulse
R	range resolution
c	speed of light
PRF	pulse repetition frequency
R_m	maximum range at which target falls within radar beamwidth
R_0	range to the point of closest approach
n_a	number of additions
n_{blocks}	number of subaperture blocks
n_{bins}	number of range bins
n_{subs}	number of subapertures
n_m	number of multiplications
T_{add}	time taken for a transputer to perform one addition
T_{mult}	time taken for a transputer to perform one multiplication
f_p	pulse repetition frequency after presumming

Chapter 1

Introduction

Synthetic Aperture Radar (SAR) is a side-looking radar imaging system which is used worldwide as a method of producing high resolution images of large areas of the earth's surface at microwave frequencies. SAR systems can be either be airborne or spaceborne. This thesis, however, will be limited to airborne SAR. Figure 1.1 shows the geometry of a SAR system where the aircraft is depicted by a point with a velocity V, travelling along the aircraft track h meters above the ground. A side-looking radar antenna transmits a radar beam illuminating the ground as shown by the shaded area. This area moves along with the aircraft, sweeping the area between the two parallel dotted lines. The distance between these two lines is the swath width, denoted by S in the diagram. The radar beam is transmitted in regular pulses at a frequency called the Pulse Repetition Frequency (PRF). Chapter 3 explains this operation in more detail.

SAR images are of great value as they can give us information about the world around us and how it is changing. SAR techniques can be used to monitor population spread, observe structures and geological features buried beneath dry earth or below thick vegetation or to monitor oil slicks or waves on the ocean surface [13]. Unlike optical techniques, SAR can operate equally well in any lighting conditions and can operate through fog, mist and rain. SAR images are often able to distinguish features where an optical photograph could not, for example different crop types may be the same colour, but may well have different radar reflectivity properties, and therefore SAR imaging techniques can be used to distinguish between these different crop types.

There are various SAR systems in use around the world today, but South Africa does not currently possess such a system. Due to the enormous costs involved in purchasing and importing such a system, there is a need for the development of a locally produced, low-cost, SAR imaging system. This is

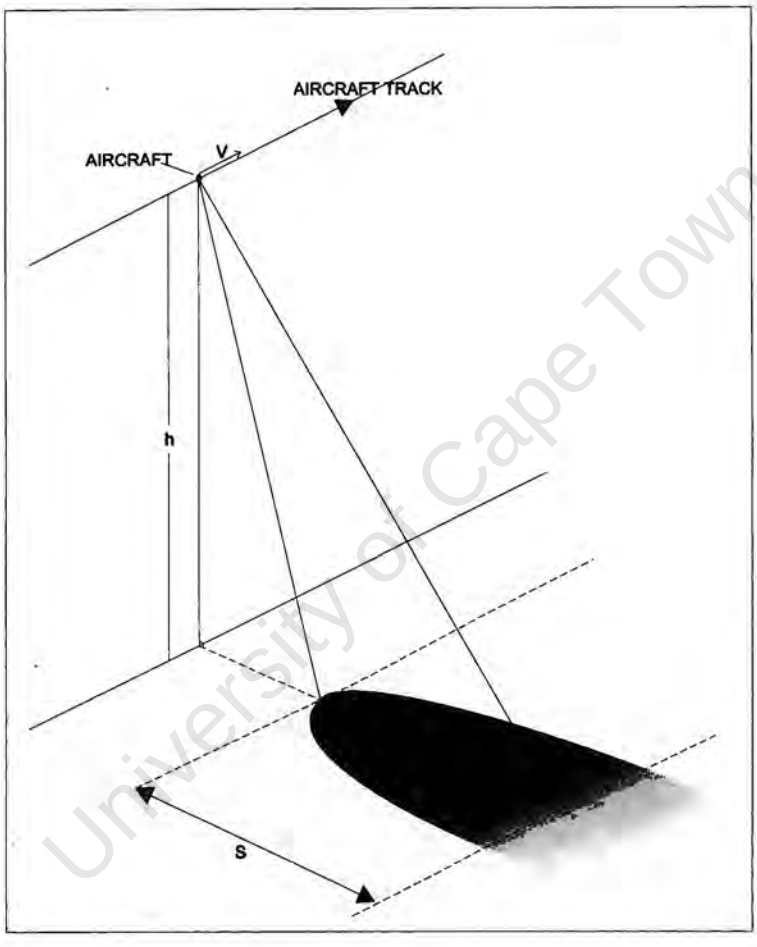

Figure 1.1: Basic SAR Geometry

exactly what SASAR is. The SASAR system is planned to be a low cost, multifrequency, multipolarization airborne SAR system, with a VHF SAR and a X-Band or C-Band SAR [2]. The SASAR recording system [3] has been designed and a prototype was constructed at the University of Cape Town (UCT) along with the SAR ground processing system.

In any SAR recording system some form of quality control of the data being recorded is needed. This usually takes the form of a real-time SAR processor. This thesis discusses the development of the real-time SAR processor called the Quick Look Processor (QLP) for the SASAR system. The aim was to design the QLP according to a given specification. At the time of development, a number of assumptions were made by the system designers about the performance of a network of transputers. At the time of design also, transputers were the only readily available processors which could easily be connected together to obtain the required performance. Based on these assumptions, it was decided that the transputers would be a suitable hardware platform on which to build the QLP. These transputers were then (erroneously) purchased specifically for the QLP development.

Unfortunately after implementation of the QLP, it was found that the system performed at about 25% of its required speed. This thesis will show the reader the route taken by the developer, how the problems manifested and were investigated. This will lead to the conclusion that some basic assumptions were incorrect and that certain computational issues were neglected.

Firstly, a brief overview of the SASAR system will be given, introducing the different subsystems and how they relate to the QLP. The QLP system requirements will then be given. The high level specification of the QLP will then be stated. The Quasi Focused Algorithm (QFA) will be explained and key parameters will be discussed in terms of computational loading and final image quality. The QFA was developed by the UCT RRSG (Radar Remote Sensing Group) specifically for the SASAR project [8]. Here it will be seen that the main concern is the speed at which the QFA can be processed. Factors which affect the processing demands are covered.

Chapter 4 deals with the system level design. Here the actual design phase is described starting with the hardware design. A PC is required for the actual displaying and recording of the image. The details of the required PC and graphics adapter are discussed. The transputer network can be configured in a number of different ways, such as a star or ring configuration. These configurations are then discussed along with the advantages and disadvantages of each. Once this has been done, it will be shown that the ring configuration is the most suitable method for the interconnection of the transputers for this application. Once the hardware platform has been covered, the software design is discussed. First the Host PC software is discussed

and issues such as achieving the demanding graphics performance required, given the limitations of current PC graphics adapters are also discussed. The transputer software is then covered. This section shows how the QFA was split up, so it could be run concurrently on a number of transputers. Methods for ensuring an efficient implementation of the QFA are also explained.

Chapter 5 reports on the actual timing tests performed to try to quantify the unexpectedly low performance. The results of these tests are given. Additional tests were performed to verify the computational performance of the transputers and the efficiency of the pipeline configuration. The results of these tests were analysed in detail and a new theoretical performance figure was obtained. During this analysis the weakness of the original assumptions became clear. The original calculations, based on the assumptions neglected the time taken for the transputer to fetch data from memory and return the results to memory.

Chapter 6 draws conclusions, summarises findings and briefly discusses the options for upgrading the QLP to perform according to specification.

Chapter 2

The SASAR System and QLP Requirements

This chapter deals with the SASAR project as a whole and specifically the recording system, of which the QLP is a subsystem. First, a broad overview of the SASAR system is given, followed by a brief overview of the recording system. This shows how the QLP fits into the SASAR system. Once the context of the QLP is established, the QLP system specifications and requirements are discussed.

2.1 Introduction

The SASAR system is currently under development at the University of Cape Town. It is intended to be a multifrequency, multipolarization airborne SAR system, with VHF SAR and an X-Band or C-Band SAR.

2.2 SASAR System Overview

The SASAR system block diagram is shown in Figure 2.1. As can be seen from the diagram, four systems reside within the SASAR system. They are depicted as shaded blocks in the diagram and can be identified as follows:

1. Platform System
2. Radar System
3. Recording System
4. Ground Processing System

Each of these systems is briefly described below.

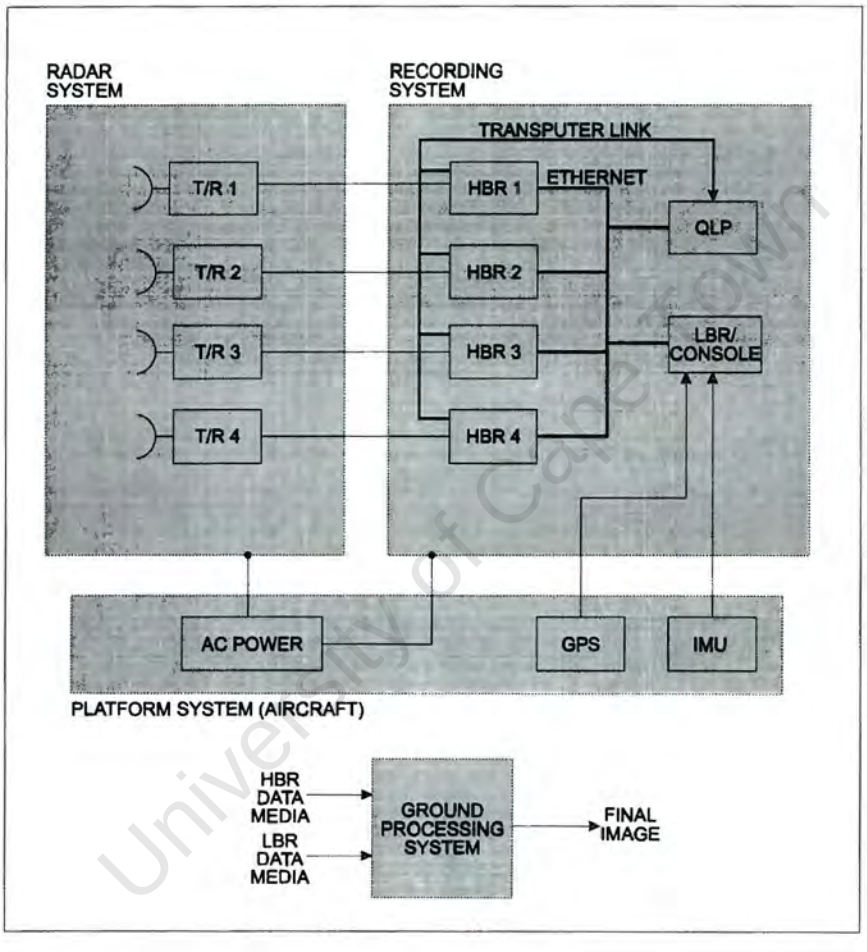

Figure 2.1: The SASAR System Block Diagram. (Adapted from the SASAR System Design Document[2].)

2.2.1 Platform System

The platform system is simply the physical environment in which the system will be flown, in other words the aircraft. This system includes factors such as aircraft velocity and maximum carrying capacity. More details can be found in Appendix B, but a full description can be found in the Platform Specification Document [5].

Other important functions of the platform are to provide power to the other systems, and to provide Global Positioning System (GPS) and Inertial Measurement Unit (IMU) information to the recording system.

2.2.2 Radar System

The radar system includes the radar transceivers and antennas. Currently only one wavelength of 2.5m is being used. The SASAR system however, is capable of running four transceivers simultaneously. Typically, the other transceivers would be used for different polarisations and/or different wavelengths.

As can be seen in Figure 2.1, each transceiver is connected to the recording system. The recording system triggers the transceiver at the correct instant and the received radar pulses are sent to the recording system for pre-processing and recording.

2.2.3 Recording System

The recording system is the heart of the SAR system. It is here where the raw radar returns are pre-processed and stored on magnetic tape. The recording system consists of the following subsystems:

1. The High Bitrate Recorder (HBR)
2. The Low Bitrate Recorder(LBR)/Console
3. The QLP

Figure 2.2 gives a block diagram of the recording system, showing how the various sub-systems fit together. This diagram was taken from the SASAR Recording System Design Document [3].

Each of these subsystems is described below. Appendix B provides the recording system interface specifications.

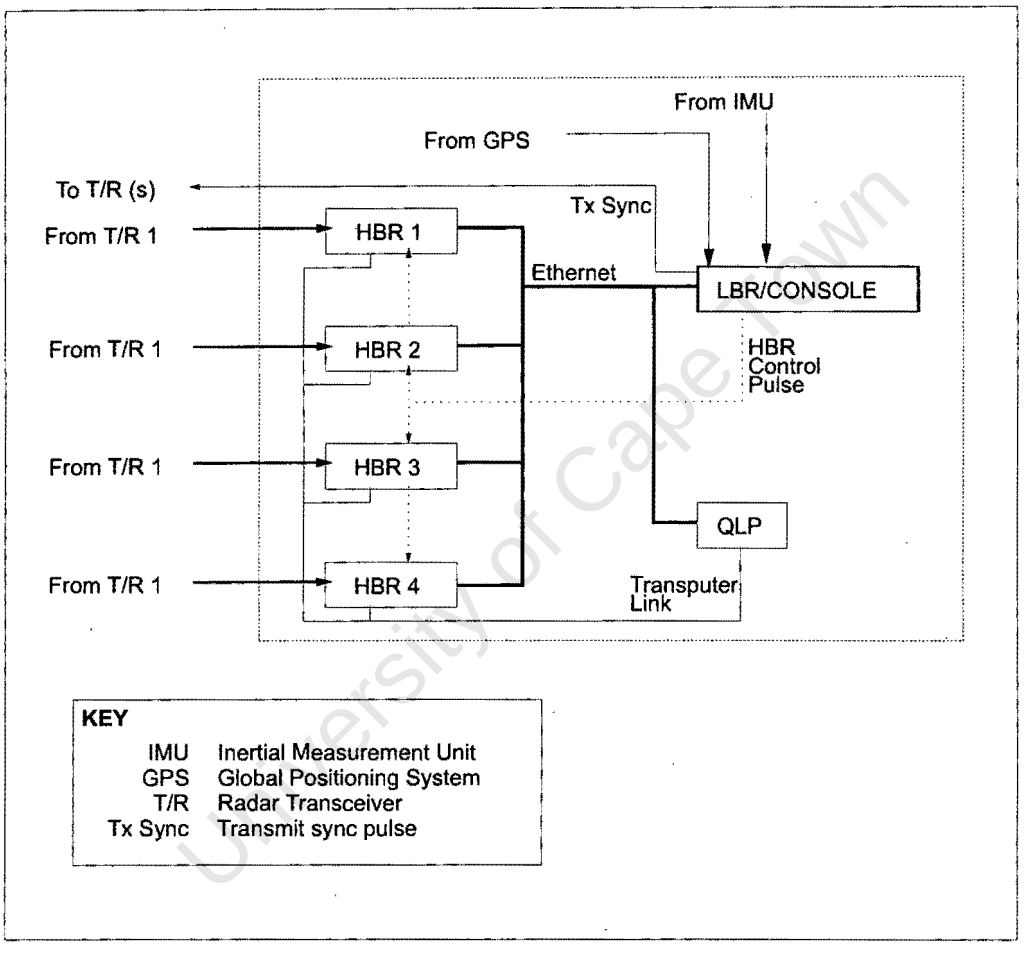

Figure 2.2: The SASAR Recording System.

The HBR

One HBR or High bitrate Recorder is required for each radar transceiver. Thus up to four HBRs can be accommodated in the system. The basic function of the HBR is to pre-process (pre sum and filter) the raw data and to store it on magnetic tape. The data is written to Exabyte tape in SAR580 format[10].

The LBR/Console

One LBR/Console subsystem or Low Bit-rate Recorder/Console subsystem is required for the recording system. This subsystem consists of two subsystems, namely the LBR and the console. The two subsystems were combined into one hardware platform consisting of a PC with a custom timing card. The LBR is required to store the navigation information from the GPS and inertial information from the IMU. This information is required for the high resolution ground processing. The navigation information is also passed on to the QLP. The console controls all the recording activities and provides the synchronisation pulses for the radar transceivers. The LBR/Console provides the control signals for the HBRs and the QLP via the timing card. Setup information for the QLP is also provided by the Console.

The QLP

The QLP is used to provide real-time medium resolution images. The QLP is connected to the HBRs to receive the pre-summed radar data, and to the LBR/Console to receive other data and control signals. The QLP will be covered in detail in the following chapters.

2.2.4 Ground Processing System

The function of the GPROCS or Ground PROCessing System is to produce high resolution SAR images. These images are produced off-line and therefore cannot produce images in real-time. The pre-processed radar data from the Exabyte tape is fed into the GPROCS along with the GPS and IMU data from the LBR. The GPROCS then processes the data to produce the final images. Refer to Appendix B for the GPROCS specifications.

Table 2.1: QLP Specifications for VHF System

Range resolution	30m
Chirp Pulse Bandwidth	5MHz
Azimuth pixel size	20m
Azimuth resolution	30m
Azimuth sidelobe level	20dB
Speckle reduction	none
Motion compensation	none
Swath width	20 km
Number of range bins	2048
image dynamic range	4 bits (24dB)
Screen resolution	1024 X 768 pixels
Image storage format	custom
PRF rate (after presumming)	39Hz

2.3 QLP System Requirements

This section discusses the required functionality of the QLP and covers the major specifications.

2.3.1 Functions of the QLP

The QLP is required to perform the following functions:

- Read in the pre-summed In phase and Quadrature (IQ) data from an HBR (This will be explained in greater detail in section 3.1.);
- Process the IQ data using the QFA;
- Produce a real-time scrolling display of the terrain ;
- Read geocoding information from the LBR and geocode scrolling image;
- Record this image to disk;
- Redisplay the recorded image when the system is off-line.

2.3.2 Specifications

Table 2.1 lists the QLP specifications for the VHF system. These specifications were given and form part of the problem statement.

Range resolution and Chirp Pulse Bandwidth.

The range resolution is defined by the radar hardware. The major parameter is the bandwidth of the pulse.

The effective bandwidth of a simple pulse is given by:

$$BW = \frac{1}{PulseWidth}$$

If a chirp pulse is used as in the case of the SASAR system, the effective bandwidth of the chirp pulse must be calculated. For the SASAR system it is 5Mhz.

The range resolution is then given by:

$$R = \frac{c}{2BW}$$

Azimuth pixel size

This is the length represented by one pixel on the QLP display in the azimuth direction, or the sampling interval in the azimuth direction measured in meters. Dividing the PRF by the aircraft velocity and multiplying by the presum ratio gives this parameter.

$$AzimuthPixelSize = \frac{PRF}{AircraftVelocity} \cdot PRESUM\ RATIO$$

The PRF is automatically varied as the aircraft speed fluctuates in such a way as to keep the azimuth sample spacing constant.

Azimuth resolution

SAR utilises the concept of a synthetic aperture to increase the azimuth resolution. This parameter is thus of prime importance in SAR processing. The azimuth resolution is dependent on the length of the synthetic aperture, and hence the number of subapertures used. Chapter 3 deals with this concept in more detail. At this stage it is important to point out the difference between the azimuth sample spacing and the azimuth resolution. The azimuth maximum obtainable resolution is set by the bandwidth of the transceiver, while the azimuth sample spacing is set by the sampling frequency (PRF) and the aircraft speed.

Azimuth sidelobe level

To measure this parameter, a simulated point target response is processed, and the ratio of the height of the point target to the first sidelobe is measured. This parameter is dependent on the algorithm used. Windowing techniques can usually be used to reduce this figure, however it is difficult in this case to apply a windowing function without dramatically increasing the computational load on the system. For this reason, windowing techniques were not implemented in this version of the system. For a more detailed explanation of the sidelobes, refer to Curlander [11].

Speckle reduction

Speckle reduction techniques are often used to reduce the speckle in radar images. These techniques require significant additional processing, which is not feasible in this implementation of the QLP. Speckle reduction is therefore not supported in this version of the QLP.

Motion compensation

Like speckle reduction, motion compensation is not implemented in this version of the QLP due to a lack of available processing power.

Swath width

The swath width defines the width of the strip being mapped. It is dependent on the number of range bins processed and the range bin size.

Number of range bins

This is the number of range bins across the swath. It is limited in practice by the maximum data rates of the recording system and the QLP.

Image dynamic range

The QLP algorithm produces output values scaled to fit between 0 and 255. This corresponds to a maximum dynamic resolution of 8 bits. This is also the dynamic range of the image stored on the hard disk. As will be explained later, the graphics adapter has dynamic range limitations. The number of grey shades used will therefore be 15 (the 16th one is reserved for geocoding in green), and therefore the image dynamic range is 4 bits on the screen.

Screen resolution

This is the maximum resolution supported at the time of implementation.

Image storage format

The image storage format is as follows:

RangeBin 0	**RangeBin 1**	**RangeBin ...**	**RangeBin n**	
$MAG_{0,0}$	$MAG_{1,0}$	$MAG_{...,0}$	$MAG_{n,0}$	**Range line 0**
$MAG_{0,1}$	$MAG_{1,0}$	$MAG_{...,0}$	$MAG_{n,0}$	**Range line 1**
$MAG_{0,...}$	$MAG_{1,...}$	$MAG_{...,...}$	$MAG_{n,...}$	**Range line ...**
$MAG_{0,m}$	$MAG_{1,m}$	$MAG_{...,m}$	$MAG_{n,m}$	**Range line m**

Where there are n range bins in each range line and m range lines were recorded. MAG is a byte representing the grey scale level from 0 (representing black), and 14 (white,) and all the shades of grey in between. Colour 15 is reserved for geocoding annotations and is set to green to allow for clear reading.

PRF

The PRF (Pulse Repetition Frequency) is the number of radar pulses that are transmitted every second. The PRF for the SASAR system is 1248Hz (see the Recording System Specification Document [7].) The data is presummed by a factor of 32, and hence the effective PRF is reduced to 39 Hz as indicated in the table.

Chapter 3

The Quasi Focused Algorithm

The Quasi focused algorithm (QFA) is an efficient algorithm for processing low to medium resolution SAR images. It was designed at UCT by the Radar Remote Sensing Group (RRSG) specifically for the QLP in the SASAR project [8].

3.1 Description of the QFA

This section contains a detailed look at how the QFA works. The SASAR system consists of a pulsed system in which coherent radar pulses are transmitted at regular intervals (corresponding to the radar PRF) along the aircraft flight track. The returns from these pulses are sampled at a rate which determines the range resolution. The output from the receiver consists of amplitude and phase information for each sampling point. This is represented as a complex number in the form of in-phase and quadrature components (I and Q).

The number of range bins is determined by the required swath width and the ADC sampling rate. The number of pulses is determined by the along track length of the image, the radar PRF and aircraft speed. SAR processing involves summing returns with an appropriate phase correction for each range bin. In full SAR processing each IQ value or vector would undergo a phase correction and summing would then be performed in the azimuth direction. Range migration is not considered in this version of the system, as the interpolations required to correct it would add further computational loading to the system. For the purpose of this algorithm, it is therefore assumed that the incoming data is orthogonal in range and azimuth directions. For a detailed explanation of range migration, see Curlander [11].

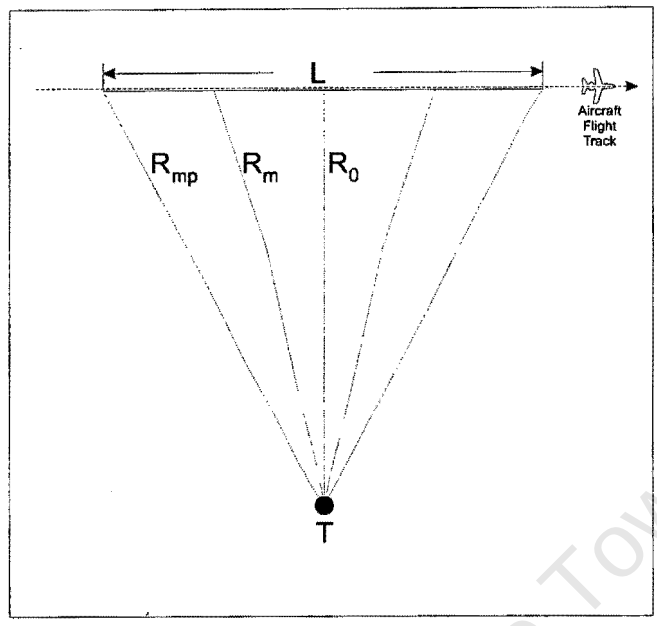

Figure 3.1: Phase Compensation Geometry

3.1.1 Unfocused SAR

Figure 3.1 shows the SAR geometry which leads to the need for phase correction. T is a target to the side of the aircraft flight track. R_{mp} is the maximum range at which the target falls within the radar beamwidth. R_0 is the range to the point of closest approach. Figure 3.2 shows a plot of the phase shift of the returned signal versus time or along the track distance. This parabolic shape indicates that small only small phase corrections are required for range values close to R_0. As one moves away from R_0, so the required phase corrections become larger and larger. Unfocused SAR does not carry out any phase corrections. At target ranges far from R_0, the phase shifts become too large to add constructively, and thus the inclusion of such returns would have a negative effect on the resultant resolution. It is thus necessary to determine the maximum array length which can be used before the returns start to add destructively. Appendix A of the SASAR QLP Design Document[6] shows Stimson's method of maximising the gain and minimising the beamwidth to calculate the longest possible unfocused aperture. Since the resolution attainable from the unfocused SAR technique is about 70m, which is below the QLP specification, the quasi focused SAR algorithm was devised. It is described in the next section.

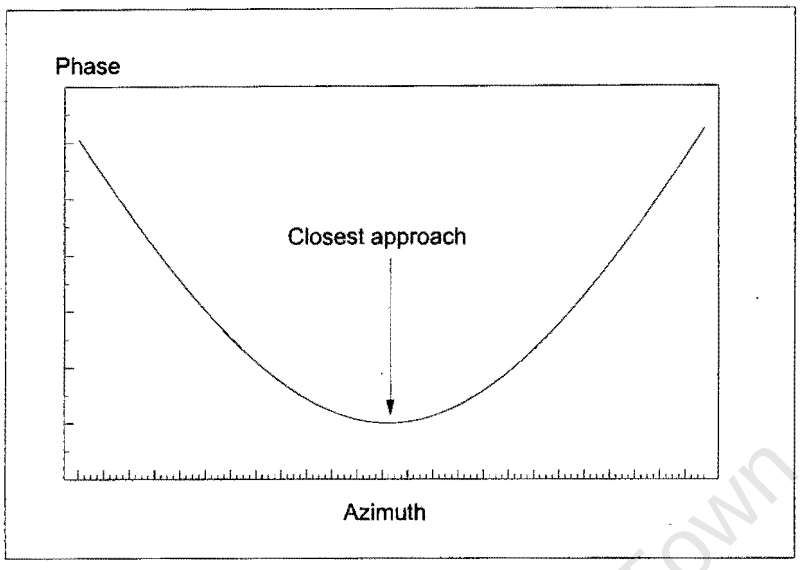

Figure 3.2: Phase shift vs. time

3.1.2 Quasi Focused SAR

In order to reduce the computational load which is demanded by the phase shifting and summing, Figure 3.3 shows a method of splitting up the aperture into several subapertures. Each of these subapertures is processed as in unfocused SAR.

Subaperture 0 is the unfocused SAR case as described previously. Subaperture 1 consists of two parts, a and b, which lie symmetrically about subaperture 0. The size of this subaperture is determined by the requirement that the phase shifts within the subaperture fall within a certain range

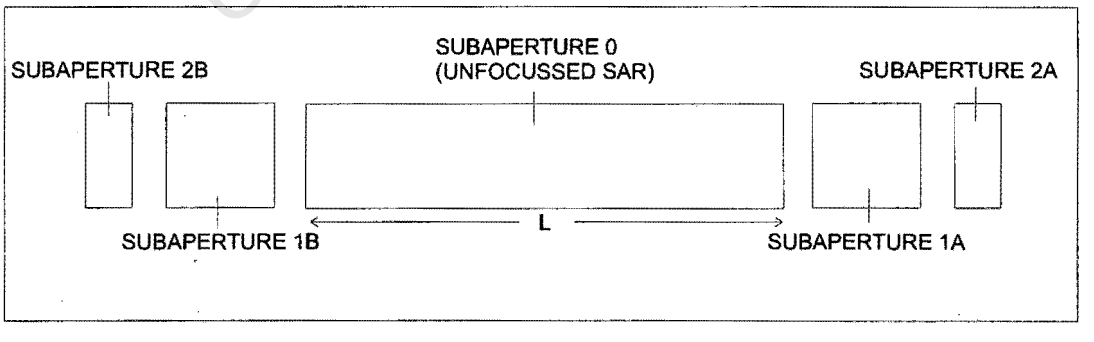

Figure 3.3: Subapertures in Quasi Focused SAR

as set by the phase tolerance criterion. Sub aperture 0 would need no phase correction. Subaperture 1 would need a phase correction equal to the tolerated spread of phase values within each subaperture. Subaperture 2 would need a phase correction of twice the tolerated phase spread, etc.

By including enough subapertures, the desired synthetic aperture length, and hence the required resolution can be obtained. Now all that needs to be done is to add up all the phase corrected subaperture sums by complex addition. The magnitude squared of this sum is then displayed on the screen as a shade of gray, representing the strength of the returned signal. The actual methods of implementing the algorithm are described in Chapter 4.

3.1.3 Key Parameters Affecting Computational Demands

Since the processing power available from the transputer network is directly related to the number of transputers used, and hence the cost of hardware, it is necessary to gain an understanding of what parameters affect the computational demands. Once this is understood, the computational demands can be adjusted to fit the hardware. There are a number of parameters which effect the processing load requirements. These include the required azimuth resolution and the number of range bins to be processed. These parameters are discussed in more detail below.

Azimuth Resolution

As the desired resolution is increased, so the length of the synthetic aperture is increased. As explained earlier in this chapter, a longer synthetic aperture requires more subapertures to be processed according to the QFA. The more subapertures processed, the more demands are made on the processing. Figure 3.4 presents a graph of required azimuth resolution vs processing requirements. The processing requirements are measured in a unit called a Transputer Processing Unit (TPU), which has been defined as the time in seconds taken for 1 transputer to process 256 range lines, 256 range bins wide, according to the QFA with all other parameters as in Table 2.1. The figures for this graph were obtained by actually running a series of tests on the algorithm after implementation and timing the processing at different resolutions.

From figure 3.4 it can be seen that the QFA is efficient at low to mid resolutions, and that efficiency drops off as required resolution increases.

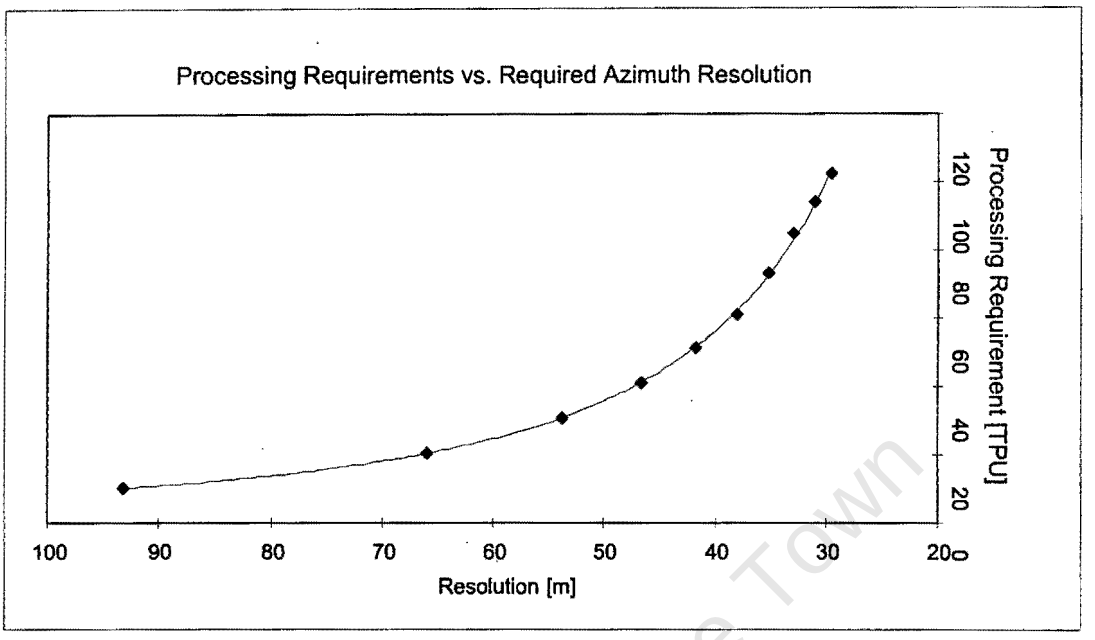

Figure 3.4: Graph showing algorithm computational requirements at different resolutions

Number of range bins

Figure 3.5 shows the linear relationship between the number of rangebins and processing requirements. Therefore the more range lines processed, the greater the processing demands placed on the QLP. This graph was also generated from tests performed once the algorithm was implemented, but it stands to reason that a linear relationship will exist.

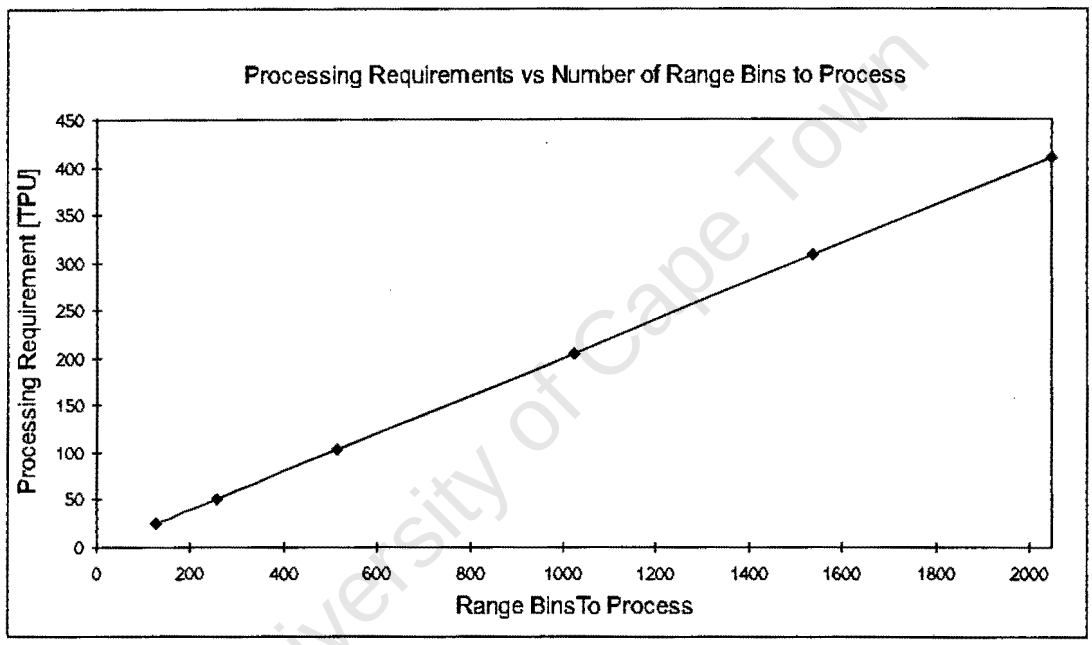

Figure 3.5: Graph showing algorithm computational requirements with a range of range bins.

Chapter 4

System Design

This chapter documents the design of the QLP. Firstly a diagram of how the data flows from the radar antenna right through to the QLP display is given in Figure 4.1

The reflected radar energy from a pulse is received by the radar transceiver in the form of an in-phase and quadrature part. This reflected pulse is then sampled at a rate of 25MHz by the 8-bit ADC. The resulting data rate is thus 25Mbytes per second per channel. Since the waveform is only sampled when a pulse is received, the average data rate is considerably lower and is in the order 2.5Mbytes per second per channel. The FIFO reduces the burst data mode to a sustained data rate of 2.5Mbytes per second. The data rate is further reduced by the presummer, which decreases the data rate to 80kbytes per second per channel, where 1kbyte = 1024 bytes. Thus 160kbytes per second of data enters the transputer link adapter for both I and Q channels. From here the data is transferred via a transputer link to the QLP for processing. After processing, the processed image is displayed on the PC display.

Now that an overview of the functionality of the QLP has been given, the detailed design is described.

4.1 Hardware Design

The QLP system is designed around a PC with a network of 9 transputers. The reasons for this choice are explained in sections 4.1.1 and 4.1.2. Figure 4.2 shows the final QLP system in the form of a block diagram. This diagram was adapted from Figure 2.1 in the SASAR QLP Design Document [6].

As can be seen from the diagram, there are two distinct sections, namely the host PC and the transputer network. These are housed in separate

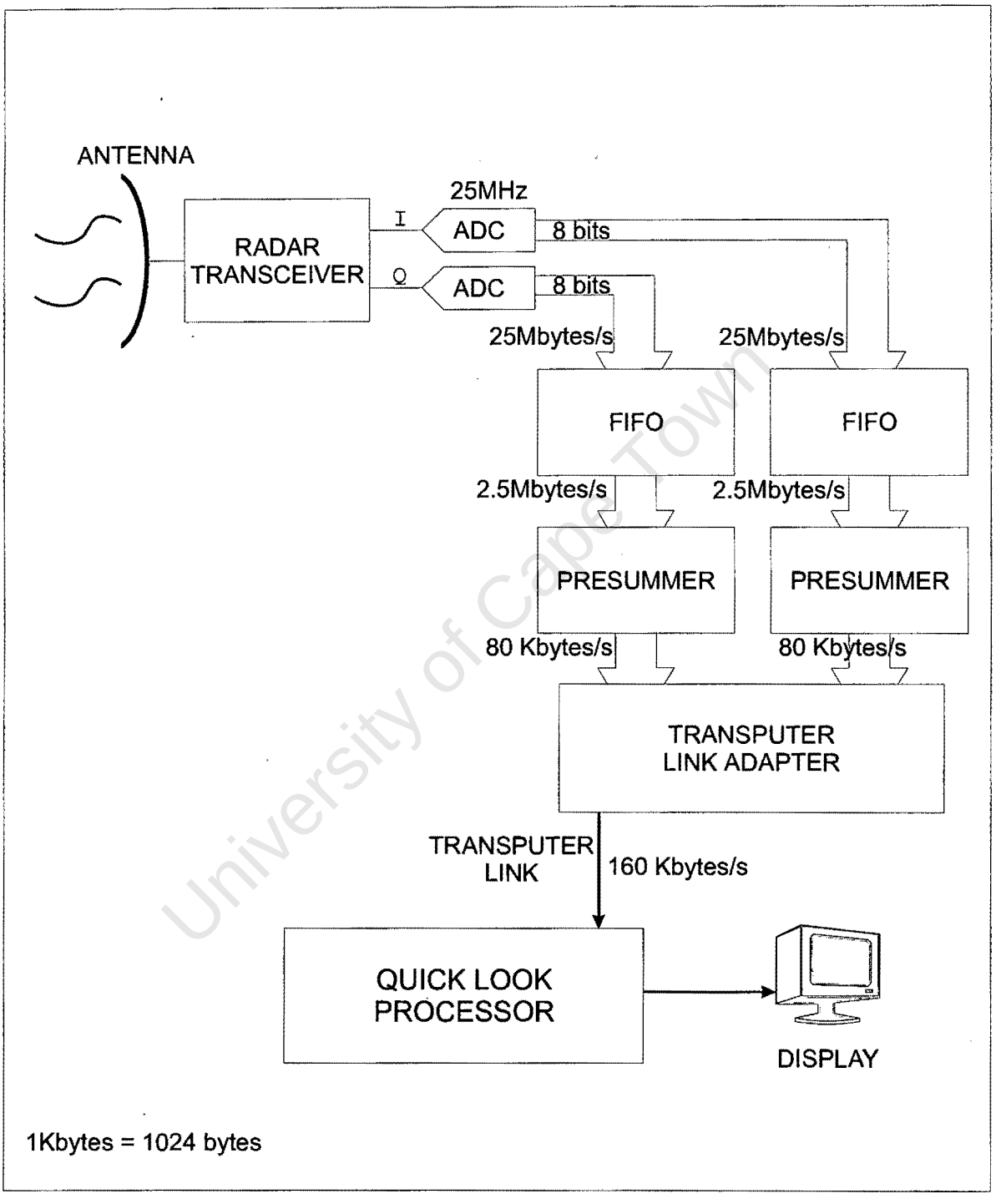

Figure 4.1: Data Flow from Radar to QLP Display.

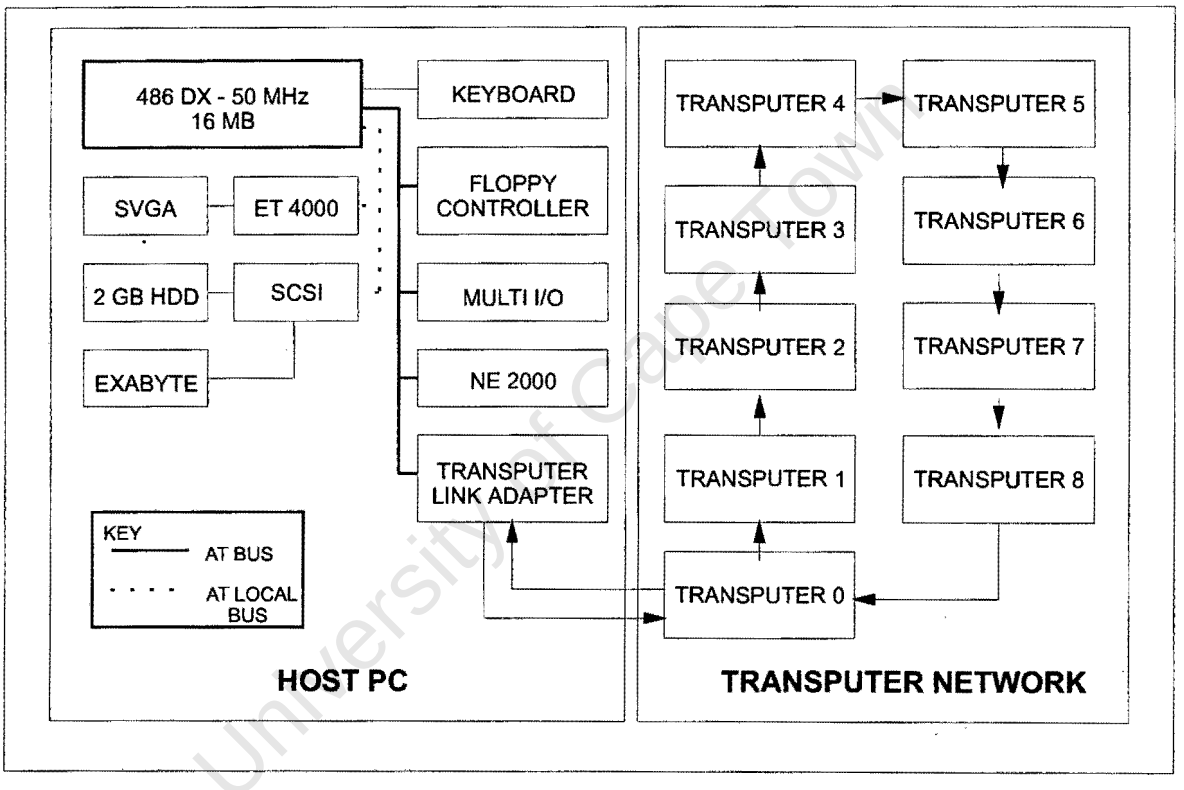

Figure 4.2: The QLP Subsystem.

chassis. The host PC is responsible for the following functions:

- Booting and uploading code to transputers;
- Receiving the configuration data from the console subsystem;
- Initialising transputers with configuration information;
- Receiving image data from transputer link adapter;
- Displaying image on a scrolling display;
- Adding geocoding annotations to the scrolling image;
- Writing QLP image to disk for later retrieval.

The master transputer is responsible for communicating with the PC via the transputer link adapter and with the slave transputers.

Each slave transputer in the network is responsible for the following functions:

- Reading in the raw data and previously processed data from the preceding transputer;
- Processing a portion of the remaining raw data;
- Passing on the remaining raw data and the processed data.

4.1.1 Host PC

The host PC has the following important features:

- 80486 DX-50 Processor;
- Local bus ET4000 Super VGA graphics adapter;
- 2 Gbyte SCSI hard disk drive;
- Transputer Link Adapter.

PC technology was selected because it provided a suitable development environment, as well as cost effectiveness. The 80486 DX-50 Processor was selected as it had the highest bus speed of PCs available at the time of implementation. A high bus speed is important for the efficient transfer of image data to the display adapter and hard drive.

A local bus ET4000 Super VGA graphics adapter was selected as the ET4000 is an industry standard video chip. In addition it supports virtual screens and hardware scrolling which makes it highly suitable for the application in mind. A local bus adapter is used to maximise data transfer rates. Pixels are plotted to the display at a rate of approximately 80,000 every second[1]. A 2 Gbyte SCSI hard disk was chosen as approximately 300 Mbytes of image data are written to the disk every hour. The transputer link adapter is required for the master transputer to communicate with the PC.

4.1.2 Transputer Network

The transputer network consists of a host transputer, which communicates with the PC via the link adapter, and 8 target or slave transputers. These are housed in a separate chassis. Although the transputer platform was given, there is a certain amount of flexibility allowed in terms of the configuration of the transputers. Transputer technology was used as at the time of implementation it was the most cost effective, scalable, high performance microprocessor available.

A transputer is a high performance microprocessor with its own local memory and with links for connecting one transputer to another transputer. By connecting many transputers together, many such microprocessors can be used concurrently in one application. The architecture allows for the transputers to be connected together with links. These links are used to transfer data between the transputers.

Each transputer has 4 links, allowing for three basic configurations. These are:

- Star configuration
- Ring configuration
- Free form configuration

  Each of these is discussed, along with the advantages and disadvantages of each.

Star configuration

In this configuration all the slave transputers are connected directly to the Master transputer. The advantage of this system is that the master transputer can communicate directly with each of the slave transputers and vice

[1] $2048\ rangebins/line \times 39\ lines/second = 79872\ rangebins/second$

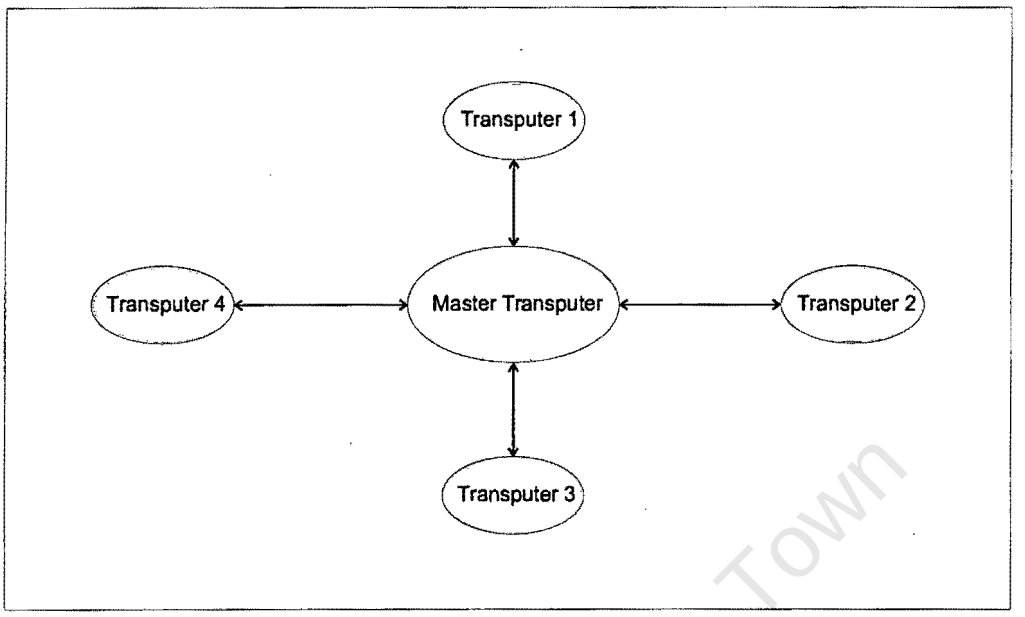

Figure 4.3: Star configuration

versa. This simplifies design of communication protocols. Unfortunately this configuration is severely limited by the fact that each transputer can have a maximum of 4 links, and therefore only 5 transputers could be connected in this fashion. Another link would still be required to connect the master transputer to the host PC.

An example of a star configuration can be seen in Figure 4.3

Ring configuration

In this configuration the transputers are connected together in a ring or pipeline. The advantage of this configuration is that any number of transputers can be connected together in this fashion without increasing the link requirements. This means that the system can be expanded as much as is required. One disadvantage is that the Master transputer is not directly connected to each transputer, and so a communication protocol needs to be set up so that data can be transferred through the intermediate transputers to the one requiring the data. This adds complexity to the program. See figure 4.4 for an example.

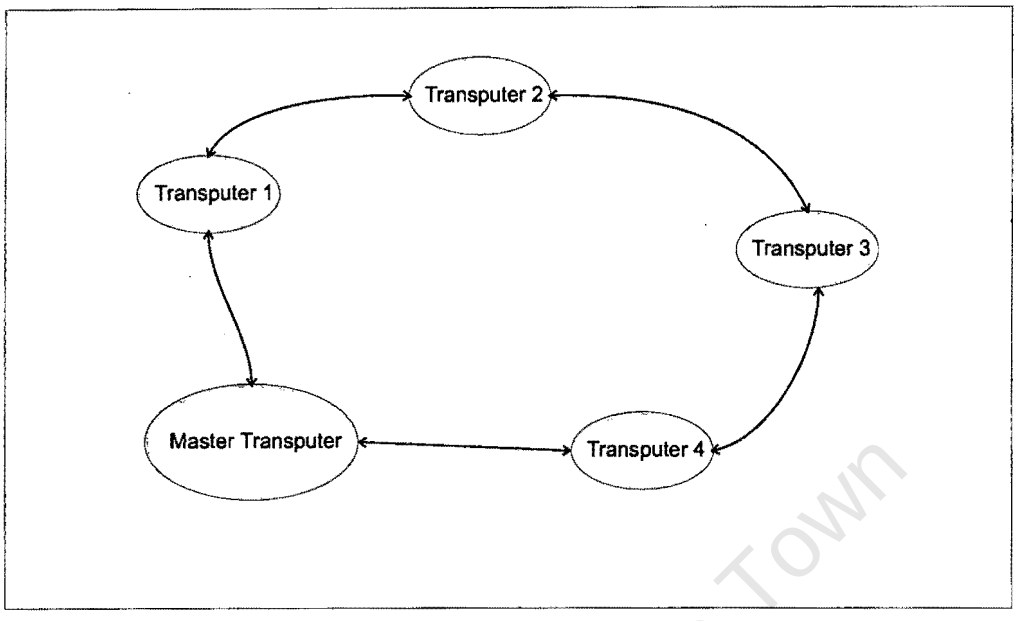

Figure 4.4: Ring Configuration

Free form configuration

Any combination of these configurations could be devised, to suit the requirements, as long as no more than 4 links per transputer are required. Figure 4.5 shows such an example. The advantage of this configuration is that the hardware can be designed to fit more closely to the software algorithm in mind, and can therefore be highly efficient. Thus, flexibility is one if this configuration's strong points. Unfortunately this system does have its limitations; It is difficult to do an overall upgrade for such a system by simply adding more transputers, and systems based on this configuration tend to get rather complex.

Based on the above information, it was decided that the ring configuration would be the most suitable for the QLP, as any number of transputers could be used, depending on the computational requirements. Although direct communication between transputers and the master transputer is not possible, this does not significantly decrease efficiency, as shall be shown in Chapter 5 where the transputer's performance is analysed.

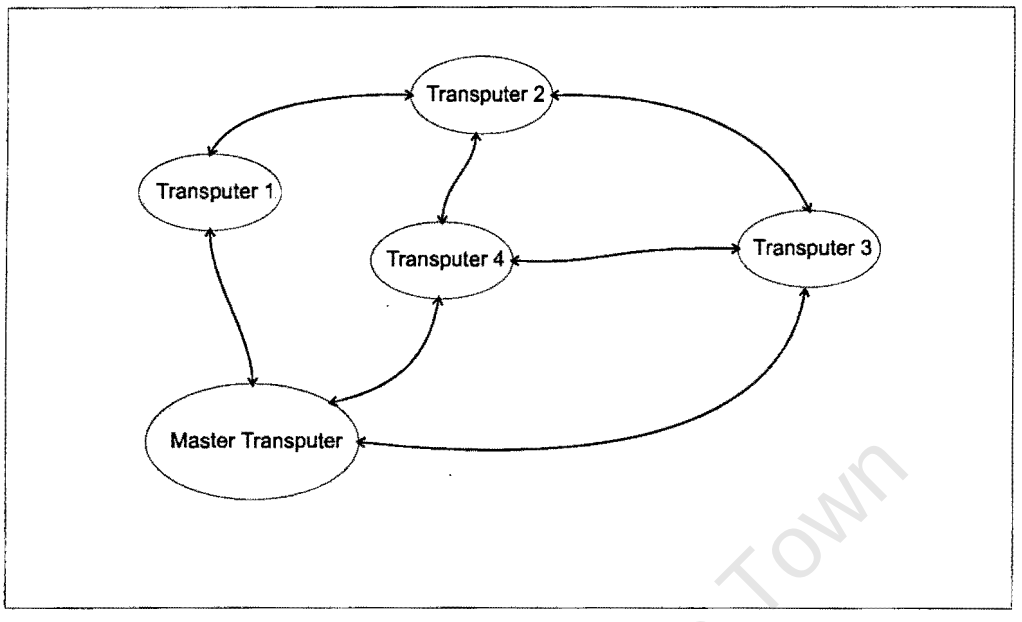

Figure 4.5: Freeform Configuration

4.2 Software Design

The Software design is described below, and is divided into two sections, namely the host PC software and the transputer software.

4.2.1 Host PC Software

The Language used for the host PC software is 'C', which is an industry standard language. Borland C++ V3.1 was used as it provides a good integrated development environment with excellent debugging facilities.

Basic program flow

Figure 4.6 shows the flow of the host PC software. First the transputers are booted by a batch file calling the *iserver* program which is supplied with the transputer development system [16]. The mouse is then initialised and the constrains of movement are defined. The parameter file is then read. This contains information such as the radar parameters and subaperture sizes. An example of a parameter file is given in Appendix C.

The IPX packet driver is initialised and an IPX listen for packet request is issued. The QLP then waits for a start signal from the Console system. Once the start signal has been issued, the transputer network is initialised

Table 4.1: Transputer Link Interface I/O Map

BASE_ADDRESS	150 Hex
DATA_IN	BASE_ADDRESS + 0
DATA_OUT	BASE_ADDRESS + 1
READY_TO_SEND	BASE_ADDRESS + 2
READY_TO_RECEIVE	BASE_ADDRESS + 3

with the relevant parameters. The correct graphics mode is set along with a grey-scale colour palette. The graphics screen is now clear, and ready to display the image. Another IPX listen for packet is issued, and the QLP enters the main loop. At this point the QLP is ready to start processing. The processing will be explained in the section on the transputer software.

Once processed data is ready, a flag in the transputer link adapter will indicate that data is ready to be read out. The host PC then reads out a range line of data, displays it on the bottom of the screen and scrolls the screen up 1 line. The range line of data is also written to the hard disk. The program then checks for network data from the LBR/Console. If a packet is waiting, it is read. If the packet is a geocoding packet, the data is formatted and output to the screen. If the packet is a stop instruction from the console, a stop signal is sent to the master transputer and the program closes the image file and exits. Otherwise the program loops back and awaits another line of processed data.

Program elements

This section explains in more detail the elements of the host PC software code.

Transputer link adapter interface The transputer link adapter is an 8 bit ISA expansion card which is installed in the host PC. Table 4.1 gives the I/O address map. To write to the transputer link interface, the address READY_TO_SEND is polled until the interface is ready, and then a write operation is performed to address DATA_OUT. Reading from the transputer link interface is done in a similar manner, but polling READY_TO_RECEIVE and reading from DATA_IN.

This interface has three primary functions:

1. Boot and upload the master transputer with its code;

2. Transmit configuration parameters to master transputer;

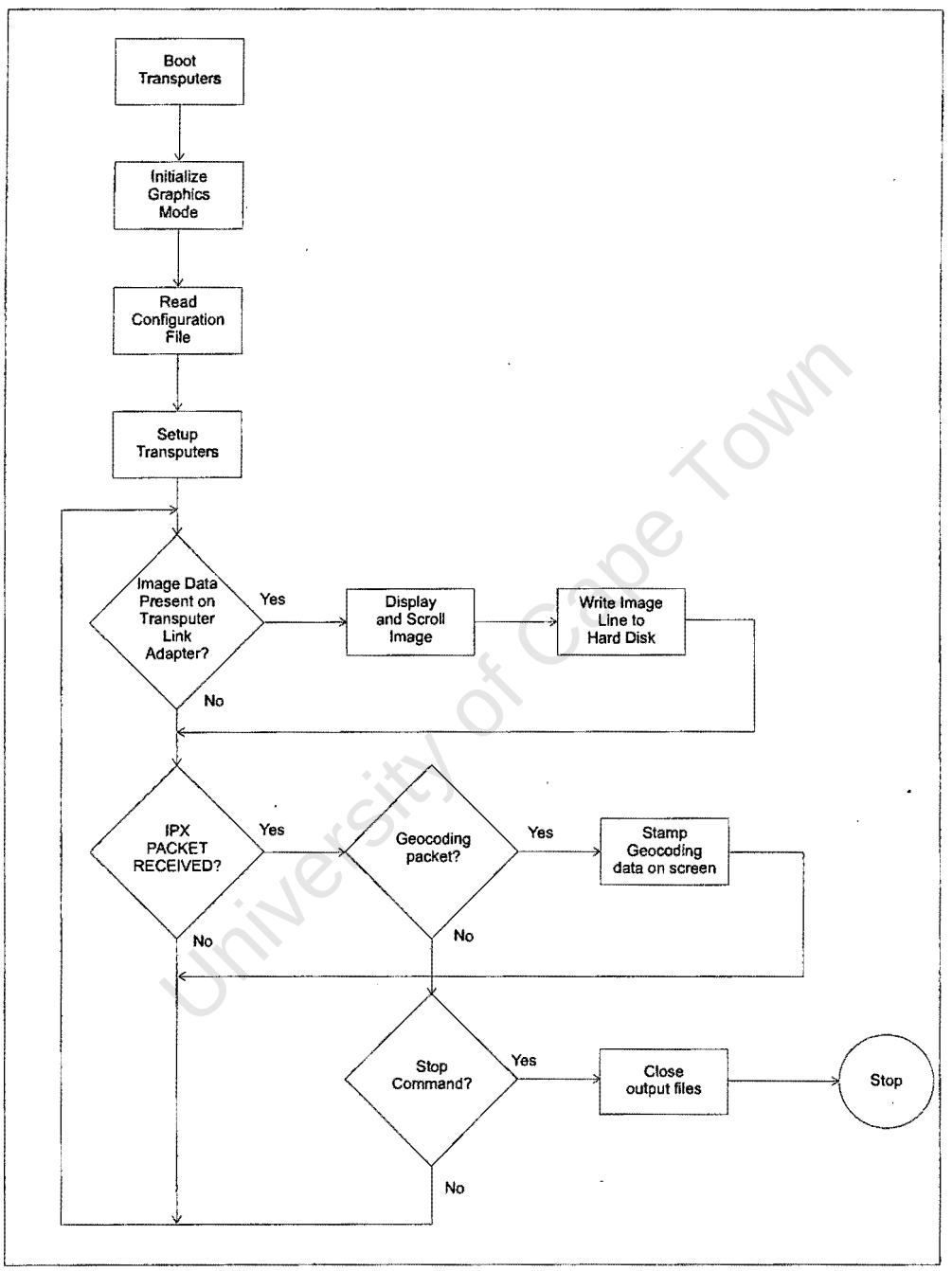

Figure 4.6: Flowchart of Host PC Software

3. Receive processed image data from transputer network.

Booting the transputers and uploading the QLP code to the master transputer is not discussed, since it is simply done by a call to an external program supplied with the transputer development system.

Transmit configuration parameters to master transputer The transputers need to be configured with the correct radar parameters so that they can process the data correctly. This information is passed to the transputers through this interface. Information such as the number of subapertures to use and the number of range bins to process are transmitted here. The parameters are as follows:

1. Scale factor;
2. DC Offset;
3. Size of Aperture;
4. Number of subapertures;
5. Subaperture markers.

Receive image data from master transputer Once the data has been processed, the host PC needs to receive the image data and display it on the screen. The image data is read from this interface.

In order to operate in real-time, the interface is required to receive 39 range lines of image data every second. This figure is the PRF measured after presumming. See Table 2.1 for more details. If this figure is multiplied by the number of range bins, we get the required sustained data rate from the interface. It has been calculated to be 78kbytes per second. This data rate is easily achieved by the ISA bus. Only image data is received through this adapter so it can be assumed that if READY_TO_RECEIVE is true, that image data is going to be received.

Graphics code In order to display and scroll the processed image on the screen, as well as overlay geocoding text, a set of high performance graphics routines needed to be developed.

Number of horizontal pixels	2048
Number of vertical pixels	768
Colour Pallette	greyscale + 1 other colour
Vertical scrolling rate	39 lines/s
Text overlaying	Yes

Table 4.2: Graphics Requirements

Required graphics specification The graphics requirements for the QLP display are summarised in the Table 4.2.

The horizontal direction corresponds to the range direction, while vertically corresponds to the azimuth direction, or the direction of flight. By multiplying the number of horizontal pixels with the vertical scrolling rate, it can be calculated that 79,872 pixels need to be plotted every second. This combined with the vertical scrolling is highly demanding on the host PC. Another concern is that at time of implementation, the maximum number of pixels that a SVGA could display across the screen is 1024. In order to see all 2048 pixels on the SVGA screen, some form of horizontal scrolling is required.

Graphics implementation To solve these problems, a VESA local bus ET4000 Super VGA adapter was used. Firstly it is a local bus card running on a 50MHz bus, therefore maximising the efficiency of writing data to the card. Secondly it supports hardware scrolling which eliminates the scrolling problem. To circumvent the problem of sideways scrolling, use was made of the virtual page support provided by the ET4000. Operating the ET4000 in 16 colour as opposed to 256 colour mode also reduces the graphics bandwidth and graphics memory requirements. Table 4.3 shows the graphics parameters used.

1024 x 768 is the highest resolution supported. The ET4000 has 1Mb of graphics RAM, in which the entire virtual screen must be contained. If running in 16 colour mode, 4 bits per pixel are required, and therefore the maximum number of pixels that can be stored in 1Mb RAM is 2,097,152. If we require a horizontal virtual screen size of 2048 pixels, this leaves us with 1024 pixels vertically. Although only 768 pixels are required, the remaining memory was utilised, allowing the QLP to store 256 extra range lines on screen. These lines can be seen by scrolling up and down as the image constantly scrolls up at 39 lines per second.

A greyscale colour palette is required. To set the colour palette on the ET4000, the RGB table is edited for each colour entry. One of the 16 colours

Screen Resolution	1024 x 768 pixels
Virtual Screen Size	2048 x 1024 pixels
Colour mode	16 colours

Table 4.3: Graphics Parameters Used

was set to green, allowing green text to be overlaid on the image for geocoding purposes.

Scrolling on the ET4000 is achieved by simply specifying the start address in the virtual page where the image must be displayed. Each time a line needs to be plotted, the screen is scrolled up 1 pixel (by changing the start address), and the new line is added to the appropriate location in the virtual screen. Horizontal and vertical scrolling is controlled by the mouse, allowing fast and flexible panning. To increase efficiency, the pixels were not plotted individually, but rather in blocks of 8 pixels. This reduced some of the overheads in addressing the ET4000 card.

The LBR/Console interface The functions of the LBR interface are as follows:

- Pass command and status messages between the Console and the QLP so that the Console can control the QLP's operation;
- Transmit geocoding data from the LBR to the QLP to enable the QLP to geocode the image data.

A network interface was chosen as the interface to the LBR system. A network connection is a highly flexible method of implementing the interface. The IPX protocol was used to develop the network routines, which allowed the high level interface code to send packets to any destination machine. A set of functions was written to send and receive the control packets. Appendix A gives a short introduction to IPX programming and explains briefly how the functions were implemented and the packet formats involved. The code for the network routines can be found in Appendix C.

Hard disk interface In order to write the image data to the hard disk in real-time a sustained data transfer rate of 78Kbytes per second must be achieved. Although a standard SCSI controller card with a SCSI hard disk drive can handle this transfer rate, latency is a problem. The reason for this is that when a block of data is written to disk, the controller first holds the PC while the disk spins round to the correct position. At this point

the data transfer takes place, and then control is returned to the PC. Upon implementation, these delays were found to affect performance detrimentally. A caching controller card was tested, and found to be the solution to the problem. When a block of data is to be written to disk, it is first transferred to the cache memory, then control is regained by the PC, while the controller card in its own time transfers the data from cache memory to disk once the disk spins to the correct position. The results of the timing tests performed to verify this are shown in the table below. The rates in the table are measured in lines per second processed, where a range line is 2048 range bins long.

Controller Card Type	Rate(No writes to disk)	Rate(Writing to disk)
Non-cached	45.5	27.5
Cached .	45.5	43

A rate of at least 39 lines/s is required. As can be seen the non-cached controller does not match up to these specifications, whereas the cached controller solves the problem.

Image retrieval software In order to view the images after flying, image retrieval software was written to read the image data off hard disk, and display it on the screen. This software is simple, and relies heavily on a subset of the routines developed for the QLP. For this reason it is not covered.

4.2.2 Transputer Software

Occam was chosen for the transputer programming. This language was designed to take full advantage of the transputer's features and parallel structures. This section will show how the algorithm was implemented on 1 transputer. Once this has been done, a description will be given of how the software and the transputer links interrelate and how the processing work load is divided up between the transputers.

Structure of algorithm

Figure 4.7 shows the structure of the algorithm in the transputer. Each transputer can be thought to have a copy of this structure in its memory. Each transputer simply gets a different set of data to process. Each small square in the figure represents an I and Q value which in turn represents a complex number, which will be referred to as an IQ value. An $n \times m$ array of IQ values is set up where n is the number of range bins and m is the length of the entire aperture. This is represented by the large shaded

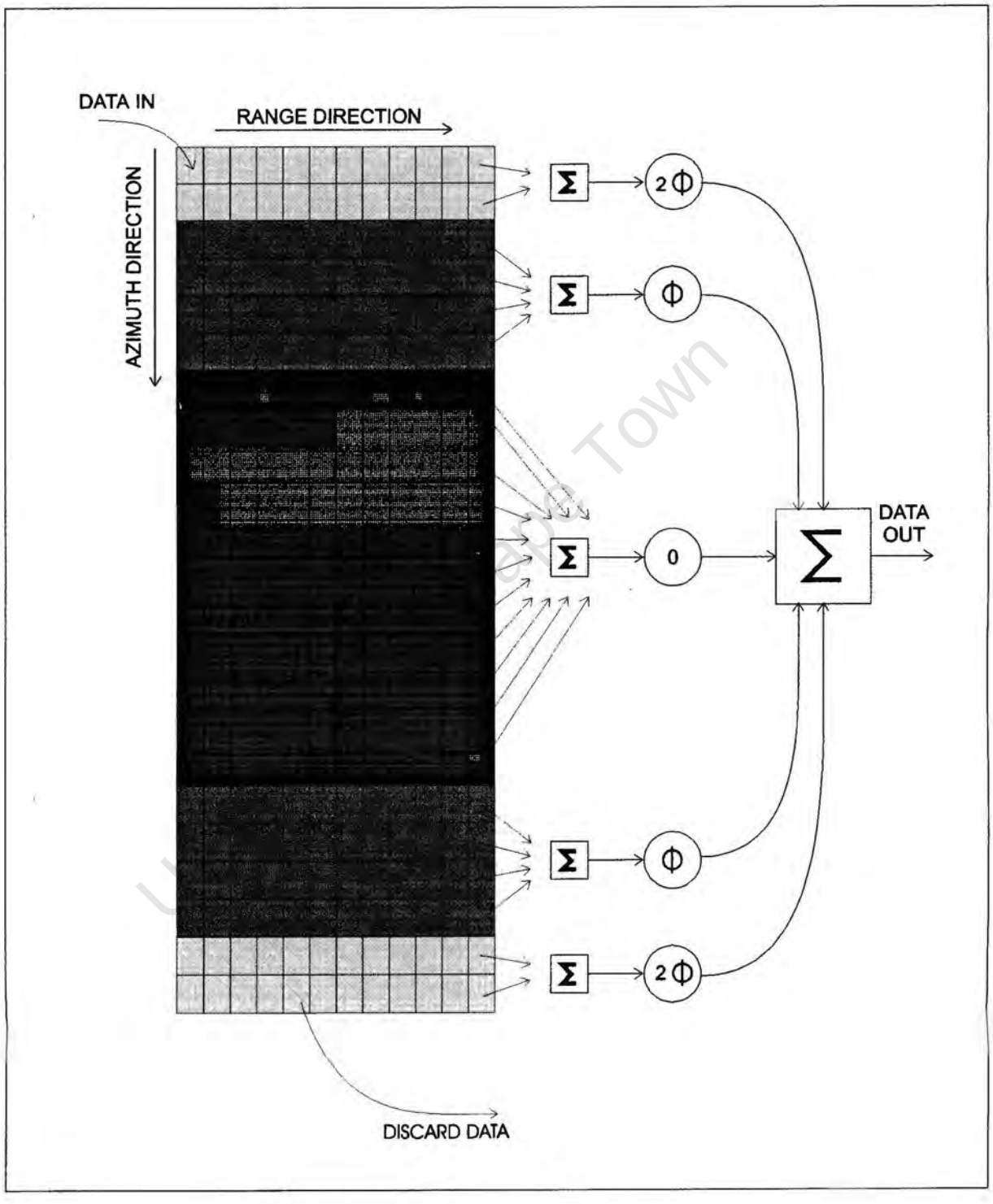

Figure 4.7: Internal Data Format of QLP

block in the diagram and is in fact the entire aperture. In this simple case $n = 12$ and $m = 23$. Typically n would be 256 and m 120. The subapertures are indicated by different shadings, in the diagram. The middle section represents the first subaperture, while the top and bottom sections represent the third subaperture pair. Again, in this example there are two subaperture pairs and a central aperture while in reality we would have perhaps as many as 19 subaperture pairs and the central aperture. The square blocks with the sigma signs inside them represent the subaperture sums for range bin n. The circles represent a phase shift of the subaperture sum. The large summation block sums all the phase shifted values together. As a new line is read in, the data needs to slide down, in the azimuth direction, and a new range line read in at the top. The range line at the bottom is discarded, as it is no longer required. Each time this happens, the transputer re-calculates the sum blocks, phase shifts and the final sum. This is done for each range bin in turn. The resulting set of sums is the final image data. This is then passed out and the process repeats itself.

In order to improve on efficiency, the data does not physically slide down the aperture as explained above. A set of pointers demarcating the divisions between the subaperture pairs slide through memory in the opposite direction, hence the effect is the same, but with a significant reduction in processing. Another area where processing is reduced is in the area of summing the subaperture magnitudes. Instead of looping through each subaperture, summing the values to obtain the subaperture sums, a running total is kept, and as each value slides out of a subaperture its value is subtracted from that running total, and as a new number enters, its value is added to this running total.

Parallelisation of the algorithm

If the QFA is analysed, it will be discovered that the algorithm is independent in the range direction. That is, each range bin, is completely independent from every other one. This means that each transputer could process a set of range bins without any knowledge of the range bins being processed by another transputer. Therefore the task of processing all n range bins can be divided up between all the available transputers.

Process allocation Each slave transputer is allocated an equal number of range bins to process. If for example we are using 8 slave transputers and we require 2048 range bins to be processed, each transputer would process $\frac{2048}{8} = 256$ range bins. In this way the processing load can be distributed evenly across the available transputers.

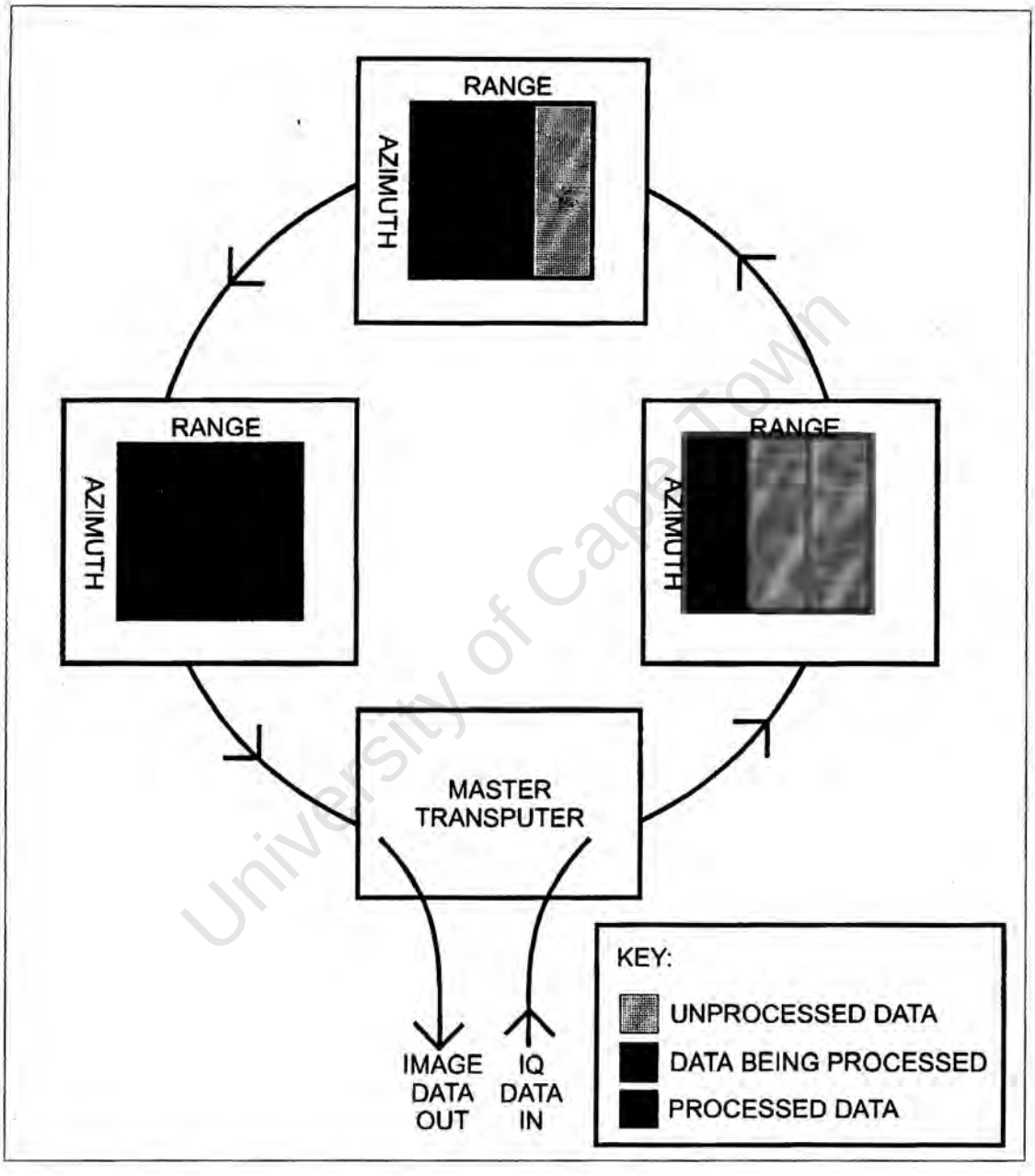

Figure 4.8: Pipeline Structure of QLP

Figure 4.8 shows the ring structure and how the process is divided up on all available transputers. For simplicity, only three slave transputers are used in the figure. IQ data is fed into the master transputer from the HBR. The data is then sent to the first slave transputer. The slave transputer processes the first third of the data, and then passes along this processed data along with the remaining two thirds of unprocessed data. The second slave transputer then processes the second third of the data, and passes along the two thirds processed data to the last transputer for processing. The last transputer processes the last section of the range and passes all the processed data back to the master transputer. The master transputer then sends the processed data back to the host PC for displaying.

Inter-relationship between software and hardware.

Figure 4.9 shows how the software and hardware inter-relate. The circles represent software processes, and the dashed squares the physical transputers on which they are executing. The transputer links are also shown with curved arrows. The links "to.helper" and "from.helper" are software links, while the rest are actual physical connections. Communication between the master transputer and the transputer link adapter (hence the host PC) is done through the "to.host" and "from.host" links. Thus the host PC software communicates with the master transputer's "main" process through these links. The HBR data is directed into the "main" process via the "from.hbr" link. The "helper" process communicates with the main process via the "to.helper" and "from.helper" links. The helper in turn transmits raw unprocessed data to transputer 1 via link $pipe_0$, and receives processed image data from link $pipe_8$ via transputer 8.

The code for the master and slave transputers is looked at more closely below:

Master transputer code

The master transputers' two running processes are described below:

"main" process

Firstly the "main" process reads the setup parameters from the host PC. The parameters read are:

- Scale factor;
- DC offset;

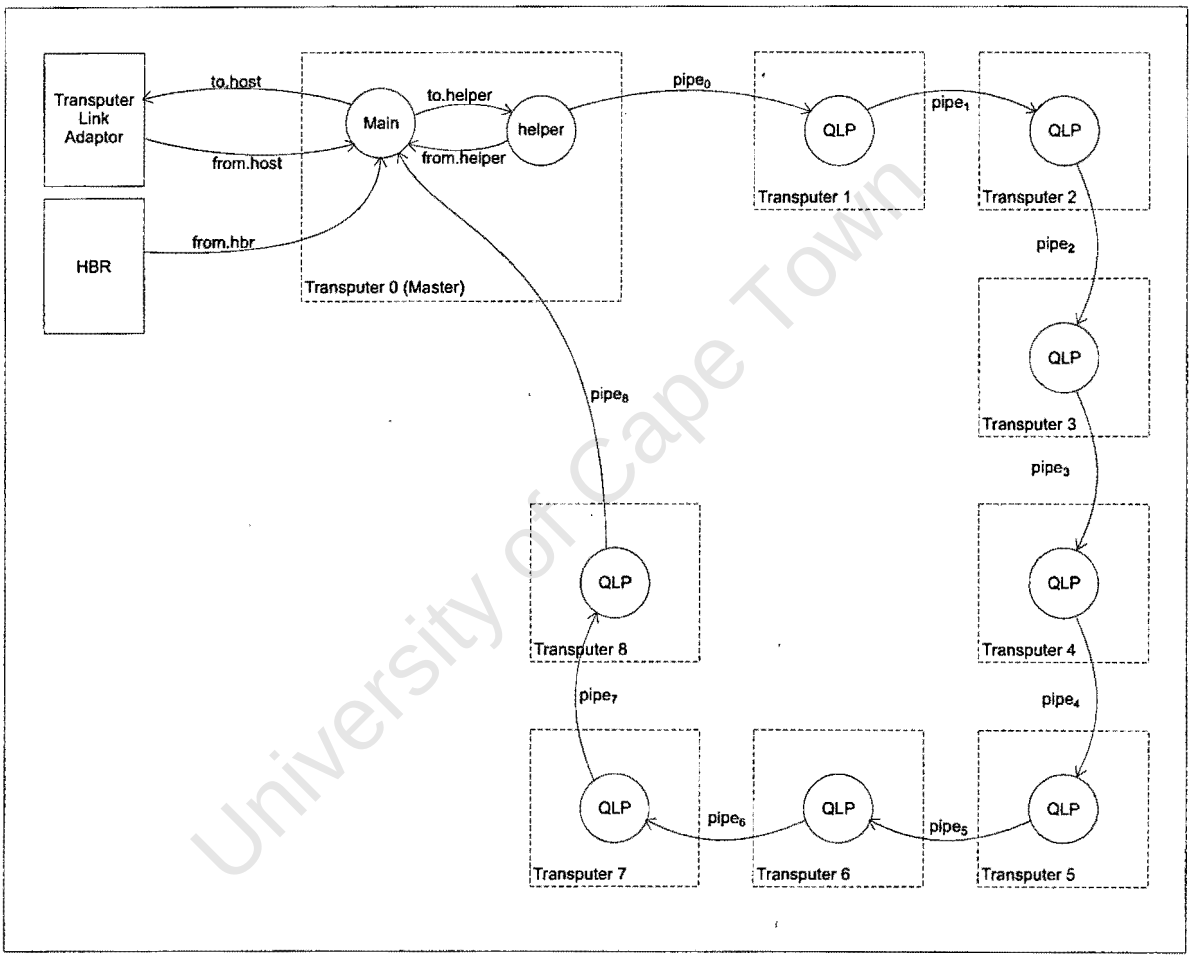

Figure 4.9: The inter-relationship of the software and transputer links

- Size of aperture;
- Number of range bins;
- Number of subapertures;
- Subaperture markers.

Once this is complete, these parameters are passed to the "helper" process. The process then waits as these parameters are passed from the "helper" process round the transputer network, and returned to the "main" process. The QLP is now configured and ready to run.

As can be seen in the diagram, there are four input links to this process, namely "from.host", "from.helper", "from.hbr" and "$pipe_8$". Each one of these inputs is monitored for an input. If a "more" request comes from "from.helper", the "main" process fetches a range line from the "from.hbr" link, and sends it to the "helper" process via the "to.helper" link. If processed data is received on the "$pipe_8$" link, it is assumed to be fully processed data, and it is in turn sent to the transputer link adapter via the "to.host" link. If a "halt" command comes from the "from.host" link, the halt is forwarded to the "helper" process, but the "main" process itself does not halt, but waits until the halt signal is returned via the "$pipe_8$" link. At this point, the "main" process terminates.

"helper" process

This process simply assists the "main" in smoothly handling the data in and out. Basically if the "helper" process receives data from the "main" process, it sends the data on to the transputer network via "$link_0$". Once the data has been transmitted (it first has to wait for transputer 1 to accept the data), a "more" signal is returned to the "main" process indicating that more data can be accepted. This allows the "main" process to multiplex and look for other inputs (e.g. data returning from the transputer network) at the same time as looking for the "more" signal. If the helper receives a "halt" signal, is simply passes it on to transputer 1 and then terminates. Clearly to initiate the whole procedure, the "main" process has to "prime" the system by sending the first range line without a "more" signal.

Slave transputer code

Each slave transputer has a copy of the "QLP" process running on it.

If the "QLP" procedure receives configuration data, it initialises its internal variables from this data, and passes the data on to the next transputer to

use. Once all transputers have been configured, the config data arrives back at the "main" process. The "main" process now knows that the transputer network is now ready.

One of the parameters is the number of range bins to process. This is set to the total number of range bins to process divided by the number of slave transputers. So for 2048 bins, each transputer needs to process 256 range bins. When the "QLP" process receives a dataset (1 full range line), it processes the correct group of 256 range bins, so for example transputer number 2 would process the second group of range bins (bin 256 to bin 511). The processed data, along with the unprocessed data is then sent to the next transputer in the chain. Once the last transputer has processed the last group of range bins, the final data represents a fully processed range line. This range line is then transmitted to the "main" process on the master transputer. From there it is sent to the host PC for displaying.

If a "halt" signal is received, the "QLP" process passes this signal on to the next transputer, and the process is terminated. In this way, the "halt" signal moves through the transputer network, halting each process as it goes. Once the halt process arrives back at the "main" process, the "main" process halts, knowing that all other processes have halted correctly.

Optimisations A number methods of optimising the transputer code were performed. These are briefly described in this section.

Use of defines Occam has an unusual method of optimising code by the use of defines. If, for example, in a loop the array IQ.Data[SubApertureMarker[i]][j] is accessed frequently, the following notation will pre-calculate the address of the start of the array:

```
VAL IQ IS IQ.Data[SubApertureMarker[i]][j]:
```

Thus to access that array, all that is required is to use IQ[i], and the address of IQ[0] will already have been calculated, and so accessing the array will be significantly quicker. Use of this has been made throughout the program to ensure it runs as efficiently as possible.

Lookup tables To compute the phase rotations, the sine and cosine of the rotation angles are required. Fortunately all these rotation angles are known before the processing begins, and so a lookup table consisting of the sine and cosine of the rotation angles can be constructed.

General techniques Other techniques were used to ensure maximum efficiency. The use of pre-calculated intermediate values and eliminating the unnecessary use of intermediate variables for example was always taken into consideration. Another optimization example is as follows: the phase shifts for each subaperture pair are identical. For this reason, the running totals can be merged together for the subaperture pair, and then only one phase shift for each subaperture pair occurs. This and other similar techniques have further sped up the processing.

Chapter 5

System Performance Analysis

Initial calculations were done within the R.R.S.G. to predict the amount of processing power required, and hence the number of transputers that would be required to implement the QLP algorithm. Upon implementation of the algorithm, these calculations proved to be inaccurate. The transputers were found to be approximately four times slower in processing the algorithm than predicted. This chapter addresses this discrepancy and looks in detail into the performance of the transputers in this application.

5.1 Actual System Performance

A series of tests was performed to measure the performance of the QLP. For all the tests performed in this chapter, the following parameters were used, unless stated:

- Number of transputers: 8
- Number of range bins: 1024

Table 5.1 summarises the QLP performance test results. These tests show the rate at which the data is processed while varying the number of subapertures, and hence the resultant azimuth resolution. The specification requires a resolution of 30m or better, which corresponds to 10 subapertures (29m). As can be seen from the table, only 3 subapertures can be processed at a rate greater than the required 39Hz. Looking at Table 5.1 it can be seen that the actual rate at 29m is 18 lines/s. To process this at the required rate of 39Hz would require a processing speed improvement factor of just over 2. Note also that this test processes 1024 range bins, and the QLP specification requires 2048 range bins to be processed. This accounts for another factor of

Table 5.1: Test Results for QLP

RESOLUTION[M]	SUBAPERTURES	RATE/[LINES/S]
93	1	> 39
65	2	> 39
54	3	> 39
46	4	34
42	5	28
29	10	18

2. Thus an improvement factor about 4 is required for the QLP to perform up to the required specification. The remainder of this chapter details why the transputers performed about 4 times slower than first predicted.

5.2 Tests Performed to Verify Results

In order to find where the performance is lacking, it is necessary to find out where in the QFA the transputers are spending the most time. The QFA algorithm can be broken down into a number of discrete processing tasks as follows:

- Subtract values from subapertures (Task A);
- Read in new IQ pair (Task B);
- Add values to subapertures (Task C);
- Phaseshift and sum subapertures (Task D);
- Calculate magnitude squared, scale and clip (Task E);
- Remainder of tasks, for example updating subaperture pointers. (Task F);
- Overhead time. This is the time taken to simply loop through the code, and other such overheads. This task is unavoidable, and is present in all tests. (Task G).

Each of these tasks was performed alone and in certain combinations, and timed. The results are given in Table 5.2.

The parameters for this test were as follows:

- Number of transputers: 5

Table 5.2: QFA Task Timing Test Results

A	B	C	D	E	F	G	Time[s]	% Tot. Time	Comments
x	x	x	x	x	x	x	40.0	100%	All tasks. Sum of all tasks + overhead time
						x	2	5%	No tasks. Measure of overhead time.
x						x	10.8	27%	Subtract values from subapertures.
	x					x	4.1	10.25%	Read in new IQ pair.
		x				x	10.4	26%	Add values to subapertures.
			x			x	18.2	45.5%	Phase shift and sum subapertures.
				x		x	3.9	9.75%	Calculate magnitude squared, scale & clip.
					x	x	2.6	6.5%	Remainder of tasks.
x		x				x	19.2	48%	Just tasks A and C.
x		x				x	6	15%	Tasks A and C without array addressing.

- Number of range bins: 256
- Number of subapertures: 10

The first test measures the time for the complete algorithm, so that it can be used as a comparison to the times of individual steps in the algorithm. Overheads are measured in the second test, where none of the identified tasks are performed they amount to 5% of the total time. The next 6 tests show the time taken for each task plus the unavoidable overhead time. To calculate the percentage time taken by each task, 5% must be subtracted from the figure given for that test to discount the overhead time. The next test shows that the algorithm spends about 43% (48%-5%=43%) of its time on tasks A and C combined. From this observation, it was decided that these tasks in particular required more investigation.

A test was done, where only tasks A and C were done, but all the array addressing in these tasks were removed. In the table, it can be seen that the time was dramatically reduced down to 15% of the total algorithm time.

Table 5.3: Radar parameters used in timing calculations.

Slant range to mid swath	20km
PRF	1250Hz
Aircraft velocity	181m/s
Wavelength	2.5m
Range bins	2048
Required azimuth resolution	30m
Number of subapertures (blocks)	19
Presum ratio	32

5.3 Analysis of Results from Performance Tests

As the above section reveals, a high percentage of the time is spent in tasks A and C, and particularly in array addressing. At this point it was therefore clear that the time taken for array addressing was the cause of the problem. The following section looks into this issue.

5.4 Timing Predictions

This section looks in more detail into the actual theory behind the operation of the transputer's micro operations and how to calculate how fast a particular piece of code will execute on a given transputer. First, however the initial calculations done by the R.R.S.G. system designers will be presented.

5.4.1 Initial Calculations

The initial calculations that were performed are briefly laid down below.

In order to estimate the number of transputers required, the total number of operations (Additions and Multiplications) was estimated, and then multiplied by the time taken to execute these operations. Table 5.3 gives the radar parameters used for the calculations.

The total number of operations required was calculated as follows: The algorithm was broken down into its various components and for each one, the number of operations was calculated. This is shown as follows:

Shift and Accumulate

Number of Additions was calculated to be:

$$n_a = 2 \cdot n_{blocks} \cdot n_{bins} \tag{5.1}$$

where:

n_a	is the number of additions
n_{blocks}	is the number of blocks used in aperture[1].
n_{bins}	is the number of range bins.

Substituting in the values from Table 5.3, $n_a = 77824$
No multiplications were performed.

Phase Shift and Combine

Number of Additions was calculated to be:

$$n_a = (n_{blocks} - 1) \cdot n_{bins} \tag{5.2}$$

For the parameters in Table 5.3, $n_a = 36864$
Number of Multiplications was calculated to be:

$$n_m = \frac{(n_{blocks} - 1)}{2} \cdot n_{bins} \tag{5.3}$$

For the parameters in Table 5.3, $n_m = 18432$

Detection

Number of Additions was calculated to be:

$$n_a = n_{bins} \tag{5.4}$$

For the parameters in Table 5.3, $n_a = 2048$
Number of Multiplications was calculated to be:

$$n_m = n_{bins} \tag{5.5}$$

For the parameters in Table 5.3, $n_m = 2048$

Complete Algorithm

The total number of additions is therefore 116736 for each range line. The total number of multiplications is therefore 20480 also for each range line. The time taken for a complex addition was taken from [15] to be 0.7μs and for a complex multiplication 2.9μs. From here, the total number of transputers was calculated from the following formula taken from [9]:

[1]The blocks relate to the number of subapertures by the following relationship: $n_{blocks} = (n_{subapertures} \times 2) - 1$

$$n_{transputers} = [T_{add} \cdot (3 \cdot n_{blocks} + 1) + T_{mult} \cdot \frac{n_{blocks}+1}{2}] \cdot n_{bins} \cdot f_p \qquad (5.6)$$

$$n_{transputers} = 5.6 \qquad (5.7)$$

where:

$n_{tranpsuters}$	is the number of transputers required
T_{add}	is the time taken to perform a complex addition $(0.7 \times 10^{-6})s$
T_{mult}	is the time taken to perform a complex multiplication $(2.9 \times 10^{-6})s$
n_{blocks}	is the number of suubaperture blocks required (19)
n_{bins}	is the number of range bins to be processed (2048)
f_p	is the PRF after presumming $(39Hz)$

Provision was made for inactive time, by increasing the number of transputers by 20 to 50%. This is explained in [8]. It was therefore (incorrectly) predicted that 8 transputers (plus the master) would be sufficient. This incorrect prediction is due to the shortcomings of the above calculations. Other issues such as array addressing and other overheads are discussed in a later section.

5.4.2 Closer Calculations

Due to the shortcomings of the initial calculations, the transputer timing was looked into closer. The QLP algorithm is heavily dependent on reading values in and out from a large 2-dimensional array. In order for the transputer to do any calculations on a particular number in that array, the transputer first has to calculate the physical address of the element in the array. If the answer is also to be placed in a 2-dimensional array, again the address of the element into which the answer must be placed needs to be calculated. As will be shown below, the time required to calculate the address of any one of the elements in the array is much greater than the time taken to do the actual addition or multiplication.

In order to calculate the time required to calculate the address of an element in a 2-dimensional array, the machine code which the OCCAM compiler generates needs to be disassembled. A dissassembler for the T8000 was not available at the time of writing, so instead the code was hand assembled using the information provided in the Occam TDS reference manuals [14], [15].

What is done below is to calculate the time required to add two elements from two 2-dimensional arrays together and store the result in another 2-dimensional array.

To calculate the required execution time, we need to calculate the total number of instruction cycles required to complete the operation.

One assumption that needs to be made is the width of the 2-dimensional array. This is because the time taken to compute the address is related to the array width as explained in [15].
Assumptions:

- The array width is 2048 (i.e. 10 bits)
- Instruction cycle = 50ns (20MHz) [15].

Assume we want to compile the following OCCAM line of code:

```
X[a,b] := Y[c,d] +Z[e,f]
```

In order to do this addition, we need to calculate the addresses of each of the three elements.

First we need to get the address of Y[c,d].

To do this, the transputer compiler would generate the following code:

Instruction Description	**Instruction**	**Clock Cycles**
load local c	LDL c	2
load constant 2048	LDC 2048	1
product	PROD	14 (b+4=10+4=14) [2]
load local d	LDL d	2
add	ADD	1
TOTAL CLOCK CYCLES		20

We now have the address of Y[c,d]. In the same way, we can calculate the address of Z[e,f] and X[a,b]. Thus to calculate all the addresses, we require 20 x 3 = 60 clock cycles. The addition then takes another one cycle. Thus totalling 61 cycles per array addition. 50ns x 61 cycles = 3.05μs Thus ignoring any other overheads such as loop counters etc., we can expect an array addition of this nature to take at least 3μs.

[2] b is the number of bits to store the number 2048

5.5 Timing Tests

In order to verify the above predictions and to attempt to quantify the transputer's performance in real terms, a number of timing tests were performed. Four basic timing tests were performed for both integer multiplication and integer addition. These are as follows:

1. Simple number multiplication or addition;
2. Simple variable operations;
3. 2-D Array operations with a constant index;
4. 2-D Array operations with a variable index.

In each of the above tests, the operation under analysis was performed 100,000 times, and the average time taken for each operation was measured.

Simple Number Multiplication or Addition

In this test, a simple operation is performed.
For example:

```
x := 1 + 3 or
x := 5 * 2
```

Simple Variable Operations

In this test, a simple variable operation is performed.
For example:

```
x := a + b or
x := a * b
```

2-D Array Operations with a Constant Index

In this test, 2-D array elements are added or multiplied together, and the result stored in another element of a 2-D array.
For example:

```
x[1,2] := y[3,52] + z[25,23] or
x[1,2] := y[3,52] * z[25,23]
```

2-D Array Operations with a Variable Index

In this test, 2-D array elements are added or multiplied together, and the result stored in another element of a 2-D array. This time, however the array indexes are not constants, but variables. This type of operation is a more realistic reflection of the type of multiplications and additions that are performed in the QLP.
For example:

```
x[i,j] := y[a,b] + z[c,d] or
newline{}x[i,j] := y[a,b] * z[c,d]
```

Below is given an example of the OCCAM source code which was used to perform these timing tests. This example was for the 2-D Array operations with a variable index test. The code was simply modified for the other tests.

```
INT a, b, c :
INT32 time1, time2, dummy :
BYTE key, result :
[1000][1000]INT test:
SEQ
so.time (fs, ts, time1, dummy)
a := 4
b := 7
SEQ i = 0 FOR 1000
SEQ
SEQ j = 0 FOR 1000
        SEQ
        test[i][j] := test[i][j] + test [i][j]

so.time (fs, ts, time2, dummy)
so.write.int64 (fs, ts, (INT64 (time2 - time1)), 0)
so.getkey (fs, ts, key, result)
```

Table 5.4: Results of Addition timing tests

Test #	Type of Test	T_{add}	Pred1	Factor1	Pred2	Factor2
1	Simple constant addition	0.55μs	0.35μs	1.6		
2	Simple variable addition	0.63μs	0.35μs	1.8		
3	2-Darray add with constant index	1.6μs	0.35μs	4.6		
4	2-Darray add with variable index	3.1μs	0.35μs	8.9	3.05	1.02

Table 5.5: Results of Multiplication timing tests

Test #	Type of Timing Test	T_{mul}	Original Pred.	Factor
1	Simple number multiplication	2.5μs	1.45μs	1.7
2	Simple variable multiplication	2.6μs	1.45μs	1.8
3	2-D array multiplication with constant index	3.1μs	1.45μs	2.1
4	2-D array multiplication with variable index	5.2μs	1.45μs	3.6

5.5.1 Addition Timing Tests Results

The results of the above tests are given in Table 5.4. These tests were performed on a single transputer. Column T_{add} is the actual time taken for an addition. Pred 1 is the first prediction time for an addition. Factor 1 is the ratio of the actual time to the first predicted time. Pred 2 is the predicted addition time after closer calculations, followed by factor 2 which is the ratio of the actual time to the second predicted time.

As can be seen in Table 5.4, the additions are up to 8.9 times slower than first predicted. In the closer calculations, an array addition with variable index was calculated to take 3.05μs. This predicted value corresponds very closely to the results from the experimental timing tests.

5.5.2 Multiplication Timing Tests Results

The results of the above tests are given in Table 5.5. These tests were performed on a single transputer.

As can be seen in Table 5.5, the multiplications are up to 3.6 times slower than first predicted. In order to verify that it is plausible that this is the major contributing factor to the unexpected performance degradation, these figures must be related back to the original algorithm. To simplify the calculation, it is assumed that the average time for an addition is based on the

average time of the last 3 types of additions tested (Addition test 2, 3 and 4). The additions found in Addition test 1 are not used in the algorithm, and therefore this timing figure is not considered when calculating the average addition time. In the same way, the average time for a multiplication is based on the average time of the last 3 types of multiplications tested (Multiplication test 2, 3 and 4). The multiplications found in Multiplication test 1 are not used in the algorithm, and therefore this timing figure is not considered when calculating the average multiplication time.

The average addition time can then be calculated to be 1.8μs, and the average multiplication time 3.6μs. Thus comparing these two figures with the original figures of 0.35μs and 1.45μs, the speed difference factors are 5.1 and 3.6 for addition and multiplication respectively.. Now in order to obtain a fair overall factor for both additions and multiplications, these factors need to be weighted in accordance with the relative number of additions and multiplications in the algorithm. As calculated previously, the algorithm performs 116736 additions and 20480 multiplications. Thus there are 5.7 times more additions as there are multiplications in the algorithm. This is the weighting factor to apply to the addition factor. Calculating the final factor between the original addition and multiplication time predictions and the measured addition and multiplication times, a value of 4.9 is obtained.

This was calculated as follows: $\frac{(5.1\times5.7)+(3.6\times1)}{5.7+1}$

What this means is that the new calculations predict that the additions and multiplications take about 4.9 times longer to process than originally predicted. Considering that in practise the complete algorithm runs about 4 times slower than originally predicted, and that the algorithm does tasks other than addition and multiplication, these results are plausible..

5.6 Pipeline Efficiency Tests

It is important to gain a feeling for how efficient the pipeline configuration is. Using the same test parameters as in section 5.1, but with the number of range bins set to 2048, and varying the number of transputers, the following results were obtained:

Number of transputers	Rate[lines/s]	Comments
1	.25	Measured
2	1.5	Measured
3	2.75	Measured
4	4.0	Measured
6	6.5	Measured
8	9.0	Measured
18	21.5	By linear extrapolation
32	39	By linear extrapolation
33	40.25	By linear extrapolation

The last three results were obtained by linear extrapolation from the first six, since the first six fall on a straight line. It can be seen that theoretically 33 transputers would be capable of meeting (and exceeding) all the specifications. This will be discussed again in Chapter 6.

Chapter 6

Summary, Conclusions and Future Work

6.1 Summary of Findings

This thesis has shown that the development and implementation the QLP is possible. To obtain the required specifications however, some improvements need to be made to the existing transputer hardware. Based on the initial assumptions about transputer performance, the existing hardware would have been acceptable, however due to the shortcomings of these assumptions the hardware was found to process the algorithm four times slower than desired. It was found that the host PC part of the system operated at the required specification without a problem.

6.2 Sample Imagery

In order to see the operation of the QLP on real data, C Band data was obtained from DLR in Oberpfaffenhofen and processed. Two examples of the imagery are given:

Figure 6.1 shows the data with only one subaperture (unfocused SAR) which corresponds to an azimuth resolution of 8.2m, while Figure 6.2 shows the same area processed to a resolution of 2.9m with 8 subapertures. The x-axis represents the range direction, and the y-axis the azimuth direction. In both cases, the azimuth pixel spacing is 20m. The vertical white line on the left of the image is a railway line.

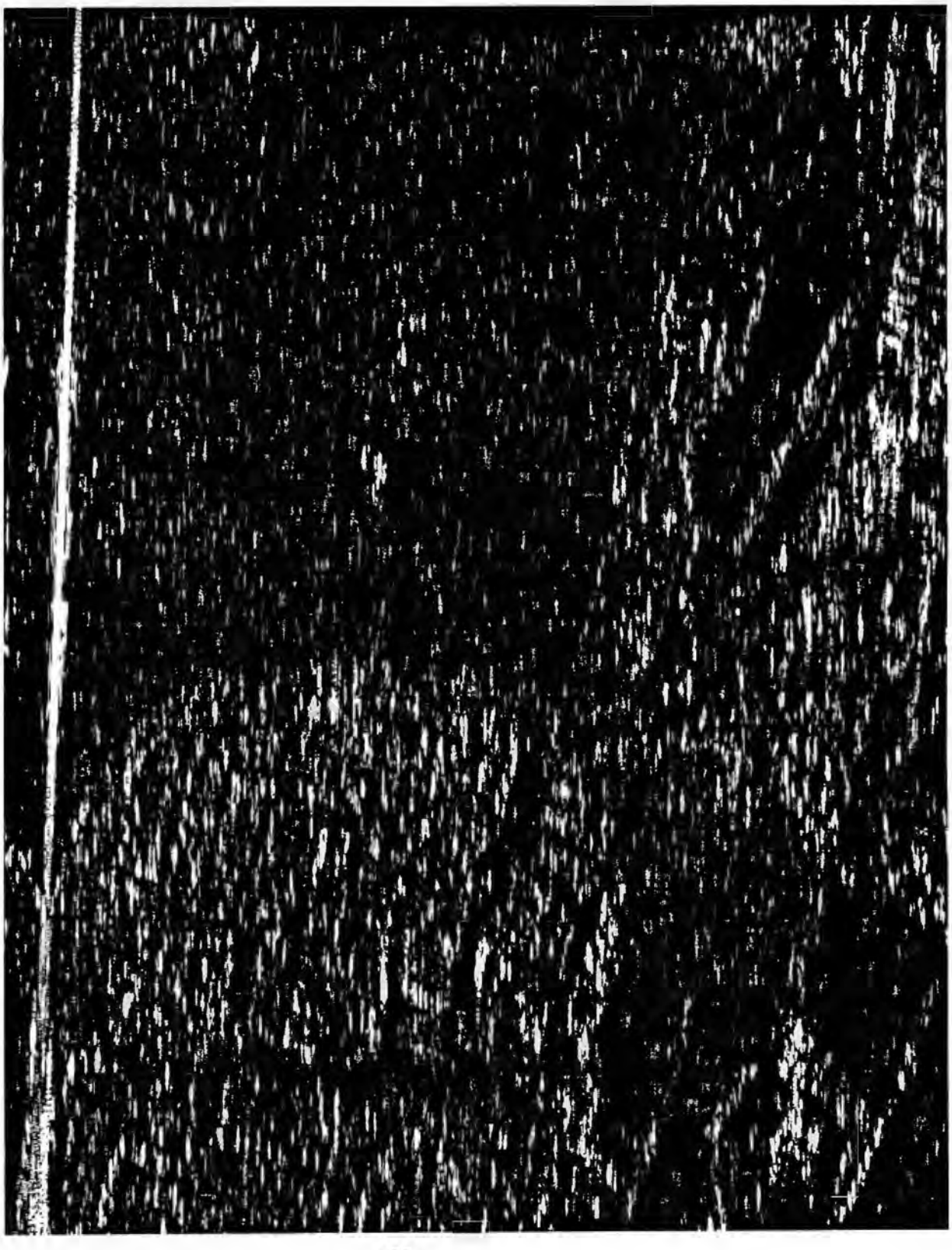

Figure 6.1: Unfocussed SAR image. Azimuth resolution 8.2m.

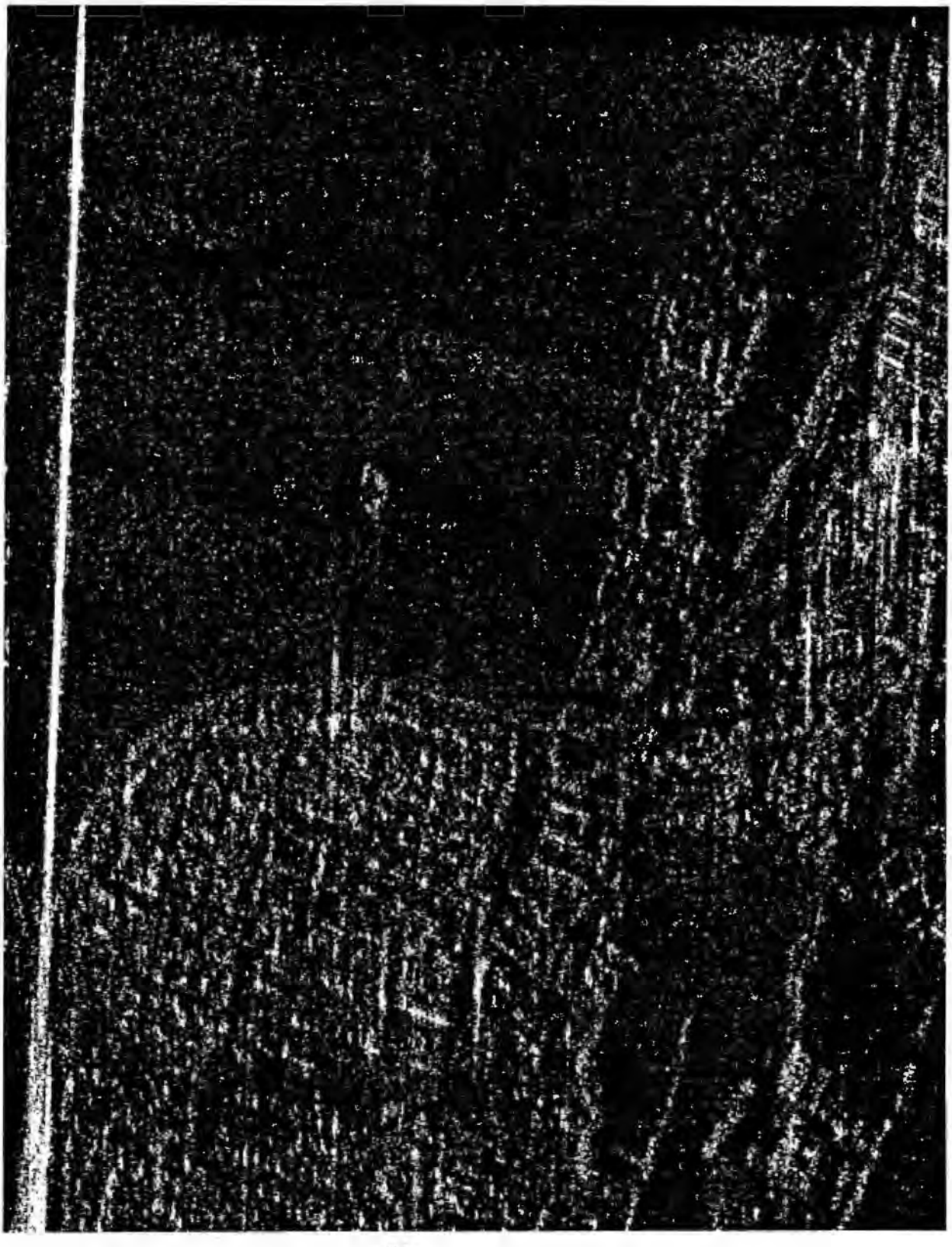

Figure 6.2: Quasi Focused SAR. Azimuth resolution 2.9m.

6.3 Future Work

To improve the processing speed and get the system running at the required specification, there are two options:

1. To add more transputers.
2. To design a new system on a different hardware technology which would support the computing demands.

6.3.1 Adding More Transputers

One master and 33 slave transputers would be sufficient to process the algorithm in real-time. Although this solution may be fairly costly in terms of purchasing the hardware, there would be no extra effort required to get the system running up to speed. By simply adding the extra transputers into the pipeline and telling the master transputer how many slaves it has, the software will automatically distribute the workload evenly between all the available slaves. Since the performance of the host PC is sufficient, this system would not require any changes.

6.3.2 The Design of a New Hardware Platform.

This option would involve an investigation into what hardware platforms would be more suitable, taking advantage of the latest technology available. Once a new hardware platform was chosen, it would need to be evaluated in terms of actual performance for the task at hand. Since the transputer software was written Occam which was designed specifically for transputer technology, the transputer software would have to be re-written for the new hardware platform. Although the performance of the host PC is sufficient, the interface between the PC and the new hardware would also have to be rewritten. Thus this option would involve a significant amount of development and redesign time

Since this option would require much redesign, it would make sense to consider using another algorithm, such as the range Doppler algorithm. The inclusion of motion compensation and speckle reduction could then be considered if processing capabilities allow.

Bibliography

[1] Stimpson, G.W. *Introduction to Airborne Radar*, pp515-562, Hughes Aircraft Co., 1983.

[2] Inggs, M.R., J.M. Horrell, P. Koeppen, L. Alexander, A.R. Knight, P.J. Archer. *The iSAR System Design Document.* Technical Report RRSG01:94, R.R.S.G. University of Cape Town, South Africa, 1994.

[3] Inggs, M.R., J.M. Horrell, P. Koeppen, L. Alexander, A.R. Knight, P.J. Archer. *The iSAR Recording System Design Document*, Technical Report RRSG16:94 R.R.S.G. University of Cape Town, South Africa, 1994.

[4] Inggs, M.R., J.M. Horrell, P. Koeppen, L. Alexander, A.R. Knight, P.J. Archer. *Design Document for the iSAR GPROCS*, Technical Report RRSG02:94 R.R.S.G. University of Cape Town, South Africa, 1994.

[5] Inggs, M.R., J.M. Horrell, P. Koeppen, L. Alexander, A.R. Knight, P.J. Archer. *The iSAR Platform Specification*, Technical Report RRSG20:94 R.R.S.G. University of Cape Town, South Africa, 1994.

[6] Inggs, M.R., J.M. Horrell, P. Koeppen, L. Alexander, A.R. Knight, P.J. Archer. *Design Document for the iSAR Real-Time Quicklook Processor*, Technical Report RRSG05:94 R.R.S.G. University of Cape Town, South Africa, 1994.

[7] Inggs, M.R., J.M. Horrell, P. Koeppen, L. Alexander, A.R. Knight, P.J. Archer. *The iSAR Recording System Specification Document*, Technical Report RRSG17:94 R.R.S.G. University of Cape Town, South Africa, 1994.

[8] Horrell J.M., Knight A.R., and Inggs M.R. *A Quicklook Processor for an Airborne SAR* IGARSS '94, Vol 2, pp 1178-1180.

[9] Horrell, J.M. Unpublished work, 1994.

[10] F.E. Guertin. *SAR-580 Video Signal Cct Format Specification.* Canada Center for Remote Sensing, December 1981.

[11] Curlander, J.C. and R.N. McDonough. *Synthetic Aperture Radar,* pp 438-440, Wiley, Canada, 1991.

[12] Schwaderer, *W. D. C Programmers' Guide to NetBIOS, IPX and SPX,* SAMS Publishing, 1992.

[13] Carrara, W.G., R.S. Goodman, R.M. Majewski. *Spotlight Synthetic Aperture Radar. Signal Processing Algorithms,* p3, Artech House, Inc., 1995.

[14] INMOS. *The Transputer Applications Notebook,* INMOS Document number 72-TRN-205-00, INMOS Limited, 1989.

[15] INMOS. *The Transputer Databook,* INMOS Document number 72-TRN-203-01, INMOS Limited, 1989.

[16] INMOS Software. *Transputer Development System,* INMOS Product number 72-TDS-142-02, INMOS Limited, 1990

Appendix A

IPX Programming

This Appendix gives a basic explanation of the IPX (Internetwork Packet Exchange) network protocol. Novell NetWare provides a shell which provides a seamless connection between DOS and the network functions.

The connection services are accessed by calling interrupt 21 Hex. Most of the functions require request and reply data structures (buffers) to contain the information passing to and from the calling machine. All the fields in the request buffer must be set up before making the service call. Also the size of the reply data buffer must be filled out. When this is done, the service call can be made and the contents of the reply buffer examined for the results. IPX is a variant on the Xerox Network Systems Internet Transport protocol. It is a high performance packet transport protocol. Stations can communicate to each other directly without the need for the interaction of the file server.

A.1 The IPX Packet Header

An IPX packet contains a 30-byte header and a data section. The IPX packet header's format is as follows:

Offset	Content	Type
0	Checksum	Word
2	Length	Word
4	Transport Control	Byte
5	Packet Type	Byte
6	Destination Network	Byte[4]
10	Destination Node	Byte[6]
16	Destination Socket	Byte[2]
18	Source Network	Byte[4]
22	Source Node	Byte[6]
28	Source Socket	Byte[2]
30	Data	Byte[0 to 546]

Before a packet can be sent, the IPX header must be filled out. The size if the data section can be from 0 to 546 bytes. In this application, the size is set to 512 bytes. Although the QLP does not require more than 20 bytes, this figure was chosen, so that a standard packet size could be used by all systems. Other systems like the console require larger packet sizes.

A.2 The Event Control Block

To send and receive packets an event control block (ECB) is required. An ECB is a structure that controls a send or receive packet event. To receive a packet, the receiving program must first issue an IPX listen for packet request. When a packet arrives, the listen for packet request is consumed. Another IPX listen for packet request must therefore be issued if another packet is to be received.

Once a packet has arrived, the packet's data is placed into a buffer specified in the ECB. The arrival of a packet can be detected by scanning the InUseFlag of the ECB.

A.3 The IPX Communication Functions

In order to simplify the appearance of the Host PC Software, a set of IPX library functions were written to handle the packet transmission and reception. The following major functions were written:

1. InitIPX
2. IPXListenForPacket

3. ListenEcb.InUseFlag

4. IPXSendPacket

A.3.1 InitIPX

This function first checks that the IPX drivers are loaded. If they are not, an error message is printed and the system exits. If the IPX drivers are running an IPX listen and an IPX send socket are opened[12]. This is necessary to enable communication with other workstations. The ECB is then filled out with the correct information and the system is ready to communicate.

A.3.2 IPXListenForPacket

This function issues a listen for packet request. This must be executed before any packet can be received.

A.3.3 ListenEcb.InUseFlag

This is a variable which detects whether or not a packet has been received. This flag has a value of 1 if the listen request is still waiting for a packet to be received. Once a packet has been received, the flag has a value of 0.

A.3.4 IPXSendPacket

This function is used to transmit a packet to some destination machine. The destination machine address is sent as one of a set of parameters to this function.

A.4 The IPX Communication Packet Protocol

Although not completely implemented in the current version of the QLP, a protocol was devised for communication between the QLP and the console system. The data block of the packet has the following format:

Where C is Command Type. Valid command types are:

- C = Configuration Data
- S = Start

- D = Shutdown
- G = Geocoding Data

A.4.1 Configuration Data (C)

When this command is received, the QLP looks at the remainder of the packet data to determine the configuration of the system, Figures such as number of subapertures and range bins to process are included here.

A.4.2 Start (S)

This command tells the QLP to start processing data.

A.4.3 Geocoding (G)

This command indicates that geocoding data is in the remainder of the packet data block. This data represents the aircraft position, and is in a string format. This string is then simply displayed on the scrolling screen.

A.4.4 Shutdown (D)

This command shuts down the software at the request of console system.

Appendix B

SASAR System Specifications

This appendix contains a summary of the important specifications in the SASAR system.

B.1 Platform Specifications

The platform consists of the following subsystems:

1. Aircraft
2. Global Positioning System (GPS)
3. Inertial Measurement Unit (IMU)
4. AC Power Supply

B.1.1 The Aircraft

The Aircraft subsystem is responsible for providing the physical environment for the other systems. The physical environment factors include altitude, local altitude, speed, maximum mass, cabin space and crew.

B.1.2 Global Positioning System (GPS)

The GPS subsystem provides aircraft positional information, which is passed on to the LBR. The following information is provided:
Longitude and Latitude
This information is then passed on to the QLP for geocoding purposes.

B.1.3 AC Power Supply.

The AC Power Supply is responsible for providing AC electrical power to the other systems. At the time of the QLP implementation these requirements had not yet been specified, as they depend on the power fittings in the aircraft.

B.2 SAR Recording System Interface Specifications

A number of interfaces are relevant to the QLP and are briefly given below. For the full specification, refer to the Recording system specification document [7].

B.2.1 LBR/CONSOLE to QLP Interface

This interface controls the quick look processing.

Physical

The Ethernet card of the LBR/CONSOLE connects to the QLP via a 50 ohm coaxial cable. The Ethernet card is fitted with a female BNC type connector.

Electrical

The interface is implemented using the Ethernet IEEE802.3 specification.

Software

Novell SPX-IPX protocol is used.

B.2.2 QLP to HBR Link (Transputer link)

This is used to send presummed raw data to the QLP for processing in real time.

Physical

The QLP and HBR units are connected via connectors to two twisted pair wires. The connectors used are all D-type 9 pin female connectors.

Electrical

The inputs can accept voltage levels of up to 7 differential volts. The outputs generate voltages of differential 5 volts.

Software

The data is sent to the QLP in packets. The packet starts with a PRIID number followed by alternating I and Q data bytes for that presummed group.

Appendix C

Source Code

This appendix contains the source code that was written to implement the QLP system. The code is divided into two sections, namely the host PC software and the Transputer software. A brief description of each listing is given below:

C.1 Host PC Software

1. qlpserve.cpp - Main QLP server. This program starts the entire QLP system.
2. miscutil.cpp - Functions used by qlpserve.cpp
3. packet.cpp - Functions for IPX peer to peer networking.
4. packet.h - Header file for packet.cpp
5. ipx-spx.h - Header file for SPX/IPX functions
6. defs.h - Header file for packet.cpp
7. mouse.cpp - Mouse functions
8. mouse.h - Header file for mouse.cpp
9. qlp.par - Default parameter file

C.2 Transputer Software

1. qlp.lis - Occam source code. Master and slave transputer code is contained in this file.

Qlpserve.cpp

```
/*

 This program is the main Quick look processor server.
 It is responsible for the following:


       * Booting and uploading code to transputers. (Call to ISERVER.EXE)

       * Receiving the configuration packet from the HBR/CONSOLE via
       IPX protocol

       * Initializing Transputers with config info

       * Receiving Magnitude squared data from transputer link adaptor

       * Displaying image on scrollable display.

*/

/* Compiled with Borland C V3.1 */

/* Code: P.J. Archer */

// Standard include files
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <process.h>
#include <errno.h>
#include <time.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>

// Custom include files
#include "PACKET.H"          // IPX packet routines header file
#include "IPX-SPX.H"         // IPX packet routines defs
#include "mouse.h"           // Mouse driver header file
#include "mouse.cpp"         // Mouse driver code
#include "miscutl.cpp"       // Misc utilities for QLP


#define PACKET_SIZE 255    // Max packet size
#define BYTE unsigned char

// The following defines are for the base address of the Transputer
// link adaptor.
#define BaseAddress 0x150
#define DataIn       BaseAddress + 0
#define DataOut      BaseAddress + 1
#define ReadyToSend  BaseAddress + 2
#define ReadyToRec   BaseAddress + 3


// The following defines define the protocol for data communication
// between the Occam and C program
#define RawData 0                    // requests a range line of raw data.
#define PRIIDNumbers  1              // indicates PRIID number coming.
```

```
#define ProcessedData 255           // indicates processed range line
coming.


#define GRC 0x3CE                  // VGA Address
#define MaxNumberOfRangeBins 2048
#define MaxNumberOfSubApertures 45

// Global Prototypes

void        ToLink (char val);     // Sends data to transputer link
unsigned char   FromLink ();          // Reads data from transpter link
void        ToLinkfast (char val); // Optimised version of sending to link
unsigned char   FromLinkfast ();      // Optimised version of reading link
void color(BYTE C,BYTE R,BYTE G,BYTE B); // Sets RGB table element
void GrayScale();                      // Sets RGB table to grayscale
void ExpScale();                  // Sets RGB table to exponential gs
void Exp2Scale();                      // Sets RGB table to invese exp gs
void GeoCode (int XOffset, int YOffset, unsigned char *string); // prints
it

BYTE   SendInt32 (long Value);       // Sends a long int to the transputer
BYTE   SetupTransputers (void);      // Sends qlp parameters to transputer
BYTE     BootTransputer (void);        // Boots transputer network
void   BlockPlot(int x,int y);       // plots a set of pixels
void   Plot(int x,int  y,long int col); // plots a single pixel
void   setvstart(unsigned int x,unsigned int y); //Sets scroll position
void   ReadParameterFile (void);     // Reads default parameter file
int    skipchar(FILE *fp,char ch);   // Used in reading parameter file
int    GetString(FILE *fp,char *string); //Used in reading parameter file
long   FileHeight,FileWidth;
int      UseNetwork,PriidsInSourceFile,WriteToFile; // Boolean variables
int    SubApLength[MaxNumberOfSubApertures/2];  // Subaperture Length


// Global variables

BYTE DCOffset;                          // DC Offset of input data
char inname[80];                        // Input raw data filename (for
tests)
char outname[80];                       // Output image data filename
int  StartRangeBin,ScaleFactor,SizeOfAperture; // Processing parameters
int  StopRangeBin;                                // Processing parameters
long NumberOfSubApertures;                        // Processing parameters
int NumberOfRangeBins; // Number of rangebins received from transputers
BYTE colorarray[8];    // Buffer used for writing small a small block to
VGA
FILE  *SetupFile;      // Input setup file for parameters
                  // the VGA screen    (8X1 block)

void main (void) // Main program starts here
{
  // Main global variables
  time_t first, second; // Used for timing/performance measurement

  BYTE InBuffer[2*MaxNumberOfRangeBins]; // buffer where data is stored
  char key;                       // Keyboard input var
  int line;                      // Current range line
  int x,y;                       // General vars
  int XPanPosition,YPanPosition; // virtual page screen start coordinates
  int Active;                    // flag indicating qlp is processing

  unsigned char IQData;          // Input buffer from transputers
```

```
unsigned char PacketReceive[PACKET_SIZE]; // Packet receive buffer
BYTE PRIIDBuffer[4];           // Incoming PRIID numbers buffer
long  NumberOfPRIIDs;          // Count of PRIID numbers received
FILE  *InFile;                 // Input file for testing
FILE  *OutFile;                // Output file for writing image
int button,NewXMouse,NewYMouse,
    XMousePos,YMousePos;       // Mouse vars

MouseInit();                  // Initialize mouse
MouseXdim (1,25);             // Set mouse constraint from 0 to 2047
MouseYdim (0,254);            // Set mouse  Y constraint from 0 to 256
MouseSet (0,192);             // Set mouse to 0,0

MouseStat(&button,&NewXMouse,&NewYMouse);
XMousePos = NewXMouse;
YMousePos = 50 + NewYMouse;
XPanPosition = (2048)-XMousePos;
NumberOfPRIIDs = 0;
clrscr ();

printf ("                          QUICK LOOK PROCESSOR.\n\n");
printf ("Reading Quicklook parameter file...\n");
ReadParameterFile (); // Reads default parameters
if (UseNetwork)
  {
    printf ("Initializing IPX Packet driver...\n");
    InitIPX(); // initialize IPX packet driver
    printf (" Done.\n");
    IpxListenForPacket(&ListenEcb,PacketReceive,PACKET_SIZE);
    printf ("Waiting for configuration data from LBR system...");
    while ((ListenEcb.InUseFlag)&&(!kbhit()));
    printf (" Done.\n");
    printf ("Received configuration data.\n");
  }
printf ("Initializing Transputer network.\n");
if ( (InFile = fopen(inname,"rb")) == NULL )
 { printf ("Error opening input file:%s\n",inname);
   exit (1) ;
 }
if ( (OutFile = fopen(outname,"w+b")) == NULL )
 { printf ("Error opening output file:%s \n",outname);
   exit (1) ;
 }
line  = 0;  // reset line number
Active = 1;

XPanPosition = 0;     // Set initial scroll position
YPanPosition = 1023; // Set initial scroll position
printf ("\nBooting and initializing Quick Look Processor...\n\n");

BootTransputer();     // boots transputers
SetupTransputers();  // downloads parameters to transputers
printf ("Transputer network initialized.");
spawnl(P_WAIT,"VGAMODE.EXE",NULL); // Initialize graphics mode 1024x768
GrayScale(); // Sets RGB table to grayscale
first = clock(); // start timing
if (UseNetwork) IpxListenForPacket(&ListenEcb,PacketReceive,PACKET_SIZE);
while (Active)
{

if (UseNetwork)
 if (ListenEcb.InUseFlag == 0)
  {
```

```
   IpxListenForPacket(&ListenEcb,PacketReceive,PACKET_SIZE);

 }
MouseStat(&button,&NewXMouse,&NewYMouse);
if (button == 1)
 {
   XMousePos = NewXMouse;
   YMousePos = 50+NewYMouse;
   XPanPosition = (2048)-XMousePos;
 }
else MouseSet (XMousePos,YMousePos);
if (button == 3) Active = 0; // both buttons exit
setvstart (XPanPosition,YPanPosition+253-YMousePos);
if (inportb(0x60) == 60) GrayScale();
if (inportb(0x60) == 61) ExpScale();
if (inportb(0x60) == 62) Exp2Scale();
if (kbhit())
{ key = getch ();
  if (key == 27) Active = 0;
  if (key == 75) XPanPosition+=8;
  if (key == 77) XPanPosition-=8;
  setvstart (XPanPosition,YPanPosition+253-YMousePos);
}
if ((inportb(ReadyToSend) & 1) == 1)
{
 IQData = inportb(DataIn);

 if (IQData == RawData) // Send raw data from hard disk
 {
   if (PriidsInSourceFile) fseek(InFile,4, SEEK_CUR); // Skip PRIIDS
   fseek(InFile,(StartRangeBin<<1), SEEK_CUR); // Skip rest
                                               // of data 4 for priids!
   fread (InBuffer,NumberOfRangeBins,2,InFile);
   for (x=0;x<NumberOfRangeBins;x++)
   {
   ToLink (InBuffer[x<<1]);
   ToLink (InBuffer[(x<<1)+1]);
  }
   fseek(InFile,(FileWidth-(StartRangeBin+NumberOfRangeBins))<<1,
          SEEK_CUR); // Skip rest of data
 }

 if (IQData == PRIIDNumbers) // Process PRIID Numbers
 {
   PRIIDBuffer[0] = FromLink();
   PRIIDBuffer[1] = FromLink();
   PRIIDBuffer[2] = FromLink();
   PRIIDBuffer[3] = FromLink();
   NumberOfPRIIDs++;
 }

 if (IQData == ProcessedData) // Display and save new range line.
 {
   YPanPosition--;
   line++;
   if (fmod(line,200) == 0) // Geocode every 200 lines...
   GeoCode (0,YPanPosition,PacketReceive);
   if (line == FileHeight-1)
   {
   line = 0;
   rewind (InFile);
   }
   if (YPanPosition == 0) YPanPosition = 1024;
```

```
      for (x=0;x <= (NumberOfRangeBins>> 3) -1; x++) // Display MAGS
      {
      for (y=0; y < 8; y++) colorarray[y] = FromLink();
      if (WriteToFile) fwrite(colorarray, sizeof(colorarray), 1,OutFile );
      BlockPlot (x*8,YPanPosition-1);
      }

   }

   }
  }
  second = clock();
  textmode (C80);
  fcloseall();
  printf ("%s Quick Look Image file generated. %d X %d pixels.\n",
        outname,NumberOfRangeBins,line);
  printf("Time: %f seconds\n", (second - first) / CLK_TCK);
  printf ("Quick look processor terminated.\nHave a nice day.\n");
  printf ("PRIIDs received:%d",NumberOfPRIIDs);
  printf ("xpanpos:%d",XPanPosition);
}

/*************************************************************************/
/* NAME: ToLink()                                                        */
/* DESCRIPTION:Sends a byte to the transputer link                       */
/*************************************************************************/
void ToLink (char val)
{
  while (!(inportb (ReadyToRec) & 1)); // wait for transputer to be ready
  outportb (DataOut, val);
}

/*************************************************************************/
/* NAME:                                                                 */
/* DESCRIPTION:Sends a byte to the transputer link without checking      */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
void ToLinkfast (char val)
{
  outportb (DataOut, val);
}

/*************************************************************************/
/* NAME:FromLinkFast()                                                   */
/* DESCRIPTION:Assumes data available from transputer and reads it       */
/*************************************************************************/
unsigned char FromLinkfast ()
{
  char val;
  while (1);
  inportb (ReadyToSend) & 1;
  val = inportb (DataIn);
  return (val);
}

/*************************************************************************/
/* NAME:FromLink ()                                                      */
/* DESCRIPTION:Reads data from the transputer link                       */
/*************************************************************************/
unsigned char FromLink ()
```

```
{
  char val;

  while (!(inportb (ReadyToSend) & 1));
  val = inportb (DataIn);
  return (val);
}

/*************************************************************************/
/* NAME:SendInt32(long Value)                                            */
/* DESCRIPTION: This function sends a long int to the transputer         */
/*************************************************************************/
BYTE SendInt32 (long Value)

{
  ToLink (Value >> 0);
  ToLink (Value >> 8);
  ToLink (Value >> 16);
  ToLink (Value >> 24);
  return (0);
}

/*************************************************************************/
/* NAME:BootTransputer (void)                                            */
/* DESCRIPTION: This function calls an external program to boot the      */
/*  transputers.                                                         */
/*************************************************************************/
BYTE BootTransputer (void)
{
  spawnl(P_WAIT,"iserver.exe","/sr","/sc","qlp.B4", NULL);
  perror("EXEC");
  return (0);
}

/*************************************************************************/
/* NAME: SetupTransputers (void)                                         */
/* DESCRIPTION:This function sends the QLP parameters to the transputer  */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
BYTE SetupTransputers (void)
#define WaitForData while ((inportb(ReadyToSend) && 1) == 0)

{
  int  Resolution;
  int  val_test;
  int  Offset;
  BYTE IQ;
  FromLink(); // 99 Check number
  Resolution        = 30; //m in Azimuth
  SendInt32 (ScaleFactor);
  SendInt32 (DCOffset);
  SendInt32 (SizeOfAperture);     // Size of aperture
  SendInt32 (NumberOfRangeBins);  // Number of range bins
  SendInt32 (NumberOfSubApertures);
  Offset = 0;
  SendInt32 (0);
  if (NumberOfSubApertures > 1)
  {
    for (int x=(NumberOfSubApertures-1)/2;x>=0;x--)
    {
     Offset += SubApLength[x];
     SendInt32 (Offset);
```

```
    }
    for (x=1;x <(NumberOfSubApertures-1)/2;x++)
    {
     Offset += SubApLength[x];
     SendInt32 (Offset);
    }
  }
  SendInt32 (0);
  if ( (Resolution != 30)  & (Resolution != 40) )
  {
    clrscr;
    printf ("Invalid resolution : Try 30m or 40m");
    exit (1);
  }
  IQ = FromLink ();
  if (IQ != 221)
  {
    clrscr;
    printf ("ERROR Initializing transputers\n");
    exit (1);
  }
  FromLink(); //222 Check Number
  FromLink(); //255 Check Number
  if (FromLink() == 111 )
  printf ("Transputer network initialized okay.\n");
  else
  {
    printf ("Transputer network initialization error.\n");
    exit (1);
  }
  return (0);
}

/*************************************************************************/
/* NAME:void ReadParameterFile (void)                                    */
/* DESCRIPTION:Reads default parameters from file                         */
/*************************************************************************/
void ReadParameterFile (void)
{
  int Offset = 0;
  if ( (SetupFile = fopen("qlp.par","rb")) == NULL )
   { printf ("Error opening qlp.par parameter file.\n");
     exit (1) ;
   }

  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%d", &WriteToFile);

  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%d", &PriidsInSourceFile);

  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%d", &UseNetwork);


  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%ld", &FileWidth);

  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%ld", &FileHeight);

  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%d", &NumberOfRangeBins);
```

```
  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%d", &StartRangeBin);
  StopRangeBin = StartRangeBin + NumberOfRangeBins;
  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%d", &DCOffset);

  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%d", &ScaleFactor);

  skipchar(SetupFile,'>');
  GetString(SetupFile,inname);

  skipchar(SetupFile,'>');
  GetString(SetupFile,outname);

  skipchar(SetupFile,'>');
  fscanf (SetupFile, "%d", &NumberOfSubApertures);



  for (int x=0;x<NumberOfSubApertures;x++)
  {
    skipchar(SetupFile,'>');
    fscanf (SetupFile, "%d", &SubApLength[x]);
  }

  for (x=0;x < NumberOfSubApertures;x++)
  {
   Offset += SubApLength[x]*2;
  }
  SizeOfAperture = Offset;
  SubApLength[0] = SubApLength[0] *2;
  NumberOfSubApertures = (NumberOfSubApertures * 2)-1;
  fclose (SetupFile);
}

/*************************************************************************/
/* NAME:Plot(int x,int  y,long int col)                                  */
/* DESCRIPTION:Plots a single pixel                                      */
/*************************************************************************/
void Plot(int x,int  y,long int col)
{
const unsigned int msk[8]  ={128,64,32,16,8,4,2,1};
const vseg = 0xa000;
 long l;
 unsigned int z;
  l = (long (y) << 8) + (x >> 3);
  outportb (GRC,3);
  outportb (GRC+1,0);
  outportb (GRC,5);
  outportb (GRC+1,2);
  outportb (GRC,8);
  outportb (GRC+1,msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),col);
}

/*************************************************************************/
/* NAME: BlockPlot(int x,int  y)                                         */
/* DESCRIPTION: Plots an 8x1 block of pixels to the screen.              */
/*************************************************************************/
```

```
void BlockPlot(int x,int  y)

{
const vseg = 0xa000;
 long l;
 unsigned int z;
  for (int loop = 0 ; loop <= 7 ; loop++)
  {
    colorarray[loop] = colorarray[loop] >> 4;
    if (colorarray[loop] == 15) colorarray[loop] = 14;
  }
  l = (long (y) << 8) + (x >> 3);
  outportb (GRC,3);
  outportb (GRC+1,0);
  outportb (GRC,5);
  outportb (GRC+1,2);
  outportb (GRC,8);

  outportb (GRC+1,128);// msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),colorarray[0] );

  outportb (GRC+1,64);// msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),colorarray[1]  );

  outportb (GRC+1,32);// msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),colorarray[2]  );

  outportb (GRC+1,16);// msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),colorarray[3]  );

  outportb (GRC+1,8);// msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),colorarray[4]  );

  outportb (GRC+1,4);// msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),colorarray[5]  );

  outportb (GRC+1,2);// msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),colorarray[6]  );

  outportb (GRC+1,1);// msk [x & 7]);
  outportb (0x3cd,((l >> 16) * 17));
  z = peekb (vseg,l);
  pokeb(vseg,unsigned (y * 256  + (x >> 3)),colorarray[7]  );

}


unsigned int rdinx(unsigned int pt,unsigned int inx)
//read register PT index INX
```

```
{
  unsigned int x;
  outportb (pt,inx);
  return (inport (pt+1));
}

void modinx(unsigned int pt,unsigned int inx,unsigned int mask,
                    unsigned int nwv)
  //In register PT index INX sets
  // the bits in MASK as in NWV
  // the other are left unchanged

{
  outportb (pt,inx);
  outportb (pt+1,(rdinx(pt,inx) & (!mask))+(nwv & mask));
}

/*************************************************************************/
/* NAME:setvstart(unsigned int x,unsigned int y)                         */
/* DESCRIPTION:Set the display start address (i.e. Scroll position)      */
/*************************************************************************/
void setvstart(unsigned int x,unsigned int y)


{
  long l;
  long bytes;
  unsigned VideoMemory;
  unsigned crtc;
  bytes = 256; // bytes per scan line
  VideoMemory = 1024;
  crtc = 980;   // address of CRTC
  l=(bytes*y+(x >> 3))*4;
  y= (l >> 18) & (  (VideoMemory-1) >> 8);
  modinx(crtc,0x33,3,y);
  x=l >> 2;
  outportb(crtc,13);
  outportb(crtc+1, x & 255);
  outportb(crtc, 12);
  outportb(crtc+1, (x >> 8 ) & 255);
}
```

miscutil.cpp

```
/*************************************************************************/
/* NAME: skipchar()                                                      */
/* DESCRIPTION: Reads characters from a file until a certain chosen      */
/*              character ch is reached                                  */
/*  PARAMETERS:                                                          */
/*        fp          local     R    Pointer to file to read            */
/*        ch          local     R    Character to read until            */
/*************************************************************************/

int skipchar(FILE *fp,char ch)

{

 char tch;       /* Store of character read in from the file */

 do
  {
   tch = getc(fp);     /* Reads character from file fp */
  }
 while ( (tch!= ch) && (tch != EOF) );  /* Reads while character is not */
                                /* the end character or the EOF */
 return(0);
}


/*************************************************************************/
/* NAME: GetString()                                                     */
/* DESCRIPTION: Reads string from a file                                 */
/*  PARAMETERS:                                                          */
/*        fp          local     R    Pointer to file to read            */
/*        string      local     R    Char pointer to start of string    */
/*************************************************************************/

int GetString(FILE *fp,char *string)

{

 int n=-1;               /* Counter of string postion */

 getc(fp);                      /* Get first space */
 do
  {
   string[++n] = getc(fp);      /* Reads character from file fp */
  }
 /* Reads while character is not a space, end of line, or end-of-file*/
 while ( (string[n] != ' ') && (string[n] != '\r')&& (string[n] != '\n')
        && (string[n] != EOF));
 string[n] = '\0';            /* Indicates end of string */
                        /* the end character or the EOF */
 return(0);
}

void color(BYTE C,BYTE R,BYTE G,BYTE B)
{
    outportb(0x3C8,C);
    outportb(0x3C9,R);
    outportb(0x3C9,G);
    outportb(0x3C9,B);
}

void Exp2Scale()
```

```
// inverse exponential gray scale
{
  color (0,0,0,0);
  color (1,9,9,9);
  color (2,18,18,18);
  color (3,25,25,25);
  color (4,31,31,31);
  color (5,36,36,36);
  color (20,41,41,41);
  color (7,45,45,45);
  color (56,49,49,49);
  color (57,52,52,52);
  color (58,54,54,54);
  color (59,57,57,57);
  color (60,59,59,59);
  color (61,60,60,60);
  color (62,62,62,62);
  color (63,13,63,13);
};

void ExpScale()
// exponential gray scale
{
  color (0,0,0,0);
  color (1,1,1,1);
  color (2,3,3,3);
  color (3,4,4,4);
  color (4,6,6,6);
  color (5,8,8,8);
  color (20,11,11,11);
  color (7,15,15,15);
  color (56,19,19,19);
  color (57,24,24,24);
  color (58,29,29,29);
  color (59,35,35,35);
  color (60,42,42,42);
  color (61,51,51,51);
  color (62,62,62,62);
  color (63,13,63,13);
}

void GrayScale()
{
  // linear gray scale
  color (0,0,0,0);
  color (1,4,4,4);
  color (2,8,8,8);
  color (3,12,12,12);
  color (4,16,16,16);
  color (5,20,20,20);
  color (20,24,24,24);
  color (7,28,28,28);
  color (56,32,32,32);
  color (57,36,36,36);
  color (58,40,40,40);
  color (59,44,44,44);
  color (60,48,48,48);
  color (61,52,52,52);
  color (62,56,56,56);
  color (63,13,63,13);

}
```

```
void GeoCode (int XOffset, int YOffset, unsigned char *string)

{
#define fontsize 16
// This code defines the font.
  BYTE FONT[fontsize][8][8] = {
  0,0,0,0,0,0,0,0,
  0,0,1,1,1,1,0,0, // 0
  0,1,0,0,0,0,1,0,
  0,1,0,0,0,0,1,0,
  0,1,0,0,0,0,1,0,
  0,1,0,0,0,0,1,0,
  0,1,0,0,0,0,1,0,
  0,0,1,1,1,1,0,0,

  0,0,0,0,0,0,0,0, //1
  0,0,0,0,1,0,0,0,
  0,0,0,1,1,0,0,0,
  0,0,0,0,1,0,0,0,
  0,0,0,0,1,0,0,0,
  0,0,0,0,1,0,0,0,
  0,0,0,0,1,0,0,0,
  0,0,1,1,1,1,1,0,

  0,0,0,0,0,0,0,0, //2
  0,0,0,1,1,1,0,0,
  0,0,1,0,0,0,1,0,
  0,0,0,0,0,0,1,0,
  0,0,0,0,0,1,0,0,
  0,0,0,0,1,0,0,0,
  0,0,0,1,0,0,0,0,
  0,0,1,1,1,1,1,0,

  0,0,0,0,0,0,0,0, //3
  0,0,1,1,1,1,1,0,
  0,0,0,0,0,1,0,0,
  0,0,0,0,1,0,0,0,
  0,0,0,0,0,1,0,0,
  0,0,0,0,0,0,1,0,
  0,0,1,0,0,0,1,0,
  0,0,0,1,1,1,0,0,

  0,0,0,0,0,0,0,0, //4
  0,0,0,0,1,1,0,0,
  0,0,0,1,0,1,0,0,
  0,0,1,0,0,1,0,0,
  0,1,1,1,1,1,1,0,
  0,0,0,0,0,1,0,0,
  0,0,0,0,0,1,0,0,
  0,0,0,0,0,1,0,0,

  0,0,0,0,0,0,0,0, //5
  0,0,1,1,1,1,1,0,
  0,0,1,0,0,0,0,0,
  0,0,1,1,1,1,0,0,
  0,0,0,0,0,0,1,0,
  0,0,0,0,0,0,1,0,
  0,0,1,0,0,0,1,0,
  0,0,0,1,1,1,0,0,

  0,0,0,0,0,0,0,0, //6
  0,0,0,1,1,0,0,0,
```

```
0,0,1,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,1,1,1,1,1,0,0,
0,1,0,0,0,0,1,0,
0,0,1,0,0,0,1,0,
0,0,0,1,1,1,0,0,

0,0,0,0,0,0,0,0, //7
0,0,1,1,1,1,1,0,
0,0,0,0,0,0,1,0,
0,0,0,0,0,0,1,0,
0,0,0,0,0,1,0,0,
0,0,0,0,1,0,0,0,
0,0,0,1,0,0,0,0,
0,0,1,0,0,0,0,0,

0,0,0,0,0,0,0,0, //8
0,0,1,1,1,1,0,0,
0,1,0,0,0,0,1,0,
0,1,0,0,0,0,1,0,
0,0,1,1,1,1,0,0,
0,1,0,0,0,0,1,0,
0,1,0,0,0,0,1,0,
0,0,1,1,1,1,0,0,

0,0,0,0,0,0,0,0, //9
0,0,1,1,1,1,0,0,
0,1,0,0,0,0,1,0,
0,1,0,0,0,0,1,0,
0,0,1,1,1,1,0,0,
0,0,0,0,1,0,0,0,
0,0,0,1,0,0,0,0,
0,0,1,0,0,0,0,0,

0,0,0,0,0,0,0,0, //N
0,1,0,0,0,0,1,0,
0,1,1,0,0,0,1,0,
0,1,0,1,0,0,1,0,
0,1,0,1,1,0,1,0,
0,1,0,0,1,1,1,0,
0,1,0,0,0,1,1,0,
0,1,0,0,0,0,1,0,

0,0,0,0,0,0,0,0, //S
0,0,1,1,1,1,0,0,
0,1,0,0,0,0,1,0,
0,0,1,1,0,0,0,0,
0,0,0,0,1,1,0,0,
0,0,0,0,0,0,1,0,
0,1,0,0,0,0,1,0,
0,0,1,1,1,1,0,0,

0,0,0,0,0,0,0,0, //E
0,1,1,1,1,1,1,0,
0,1,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,1,1,1,1,0,0,0,
0,1,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0, //W
0,1,0,0,0,0,1,0,
```

```
0,1,0,0,0,0,1,0,
0,1,0,0,1,0,1,0,
0,1,0,0,1,0,1,0,
0,1,0,1,1,0,1,0,
0,1,1,1,0,1,1,0,
0,0,1,0,0,1,0,0,

0,0,0,1,1,0,0,0, //+
0,0,0,1,1,0,0,0,
0,0,0,1,1,0,0,0,
0,1,1,1,1,1,1,1,
0,1,1,1,1,1,1,1,
0,0,0,1,1,0,0,0,
0,0,0,1,1,0,0,0,
0,0,0,1,1,0,0,0,

0,0,0,0,0,0,0,0, //.
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,1,1,0,0,0,
0,0,0,1,1,0,0,0,
};

BYTE character,x,y,point;

for (y = 0; y < 8;y++)
{
 for (x = 0; x < 8;x++)
 if (FONT[14][y][x] == 1) Plot(XOffset+1+x,YOffset-1+y,15);
}

for ( character = 0 ; string[character] != 0; character++)

if (((string[character]) >= 47) && ((string[character]) <= 58))
  for (y = 0; y < 8;y++)
    {
     for (x = 0; x < 8;x++)
     if (FONT[string[character]-48][y][x] == 1) Plot(XOffset+10+x+
         (character*8),YOffset-1+y,15);
    }
else
if (string[character] == 'S')
  for (y = 0; y < 8;y++)
    {
     for (x = 0; x < 8;x++)
     if (FONT[11][y][x] == 1) Plot(XOffset+10+x+ (character*8),
         YOffset-1+y,15);
    }
else
if (string[character] == 'E')
  for (y = 0; y < 8;y++)
    {
     for (x = 0; x < 8;x++)
     if (FONT[12][y][x] == 1) Plot(XOffset+10+x+ (character*8),
         YOffset-1+y,15);
    }
else
if (string[character] == 'W')
  for (y = 0; y < 8;y++)
    {
```

```
        for (x = 0; x < 8;x++)
        if (FONT[13][y][x] == 1) Plot(XOffset+10+x+ (character*8),
            YOffset-1+y,15);
      }
  else
  if (string[character] == 'N')
    for (y = 0; y < 8;y++)
      {
       for (x = 0; x < 8;x++)
       if (FONT[10][y][x] == 1) Plot(XOffset+10+x+ (character*8),
           YOffset-1+y,15);
      }
  else
  if (string[character] == '.')
    for (y = 0; y < 8;y++)
      {
       for (x = 0; x < 8;x++)
       if (FONT[15][y][x] == 1) Plot(XOffset+10+x+ (character*8),
           YOffset-1+y,15);
      }
}
```

packet.cpp

```
/*-------------------------------------------------------------*/
/*                                                             */
/*     This module contains the functions to support           */
/*     peer-to-peer IPX packet sending.                        */
/*     Written by Paul Archer.                                 */
/*-------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <mem.h>
#include <conio.h>
#include "defs.h"
#include "ipx-spx.h"
#include "packet.h"

/*****************************************************************************
*
 *
*
 *               G L O B A L     V A R I A B L E S
*
 *
*

*****************************************************************************
/


void ((far *IpxSpxCallPtr)(void));

struct IpxHeader HeaderForIpxSendPacket,
            HeaderForIpxListenPacket;

struct Ecb   ListenEcb,
            SendEcb;


USGC GlobalLocalTargetAddress[NODE_ADR_LENGTH];

struct NetworkAddress // NetworkAddressArray[IPX_MAX_CONNECTIONS],
                  LocalAddressInfo,
                  InputAddressInfo;

byte SendDataArea[PACKET_LENGTH];

byte ListenDataArea[PACKET_LENGTH];


/*****************************************************************************
*
 *
*
 *               Exits program if IPX is not present
*
 *
*

*****************************************************************************
/
```

```
void CheckForIpxPresent(void)
{
  struct SREGS SegReg;
  union  REGS  InReg, OutReg;
  InReg.h.ah = 0x7A;
  InReg.h.al = 0x00;

  int86x(DOS_MULTIPLEX, &InReg, &OutReg, &SegReg);

  if (OutReg.h.al != IPX_SPX_INSTALLED)
  {
      cprintf("\n IPX is not installed...");
      exit (-1);
  }
  else
  IpxSpxCallPtr = (void (far *)()) ((((long) SegReg.es) << 16) +
                                ((long) OutReg.x.di));

}

/*************************************************************************
*
 *
*
 *                Opens a socket for communication.
*
 *
*

*************************************************************************
/

void IpxSpxOpenSocket(USGI SocketNumber)
{
  USGC CompletionCode;
  USGI SwappedSocketNumber;
  USGI AssignedSocketNumber;

  swab((char *)&SocketNumber, (char *)&SwappedSocketNumber, sizeof(USGI));

  ASM mov AL,SHORT_LIVED_SOCKET
  ASM mov BX,IPX_SPX_OpenSocket
  ASM mov DX,SwappedSocketNumber

  ASM push BP

  IpxSpxCallPtr();  /* completion code provided in AL */

  ASM pop  BP

  ASM mov CompletionCode,AL
  ASM mov AssignedSocketNumber,DX

  swab((char *)&AssignedSocketNumber, (char *)&AssignedSocketNumber,
        sizeof(USGI));

  switch(CompletionCode)
  {
    case 0x00 :          break;
    case 0xF0 : printf("[Failure, IPX not installed]");
            break;
    case 0xFE : printf("[Failure, full socket table]");
```

```
            break;
    case 0xFF : printf("[Failure, socket already open]");
            break;
    default   : printf("[Unknown completion code (%02X)]",CompletionCode);
            break;
  }

}

/*************************************************************************
*
 *
*
 *                Sets up Event Control Block (ECB)
*
 *
*

*************************************************************************
/

void SetupECB(USGI ListenSocketNumber)
{
  USGI SwappedSocketNumber;

  swab((char *)&ListenSocketNumber, (char *)&SwappedSocketNumber,
        sizeof(USGI));

    ZeroEcb(&ListenEcb);   /* sets Esr pointer == 0000:0000 ==> no Esr */
    ListenEcb.SocketNumber = SwappedSocketNumber;
    ListenEcb.FragmentCount = 2;
    ListenEcb.FragmentDescriptor[0].Ptr =
                (void far *) &HeaderForIpxListenPacket;
    ListenEcb.FragmentDescriptor[0].Length =
                        sizeof(HeaderForIpxListenPacket);
    ListenEcb.FragmentDescriptor[1].Ptr =
                 (void far *) ListenDataArea;
    ListenEcb.FragmentDescriptor[1].Length =
                        sizeof(ListenDataArea);
    ZeroIpxHeader(&HeaderForIpxListenPacket);

//    IpxListenForPacket(&ListenEcb);
}


/*************************************************************************
*
 *
*
 *                Sends an IPX listen for packet request
*
 *
*

*************************************************************************
/

void IpxListenForPacket(struct Ecb *EcbPtr,unsigned char far
*ReplyDataArea,
                        unsigned char PacketSize)
{
  USGC CompletionCode;
  void far *FarPtr;
```

```
  USGI SegmentValue,OffsetValue;
  ListenEcb.FragmentDescriptor[1].Ptr =
                  (void far *) ReplyDataArea;
  ListenEcb.FragmentDescriptor[1].Length = PacketSize;

  FarPtr = (void far *) EcbPtr;

  SegmentValue = FP_SEG(FarPtr);
  OffsetValue  = FP_OFF(FarPtr);

  ASM mov BX,IPX_ListenForPacket

  ASM mov ES,SegmentValue
  ASM mov SI,OffsetValue

  ASM push BP

  IpxSpxCallPtr();            /* completion code provided in AL */

  ASM pop  BP

  ASM mov  CompletionCode,AL

  switch(CompletionCode)
  {
  case 0x00 : break;
  case 0xFF : printf("[Failure, nonexistant listening socket]");
              break;
  default   : printf("[Unknown completion code (%02X)]",
                                 CompletionCode);
              break;
  }

}

/*************************************************************************
*
 *
*
 *                Clears the IPX packet header
*
 *
*

*************************************************************************
/

ZeroIpxHeader(struct IpxHeader *IpxHeaderPtr)
{
  memset(IpxHeaderPtr, 0, sizeof(struct IpxHeader));
  return 0;
}

/*************************************************************************
*
 *
*
 *                Clears the ECB
*
 *
*
```

```
*****************************************************************************
/

ZeroEcb(struct Ecb *EcbPtr)
{
  memset((void *) EcbPtr, 0, sizeof(struct Ecb));
  return 0;
}


IpxRelinquishControl(void)
{
  ASM mov BX,IPX_RelenquishControl
  ASM push BP
  IpxSpxCallPtr();            /* no completion code provided in AL */
  ASM pop  BP
  return 0;
}

/*****************************************************************************
*
 *
*
 *                Sets up data for a packet headers, ECB, etc
*
 *
*

*****************************************************************************
/

void IpxSendSetup(USGC ADDR0,USGC ADDR1,USGC ADDR2,
              USGC ADDR3,USGC ADDR4,USGC ADDR5)
{
  struct NetworkAddress  RemoteNetworkAddress;
  USGI Index;
  USGI SocketNumber, SwappedSocketNumber;

  InputAddressInfo.NetworkNumber[0] = 0;
  InputAddressInfo.NetworkNumber[1] = 0;
  InputAddressInfo.NetworkNumber[2] = 0;
  InputAddressInfo.NetworkNumber[3] = 0;
  InputAddressInfo.NodeAddress[0] = ADDR0;
  InputAddressInfo.NodeAddress[1] = ADDR1;
  InputAddressInfo.NodeAddress[2] = ADDR2;
  InputAddressInfo.NodeAddress[3] = ADDR3;
  InputAddressInfo.NodeAddress[4] = ADDR4;
  InputAddressInfo.NodeAddress[5] = ADDR5;

  IpxGetLocalTarget(InputAddressInfo.NetworkNumber,
                    InputAddressInfo.NodeAddress,
                    WDS_LISTEN_SOCKET_NUMBER,
                    GlobalLocalTargetAddress);

         /*----- first set up the packet header -----*/

  ZeroIpxHeader(&HeaderForIpxSendPacket);

  HeaderForIpxSendPacket.PacketType = IPX_PACKET_TYPE;

  memmove(HeaderForIpxSendPacket.DestinationNetwork,
        InputAddressInfo.NetworkNumber, NETWORK_ADR_LENGTH);
```

```
  memmove(HeaderForIpxSendPacket.DestinationNode,
        InputAddressInfo.NodeAddress, NODE_ADR_LENGTH);

  SocketNumber = LISTEN_SOCKET;  /* target's socket */
  swab((char *)&SocketNumber, (char *)&SwappedSocketNumber, sizeof(USGI));
  HeaderForIpxSendPacket.DestinationSocket = SwappedSocketNumber;
            /*----- now set up the Ecb -----*/

  ZeroEcb(&SendEcb);   /* sets Esr pointer == 0000:0000 ==> no Esr */
  SocketNumber = SEND_SOCKET;
  swab((char *)&SocketNumber, (char *)&SwappedSocketNumber, sizeof(USGI));
  SendEcb.SocketNumber  = SwappedSocketNumber;

  memmove(SendEcb.ImmediateAddress,
        GlobalLocalTargetAddress, NODE_ADR_LENGTH);

  SendEcb.FragmentCount = 2;

  SendEcb.FragmentDescriptor[0].Ptr =
                      (void far *) &HeaderForIpxSendPacket;
  SendEcb.FragmentDescriptor[0].Length =
                           sizeof(HeaderForIpxSendPacket);
  SendEcb.FragmentDescriptor[1].Ptr =
                      (void far *) SendDataArea;
  SendEcb.FragmentDescriptor[1].Length =
                           sizeof(SendDataArea);
}

void IpxGetLocalTarget(USGC *DestNetworkPtr,
                   USGC *DestNodePtr,
                   USGI  DestSocket,
                   USGC *LocalTarget)
{
  void far *FarPtr;

  USGI Index;
  USGC CompletionCode;
  USGI TransportTime;
  USGI SwappedSocketNumber;
  USGI SegmentValue, OffsetValueRequest, OffsetValueReply;

  struct {
         USGC DestinationNetwork[4];
         USGC DestinationNode[6];
         USGI DestinationSocket;
       } RequestBuffer;

  struct {
         USGC LocalTarget[6];
       } ReplyBuffer;


  swab((char *)&DestSocket, (char *)&SwappedSocketNumber, sizeof(USGI));

  memmove(RequestBuffer.DestinationNetwork,
        DestNetworkPtr, NETWORK_ADR_LENGTH);

  memmove(RequestBuffer.DestinationNode,
        DestNodePtr, NODE_ADR_LENGTH);

  RequestBuffer.DestinationSocket = SwappedSocketNumber;
```

```
  FarPtr = (void far *) &RequestBuffer;

  SegmentValue        = FP_SEG(FarPtr);
  OffsetValueRequest  = FP_OFF(FarPtr);

  FarPtr = (void far *) &ReplyBuffer;

  OffsetValueReply = FP_OFF(FarPtr);

  ASM mov BX,IPX_GetLocalTarget

  ASM mov ES,SegmentValue
  ASM mov SI,OffsetValueRequest
  ASM mov DI,OffsetValueReply

  ASM push BP

  IpxSpxCallPtr();            /* completion code provided in AL */

  ASM pop  BP

  ASM mov CompletionCode,AL
  ASM mov TransportTime,CX

  memmove(LocalTarget, ReplyBuffer.LocalTarget, NODE_ADR_LENGTH);


  switch(CompletionCode)
  {
    case 0x00 : break ;
    case 0xFA : printf("[No path to destination found]");
            break;
    default   : printf("[Unknown completion code (%02X)]",
                                    CompletionCode);
            break;
  }
}


/*************************************************************************
*
 *
*
 *              Send a packet to remote machine
*
 *
*

*************************************************************************
/

void IpxSendPacket(USGC far * SourcePointer,USGC PacketSize)
{
  void far *FarPtr;
  USGI SegmentValue, OffsetValue;
  SendEcb.FragmentDescriptor[1].Ptr =
                    (void far *) SourcePointer;
  SendEcb.FragmentDescriptor[1].Length = PacketSize;


  FarPtr = (void far *) &SendEcb;

  SegmentValue = FP_SEG(FarPtr);
```

```
  OffsetValue  = FP_OFF(FarPtr);


  ASM mov BX,IPX_SendPacket

  ASM mov ES,SegmentValue
  ASM mov SI,OffsetValue

  ASM push BP

  IpxSpxCallPtr();            /* no completion code provided in AL */

  ASM pop  BP

}

void InitIPX (void)
{
   CheckForIpxPresent (); /* exits if IPX not present */
   IpxSpxOpenSocket (LISTEN_SOCKET);
   IpxSpxOpenSocket (SEND_SOCKET);
   SetupECB (LISTEN_SOCKET);
}
```

packet.h

```
#define USGI unsigned int
#define USGC unsigned char
#define USGL unsigned long

//
// Exported functions and variables from this module
//

extern struct Ecb ListenEcb,
                  SendEcb;

extern void ((far *IpxSpxCallPtr)());

//extern byte SendDataArea[PACKET_LENGTH];

//extern byte ListenDataArea[PACKET_LENGTH];


extern void CheckForIpxPresent(void);
extern void IpxSpxOpenSocket(USGI SocketNumber);
extern void SetupECB(USGI ListenSocketNumber);
extern void IpxListenForPacket(struct Ecb *EcbPtr,unsigned char far
             *ReplyDataArea,unsigned char PacketSize);
void IpxSendSetup(USGC ADDR0,USGC ADDR1,USGC ADDR2,
                  USGC ADDR3,USGC ADDR4,USGC ADDR5);
extern void IpxSendPacket(USGC far * SourcePointer,USGC PacketSize);

extern void IpxGetLocalTarget(USGC *DestNetworkPtr,
                    USGC *DestNodePtr,
                    USGI  DestSocket,
                    USGC *LocalTarget);

extern ZeroIpxHeader(struct IpxHeader *IpxHeaderPtr);
extern ZeroEcb(struct Ecb *EcbPtr);
extern IpxRelinquishControl(void);
extern void InitIPX (void);
```

ipx-spx.h

```
#define TRUE  1
#define FALSE 0

#define LISTEN 1
#define SEND   2

#define BORLAND    TRUE
#define MICROSOFT !BORLAND

#if BORLAND
#define ASM asm
#else
#define ASM __asm
#endif

#define IPX_SPX_OpenSocket           0x00
#define IPX_SPX_CloseSocket          0x01
#define IPX_GetLocalTarget           0x02
#define IPX_SendPacket               0x03
#define IPX_ListenForPacket          0x04
#define IPX_Schedule_IPX_Event       0x05
#define IPX_CancelEvent              0x06
#define IPX_GetIntervalMarker        0x08
#define IPX_GetInternetworkAddress   0x09
#define IPX_RelenquishControl        0x0A
#define IPX_DisconnectFromTarget     0x0B

#define SPX_Initialize               0x10
#define SPX_EstablishConnection      0x11
#define SPX_ListenForConnection      0x12
#define SPX_TerminateConnection      0x13
#define SPX_AbortConnection          0x14
#define SPX_GetConnectionStatus      0x15
#define SPX_SendSequencedPacket      0x16
#define SPX_ListenForSequencedPacket 0x17

#define SPX_INSTALLED      0xFF
#define SPX_NOT_INSTALLED  0x00

#define SPX_WATCHDOG_ENABLED     1
#define SPX_WATCHDOG_DISABLED    0
#define SPX_DEFAULT_RETRY_COUNT  0

#define SPX_POOL_SIZE        10

#define NODE_ADR_LENGTH       6
#define NETWORK_ADR_LENGTH    4

#define IPX_PACKET_TYPE       4
#define SPX_PACKET_TYPE       5

#define IPX_MAX_DATA_LENGTH 546
#define SPX_MAX_DATA_LENGTH 534

#define DOS_INT            0x21
#define DOS_MULTIPLEX      0x2F
#define IPX_SPX_INSTALLED  0xFF

#define MAX_NAME_SIZE        47
#define MAX_FRAGMENT_COUNT    2
#define IPX_MAX_CONNECTIONS 100
```

```
#define NETWARE_CONNECTION_SVCS_0xE3    '\xE3'
#define GET_CONNECTION_NUMBER           '\xDC'
#define GET_INTERNET_ADDRESS            '\x13'
#define GET_OBJECT_CONNECTION_NUMBERS   '\x15'
#define GET_CONNECTION_INFORMATION      '\x16'

#define  USER_BINDERY_OBJECT_TYPE 0x0001

#define LONG_LIVED_SOCKET  0FFh
#define SHORT_LIVED_SOCKET 000h

#define WDS_SEND_SOCKET_NUMBER   0x7655
#define WDS_LISTEN_SOCKET_NUMBER 0x7654

#define AES_EVENT_TIME_DELAY_VALUE ((USGI)(18.2 * 3.0))

struct ConnectionNbrArray {
                      USGI Length;
                      USGC Count;
                      char Connection[IPX_MAX_CONNECTIONS];
                     };

struct NetworkAddress {
                  USGC NetworkNumber[NETWORK_ADR_LENGTH];
                  USGC NodeAddress[NODE_ADR_LENGTH];
                  };


struct Ecb {
    void far *LinkAddress;
    void (far *EventServiceRoutine)();          /* you fill this in  */
    USGC     InUseFlag;
    USGC     CompletionCode;
    USGI     SocketNumber;                      /* you fill this in  */
    union    {
             struct {
                  USGC WorkSpace[4];
                  } Ipx;
             struct {
                  USGI ConnectionId;
                  USGC Unused[2];
                  } Spx;
           } UnionArea;
    USGC     DriverWorkSpace[12];
    USGC     ImmediateAddress[NODE_ADR_LENGTH];  /* IPX send ==> you fill
in */
    USGI     FragmentCount;                     /* you fill this in  */
    struct   {
            void far *Ptr;                  /* you fill these in */
            USGI      Length;               /* you fill these in */
           } FragmentDescriptor[MAX_FRAGMENT_COUNT];
};

struct IpxHeader {
       USGI CheckSum;
       USGI Length;
       USGC TransportControl;
       USGC PacketType;                /* IPX send ==> you fill this in */
       USGC DestinationNetwork[NETWORK_ADR_LENGTH]; /* you fill this in */
       USGC DestinationNode[NODE_ADR_LENGTH];       /* you fill this in */
       USGI DestinationSocket;                      /* you fill this in */
       USGC SourceNetwork[NETWORK_ADR_LENGTH];
```

```
        USGC SourceNode[NODE_ADR_LENGTH];
        USGI SourceSocket;
    };

struct SpxHeader {
        USGI CheckSum;
        USGI Length;
        USGC TransportControl;
        USGC PacketType;
        USGC DestinationNetwork[NETWORK_ADR_LENGTH];
        USGC DestinationNode[NODE_ADR_LENGTH];
        USGI DestinationSocket;
        USGC SourceNetwork[NETWORK_ADR_LENGTH];
        USGC SourceNode[NODE_ADR_LENGTH];
        USGI SourceSocket;
        USGC ConnectionControl;
        USGC DataStreamType;              /* you optionally fill this in */
        USGI SourceConnectionId;
        USGI DestinationConnectionId;
        USGI SequenceNumber;
        USGI AcknowledgeNumber;
        USGI AllocationNumber;
    };

struct StatusBuffer {
                    USGC ConnectionState;
                    USGC WatchDogState;
                    USGI LocalConnectionId;
                    USGI RemoteConnectionId;
                    USGI SequenceNumber;
                    USGI LocalAcknowledgeNumber;
                    USGI LocalAllocationNumber;
                    USGI RemoteAcknowledgeNumber;
                    USGI RemoteAllocationNumber;
                    USGI LocalSocket;
                    USGC ImmediateAddress[6];
                    USGC RemoteNetwork[4];
                    USGC RemoteNode[6];
                    USGI RemoteSocket;
                    USGI RetransmissionCount;
                    USGI EstimatedRoundTripDelay;
                    USGI RetransmittedPackets;
                    USGI SupressedPackets;
                };
```

defs.h

```
#define  byte unsigned char
#define  word insigned int
#define  MaxPacketBuffers 4
#define  PacketsInPlayBuffer 10
#define  PacketBufferSize 512

#define  LISTEN_SOCKET 0x7654
#define  SEND_SOCKET   0x7655

#define  CONTROL_LISTEN_SOCKET 0x7656
#define  CONTROL_SEND_SOCKET   0x7657

#define  NumberOfPacketDelays 4
#define  PACKET_LENGTH 512      /* 512 bytes per packet */
#define  FALSE 0
#define  TRUE  1
#define  OFF   0
#define  ON    1


#define BYTE unsigned char
#define WORD unsigned int
#define LONG unsigned long

#define USGI unsigned int
#define USGC unsigned char
#define USGL unsigned long

/* defines for bioskey */
#define ESC  0x11B
#define CURSOR_UP   0x4800
#define CURSOR_DOWN 0x5000
#define CR      13
```

mouse.cpp

```
static      int   Mouse = 0;

/*********************************************************
/
/   Functions Used Externally
/
*********************************************************/

/***********************************************
/ Mouse function 0. Initialize mouse.
/ If mouse is present then sets Mouse = 1 for
/ later presence indication.
***********************************************/
int   far   MouseInit( void)
{
   struct   REGPACK     regs;

   regs.r_ax = 0;
   intr( 0x33, &regs);
   if( regs.r_ax == (unsigned)-1) Mouse = 1;
   else Mouse = 0;
   return( regs.r_ax);
}

/***********************************************
/ Mouse function 1. Show mouse.
***********************************************/
void  far   MouseShow( void)
{
   struct   REGPACK     regs;

   if( Mouse == 1)
      {
       regs.r_ax = 1;
       intr( 0x33, &regs);
      }
}

/***********************************************
/ Mouse function 2. Hide mouse.
***********************************************/
void  far   MouseHide( void)
{
   struct   REGPACK     regs;

   if( Mouse == 1)
      {
       regs.r_ax = 2;
       intr( 0x33, &regs);
      }
}

/***********************************************
/ Mouse function 3. Get mouse position and button
/ status.
***********************************************/
int   far   MouseStat( int * button, int * xpos, int * ypos)
{
   struct   REGPACK     regs;

   if( Mouse == 1)
```

```
        {
         regs.r_ax = 3;
         intr( 0x33, &regs);
         *button = regs.r_bx;
         *xpos = regs.r_cx;
         *ypos = regs.r_dx;
        }
    return( Mouse);
}

/************************************************
/ Mouse function 4. Set mouse position.
************************************************/
void  far   MouseSet( int xpos, int ypos)
{
    struct   REGPACK     regs;

    if( Mouse == 1)
        {
         regs.r_ax = 4;
         regs.r_cx = xpos;
         regs.r_dx = ypos;
         intr( 0x33, &regs);
        }
}

/************************************************
/ Mouse function 6. Get button release information.
************************************************/
int   far   MouseRelease( int * button, int * xpos, int * ypos)
{
    struct   REGPACK     regs;

    if( Mouse == 1)
        {
         regs.r_ax = 6;
         regs.r_bx = *button;
         intr( 0x33, &regs);
         *button = regs.r_bx;
         *xpos = regs.r_cx;
         *ypos = regs.r_dx;
        }
    return( Mouse);
}

/************************************************
/ Mouse function 7. Set horizontal motion limits.
************************************************/
void  far   MouseXdim( int left, int right)
{
    struct   REGPACK     regs;

    if( Mouse == 1)
        {
         regs.r_ax = 7;
         regs.r_cx = left;
         regs.r_dx = right;
         intr( 0x33, &regs);
        }
}

/************************************************
/ Mouse function 8. Set vertical motion limits.
```

```
*************************************************/
void  far   MouseYdim( int top, int bottom)
{
   struct   REGPACK     regs;

   if( Mouse == 1)
      {
       regs.r_ax = 8;
       regs.r_cx = top;
       regs.r_dx = bottom;
       intr( 0x33, &regs);
      }
}
```

mouse.h

```
/*********************************************************
/
/   Mouse Module Header File  MOUSE.H
/
*********************************************************/

extern      int   far   MouseInit( void);
extern      void  far   MouseShow( void);
extern      void  far   MouseHide( void);
extern      int   far   MouseStat( int * button, int * xpos, int * ypos);
extern      void  far   MouseSet( int xpos, int ypos);
extern      int   far   MouseRelease( int * button, int * xpos, int *
ypos);
extern      void  far   MouseXdim( int left, int right);
extern      void  far   MouseYdim( int top, int bottom);
```

qlp.par

```
Write image to file Boolean (0/1)                          ==> 0
PRIIDS in Source file Boolean (0/1)                        ==> 1
Use Network Boolean (0/1)                                  ==> 0
FileWidth [No. of range bins in input file]                ==> 2048
FileHeight [Number of range lines in onput file ]          ==> 1999
NumberOfRangeBins [No. of range bins to be processed]==> 2048
StartRangeBin [FirstRangeBin to Process]                   ==> 0

DCOffset          [DC offset]                              ==> 127
ScaleFactor [Scale factor for output magnitude]            ==> 4
INNAME    [Input raw data file name]                       ==> ESAR.RNC
OUTNAME   [Output image file name]   QuickLookImage        ==> test.QLI
NumberOfSubApertures                                       ==> 4


SubApLength[0]          ==> 7
SubApLength[1]          ==> 3
SubApLength[2]          ==> 2
SubApLength[3]          ==> 2
SubApLength[4]          ==> 2
SubApLength[5]          ==> 1
SubApLength[6]          ==> 1
SubApLength[7]          ==> 1
SubApLength[8]          ==> 1
SubApLength[9]          ==> 1
SubApLength[10]         ==> 1
SubApLength[11]         ==> 1
SubApLength[12]         ==> 1
SubApLength[13]         ==> 1
SubApLength[14]         ==> 1
SubApLength[15]         ==> 1
SubApLength[16]         ==> 1
SubApLength[17]         ==> 1
SubApLength[18]         ==> 1
```

qlp.lis

```
#USE strmhdr
--{{{  Constants
VAL MaxApertureLength IS 250:
VAL MaxNumberOfRangeBins IS 128:
VAL MaxNumberOfSubApertures IS 25:  -- 25 works
VAL MaxTotalNumberOfRangeBins IS 512:
--}}}
--{{{  Protocols
PROTOCOL PIPELINE
  CASE
    config; INT; INT; INT; INT; [MaxNumberOfSubApertures + 1] INT
    data; INT::[] BYTE; INT::[] BYTE
    halt
:
PROTOCOL REQUEST
  CASE
    more
:
--}}}
--{{{  SC QLP Procedure
--:::A  3 10
--{{{F QLP Procedure
--:::F main04.tsr
--{{{  Headers
#USE strmhdr
#USE sphdr
#USE streamio
#USE ioconv
#USE extrio
#USE strings
#USE solib
#USE splib
#USE sklib
#USE snglmath
#USE linkaddr
--}}}
--{{{  Constants
VAL MaxApertureLength IS 250:
VAL MaxNumberOfRangeBins IS 128:
VAL MaxNumberOfSubApertures IS 42: -- 25:
VAL MaxTotalNumberOfRangeBins IS 512:
VAL REAL IS 0 :
VAL IMAG IS 1 :
VAL SINE IS 0 :
VAL COSINE IS 1:
VAL PI IS 3.14159265 (REAL32):
--}}}
--{{{  Protocols
PROTOCOL PIPELINE
  CASE
    config; INT; INT; INT; INT; [MaxNumberOfSubApertures + 1] INT
    data; INT::[] BYTE; INT::[] BYTE
    halt
:
PROTOCOL REQUEST
  CASE
    more
:
--}}}

VAL maxfilenamelen IS 80 :
```

```
PROC QLP (CHAN OF PIPELINE in,out,VAL INT RangeBinsToProcess)
  --{{{  Headers
  #USE strmhdr
  #USE sphdr
  #USE streamio
  #USE ioconv
  #USE extrio
  #USE strings
  #USE solib
  #USE splib
  #USE sklib
  #USE snglmath
  --}}}
  --{{{  Vars
  INT ScaleFactor,DC.Offset,NumberOfRangeBins :
  INT32 FileSize:
  INT32 InFile,OutFile :
  INT i,j :
  INT32 FilePosition :
  INT CurrentRangeBin :        -- Place for range data
  INT NumberOfSubApertures :
  REAL32 PHASESHIFT :          -- Phase Shift between subapertures
  INT SizeOfAperture :         -- Total length of Aperture
  [MaxApertureLength][MaxNumberOfRangeBins][2]INT IQ.Data :
  [MaxNumberOfSubApertures + 1]INT SubApertureMarker :
  [MaxNumberOfSubApertures][MaxTotalNumberOfRangeBins][2]INT SubApertureSum
:
  [MaxTotalNumberOfRangeBins]INT ApertureMagnitude :
  [2]BYTE IQ.Pair :
  [MaxNumberOfSubApertures]REAL32 PhaseShift :
  [MaxNumberOfSubApertures][2]REAL32 SinCosTable :
  BYTE CloseResult :
  INT Overflows:
  INT32 StartTime, EndTime, Dummy:
  BYTE Key, DummyByte :
  BOOL active :
  BYTE result :
  INT  size   :
  BOOL errorflag:
  INT32 SetupFile:
  [MaxTotalNumberOfRangeBins*2]BYTE IQBuffer:
  [MaxTotalNumberOfRangeBins]BYTE MagBuffer:
  INT IQblocklen:
  INT Magblocklen:
  --}}}
  SEQ

    active := TRUE
    size := 1
    WHILE active
      in ? CASE
        data; IQblocklen::IQBuffer;Magblocklen::MagBuffer
          SEQ
            --{{{  Do Quick Look Algorithm on a line of data
            --{{{  Variable Defs
            BYTE temp:
            INT BytesWritten:
            INT  tempint:
            INT IQValsLeft:
            INT NumberOfMagVals:
            [1]BYTE forfile:
            --}}}
```

```
            SEQ
              --{{{  Decrement and wraparound SubApertureMarkers
              SEQ i = 0 FOR NumberOfSubApertures + 1
                Marker IS SubApertureMarker[i] :
                SEQ
                  IF
                    Marker = 0
                      Marker := SizeOfAperture - 1
                    Marker > 0
                      Marker := Marker - 1
                    TRUE
                      SKIP
              --}}}
              SEQ CurrentRangeBin = 0 FOR RangeBinsToProcess
                SEQ
                  --{{{  Subtract from SubApertureSums
                  SEQ i = 0 FOR NumberOfSubApertures
                    Sum IS SubApertureSum[i][CurrentRangeBin] :
                    VAL IQ IS
IQ.Data[SubApertureMarker[i+1]][CurrentRangeBin] :
                    SEQ
                      Sum[REAL] := Sum[REAL] - IQ[REAL]
                      Sum[IMAG] := Sum[IMAG] - IQ[IMAG]
                  --}}}
                  --{{{  Read in new IQ Pair ( can optimize )
                  IQ.Pair[REAL] := IQBuffer[CurrentRangeBin*2]
                  IQ.Pair[IMAG] := IQBuffer[(CurrentRangeBin*2)+1]

                  IQ.Data[SubApertureMarker[0]][CurrentRangeBin][REAL] :=
(INT IQ.Pair[REAL]) - DC.Offset
                  IQ.Data[SubApertureMarker[0]][CurrentRangeBin][IMAG] :=
(INT IQ.Pair[IMAG]) - DC.Offset
                  --}}}
                  --{{{  Add from SubApertureSums
                  SEQ i = 0 FOR (NumberOfSubApertures )
                    Sum IS SubApertureSum[i][CurrentRangeBin] :
                    VAL IQ IS
IQ.Data[SubApertureMarker[i]][CurrentRangeBin] :
                    SEQ
                      -- Add new value to sub-aperture

                      Sum[REAL] := Sum[REAL] + IQ[REAL]
                      Sum[IMAG] := Sum[IMAG] + IQ[IMAG]
                  --}}}
                  --{{{  Sum SubApertures with phase shifts...
                  [2]INT ApSum :
                  SEQ
                    ApSum[REAL] := 0
                    ApSum[IMAG] := 0

                    SEQ i = 0 FOR NumberOfSubApertures
                      VAL Sine IS SinCosTable[i][SINE] :
                      VAL Cosine IS SinCosTable[i][COSINE] :
                      VAL RealSubApSum IS (REAL32 ROUND
SubApertureSum[i][CurrentRangeBin][REAL]) :
                      VAL ImagSubApSum IS (REAL32 ROUND
SubApertureSum[i][CurrentRangeBin][IMAG]) :
                      SEQ
                        --DO PHASE SHIFT FOR REAL PART
                        ApSum[REAL] := ApSum[REAL] +
                          (INT ROUND ((RealSubApSum * Cosine) +
(ImagSubApSum * Sine)))
```

```
                        -- NOW DO PHASE SHIFT FOR IMAG PART
                        ApSum[IMAG] := ApSum[IMAG] +
                          (INT ROUND ((ImagSubApSum * Cosine) +
(RealSubApSum * Sine)))

                    ApertureMagnitude[CurrentRangeBin] := ( ApSum[REAL]*
                      ApSum[REAL]) + (ApSum[IMAG] * ApSum[IMAG])
                  --}}}
                  --{{{  Calc Mag and clipping.
                  VAL tempint IS (ApertureMagnitude[CurrentRangeBin] /
ScaleFactor) :
                  IF
                    tempint > 255
                      SEQ
                        temp := 255 (BYTE)
                        Overflows := Overflows + 1
                    tempint <= 255
                      temp := BYTE tempint

                  MagBuffer[Magblocklen + CurrentRangeBin] := BYTE temp
                  --}}}
                  IQValsLeft := IQblocklen - (RangeBinsToProcess*2)
                  NumberOfMagVals := Magblocklen + RangeBinsToProcess
              out! data;IQValsLeft::[IQBuffer FROM (RangeBinsToProcess*2)
FOR IQValsLeft];
                   NumberOfMagVals::[MagBuffer FROM 0 FOR NumberOfMagVals]

            --}}}

        config; ScaleFactor; DC.Offset; SizeOfAperture;
NumberOfSubApertures; SubApertureMarker
          SEQ
            out ! config; ScaleFactor; DC.Offset; SizeOfAperture;
NumberOfSubApertures; SubApertureMarker
            --{{{  Initialize Vars etc
            PHASESHIFT := 0.36 (REAL32)
            Overflows := 0
            SEQ i = 0 FOR (MaxApertureLength)
              SEQ j = 0 FOR (MaxNumberOfRangeBins)
                SEQ
                  IQ.Data[i][j][REAL] := 0
                  IQ.Data[i][j][IMAG] := 0
            SEQ i = 0 FOR (MaxNumberOfSubApertures )
              SEQ
                SEQ j = 0 FOR (MaxNumberOfRangeBins)
                  SEQ
                    SubApertureSum[i][j][REAL] := 0
                    SubApertureSum[i][j][IMAG] := 0
            FilePosition := INT32 0
            SEQ i = 0 FOR (NumberOfSubApertures)
              REAL32 TempNum:
              SEQ
                TempNum := ABS (REAL32 ROUND (i - (( NumberOfSubApertures-
1)/2)))
                PhaseShift[i] := (TempNum * PHASESHIFT) * ((2.0 (REAL32)) *
PI)    ---TempNum
            SEQ i = 0 FOR (NumberOfSubApertures)
              REAL32 TempNum:
              SEQ
                TempNum := PhaseShift[i]
                TempNum := SIN (TempNum)
                SinCosTable[i][SINE] := TempNum
                TempNum := PhaseShift[i]
```

```
                TempNum := COS (TempNum)
                SinCosTable[i][COSINE] := TempNum
            --}}}

        halt
          SEQ
            out ! halt
            active := FALSE
:
--}}}F
--...F code HT8
--:::A  1 2
--:::F main04.dcd
--...F descriptor
--:::A  1 4
--:::F main04.dds
--...F link
--:::A  1 9
--:::F main04.dlk
--...F debug
--:::A  1 5
--:::F main04.ddb
--...F CODE SC QLP Procedure 15:19:11 16-May-94
--:::A  2 10
--:::F main04.csc
--}}}
--{{{  SC Main Procedure
--:::A  3 10
--{{{F Main QLP Procedure
--:::F main0102.tsr
--{{{  Headers
#USE strmhdr
#USE sphdr
#USE streamio
#USE ioconv
#USE extrio
#USE strings
#USE solib
#USE splib
#USE sklib
#USE snglmath
#USE linkaddr
--}}}
--{{{  Constants
VAL MaxApertureLength IS 250:
VAL MaxNumberOfRangeBins IS 128:
VAL MaxTotalNumberOfRangeBins IS 512:
VAL MaxNumberOfSubApertures IS 42: --25:
VAL REAL IS 0 :
VAL IMAG IS 1 :
VAL SINE IS 0 :
VAL COSINE IS 1:
VAL PI IS 3.14159265 (REAL32):
--}}}
--{{{  Protocols
PROTOCOL PIPELINE
  CASE
    config; INT; INT; INT; INT; [MaxNumberOfSubApertures + 1] INT
    data; INT::[] BYTE; INT::[] BYTE
    halt
:
PROTOCOL REQUEST
  CASE
```

```
    more
:
--}}}

VAL maxfilenamelen IS 80 :

--{{{  Skip to > procedure
PROC skipgt (CHAN OF SP fs, ts, VAL INT32 streamid, INT len,
             []BYTE data)
  [maxfilenamelen] BYTE inline :
  INT pos :
  BYTE result :
  SEQ
    so.gets (fs, ts, streamid, len, inline, result)
    pos := char.pos ('>', inline) + 1
    WHILE ((inline [pos] = ' ') AND (pos < len))
      pos := pos + 1
    len := len - pos
    [data FROM 0 FOR len] := [inline FROM pos FOR len]
:
--}}}
--{{{  Get int procedure
PROC getint (CHAN OF SP fs, ts, VAL INT32 streamid, INT data,
             BOOL errorflag)
  [maxfilenamelen] BYTE inline :
  INT len :
  BOOL error :
  SEQ
    skipgt (fs, ts, streamid, len, inline)
    STRINGTOINT (error, data, [inline FROM 0 FOR len])
    errorflag := errorflag OR error
:
--}}}
--{{{  Get int32 procedure
PROC getint32 (CHAN OF SP fs, ts, VAL INT32 streamid, INT32 data,
             BOOL errorflag)
  [maxfilenamelen] BYTE inline :
  INT len :
  BOOL error :
  SEQ
    skipgt (fs, ts, streamid, len, inline)
    STRINGTOINT32 (error, data, [inline FROM 0 FOR len])
    errorflag := errorflag OR error
:
--}}}
--{{{  Get real procedure
PROC getreal (CHAN OF SP fs, ts, VAL INT32 streamid, REAL32 data,
             BOOL errorflag)
  [maxfilenamelen] BYTE inline :
  INT len :
  BOOL error :
  SEQ
    skipgt (fs, ts, streamid, len, inline)
    STRINGTOREAL32 (error, data, [inline FROM 0 FOR len])
    errorflag := (errorflag OR ISNAN (data)) OR error
:
--}}}

PROC Main (CHAN OF ANY from.hbr, from.host, to.host, CHAN OF PIPELINE
in,out, CHAN OF REQUEST fetch)

  --{{{  Headers
  #USE strmhdr
```

```
#USE sphdr
#USE streamio
#USE ioconv
#USE extrio
#USE strings
#USE solib
#USE splib
#USE sklib
#USE snglmath
--}}}

--{{{  Vars
INT ScaleFactor,DC.Offset,NumberOfRangeBins :
INT32 FileSize:
INT32 InFile,OutFile :
INT i,j :
INT32 FilePosition :
INT CurrentRangeBin :        -- Place for range data
INT NumberOfSubApertures :
REAL32 PHASESHIFT :          -- Phase Shift between subapertures
INT SizeOfAperture :         -- Total length of Aperture
[MaxApertureLength][MaxNumberOfRangeBins][2]INT IQ.Data :
[MaxNumberOfSubApertures + 1]INT SubApertureMarker :
[MaxNumberOfSubApertures][MaxTotalNumberOfRangeBins][2]INT SubApertureSum
:
[MaxTotalNumberOfRangeBins]INT ApertureMagnitude :
[2]BYTE IQ.Pair :
[MaxNumberOfSubApertures]REAL32 PhaseShift :
[MaxNumberOfSubApertures][2]REAL32 SinCosTable :
BYTE CloseResult :
INT Overflows:
INT32 StartTime, EndTime, Dummy:
BYTE Key, DummyByte :
BOOL active :
BYTE result :
INT  size   :
BOOL errorflag:
[MaxTotalNumberOfRangeBins*2]BYTE InIQBuffer:
[MaxTotalNumberOfRangeBins*2]BYTE OutIQBuffer:
[MaxTotalNumberOfRangeBins]BYTE InMagBuffer:
[MaxTotalNumberOfRangeBins]BYTE OutMagBuffer:
INT FinalNumberOfRangeBins,dummyval:
--}}}
SEQ
  active := TRUE
  to.host ! BYTE (99)
  size := 1
  --{{{  Read QLOOK data from qserver
  from.host ? ScaleFactor -- Integer
  from.host ? DC.Offset
  from.host ? SizeOfAperture
  from.host ? NumberOfRangeBins
  from.host ? NumberOfSubApertures
  PHASESHIFT := 0.36 (REAL32)
  SEQ i = 0 FOR NumberOfSubApertures + 1
    SEQ
      from.host ? SubApertureMarker[i]
  --}}}
  ScaleFactor := ScaleFactor * ScaleFactor
  to.host ! BYTE (221)
  out ! config; ScaleFactor; DC.Offset;
        SizeOfAperture; NumberOfSubApertures;
        SubApertureMarker
```

```
    to.host ! BYTE (222)
    in ? CASE
      config; ScaleFactor; DC.Offset; SizeOfAperture; NumberOfSubApertures;
SubApertureMarker
        SEQ
          to.host ! BYTE (255)  -- Network setup
    to.host ! BYTE (111)
    to.host ! BYTE (0)
    from.host ? [InIQBuffer FROM 0 FOR (NumberOfRangeBins *2)]    -- In
meaning INto xputer network
    out ! data; NumberOfRangeBins*2::InIQBuffer;0::InMagBuffer

    WHILE active
      ALT
        fetch ? CASE
          more
            SEQ
              --to.host ! BYTE (43)
              to.host ! BYTE (0)
              from.host ? [InIQBuffer FROM 0 FOR (NumberOfRangeBins *2)]
-- In meaning INto xputer network
              --to.host ! BYTE (44)
              out ! data; NumberOfRangeBins*2::InIQBuffer;0::InMagBuffer

        in ? CASE
          data; dummyval::OutIQBuffer;FinalNumberOfRangeBins::OutMagBuffer
            SEQ
              to.host ! BYTE (255) -- Flag to indicate mag data returning
              to.host ! [OutMagBuffer FROM 0 FOR FinalNumberOfRangeBins]
            --to.host ! BYTE (45)

          halt
            active := FALSE

        --to.host ! BYTE (55)
:
--}}}F
--...F code HT8
--:::A  1 2
--:::F main0102.dcd
--...F descriptor
--:::A  1 4
--:::F main0102.dds
--...F link
--:::A  1 9
--:::F main0102.dlk
--...F debug
--:::A  1 5
--:::F main0102.ddb
--...F CODE SC Main Procedure 15:19:41 16-May-94
--:::A  2 10
--:::F main0102.csc
--}}}
--{{{  SC Helper procedure
--:::A  3 10
--{{{F Helper procedure
--:::F helper11.tsr
--{{{  COMMENT Include files
--:::A COMMENT FOLD
--{{{  Include files
#USE ioconv
#USE extrio
#USE sphdr
```

```
#USE snglmath
#USE mathhdr
#USE strings
--}}}
--}}}
--{{{  Constants
VAL MaxApertureLength IS 250:
VAL MaxNumberOfRangeBins IS 128:
VAL MaxNumberOfSubApertures IS 42: --25:
VAL MaxTotalNumberOfRangeBins IS 512:
VAL REAL IS 0 :
VAL IMAG IS 1 :
VAL SINE IS 0 :
VAL COSINE IS 1:
VAL PI IS 3.14159265 (REAL32):
--}}}
--{{{  Protocols
PROTOCOL PIPELINE
  CASE
    config; INT; INT; INT; INT; [MaxNumberOfSubApertures + 1] INT
    data; INT::[] BYTE; INT::[] BYTE
    halt
:
PROTOCOL REQUEST
  CASE
    more
:
--}}}

PROC helper (CHAN OF PIPELINE in, out, CHAN OF REQUEST fetch)
  [MaxTotalNumberOfRangeBins*2]BYTE IQBuffer:
  [MaxTotalNumberOfRangeBins]BYTE MagBuffer:
  INT IQblocklen:
  INT Magblocklen:
  --{{{  Vars
  INT ScaleFactor,DC.Offset,NumberOfRangeBins :
  INT32 FileSize:
  INT32 InFile,OutFile :
  INT i,j :
  INT32 FilePosition :
  INT CurrentRangeBin :          -- Place for range data
  INT NumberOfSubApertures :
  REAL32 PHASESHIFT :            -- Phase Shift between subapertures
  INT SizeOfAperture :           -- Total length of Aperture
  [MaxApertureLength][MaxNumberOfRangeBins][2]INT IQ.Data :
  [MaxNumberOfSubApertures + 1]INT SubApertureMarker :
  [MaxNumberOfSubApertures][MaxTotalNumberOfRangeBins][2]INT SubApertureSum
:
  [MaxTotalNumberOfRangeBins]INT ApertureMagnitude :
  [2]BYTE IQ.Pair :
  [MaxNumberOfSubApertures]REAL32 PhaseShift :
  [MaxNumberOfSubApertures][2]REAL32 SinCosTable :
  BYTE CloseResult :
  INT Overflows:
  INT32 StartTime, EndTime, Dummy:
  BYTE Key, DummyByte :
  BOOL active :
  BYTE result :
  INT  size   :
  BOOL errorflag:
  INT32 SetupFile:
  [MaxTotalNumberOfRangeBins*2]BYTE IQBuffer:
  [MaxTotalNumberOfRangeBins]BYTE MagBuffer:
```

```
  INT IQblocklen:
  INT Magblocklen:

  --}}}

  BOOL active :
  SEQ
    active := TRUE
    WHILE active
      in ? CASE
        data; IQblocklen::IQBuffer;Magblocklen::MagBuffer
          SEQ
            out ! data; IQblocklen::IQBuffer;Magblocklen::MagBuffer
            fetch ! more
        config; ScaleFactor; DC.Offset; SizeOfAperture;
NumberOfSubApertures; SubApertureMarker
          SEQ
            out ! config; ScaleFactor; DC.Offset; SizeOfAperture;
NumberOfSubApertures; SubApertureMarker
        halt
          SEQ
            out ! halt
            active := FALSE

:
--}}}F
--...F code HT8
--:::A  1 2
--:::F helper11.dcd
--...F descriptor
--:::A  1 4
--:::F helper11.dds
--...F debug
--:::A  1 5
--:::F helper11.ddb
--...F CODE SC Helper procedure 15:19:58 16-May-94
--:::A  2 10
--:::F helper11.csc
--}}}
#USE linkaddr
VAL Transputers IS 5 :
CHAN OF ANY from.hbr,to.host,from.host :
[Transputers] CHAN OF PIPELINE pipe :
PLACED PAR
  PROCESSOR 0 T8
    CHAN OF REQUEST fromhelper :
    CHAN OF PIPELINE tohelper :
    PLACE from.hbr AT link1.in :
    PLACE from.host AT link0.in :
    PLACE to.host AT link0.out :
    PLACE pipe[0] AT link2.out :
    PLACE pipe[Transputers-1] AT link3.in :
    PAR
      Main (from.hbr, from.host, to.host,pipe[Transputers-
1],tohelper,fromhelper)
      helper (tohelper,pipe[0], fromhelper)
  PLACED PAR transputer = 1 FOR (Transputers - 1)
    PROCESSOR transputer T8
      PLACE pipe[transputer-1] AT link0.in :
      PLACE pipe[transputer] AT link2.out :
      QLP  (pipe[transputer-1],pipe[transputer],32 / (Transputers-1))
```

```
      Main (from.hbr, from.host, to.host,pipe[Transputers-
1],tohelper,fromhelper)
      helper (tohelper,pipe[0], fromhelper)
  PLACED PAR transputer = 1 FOR (Transputers - 1)
    PROCESSOR transputer T8
      PLACE pipe[transputer-1] AT link0.in :
      PLACE pipe[transputer] AT link2.out :
      QLP   (pipe[transputer-1],pipe[transputer],32 / (Transputers-1))
```