The application of digital techniques to an automatic radar track extraction system

Spearman, Richard R.

THE

APPLICATION OF DIGITAL TECHNIQUES

TO AN

AUTOMATIC RADAR TRACK EXTRACTION SYSTEM

by

Richard R. Spearman

A thesis submitted in accordance with the regulations for the degree of Doctor of Philosophy in the University of Durham, March 1988.

The Application of Digital Techniques to an Automatic Radar Track Extraction System

Richard R. Spearman

ABSTRACT

'Modern' radar systems have come in for much criticism in recent years, particularly in the aftermath of the Falklands campaign. There have also been notable failures in commercial designs, including the well-publicised 'Nimrod' project which was abandoned due to persistent inability to meet signal processing requirements. There is clearly a need for improvement in radar signal processing techniques as many designs rely on technology dating from the late 1970's, much of which is obsolete by today's standards.

The Durham Radar Automatic Track Extraction System (RATES) is a practical implementation of current microprocessor technology, applied to plot extraction of surveillance radar data. In addition to suggestions for the design of such a system, results are quoted for the predicted performance when compared with a similar product using 1970's design methodology. Suggestions are given for the use of other VLSI techniques in plot extraction, including logic arrays and digital signal processors. In conclusion, there is an illustrated discussion concerning the use of systolic arrays in RATES and a prediction that this will represent the optimum architecture for future high-speed radar signal processors.

THE

APPLICATION OF DIGITAL TECHNIQUES

TO AN

AUTOMATIC RADAR TRACK EXTRACTION SYSTEM

by

Richard R. Spearman

A thesis submitted in accordance with the regulations for the degree of Doctor of Philosophy in the University of Durham, March 1988.

Thanks and acknowledgements...

...To Mr A. Shepherd and Dr. J.H. Miles, of the Admiralty Research Establishment in Portsmouth, who suggested a study of enhancements to the RATES prototype, and subsequently arranged funding for this research.

...To my supervisor, Professor C.T. Spracklen, of Aberdeen University for introducing me to the complexities of modern radar systems, and for the help he has given in all aspects of the work, especially following his move away from Durham. I wish him continued success as Professor of Electronics at Aberdeen.

...To my wife, Debby, for putting up with my late nights in the final stages of work on this thesis, and providing general encouragement throughout.

...To the staff of Digitus Ltd., and in particular Peter Woolfenden, for the invaluable help given through the loan of an Apple Macintosh and printing facilities. Without these, this thesis would never have been completed.

CONTENTS

1. Principles and Evolution of Radar

   1.0 Introduction
   1.1 Early History
   1.2 Basic Principles
   1.3 Radar Developments in the 1930's
   1.4 Radar in the 1940's
   1.5 Pulsed Radar Principles
   1.6 Post-War Developments
   1.7 Future Development Areas

2. Radar Signal Processing

   2.0 Introduction
   2.1 The Requirement for Signal Processing
   2.2 Detection of Targets in Clutter
   2.3 Filtering Techniques in Homogenous Clutter
   2.4 The Second Threshold Detector
   2.5 Data Reduction Through Plot Forming
   2.6 Summary

3. The RATES Radar System

   3.0 Introduction
   3.1 The RATES Specification
   3.2 The Input Interface
   3.3 The Constant False Alarm Processor
   3.4 The Within Beam Integrator
   3.5 The Plot Forming Processor
   3.6 System Control and Timing Functions
   3.7 The Interface to the Computer
   3.8 Electronic Counter Measures Detector
   3.9 The Display Driver
   3.10 The Control Unit
   3.11 Filtering and Tracking Computers

4. Introduction to VLSI Technology inRATES

   4.0 Introduction
   4.1 Deficiencies in the RATES Prototype
   4.2 Introduction to the Durham RATES
   4.3 Digital Design Approach
   4.4 A Solution in Custom VLSI?
   4.5 Programmable Logic in RATES
   4.6 Microprocessors in Radar Signal Processing
   4.7 A Summary of some other VLSI Devices

5. Applications of Digital Signal Processors

   5.0 Introduction
   5.1 Digital Signal Processors
   5.2 DSP Design of a CFAR Processor
   5.3 The Within-Beam Integrator
   5.4 Plot Forming Processing
   5.4 Summary of DSP Applications

6. A Microprocessor-based Tracking System

6.0 Introduction
6.1 The RATES Prototype Tracking System
6.2 Development System Computer Hardware
6.3 The Computer Interface
6.4 A Custom DMA Interface
6.5 Interface Design Using Proprietary Devices
6.6 RATES Software Design
6.7 Summary

7. Testing & Performance of the Development System

7.0 Introduction
7.1 Testing the Plot Extractor
7.2 Testing Using Recorded Radar Data
7.3 The RATES Test Target Generator
7.4 Test Scenarios and Results
7.5 Interface Performance

8. Systolic Architectures for Radar Signal Processors

8.0 Introduction
8.1 The Need for a Different Architecture
8.2 Principles of Systolic Arrays
8.3 Systolic Arrays in RATES
8.4 Simulation of a Systolic Plot Extractor
8.5 Summary and Suggestions for a Practical Design

9. Conclusions and Recommendations of Further Work

9.0 Research Objectives
9.1 Summary and Conclusions
9.2 Recommendations for Further Work

Appendices:

A References and Bibliography
B Backplane Connections for the Motorola 68-KDM Board.
C RATES Development Software, Source Listing.
D Test Target Generator, Source Listing
E Source Listing for Systolic CFAR Simulation
F "The Application of Systolic Arrays to Radar Signal Processing", (Spearman, Spracklen & Miles, Proc. IEEE Radar '86).

1.0 Introduction

In evaluating the current techniques and problems in modern radar systems it is considered necessary to provide an overview of previous work and developments in this field, presenting the concepts upon which radar detection (or radio-location, as it was formerly known) is based. Radar techniques from the 1930's and 1940's are covered in detail and an overview of more recent developments is also given. In conclusion some aspects of modern radar research are outlined, with particular reference to the type of systems involved in this thesis.

1.1 Early History

Radar is a system by which we are able to detect the position of an object, in space, without any active co-operation on the part of the object, save for the fact that it must be capable of reflecting radio waves.

As early as 1888, Heinrich Hertz [1] had shown the basic principles of radio-location, but it was not until 1904, following successful trials on ship targets, that a British patent was issued to Christian Hulsmeyer on this subject [2]. In spite of these demonstrations no further work was carried out, although Marconi, speaking to the American Institute of Radio Engineers in 1922 [3], stated:

> *"I should like to refer to another possible application of these (short) waves, which, if successful would be of great value to navigators. As was first shown by Hertz, electric waves can be completely reflected by conducting bodies. In some of my tests, I have noticed the effects of reflection and deflection of these waves by metallic objects miles away. It seems to me that it should be possible to design apparatus by means of which a ship could radiate or project a divergent beam of these rays in any desired direction, which, if coming across a metallic object such as another steamer or ship, would be reflected back to a receiver screened form the local transmitter on the sending ship and thereby immediately reveal the presence and bearing of ships, even though these ships be unprovided with any kind of radio."*

This suggested that the *bearing* of an object could be determined by reflection of radio-waves from a floodlighting source. To calculate its exact position, it implies that it would be necessary to have two receiving stations, a set distance apart, each providing bearing information. The transmitted beam would, of course, have to be very narrow for any degree of accuracy, but the principle is sound. Ideally, a system would be required which would be capable of determining the exact position of an object using a single combined transmission & reception point.

1.2 Basic Principles

In practice, the relative position of any object in space can be represented by 3 coordinates: **range** , **bearing** (the term **azimuth** is also used), and **elevation** , (**Figure 1.1**).

If a signal is transmitted towards an object, its bearing and/or elevation may be determined from the direction and angle of maximum reflection, however calculation of its distance requires some measurement of the time taken for the echo to return. If a continuous waveform were transmitted, then it would not be possible to detect an echo since the wave carries no unique points of identification. However, if the wave were modulated by some means, it would be possible to identify some significant point on the signal. Two techniques of modulation have been developed: frequency and pulse (amplitude) modulation and are now outlined in principle.

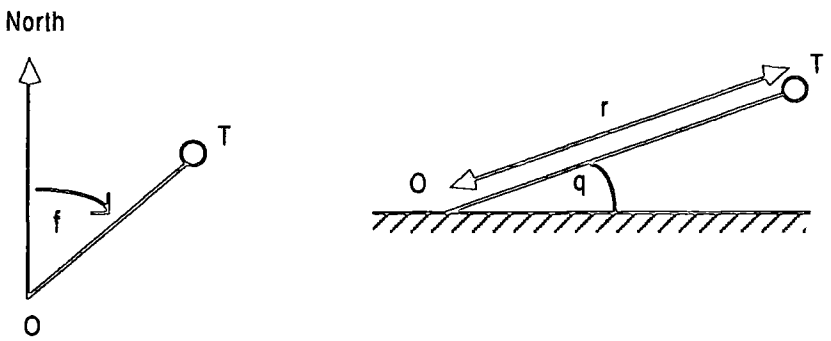


Fig. 1.1 : Determination of Position of Object (T) Using 3 Co-ordinates

1.2.1 Frequency Modulation

With this technique, the carrier frequency of the transmitter is swept through a linear sawtooth, (**Figure 1.2 : Solid line**) and a reflected signal is received, delayed by a period proportional to the distance of the target. Since the waveforms are otherwise identical, the result of combining them is a beat corresponding the frequency difference at a given point in time and it may be noted that :

$$\text{delay time} = \frac{\text{beat frequency}}{\text{rate of change of carrier frequency}}$$

This method was initially used, in 1924, to measure the height of the ionosphere but was later discarded in favour of amplitude modulation (developed in the U.S. by Breit & Tuve) which simplified multiple target detections.

1.2.2 Amplitude Modulation

Amplitude modulation uses a combination of the carrier frequency and short r.f. pulses, transmitted at regular intervals. The pulse travels out to the target and is reflected back, enabling a direct measurement of distance from the time taken for the echo to be received. With careful synchronisation of the initial pulse and a linear timebase, a cathode ray tube (C.R.T.) may be used to give a steady picture of the returned echoes [cf : Section 1.4.1].

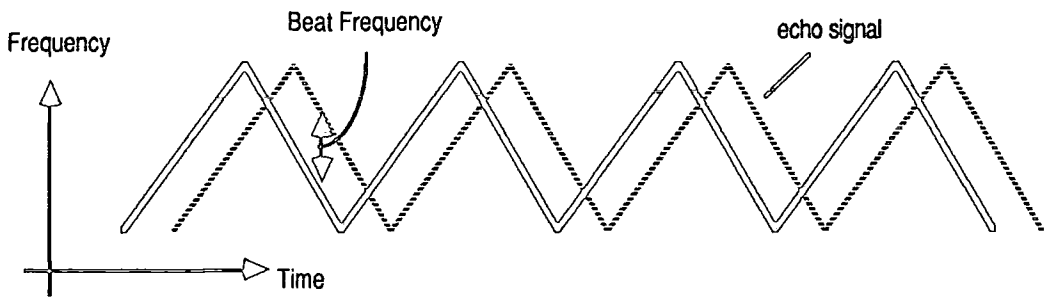


Fig. 1.2 : Basic Principle of F.M. Radar

The first record of any practical observation of these techniques was in Britain in Dec. 1931, when it was noted that an aircraft flying near a 5m (ultra-short-wave) receiving station induced a beat frequency which was related to the speed of the aircraft. Similar observations were noted in the U.S. the following year and in 1934 Young began work on the detection of aircraft by pulsed radio. He demonstrated a working version in December of that year showing that echoes could be detected, at short range, from an aircraft flying across the transmitter beam. More importantly, he proved that the pulse transmitter and receiver worked independantly with neither interference nor receiver blocking from the transmitted pulses.

This work gave the Americans a lead in transmitter design and work was then started on shorter wavelengths, in the order of 150cm. In mid 1936, using a pair of directed antennae, high signal echoes were received from the city of Washington D.C. at 8 miles, but no aircraft could be detected beyond 4 miles. It was later observed that use of the same antenna for transmission and reception, by way of a duplexer, led to a fourfold increase in the sensitivity of the radar. In Spring 1937, this was confirmed when such an aerial, mounted on a 5" gun turret on a ship, was used to detect aircraft movements to a distance of 16 miles. With the introduction of more powerful transmitting tubes in 1938, the duplexer was abandoned in favour of spark gap switching and ship-based radar, capable of detecting ships, aircraft, buoys, birds and 14" shells became standard equipment in the American Navy.

Early work in the States introduced the GAIN x RESPONSE-TIME ratio as a measure of the performance of a radar, and new valves were developed to cope with the higher frequencies and gains required. In receivers, amplitude limiting was forced on the received pulses in order to improve the response and recovery times, although the high voltage gains employed, (in the order of 10^6) meant that the screen of the c.r.t. was filled with input noise.

1.3 Radar Developments in the 1930's

With the approach of war in the mid 30's, these secret developments were hidden from other countries, who independently continued with their own research. Notably in Britain the development of radar followed from heightening military preparations in Germany. Sir Robert Watson-Watt [5] was initially asked to study the uses of radio waves to destroy enemy aircraft, but when calculations proved this to be impossible, suggested their use for detection of the enemy rather than its destruction. In Feb. 1935, he submitted an historic memorandum entitled "Detection and location of aircraft by radio methods" in which he discounted the possibilities of using noise emitted from targets but showed that reflected radiation would be of sufficient strength to be detected over ranges as great as 100 miles.

1.3.1 Watson-Watt's Experiments

Watson-Watt's proposals suggested the use of transmitted pulses and a method of timing the echoes on a cathode ray tube screen to indicate the distance of the target. Although accurate positional information could be calculated using distance measurements from three receiving stations, he promoted the use of a rotating beam from a single transmission and reception site, stressing the importance of continuing work into shorter wavelengths. He concluded that aircraft could also be fitted with a transponder, which would transmit a coded message on the same or slightly differing frequency as an interrogating radar beam, enabling identification of friendly and hostile targets (I.F.F.).

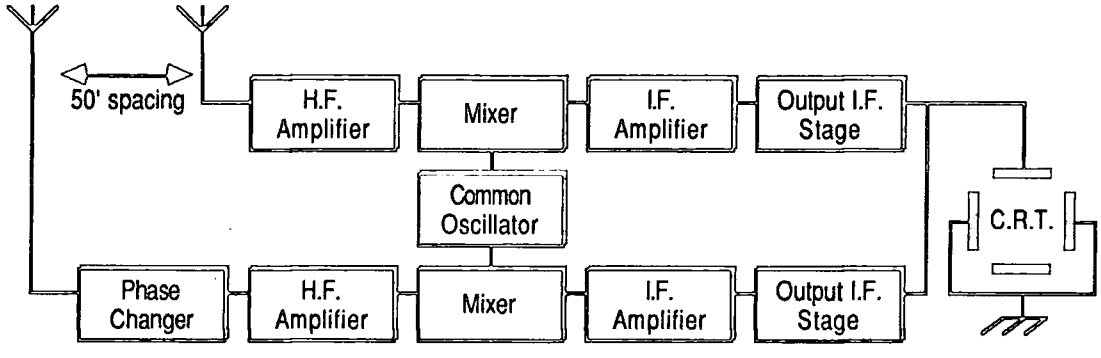


Fig. 1.3 : Receiver used in Watson-Watt's Demonstration

Watson-Watt demonstrated these proposals on 26th Feb. 1935, using a specially transmitted continuous wave (C.W.) signal from a B.B.C. short-wave transmitter, radiating 10kW on 49.8 m in a southerly direction. An aircraft flew up and down in the beam at a speed of 100mph and the receiver combined the heavily attenuated ground wave with the reflected signal, successfully indicating a beat frequency on the cathode ray tube.

The demonstration made use of the receiving circuit shown in Figure 1.3, producing a small vertical line on the c.r.t. (introduced through television work) which varied in length according to the distance of the target.

This initial experiment, although somewhat crude, provided the basis for future radar work in Britain. In the early days, receiving circuits capable of sensitive detection over wide bandwidths, such as the superheterodyne, were well developed from previous work in television so whilst transmitter and antenna design progressed apace during the early 30's, the techniques used to detect the returned echo did not change so rapidly. Special valves were developed, capable of several hundred kilowatts output for short periods of time and resulted in the introduction of the "Chain Home" radar network in 1937. These stations were put on 24 hour operation in early 1938 and constituted the world's first fully operational detection system, playing a major part in the success of the Battle of Britain.

1.3.2 The Chain Home Radar Network

For the Chain Home, the technique of reception remained unchanged but the display was modified such that the echo appeared as a glitch at some point along a timebase-driven x-axis. A linear timebase enabled the position of the echo to be directly proportional to the distance of the target from the transmitter. Precision ranging could be accomplished using a display which presented the ground wave and echoes on the upper trace whilst showing a series of calibration marks on the lower, (**Figure 1.4**). A portion of the x-axis, in the region of the ranging pip, could be selected and expanded by turning a hand wheel and a trained operator had only to align the left-hand edges of the two pulses in order to obtain an accurate range indication. Later refinements removed the calibration pips and replaced them with a moveable ranging spot.

The Chain Home made use of a floodlighting bistatic radar system, i.e. separate transmitting and receiving sites with aerials mounted on 360' steel transmitting towers and 240' wooden towers for receiving. This was capable of providing accurate range information but was not so good at determining elevation and bearing. If tracks from several receiving stations were associated, target bearings could be determined, however the problems associated with this were obvious as it was possible, for example, for site A to measure range on aircraft X : site B to measure range on aircraft Y and in associating it with the site A detection, give a range on an imaginary aircraft Z.

In Autumn 1938, however, monostatic radar (using a single transmitting and receiving site) was introduced, capable of measuring range to 1km and bearing to 1.5°. This use of the same antenna for transmission and reception followed development of the duplexer,

which was essentially a high speed switch based on spark gap and quarter-wave tube principles: When the transmitter was active the spark gap arced and presented a low impedance, i.e. a closed switch, but the end of the transmission pulse would break the arc and cause the "switch" to open. The quarter wave tubes enabled selective switching at different points in the circuitry.

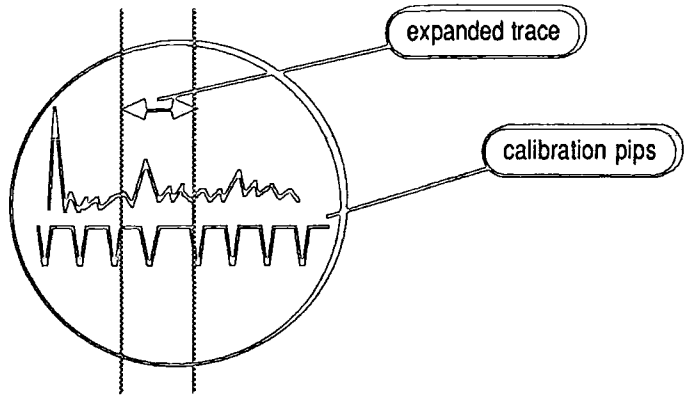


Fig. 1.4 : A-Scope Presentation of Radar Echoes

A network of coastal stations was established to supplement the Chain Home (The Chain Home Low), making use to the new technique to provide more accurate range and bearing data. This invention was introduced to the French in 1939, as their developments in radar were lagging behind at that time. The Germans, however, led the field at this point, having successfully deployed a large variety of ship and aircraft based radars, operating at V.H.F. and providing coverage from 10 nmi (for fire control) to 150 nmi (for early warning) and had it not been for further research in Britain, the outcome of the war would arguably have been different.

1.3.3 The Need for High Frequency Radar

Sir Edward Appleton had already shown the use of pulse techniques in determining the height of the ionosphere as early as 1925 [4], but this made use of pulses of 1ms duration which were highly unsuitable for detection of aircraft as a pulse could only travel 2 miles in 10 μs. To enable accurate detection, pulse lengths of less than 10 μs, combined with high power transmitters and sensitive receivers were essential. In addition, a solution to the problems of ground wave reflection at low altitudes had to be reached. Longer wavelengths incurred a phase change of approx. 180° upon ground reflection, so at low altitudes the ground and target reflected signals were almost identical, causing attenuation of the resultant signal and making detection of low-flying aircraft of ships difficult. With shorter wavelengths this problem diminished and had the added advantage of making portable radar a practical possibility, since an antenna with small dimensions, proportional to the wavelength became a reasonable proposition.

1.4 Radar in the 1940's

During the 1930's all practical radar development had been with high frequency (in the case of the Chain Home) or V.H.F. frequencies but these, as mentioned, incurred penalties of wide beamwidth, narrow bandwidth and high noise levels. Many countries had attempted to use microwaves as a solution but all had given up since valve techniques did not enable the combination of high power and frequency, however in 1939 in Britain, a new valve was developed by Randal and Boot [11] based on a combination of two successful high frequency valves - the klystron and the magnetron. The resulting "cavity magnetron" revolutionised centimetre wavelength techniques as it was capable of producing several hundred kilowatts pulse power. Waveguides were developed to carry the generated signal to the combined transmitting / receiving antenna, which could now have dimensions as small as 1/2 m diameter.

The invention of the cavity magnetron was disclosed to the Americans in the historic "Tizzard Mission" of 1940, solving their immediate problems and opening up the development of large ground-based surveillance radars and small airborne radars, working on the 3cm wavelength. The smaller aerials had the advantage that they could be rotated in both azimuth and elevation giving a complete picture of the search area over a range of heights.

For airborne radars, the possibilities of detecting small objects near to the ground were opened up by centimetre wavelengths, since lower frequency systems suffered greatly from wide beamwidths and reflected ground waves, which made aircraft - aircraft detection impossible if the target was further away than the height of the source, due to swamping by the ground echoes. With shorter wavelengths, smaller antennas could be used and if gyroscopically stabilised and rotated would provide a clear picture of the area surrounding the aircraft, permitting blind bombing raids during the war and introducing techniques such as radar navigation, making use of the differing strengths of ground reflection from land, water and populated areas.

1.4.1 Advances in Display Techniques

These new applications were of particular use in the light of the newly introduced display methods. The A-scope, as used in the Chain Home, provided accurate range determination and could be of limited use in direction finding, but it was not until 1939 that the Plan Position Indicator (P.P.I.) was introduced, although this revolutionary device had been proposed by E.G. Bowen in 1935. The P.P.I. [6] worked on the principle of rotating a linear range scale about its zero at the centre of the c.r.t., synchronised with the rotation of the directive pattern of the antenna and achieved in practice by mechanically turning the magnetic deflection coils of the tube using a synchronised motor. The brightness of the trace

was modulated in proportion to the strength of the received signal, and a map could usually be superimposed above the display giving accurate positional information. The P.P.I. relied on a highly directive receiving antenna, hence its increasing importance as transmission wavelengths fell, with the introduction of the centimetre band. In addition, displays such as the Skiatron were made available, with very long persistence enabling the extrapolation of aircraft tracks on the surface of the screen. They could usually be erased by exposure to either warm air or bright light.

1.4.2 Radar Deployment in the Allied Forces

Many radars were developed during the 1940's, although the German effort was noticibly behind that of the Allies with no equipment working in the centimetric waveband. The Germans cancelled further research in anticipation of a German victory, but when it was realised that this was not to be, their radar work was too far behind to be of use during the war. The Americans developed and installed a large number of radars in their aircraft, mostly operating on UHF and microwave frequencies, and covering a great range of applications. Their largest procurement consisted of 26,000 pieces capable of use in bombing raids as well as airborne intercept and detection of sea targets. They operated at 515MHz and transmitted 2μs pulses at a repitition frequency of 400Hz. Other more specialised radars were developed for greater accuracy over shorter ranges or for long-range early warning.

The Navy and Army also made use of radar during the war, with the Navy using P.P.I. displays to locate targets, then employing precision ranging to establish their distance. The Army, meanwhile, used it for anti-aircraft fire control, requiring a degree of prediction in order to determine the likely position of a target after a certain delay. Trained operators were capable of following aircraft, but obtaining accurate predictions was more difficult. A great deal of research was carried out into automatic lock-and-follow systems, which were capable of rotating an antenna and tracking a target once it had been identified [14].

1.5 Pulsed Radar Principles

The main components of a basic pulsed radar system are illustrated in Figure 1.5, demonstrating the uses of the P.P.I display tube. Each pulse is actually a burst of high-frequency electromagnetic energy, and as such travels at the speed of light, (approx. 1 mile / 10μs). If the time is measured from the start of the transmitted pulse to the reception of an echo, it is therefore possible to calculate the distance of the target from the transmitter. An oscillator produces a synchronising pulse which triggers the transmitter causing it to emit a high frequency signal through the waveguides to the antenna where it is radiated into space. At the same time, the display starts scanning outwards from the centre of the P.P.I. tube. The echo is picked up by the same antenna and guided down to the receiver where it is demodulated and amplified before being used to modulate the intensity of the display beam.

The rotation of the antenna and display trace are synchronised and in land-based radars it is usual to have a north bearing at the head of the P.P.I. display, enabling a direct range and bearing readout of surrounding targets.

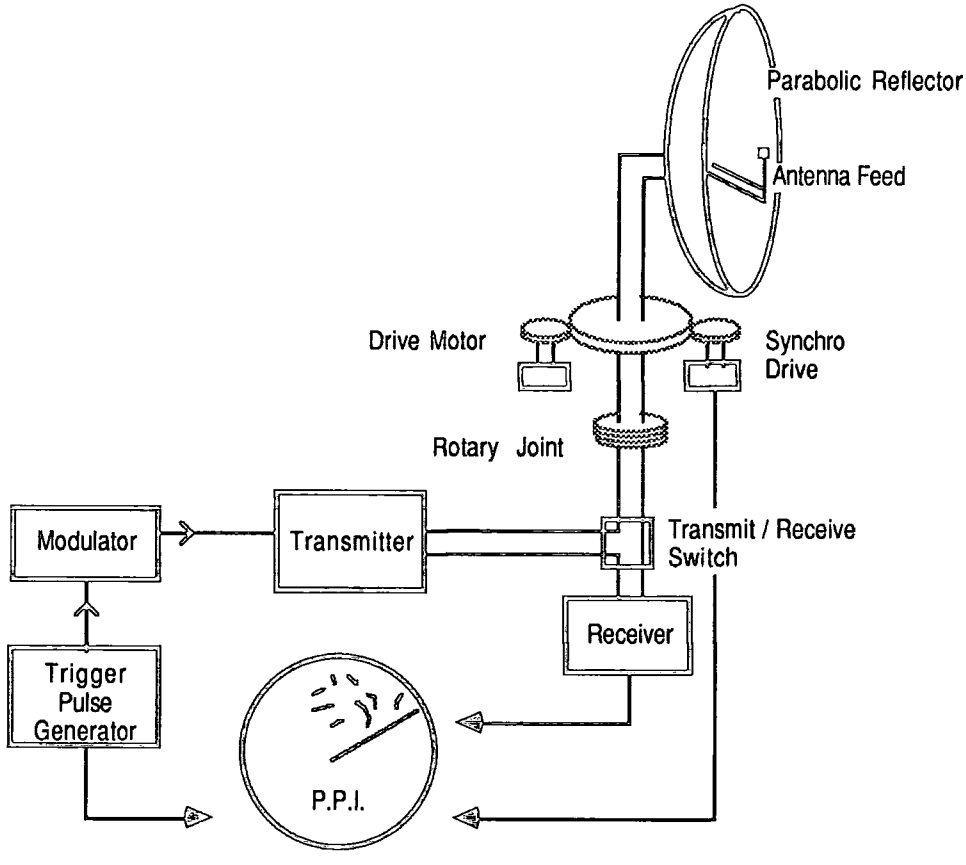


Fig. 1.5 : Pulsed Radar Schematic

1.5.1 Factors Affecting Pulsed Radar Operation

The operation of the general pulse radar can be influenced by a number of factors, which in turn have an effect on the signal processing needs of such a system:-

(i) The maximum range of the radar is partially a function of the pulse repetition frequency, since a single pulse will travel 1 mile in approx. 5μs. To avoid ambiguity between received echoes, the next pulse could not be transmitted until after a time corresponding to the maximum radar range, thus a range of 90 miles would permit a repetition frequency of 1KHz. In practice this figure would be lower to allow for switching between transmit /receive modes.

(ii) The maximum range is also a function of the wavelength used in transmission, since atmospheric attenuation becomes more significant at higher frequencies where the antenna gives the most directive characteristics. A careful balance needs to be sought between beamwidth and range considerations.

(iii) The transmitted pulse width effects the range resolution of the radar since successful distinction of two targets on equal bearing requires a distance equivalent to 0.5 pulse widths separation. A system with 10μs pulses could only resolve targets greater than 1 mile apart. At very short ranges the signal consists mainly of ground returns, requiring some form of attenuation to prevent them from entering the receiver or display stages.

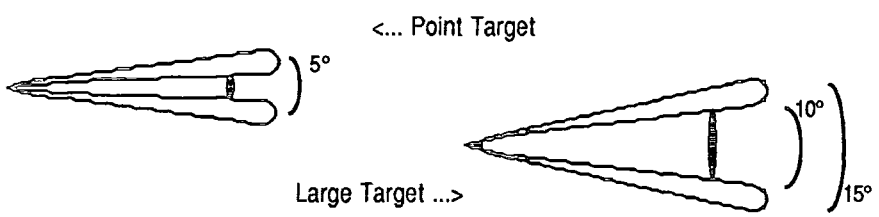


Fig. 1.6 : Bearing Ambiguity due to a Large Target

(iv) The beamwidth affects the azimuth resolution of the radar, and considering point and large targets this is illustrated in **Figure 1.6**. For the point target, only a few pulses are reflected back while it is illuminated by the beam, (note that several pulses are transmitted in one beamwidth), so the position of the target can be accurately identified. The larger target, however, also returns echoes when only partially illuminated by the beam so appearing to extend over a larger bearing. To effectively distinguish between two adjacent targets it is therefore necessary to have a separation of approx. one beamwidth.

(v) The rotation or scanning rate of a radar affects the final data rate of the system. It must be adapted to suit the detection requirements and is particularly important when target tracking is required. Rapid scanning antennae suffer from performance degradation, but greatly ease the process of tracking fast-moving targets whereas a slow-scanning aerial would work best with low velocity or stationary targets.

1.6 Post-War Developments

Owing to the pressures of war, many problems encountered in radar development had to be left until after the war years. Research into microwave radar continued after the war but at a much reduced pace, primary needs being for radars with greater range and better search capabilities in the vertical plane. In addition some means of counteracting enemy radar had to be found. Many early pioneers in the field of radar systems had moved into other areas and a new generation of engineers took over. The MIT Radiation Laboratories in the U.S., which had been responsible for research during the war, published many of its discoveries [20] and led to the introduction of a number of theoretical concepts surrounding radar development.

With the new threat from long-range ballistic missiles and bombers, older techniques in VHF and UHF radar were revived, since lower frequencies gave more reliable detection in the presence of rain and other atmospheric clutter especially over long ranges, and in addition gave better moving target indication (M.T.I.). MTI was initially studied during the war [15] but remained a classified theory until 1954 when signal processing techniques had improved sufficiently to permit its development. It made use of such properties as the Doppler frequency shift given to an echoed pulse by a moving object and is found in almost all modern surveillance radars where it enables targets to be distinguished from surrounding clutter.

Several new transmission methods were introduced, one of the most significant being the phased array radar which was no longer mechanically rotated but could be steered electronically. The transmitted beam was capable of being scanned at will across a large area, either continuously or irregularly, and could be locked onto a particular target for tracking purposes. This was accomplished by transmitting many beams simultaneously, with calculated phase differences between them, which then interfered resulting in an accurately directed beam pattern. By varying the phase differences, the interference pattern and hence the beam could be steered as required. The ability of this type of radar to detect and track space vehicles was demonstrated in the early 1960's when it was used to follow satellites and has been developed considerably since then. Its success relies on another new technology from the 60's, namely that of digital signal processing, which caused a major revolution in system development work which still continues. Almost all modern radar systems rely on digital signal processing at some stage, whether of phased array or other design. It enables a significant reduction in the quantity of unwanted data being presented to the operator, an essential factor in systems with high sensitivity and long range coverage.

1.6.1 Digital Radar Processing

Digital technology permitted the introduction of many new techniques which had been described in the theoretical work of the 1950's but had remained untested because of practical limitations with analogue signal processing. Emerson's M.T.I. theories [21] were a good example of this, since they suggested filtering the received signal using a quartz delay line or using a switched system which sampled the echo at successive range gates. These analogue designs required extra processing and were difficult to put into practice because of the high tolerances needed in the delay lines, but could easily be implemented using digital processing methods.

Development on M.T.I. continues, benefitting from another aspect of the digital processing revolution, namely the introduction of the computer into radar systems design.

1.6.2 Computers in Radar Design

The conventional radar system is considered to operate with fixed parameters, such as transmitting waveform, modulation and direction or receiver sensitivity and clutter control. Clearly, a better system could be designed if the parameters were dynamic, e.g. permitting transmission at higher powers in the direction of targets or reduced power in clutter areas. This could also be adapted to improve reception, such that higher clutter levels could be accomodated in certain directions without disrupting detection performance in other directions.

The phased array radar is a good example of the early use of computing techniques, with the computer being used to control the phase shift on each of the radiators. This has enabled the construction of very large phased arrays such as the Cobra Dane [7], which was installed in the Aleutians in the late 70's for observing and tracking inter-continental ballistic missiles as they re-entered the atmosphere. Many similar systems have been developed such as the Cobra Judy, a shipborne version of the Dane with a diameter of 7m and the UHF Pave Paws featuring two antenna faces each with a diameter of 30m and 2560 active radiators per face. These two systems are designed for ballistic missile detection, with the latter only becoming operational in the past year. Their operation would be impossible without the use of modern computer technology.

Computers may also be used to advantage in mechanically scanned radars, where the rotation rate and direction of the transmission is often fixed, but other parameters are subject to change. Transmission frequency and waveform can be modified to increase the probability of detecting particular objects in differing clutter environments and the receiver and post-detection attributes may too be altered to improve target identification. M.T.I can be implemented on computers by application of Fourier transforms to the received pulses, or by

analysis of filtered target coordinates, in a manner analagous to early British M.T.I. schemes. In considering the implementation of computer-assisted post-detection systems, however, it is important to study the exact nature of the problem. A careful balance usually has to be sought between the hardware and software and extra front-end processing is often needed in order to prevent overloading the data rate of the processor. This limitation is apparent in both large phased array systems and, more obviously, smaller portable surveillance radars. The introduction of computers, therefore, does not provide a complete solution to modern processing problems and hardwired logic and interfacing is likely to be an essential extra for some time to come.

1.6.3 Recent Developments in Surveillance Radar

In recent times the Falklands Conflict has illustrated the need for small surveillance radar systems, which are highly portable and capable of providing early warning of approaching aircraft and long range missiles especially at low altitudes. The problems associated with this type of coverage can be solved with the use of airborne systems, however at the time of the Falklands conflict the British AEW fleet had been disbanded. After the attack on HMS Sheffield it was decided to reintroduce this capability and rapid development work on the Thorn-EMI 'Seasearcher' resulted in a radar installed on a Westland Sea King helicopter which was able to detect long distance, low level attacks from Argentinian aircraft and give warning in time to scramble intercept Harriers. The radar was installed and sent down to the South Atlantic only 3 months after initial procurement decisions, due in part to the simplicity of the final design which comprised of a mechanically rotated antenna, feeding long range data to a monitoring and tracking processor.

Surveillance radar developments in recent years have been influenced considerably by their final usage. The Seasearcher in use in the Falklands was adapted from a similar unit designed for the Nimrod early warning aircraft, with modifications to the signal processing system which removed its capability to detect and track shipping targets.

Ground based radars tend to be the largest in current use, frequenly being phased array installations with coverage in certain sectors only. They often provide an ability to transmit on several different frequencies with varying methods of polarisation, giving optimum detection and immunity from interference.

Maritime surveillance radars tend to be of the mechanically scanned variety, possibly with additional radiators to provide elevation coverage. A recent production model offers a maximum range of 250 miles using 0.7μs pulses transmitted on 1 - 2GHz with a beamwidth of 2.3°. Signal processing techniques for maritime radars are all digital and tend to have the

same general characteristics, regardless of manufacturer or radar type. Suppression of clutter from waves is a major problem on ship based systems and almost all modern signal processors include some form of adaptive clutter cancelling by changing the detection threshold level every scan. Recent innovations include the provision of automatic tracking on maritime radars, this being previously restricted to ground based systems. A typical system might be capable of detecting and tracking up to 20 targets simultaneously and work is in progress in an attempt to increase this figure using faster processing methods. Maritime equipment is frequently restricted in both size and power requirements so compactness and efficiency is an important feature of any new radar system.

1.7 Future Development Areas

Short-term prediction of radar developments is relatively straightforward as much of the current work is scheduled for operation in the next few years. Several significant research fields can be identified and it is expected that these will contribute to future system designs.

- **Mechanically Steered Antennae** — These are still very much in use, in spite of predictions in the 60's that phased array radars would quickly take over. This is most likely to be due to the prohibitively high cost of phased arrays, even after many years of development.

- **Clutter Reduction** — This is still of paramount importance as targets are rarely visible without interference from background clutter. Research into both pre and post detection methods continues, the latter being of particular interest as automatic tracking computers are introduced as they can easily be overloaded by an excess of falsely identified targets. Post-detection clutter control always reduces the sensitivity of the radar so care is needed to avoid losing wanted targets.

- **Signal Processing** — Digital technology has only been widely available for the last 15 years and semiconductor integration methods are constantly advancing. Systems designed in the late 70's with small and medium scale integrated components are now outdated and work is in progress to redesign these using current VLSI devices and new generations of programmable components in an attempt to improve efficiency, performance and reliability.

- **Operator Intervention** — This is likely to be reduced in future systems, in terms of both maintenance and control functions. Automatic detection and tracking of targets is available on many of the larger installations and is being developed to a point where no operator will be required to initiate target tracking, even on small portable systems.

- **Improved Target Data** — Present systems generally provide target detection details as output, although other information can frequently be calculated, such as physical dimensions of the target. New radars are likely to provide additional data to enable precise identification, possibly construction or surface information.

2.0 Introduction

Surveillance radar systems have traditionally relied upon human interpretation of a signal which is displayed on a cathode-ray tube, however there are many problems associated with this approach and operators have required considerable training to enable them to recognise all forms of target against a variety of backgrounds. Additionally the speed of modern aircraft (in particular) calls for rapid reactions if a defensive roll is to be adopted. To overcome these problems and to assist the operator, systems have evolved which extract information from the incoming signal and re-display it in a more readily identifiable format.

A typical modern radar system may therefore be regarded as a form of communications device, receiving a large amount of information, only part of which is required by the user. The signal processing involved in such a system is complex, since the desired portions of the incoming signal exist alongside reflections from unwanted targets and noise from transmitter and receiver circuits, and the design of a suitable signal processor is therefore of great importance in the overall radar system.

2.1 The Requirement for Signal Processing

The early radar systems, in use during the Second World War, relied entirely on manual operation, combining tracking information from the operators with reports from visual observers. All confirmed movements were then displayed on large plotting tables. As aircraft performance improved it became necessary to reduce the time taken to correlate radar track information and display the resulting data. In the 1950s', analogue computers enabled such features as electronic markers to be introduced in plot display, giving a means of calculating target position directly from the P.P.I display. This could be accurate to as much as ±0.5%, but the equipment had to be regularly adjusted to maintain this figure. With the introduction of digital techniques in the 1960s', it was therefore a natural decision to transfer the analogue signal processing developments to the new technology, thus removing the problems of operational drift and enhancing the accuracy of the system.

Digital technology brings with it the power of rapid signal processing, and attempts have been made to completely automate radar track extraction, however the process is never 100% reliable and so the operator remains, not in a detecting and tracking capacity, but as a monitor of the automatic process, adjusting system parameters to optimise performance and overriding false tracking decisions.

2.1.1 Signal Processing: Terms of Reference

In this thesis it is intended to consider the general purpose surveillance radar, such as shown earlier in **Figure 1.5**, as the source of the returned signals although many of the techniques to be described can be applied to other radars. It should be noted, however, that the distinguishing feature of this type of radar is the fact that it rotates continuously and therefore all areas of coverage are illuminated for an equal period of time. As a consequence, the time available to detect and process targets in one area is equal to that available in any other area. Continuous high speed processing is therefore a prime requirement for this type of system.

Taking this type of radar system, the receiver circuits may be subdivided into a number of sections as in **Figure 2.1**. In this example, the received echo is mixed, amplified and then demodulated in an envelope detector to produce a signal which, in early radar systems, would be used to directly modulate the Z-axis of a P.P.I. display. The brightness of the trace on the cathode-ray tube would give an indication of the target's size or reflecting area, but this method of display required trained operators, who were able to resolve targets from unwanted reflections (**clutter**) which could often be visible at comparable brightness.

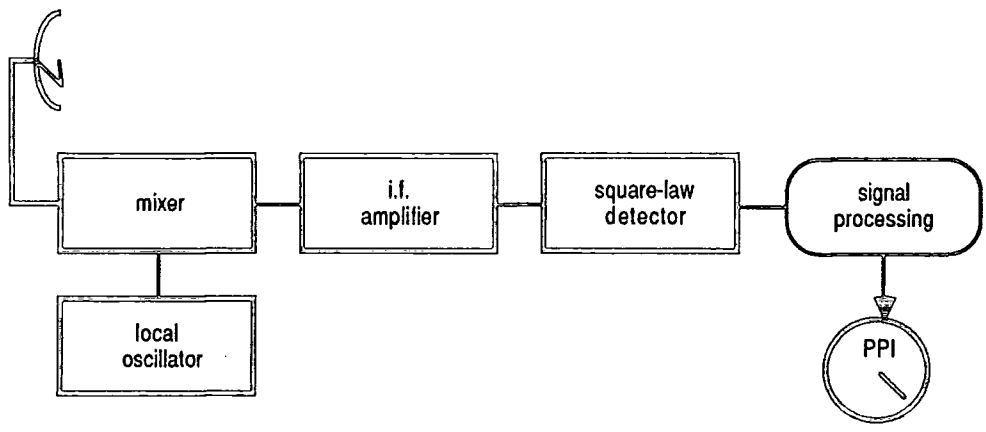


Fig. 2.1 : Typical Receiver Configuration

The exact definition of "unwanted reflections" is clearly dependent on the use to which the radar is put, but for a ship-based radar, these could be defined as land-masses, buoys, rain, birds (or angels) or intermittent echoing objects such as waves.

The signal processing section, present in modern radar systems, attempts to assist the operator in his identification of targets, whether moving or fixed, by careful filtering of the unwanted clutter from the incoming echo signal, without reducing the visibility of genuine targets. The reduction in the number of false tracks is particularly important in a military environment, where time and resources can not be wasted on reacting to non-existent targets.

2.1.2 Characteristics of a Post-Detection Signal

In describing the operation of typical signal processing techniques, it is important to understand the characteristics of the echo signal received by the aerial, and it is proposed to discuss these in conjunction with other synchronisation data which is passed on to the post-detector processing.

It should already be appreciated that the basic surveillance radar functions by transmitting a regular train of pulses which are modulated onto a high frequency carrier signal. These are radiated from their source in the form of a narrow beam, which is swept through 360° by way of the continuous rotation of the antenna. A pulse travels out along the beam at a known velocity and (ideally) is reflected back only when it encounters some object in its path.

The time interval between transmitted pulses is known as the **pulse repetition interval** (p.r.i.) and its inverse (1/p.r.i.) is the **pulse repetition frequency** (p.r.f.). Values for the p.r.f. are linked to the required range and may be typically 768Hz (30-90 miles) or 256Hz (250 miles). If the frequency is too high for the desired range, then ambiguities result from long-range echoes due to one sweep being mistaken for short-range ones due to the successive sweep. The transmitter generates a synchronisation signal as each pulse is transmitted, to enable accurate measurement of the echo time.

In assessing the p.r.f., an additional parameter should be considered - the **rotation rate** of the antenna. This is directly linked to the desired range and should enable estimation of the number of pulses which are likely to be reflected from a target at a given range. Typical rotation rates are 24 r.p.m. (30 mile range), 15 r.p.m. (90 mile) and 7.5 r.p.m. (150+ miles).

The length of the transmitted pulse is designed around the targets to be detected, but an important feature is that each pulse carries the full energy of the transmitter and hence determines the strength of the echo that may be returned. Although a longer pulse will enable a greater energy to be conveyed, this would be at the expense of the receiver's selectivity since neighbouring targets may only result in a single echo. The solution achieved by transmitter designers is to generate a 'wide' pulse then compress it, prior to transmission, into a single high-energy impulse. Using compression techniques has enabled long range (250 mile) radars to be designed which still produce an acceptable echo at the limit of their range. Typical values for pulse length (τ) are in the range 0.25 - 2μs.

The width of the transmitted beam is ideally as narrow as possible, to enable a more selective response to echoes reflected back from a target and to counteract the effects of beam

spreading at long ranges. The disadvantages of a narrow beamwidth are the increased effects of the side-lobes, which result in the signal being transmitted in directions other than the main beam. With typical values of beamwidth being around 1 - 2.5°, it should be clear that a p.r.f. of 768Hz and rotation rate of 15 r.p.m. gives rise to between 9 and 20 pulses illuminating a single beamwidth and a large number of echoes can therefore be expected from a single target.
These parameters are summarised for the radar types discussed in this thesis in Table 2.1.

Type	P.R.F.	Pulse Length	Effective Range	RATES Range	Beamwidth	Rotation Rate
A	768 Hz	2 µs	96	90	1.2°	15rpm
B	768 Hz	.5 µs	30	42	2°	24rpm
C	700 - 1400 Hz	.25 µs	30	42	1°	24rpm
D	256 Hz				2.3°	7.5rpm

Table 2.1 : Summary of Radar Parameters

The signal received by the antenna will be due to a pulse originally transmitted, say, at time t=0, which has been reflected back by a target encountered at t_1 and reaches the receiver at time t_2, where $t_2 = 2 * t_1$. Since the pulse is of electromagtetic nature, it is possible to calculate the speed of the pulse, and hence determine the distance from the aerial to the target.

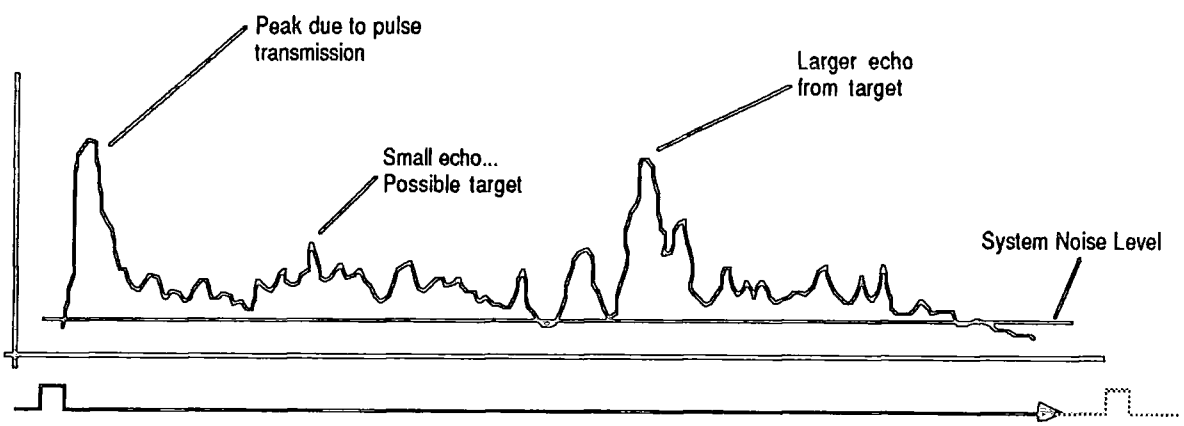


Fig. 2.2 : Characteristics of the Received Signal

A typical received signal will have the characteristics shown in Figure 2.2. It consists largely of a permanent offset noise, which may be generated by the receiving circuits or thermal effects, and a clutter level which is due to reflections from side-lobes, ground effects,

waves, birds, precipitation and weather effects. Occasionally there may be genuine targets amongst the clutter, which usually show as peaks in the background clutter level - the purpose of signal processing is to distinguish these from the permanent background clutter.

2.1.3 Sources of Clutter

The exact definition of clutter is based upon the situation in which the radar is used: To a weather radar designer, both aircraft and land would be regarded as clutter, but not the effects of cloud or precipitation. In this thesis it is intended to discuss the application of signal processing to a maritime surveillance radar and in this context the desired targets are:

- AIRCRAFT
- SHIPPING

The unwanted targets, (and hence clutter) can be identified as :

- Wave reflections
- Land masses
- Local echoes
- Precipitation
- System and thermal noise
- Buoys
- Angels
- Clouds
- Radio Frequency Interference

Study of these clutter types will reveal some constant or near-constant signals, (system and thermal noise, land-masses, buoys) and a number of moving noise sources. An ideal signal processor will therefore have facilities for removing or supressing clutter from all of these sources, while simultaneously maintaining sufficient sensitivity to detect any genuine targets.

The clutter types may be further subdivided according to the spread of reflecting surfaces within a group. The term **homogenous** is used to describe clutter which exhibits the same reflective characteristics, irrespective of position within a clutter area. **Non-homogenous** clutter may reflect differently at varying points in its area of coverage and the term **Point Clutter** is introduced to describe clutter over an area which is too small to be analysed in sections. Analysis of these three types of clutter leads to the following subdivisions:

- Homogenous clutter : wave reflections, precipitation, clouds.
- Non-homogenous clutter : echoes from land masses,
- Point clutter : angels, buoys, discrete targets.

2.2 Detection of Targets in Clutter

The designer of any radar system is required to maximise the signal/clutter ratio (whatever form the clutter takes), to achieve the best possible results from the system. This cannot be achieved, however, by simply raising the power of the transmitter, since this only serves to increase the strength of *all* returned echoes including those from unwanted targets. In practice, the clutter rejection ability of a system is determined by the transmitted signal parameters and the design of the signal processor.

2.2.1 Optimisation of Transmission Parameters

Optimisation of the transmitted signal is carried out by matching the range/angle resolution cell to the size of the target to be detected. (**Figure 2.3**). The depth of the cell corresponds to the depth of field covered by the transmitted pulse at its given frequency, and for general detection is set between 0.1 - 1μs. Pulses of less than 0.1μs duration result in break-up of the target over many cells. The width of the cell is determined by the beamwidth and rotation rate of the antenna, although beamwidths of less than 1° are difficult to obtain due to antenna size requirements. Transmission frequency is carefully chosen as frequencies above 1GHz tend to result in a radar susceptible to interference from precipitation, wind-sheer or turbulence. In many radar systems, the transmitter parameters are already fixed and the onus of reliable target detection lies with the signal processing section.

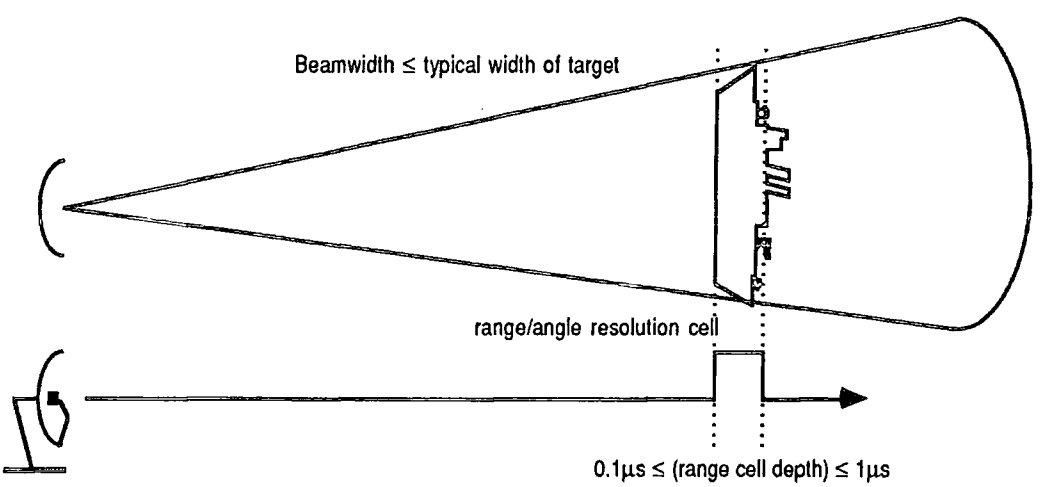


Fig. 2.3 : Optimisation of the Transmitted Signal

2.2.2 Detection of Targets in the Received Signal

The simplest form of target discrimination is between a moving and a fixed target, since the principle of Doppler Shift (e.g. observed as the change in the sound of a moving object

passing a stationary observer) may be applied to the received radar signal and used to velocity-discriminate the echoes. This process of Moving Target Indication, (M.T.I.), is frequently used, having the advantage that it may precede a standard signal processing section as a replacement for the conventional detector.Where more selective discrimination is required, the subdivisions of clutter types may be used to establish standard techniques for the filtering of unwanted data.

M.T.I. can often be of use in early processing stages particularly where filtering of point clutter is important, although its complexity may outweigh its advantages, especially as other techniques exist for filtering moving targets. The basic principle of MTI, however, is given below although it is proposed to discuss some other methods in greater detail.

2.2.3 Moving Target Indication

A large proportion of the clutter environment may be identified as stationary, consisting of targets such as land-masses, buildings, ships at anchor and navigation buoys. If some form of motion detector were introduced into the receiver it would be possible to remove all occurrences of these unwanted targets by suitable filtering. The easiest method of detection is to make use of the Doppler frequency shift given to a transmitted pulse by a moving target, requiring some means of comparing the reflected signal frequency and phase with the transmitted signal. This process relies on the fact that a target will usually reflect several pulses while illuminated by the antenna beam, enabling an indication within one scan.

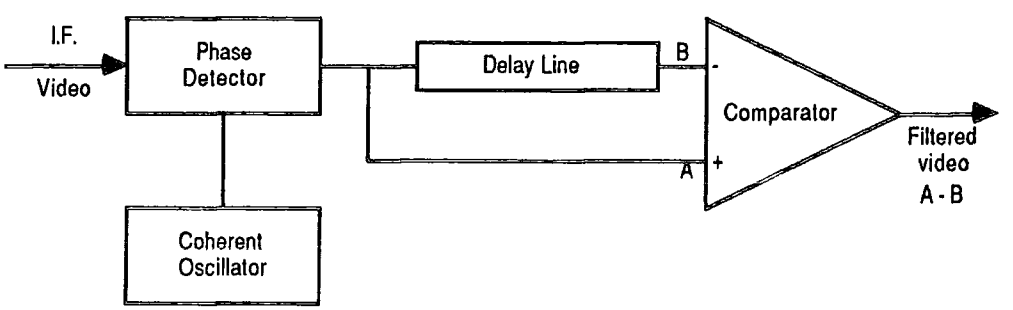


Fig. 2.4 : M.T.I. Operating Principles

The basic principle of M.T.I. detection is illustrated in **Figure 2.4**, the oscillator generating a signal in phase with the one transmitted. The phase detector compares the phase of the incoming intermediate frequency signal with the generated signal and produces a pulse output which varies in amplitude according to the detected Doppler shift. A stationary target results in zero Doppler shift, so consecutive pulses would be of equal amplitude. The comparator declares a target indication if the magnitude of the incoming signal is different from that produced by the previous range sweep.

In early M.T.I systems the delay line would be implemented in analogue form, using a water or mercury-filled tube, but later techniques permitted the construction of quartz delay lines. With the advent of digital technology, and semiconductor devices in particular, it is now possible to use charge-coupled devices for analogue delay lines or shift-registers for a digital delay. The M.T.I. filter can be made more selective by increasing the number of stages, thus enabling certain velocity ranges to be accurately detected whilst blocking unwanted speeds. Limited filtering of angels or wave targets is possible by selective M.T.I. filtering, but difficulties arise in increasing the number of M.T.I stages as the delay lines have to be more accurately constructed. This problem is solved to some extent in digital systems, since the delay can be completely controlled by its clock frequency and does not tend to drift with age as is the case with analogue components.

Although M.T.I. provides useful stationary target filtering, it suffers from a number of disadvantages. The frequency range that can be used to define a target's speed must be less than the pulse repetition frequency otherwise the radar is said to be ambiguous in Doppler, resulting in certain 'blind' speeds in the radar's detection capability where the Doppler shift is an exact multiple of the repetition frequency. Also, the transmitter frequency must not drift as this type of coherent processing would be impossible with an unstable frequency source. In these situations an M.T.I. filter would have to be replaced with an envelope detector, and some other method of target processing introduced.

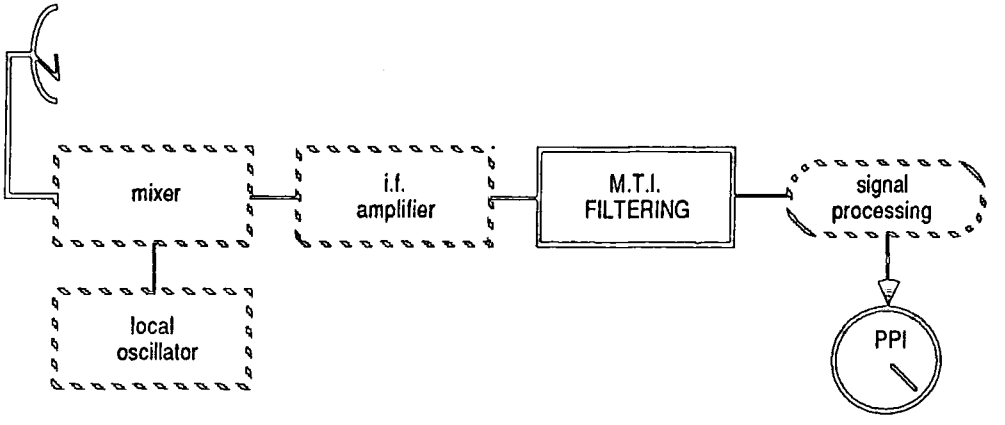


Fig. 2.5 : Inclusion of an M.T.I Filter

The inclusion of a typical M.T.I. filter is illustrated in **Figure 2.5**, where it should be clear that this circuit replaces the receiver's square-law detector, although its output conveys the same positional information with respect to time. The signal amplitude is an indication of the Doppler shift introduced by the target's motion and hence a measure of its speed, rather than its reflecting area or proximity. In considering this, it would not be unreasonable to perform some technique of amplitude-based filtering on either type of signal, although the exact implications of the results would necessarily be different and an ideal signal processor will therefore accept data from either source, taking advantage of the features offered by each.

2.2.4 Alternatives to M.T.I. Processing

In certain situations, the Moving Target Indicator is either insufficient or cannot be adapted to handle the returned echo signal and in these instances some other form of processing is necessary to remove the clutter and identify the genuine targets. Under these conditions, processing should be regarded as occuring post-detection and it may be subdivided into distinct units, each performing some form of filtering on the incoming data. Additional processing may be provided at this stage to offer extra facilities to assist the operator such as automatic tracking or I.F.F. identification decoding.

Whilst early systems handled all processing using analogue techniques, the advantages offered by new technology have resulted in current signal processors being almost entirely digital, and it is this type of design that will be considered in the coming chapters.

2.2.5 Digital Signal Processing

The first stage of any modern radar signal processing system, following the detector/M.T.I., will typically include amplification and/or digitisation of the detected video signal, since use of digital processing gives advantages of high data throughput with no drifting and an accuracy which may be set to suit the requirements of the application.

The digitisation process involves some form of quantisation of the signal into discrete amplitude levels and time periods, and may be regarded as a series of samples at different ranges: the distance between one sample and the next forms a **range cell.** The size of the range cell, and hence the digitisation rate, is another important factor in the performance of a processing system, being ideally related to the transmitted pulse width. If the rate is too high, then an echo will cover too many range cells, whilst a low rate will result in the loss of targets due to poor selectivity.

The quantisation of the signal amplitude introduces some noise into the system, as a result of the error between the true analogue input and its digital representation, however the analogue signal is usually amplified to such a level that the system noise component exactly corresponds to the least significant bit of the result, with the effect that weak signals are not lost in the digitisation process. The accuracy and dynamic range of the result is dependent on the number of bits used to represent it and can therefore be easily improved upon by increasing this parameter, subject to the constraints of background noise levels.

Once the signal has been digitised, a number of processing options are open to the designer and it is proposed to discuss those relevant to the RATES experimental radar system in detail. In particular, various methods of clutter reduction will be discussed together with suggestions for their implementation within a practical design. For details of the implementation of the prototype system, the reader is referred to **Chapter 3** and ideas for future consideration are discussed in **Chapters 4 & 5.**

2.3 Filtering Techniques in Homogenous Clutter

Study of Figure 2.2 should reveal the two separate 'noise' components which were described in the accompanying section. The system noise is a permanent offset to the remainder of the signal, fluctuating slowly with variations in atmospheric conditions, and could be removed by straightforward application of a fixed threshold to the input signal. The more dominant clutter level varies considerably and it may be regarded, as discussed previously, in terms of the nature of its components.

Homogenous clutter, as typified by rain, chaff or sea reflections, tends to be evenly dispersed over 1 mile or more in range and would therefore be covered by as many as 45 range cells using the parameters in Table 2.1. With no sharp edges to this type of clutter region, the simplest method of extracting targets (which are visible as peaks in the surrounding mean level) is to establish a value for the *local* mean clutter level at all range cells, and then to compare the received video signal amplitude against this. Successful calculation of a mean, which adapts to changing conditions without loss of selectivity or sensitivity is one of the major challenges to the radar designer, as loss of genuine targets due to clutter could render the system useless.

2.3.1 Cell Averaging

Although application of a fixed threshold is sufficient to counteract thermal and system noise components, the use of the same technique to filter 'moving' clutter would result in either the loss of many targets due to a high threshold or ineffective filtering due to a low threshold. An ideal solution would involve some means of adapting the threshold level to surrounding conditions, thus enabling the detection of small targets in low clutter environments whilst masking the effects of areas of high clutter level.

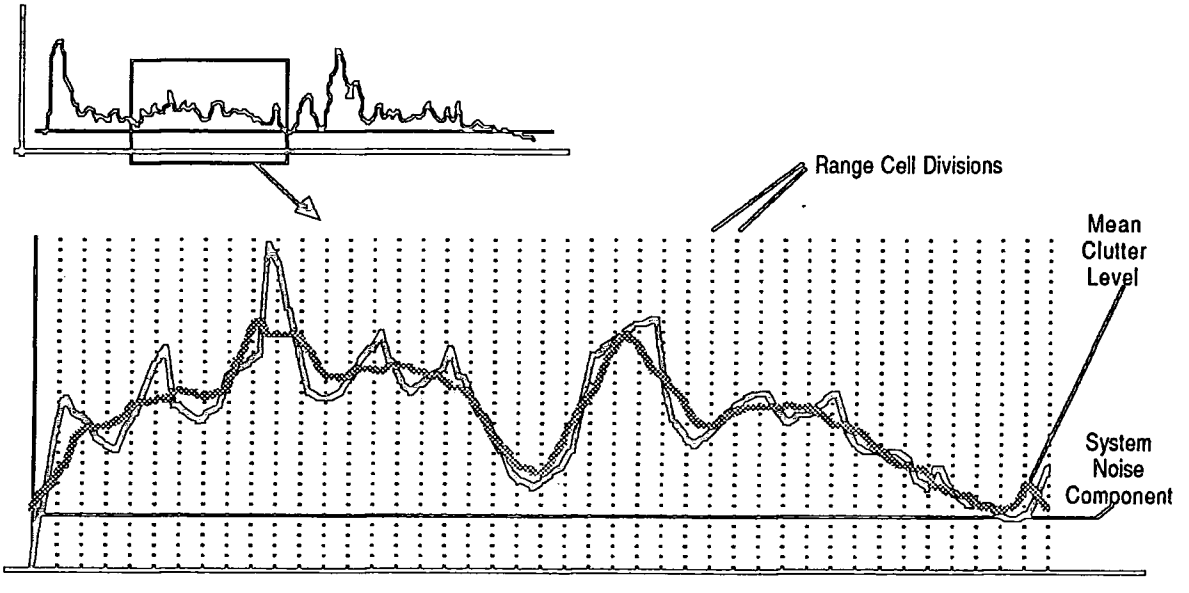


Fig. 2.6 : Expanded Portion of an Echo Signal

Consider Figure 2.6, which shows an expanded portion of Figure 2.2 and takes account of the division of the incoming echo signal into range cells : If, for every cell from the left hand side to the right hand side of the diagram, the average level of (say) the three cells on either side of it is calculated, the result would approximate to the line shown as the Mean Clutter Level. Through a binary comparison of the level in the cell of interest, with the calculated average, the peaks and troughs of the background clutter are reduced to the 14 target declarations shown in Figure 2.7a. In this diagram, the clutter level is shown as a heavy line against the original signal amplitude, and the resultant binary comparison shown at the foot of the graph.

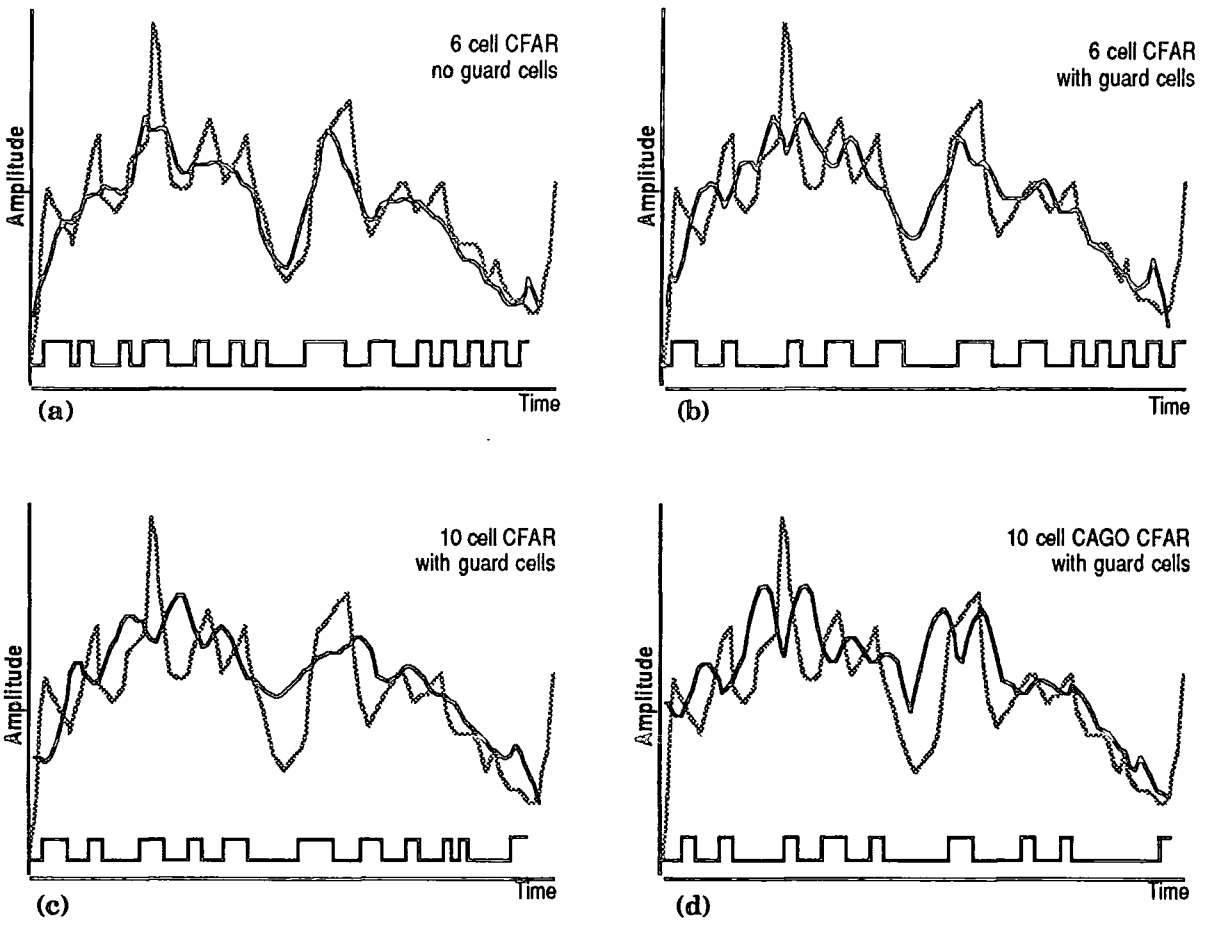


Fig. 2.7 : Relative Performance of Typical CFAR Processors
(Solid line :- Mean Clutter Level)

Although clearly not perfect, the efficiency of this method of clutter reduction has led to its adoption as the basis of most clutter processors. It is usually referred to as a "sliding-window detector" and its use always involves some means of averaging the signal level over a particular area surrounding each range cell in order to derive a value for the corresponding threshold.

2.3.2 The Sliding-Window Detector

The objective of the sliding-window detector is to reduce the probability of a a clutter 'peak' being accepted as a genuine target, down to a manageable (or zero) level at which point further processing circuits can take over to remove remaining unwanted targets. For this reason, the detector forms part of a **Constant False Alarm Rate** processor (or **CFAR**), and the comparison of each cell's amplitude with a mean clutter level forms a **First Threshold Detector.**

Much work has been carried out on the theoretical design aspects of CFAR processors based on mathematical models for targets and clutter situations, with particular attention paid to the probability of a false detection occurring or the probability of a missed detection. These are well documented in other sources [23, 36, 37, 38], so it is not proposed to discuss these theoretical aspects within the scope of this document, which will instead concentrate on aspects of their practical implementation. In practice, the detectors to be discussed exhibit a probability of missed or incorrect detection in the region of 10^{-6}, which equates to approximately 12 / sweep.

In its simplest form, the first threshold detector is entirely range based, with no interaction between detections on successive azimuth bearings. For every range cell on a given bearing, a window may be considered to exist covering a group of cells on either side of a particular cell of interest (**Figure 2.8**). Within this window, the video level in the cells is averaged to provide a mean value, hence this is often referred to as a **Cell-Averaging (CA) CFAR.** To give an improved performance in situations where a target is located on the boundary between cells, it is common practice to ignore the two cells immediately adjacent to the central one when calculating the threshold level. In the example quoted, this modification to the design would result in the mean level shown in **Figure 2.7b**, with the number of binary 'hits' falling from 14 to 12, without loss of significant target data.

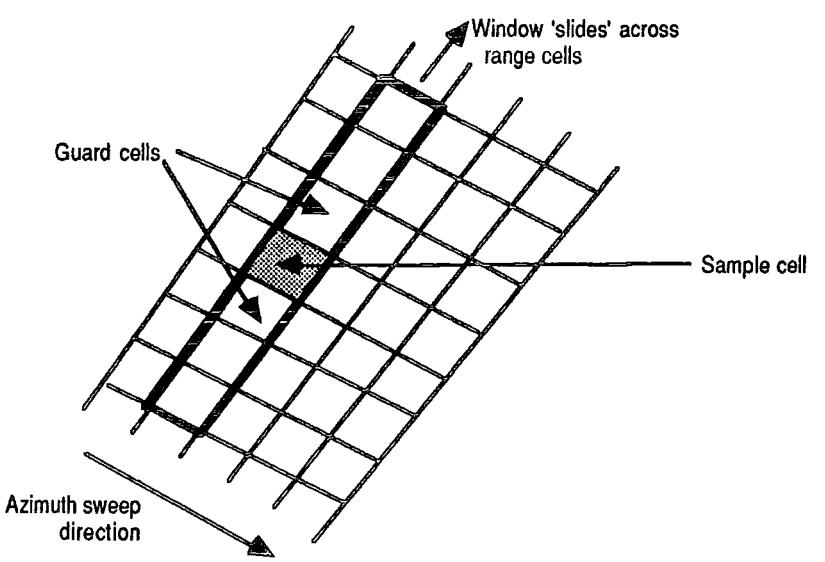


Fig. 2.8 : The 'Sliding Window' Detector

The previous two examples have involved a window of 3 active cells on either side of the cell of interest. Practical implementation of this design would be relatively easy, using adders to evaluate the sum and PROMs or shift-registers to produce the mean, however the performance of the detector with this window size tends to result in a high rate of missed detections as it responds very quickly to changes in the original signal. The result of increasing the window size to 5 cells (plus guard cell) on either side of the cell of interest is shown in **Figure 2.7c**: the mean level changes more slowly over fluctuations in the received signal amplitude, whilst short peaks in the signal, such as may be caused by targets, tend not to influence the mean, thus increasing the likelihood of a detection without compromising the clutter-rejection capabilities of the design. If a larger number of cells is used to establish the mean, the resulting threshold is found to follow the clutter-level more accurately, but at the expense of performance in situations of closely adjacent or smaller targets.

The cell averaging process discussed so far has assumed that clutter is distributed uniformly within the area enclosed by the window, and its performance is therefore degraded by any clutter edges which may occur in this region. A further enhancement to this process takes account of more realistic situations where boundaries might exist, such as at the edge of a coastline or weather clutter, and this has been evaluated successfully by other sources [38, 39]. The technique involves taking the greater of the two averages produced either side of the test cell, then using this as the basis for the detection threshold, and for this reason it is known as a **Cell Averaging, Greatest-Of** (or CAGO) **CFAR**. Its performance using the test data is shown in **Figure 2.7d**, where it can clearly be seen that the number of false target declarations has been reduced still further by the improved handling of clutter edges. Its performance in regions of uniform clutter is not so good as the CA CFAR, but it has been shown [39] that this leads to an decrease of only 0.3dB in the signal-to-noise ratio of the CFAR output.

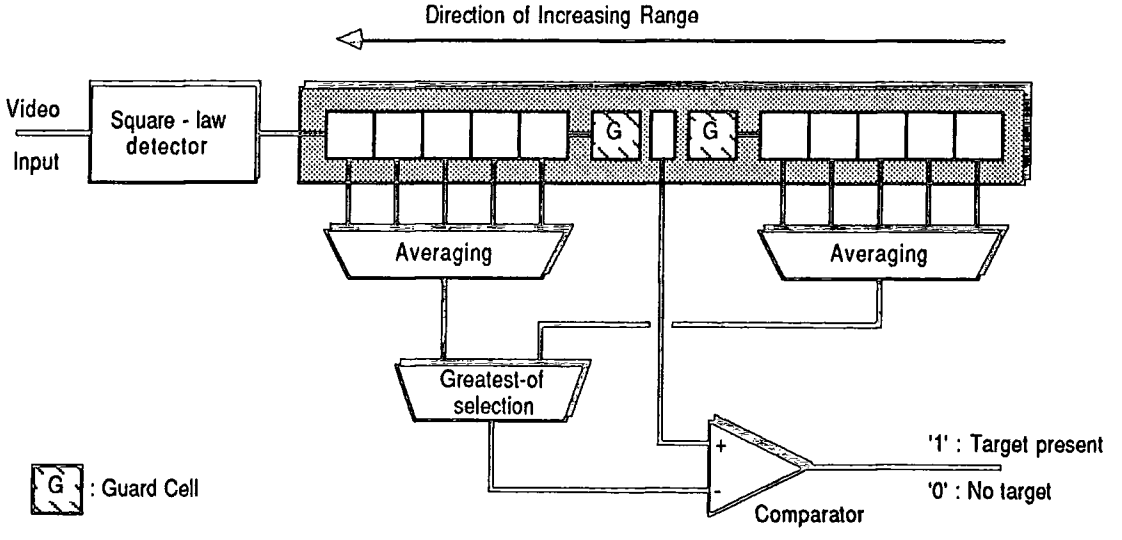


Fig. 2.9 : Implementation of a CAGO CFAR

The expressions for the mean levels (Z_{ca} & Z_{cago}) of the CA and CAGO CFARs are given below. In this case, the cell averager comprises 'N' cells, each of amplitude X_i, with 'M' cells on either side of the guard cells contributing to the mean.

$$Z_{ca} = \frac{1}{N} \cdot \sum_{I=1}^{N} X_I \qquad Z_{cago} = \frac{1}{M} \cdot MAX\left(\sum_{I=1}^{M} X_I , \sum_{J=M+1}^{N} X_J \right)$$

A practical application of a CAGO CFAR processor is described in Chapter 3, but typical implementation of such a device, using a window size of 5 cells, is illustrated in Figure 2.9. If the input signal was not digitised, the shift register could be realised using analogue techniques and the final output would still be digitised to single-bit accuracy, however the circuit would be difficult to design due to drifting and problems with balancing the delay lines. Alternatively the signal could be digitised before input to a digital shift-register, solving balancing problems but requiring some means of summing the outputs from each stage of the register. In practice, the signal output from the CFAR averager is rarely used for direct comparison with the test video, and is more often combined with manual offsets (to account for system and thermal noise) and computer-generated offsets which compensate for other types of clutter (Section 2.3.3).

2.3.3 Non-Homogenous Clutter Cancellation

The cell averaging techniques discussed so far can be applied with success to regions of homogenous clutter which by definition exhibit an even distribution of clutter levels with no sharp boundaries. Land masses, however, can produce strong echoes without being evenly distributed and can often lead to sharp transitions, say between a sea and land clutter boundary. Elimination of this type of clutter requires some knowledge of the movements of strongly echoing targets, which in turn can be fed back into the clutter rejection circuits.

In such a situation, an additional weighting factor is applied to the output from the threshold averaging circuits, before it is compared with the test video level. The weighting is typically derived from the tracking computers, which collect details of all plots which have not been rejected by the CFAR processors and group them according to the region they occur in. Any plots which do not move or show a low consistency count (i.e. do not correlate from scan to scan) are regarded as clutter and can therefore be filtered out, an action which is performed by raising the threshold level in the regions where the clutter count is high.

In practice, it is neither possible nor desirable to provide a computer-generated threshold value for every input range cell, so the radar's area of coverage is divided into a 'map' of clutter regions which are updated periodically by the tracking computers (Figure

2.10). The azimuth resolution of each cell approximates to that of the antenna beam, so as to maintain an effective area of coverage.

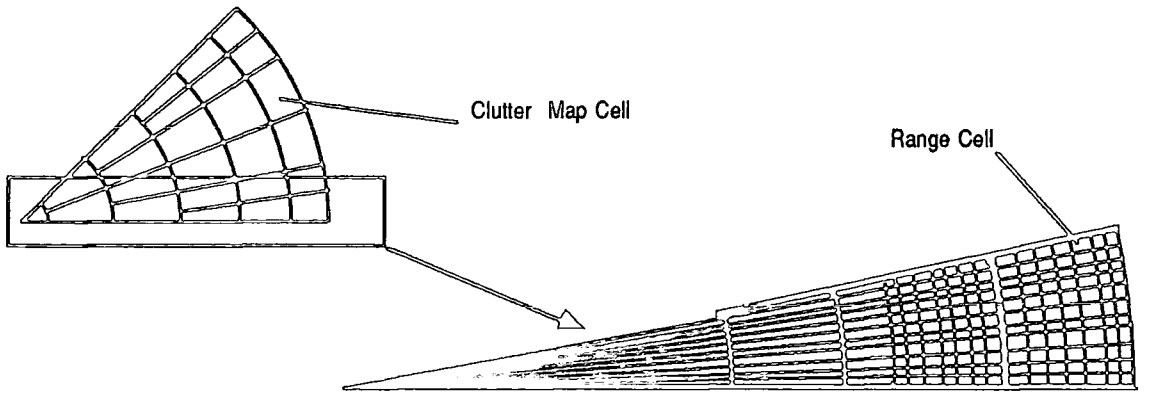


Fig. 2.10 : Implementation of a Clutter Map
(Showing typical relationship between clutter map and range cell sizes)

The map is typically held as part of the first threshold processor and is addressed by the current azimuth and range values. The output is scaled by a pre-determined constant and then added to the mean threshold to form a value which can be compared with the test video level. It should be noted that the clutter map is often arranged so that the area covered by each cell is approximately constant, allowing for the spread of the beam at greater ranges. In operation, the use of a clutter map can be shown to be particularly effective against weather-clutter edges or static clutter against a fixed level background (e.g. buildings against land clutter).

Typical implementation of a complete CAGO CFAR with clutter map is shown in Figure 2.11, below.

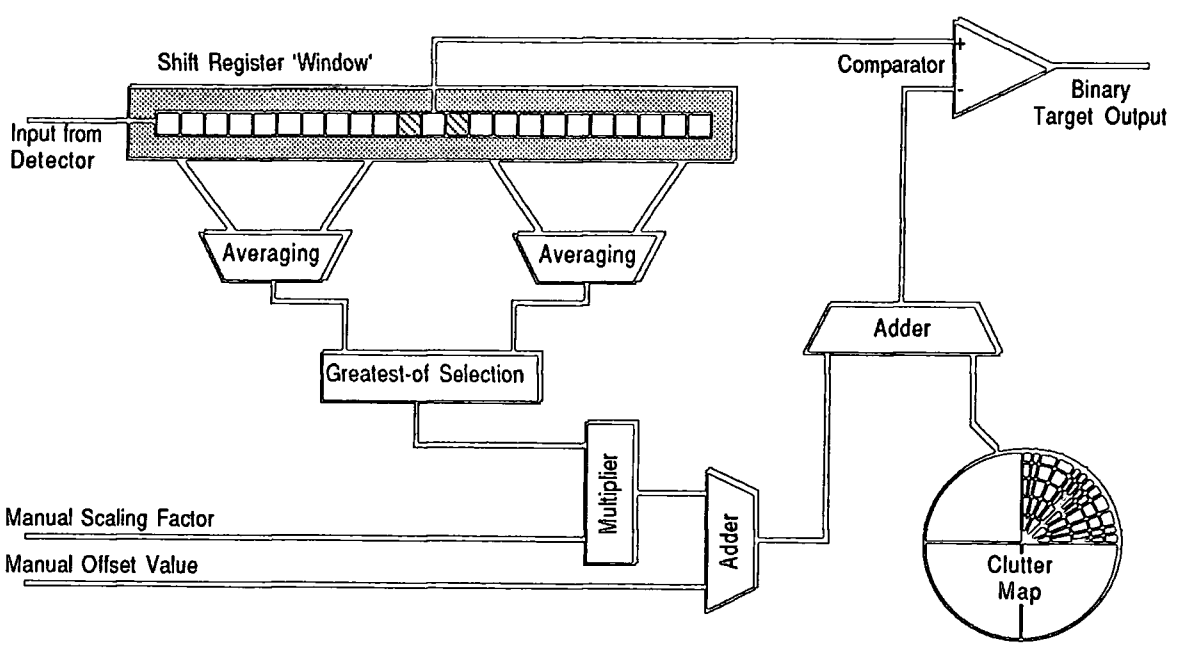


Fig. 2.11 : Complete CAGO CFAR Schematic

2.3.4 Alternatives to the Basic Cell Averaging Processor

The performance of the CA and CAGO processors in conditions of homogenous clutter has been shown to be adequate for most requirements, however these conditions rarely exist on their own in a practical situation. There is therefore a need for some alternative or enhancement to the basic CFAR, to enable it to process non-homogenous and point clutter. The map described above is capable of reacting to fixed clutter, but is too slow to react to sharp edges, such as occur in multiple target situations, and does not adapt readily to the type of clutter that may be caused by bird flocks.

2.3.4.1 Area Thresholding

This method of clutter supression has proved highly effective in dealing with bird flocks and relies on the differences in amplitude distribution between clutter and desired targets (such as aircraft or shipping). It is more suitable for application to post-M.T.I. processing, but can equally be used following a square-law detector.

An area map is established, similar in size to the clutter-map described previously, and the smallest genuine target in each cell is chosen as a reference. A detection threshold is applied to filter the low-level clutter and then two counts are accumulated - one comprising those detections which exceed the reference amplitude and the other, those which exceed the threshold but not the reference level. If too many small targets are detected the threshold is raised slightly and it is lowered if there are too few, thus keeping the number of small target detections low enough to avoid overloading the tracking computers. The disadvantage of this technique is that it takes some time to establish an adequate threshold and the computation is such that it requires a high-speed computer to evaluate it.

2.3.4.2 Rank Order CFAR Processors

Range-based rank-ordered CFAR processors are intended as an improvement on the traditional cell-averaging CFARs, especially in situations of multiple targets or sharp clutter edges, and they require less computational time than the area-thresholding processors described in Section 2.3.4.1.. Much work has been carried out by Rohling [40,41], comparing their efficiency against CA and CAGO types, with the conclusion that a rank-based CFAR allows the use of a much larger window whilst retaining the rapid response associated with a much smaller window size. The function of a rank-order CFAR processor is as follows:

Consider a threshold window of size 'N' cells (excluding the test cell), with video level in each cell, given by X_i (i=1...N). The individual values of X_i are firstly sorted to give the sequence:

$$X_1 \leq X_2 \leq X_3 \leq \ldots\ldots \leq X_N$$

Rohling terms this resulting sequence an Ordered Statistic, and suggests that a value X_k is chosen, where $k \in [1,2,3...N]$, and this is used as the basis of the threshold level. The parameter k is dependent only on the size of the window (N) and practice suggests a value approximately three-quarters of N. Hence a rank-order CFAR can be described by:

$$Z_{os} = X_k \qquad k = 3N/4$$

Using the example given earlier, Figure 2.12 gives a comparison between a 10-cell CAGO and 10-cell rank CFAR, illustrating the more rapid response to threshold peaks that is achieved with the rank-based processor.

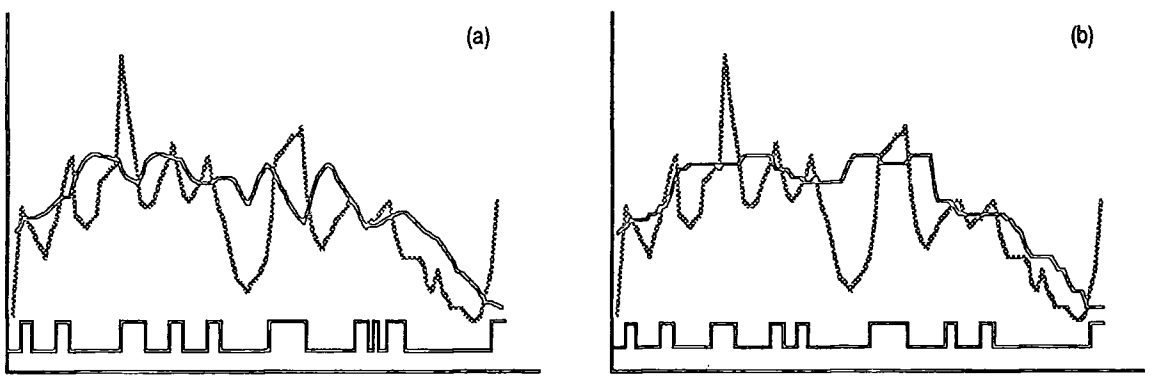


Fig. 2.12 : Comparison of CAGO (a) and OS (b) CFAR Methods
(Solid Line :- Mean Threshold Level)

As the rank-based CFAR introduces a higher average threshold level than a CAGO processor of similar window size, Rohling has introduced a measure of the loss (in dB) which is incurred by the replacing the CAGO with an OS CFAR, based on the Average Decision Threshold (ADT) of each. This is given by the expression:

$$D[dB] = 10 \log_{10} (ADT_{os} / ADT_{cago})$$

For the test data shown in Figure 2.12, this indicates a loss of approximately 0.451 dB, which compares well with the results given in [41]. Variations in the value of 'K' tend to raise or lower the mean threshold value for the OS CFAR (Figure 2.13): a value of K=3N/5 gives similar threshold performance to the CAGO, whilst a value of K=9N/10 gives a high threshold with slow response. For the OS CFAR to function successfully, it is suggested that K should be greater than N/2 and the difference N-K should not be less than double the expected target depth (in range cells) so as to retain the ability to detect multiple adjacent targets.

Rohling has concluded that in multiple target (equal amplitude) situations, use of a CA or CAGO CFAR will often result in loss of both targets due to a raised threshold. For targets of differing amplitude these CFARs tend to lose the one of lower amplitude, whilst the ordered-statistic CFAR can be shown to detect both targets regardless of their amplitudes.

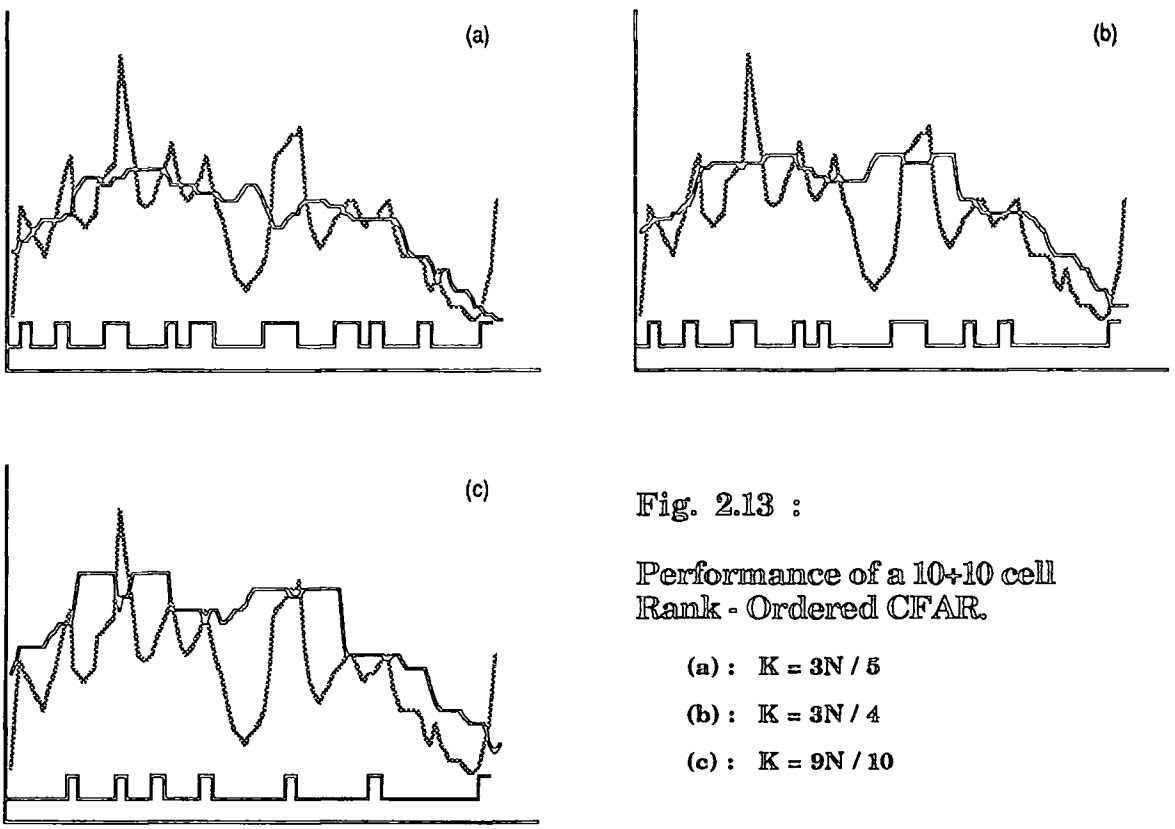


Fig. 2.13 :

Performance of a 10+10 cell Rank - Ordered CFAR.

(a) : K = 3N / 5

(b) : K = 3N / 4

(c) : K = 9N / 10

2.3.4.3 Non-Parametric Detectors

The Cell Averaging, CAGO and Ordered Statistic CFARs discussed previously can all be termed parametric detectors, by way of the methods they use to establish a mean threshold level. All make assumptions about the distribution of the clutter which they are required to filter and therefore tend not to perform well in situations where these rules are not adhered to. For example the CA and CAGO processors are based on an assumption of a Raleigh clutter distribution within the window, whilst OS types expect a certain size of target and clutter environment to function effectively. An alternative class of CFAR which is designed to function independently of the clutter distribution is termed a **non-parametric** detector, and a practical implementation of such a processor has been suggested by Mao et al [42] due to earlier work by Vogel [43] and Dillard [44].

The non-parametric CFAR uses the statistical ranking of a test cell's amplitude within a range window as the basis of a simple thresholding test, however it is important to note the differences between this and Rohling's Ordered Statistic CFAR which is also rank-based.

If, for range sweep 'i', a window of size N cells (excluding test cell) is considered, with corresponding video level X_{ik} ($k \in 1...N$) in each cell, and a test video level of X_{ij}, then the 'rank' of the test cell (R_{ij}) is given by the expression:

$$R_{ij} = \sum_{k=1}^{N} C(X_{ij} - X_{ik})$$

where C is an operator performing the function:

$$C(x) = \begin{cases} 1 : \text{for } x>0 \text{ or } (x=0 \ \& \ i-j \text{ is odd}) \\ 0 : \text{for } x<0 \text{ or } (x=0 \ \& \ i-j \text{ is even}) \end{cases}$$

From this we can deduce that the value of R_{ij} is an integer in the range 1...N, and this can then be used as either an input to the CA CFAR described previously or simply compared with a pre-determined rank number to establish a binary output.

As with the OS CFAR, the ideal rank threshold number is dependant on the window size and the required false alarm probability. For a large window size, it has been discovered that this approximates to $0.8(N+1)$, however the large window introduces a slow response to changes in clutter level whilst a smaller window increases the false alarm probability. Mao suggests a window size of between 8 and 16 cells, with a calculated false alarm probability of $0.84 * 10^{-6}$ for a 15 cell window.

The suggested method of implementation is to use an analogue charge-coupled device as a shift-register, with output taps being fed to comparators which produce a binary output depending on the result of comparison with the test cell amplitude (**Figure 2.14**). Charge coupled devices currently available can easily support the clock rate required to give an adequate sampling rate for most surveillance radars, with an accuracy which corresponds to an 8-bit digitisation process. A size of at least 20 CCD cells is recommended to give adequate CFAR response.

The outputs from the comparators are summed in an PROM (to speed calculation) and the output of this gives a direct measure of the rank of the test cell. It must be remembered, however, that this figure must be processed further in order to evaluate a binary target declaration, and in the example given, the rank is compared with a manually preset value. The output is the same as would result from a conventional A/D convertor followed by a CA or similar CFAR, but avoids the problems normally associated with this type of processing.

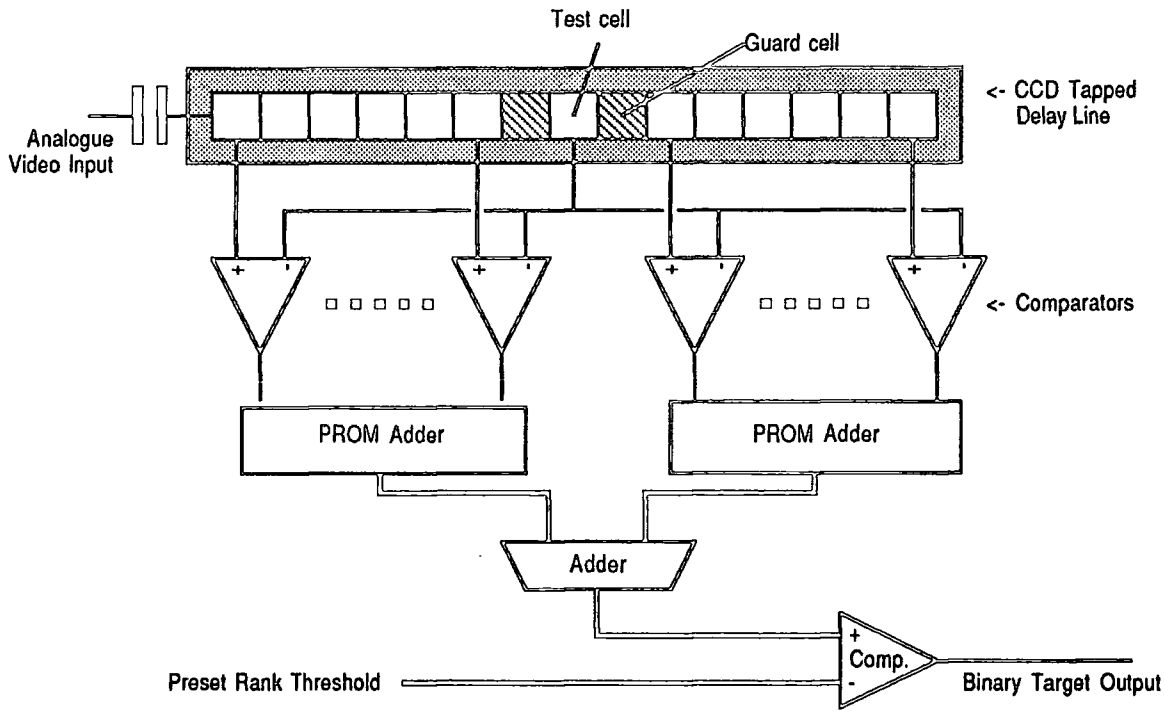


Fig. 2.14 : Implementation of a Non-Parametric CFAR

2.4 The Second Threshold Detector

The clutter-elimination techniques discussed so far have mostly concentrated on the range ordered aspects of clutter regions, processing data that is derived directly from the detection stages of the radar receiver. In practice, the width of the antenna beam is such that several pulses will be reflected from any genuine target and it is therefore possible to look for correlation between pulse repetitions in order to establish the presence of a target against a background of range-correlated clutter. A system which will work within the scope of these requirements is termed a 'Second Threshold Processor' or, by way of its method of function, a **'within-beam integrator'**. The processor is usually coupled with a CA or similar CFAR, as described earlier, to form a complete **double-threshold detector**. Original work in this area is due to a number of sources [45,46] and improvements to the basic design have been suggested by Marcoz and Galati [47]. This latter design is particularly easy to implement and it is therefore proposed to discuss this solution in detail.

2.4.1 The Marcoz & Galati Detector

The Marcoz and Galati (M&G) detection process consists of four basic steps which are evaluated for every range cell over a continuous azimuth scan. The first step is performed by a first-threshold processor of the type described previously or in its simplest form by comparison of the input signal with a fixed threshold voltage. The binary 'hits' which result from this stage are used to clock a counter which counts up if a hit is recorded or down if there is no hit. The second threshold is applied to the output of the counter such that a target is declared if the counter goes above a certain preset level and continues to be declared until the counter value falls below a second preset level. In the basic 'second threshold detector', the counter increments and decrements would both be set to a value of 1, whilst the Marcoz and Galati variation sees a larger increment before a target is declared and a larger decrement after the target has been declared. The following parameters are used in the evaluation of a second threshold:

i	The number of the sweep, i.e. the *i*th p.r.i.
X_i	Output from the first threshold detector
Y_i	Stored accumulator value
Z_i	Target detection status (1 - target present, 0 - no target)
U	Upper second threshold
L	Lower second threshold
INC	Counter increment
DEC	Counter decrement
α,β	Counter increment/decrement constants ($\alpha > \beta$)

The detector is initialised, at power-on, with accumulator (Y) and target detection status (Z) set to zero. The following algorithm can then be applied for every range cell in successive p.r.i. cycles:

```
IF ( Z(i-1) = 1 )      THEN
      INC = β, DEC = α
ELSE
      INC = α, DEC = β

IF ( Xi = 1 ) THEN
      Yi = Y(i-1) + INC
ELSE
      Yi = Y(i-1) - DEC

IF (Yi ≥ U ) THEN  Zi = 1

IF ( Z(i-1) = 0 ) & ( Zi = 1 )  THEN
      (start of target declaration)
ELSE
      IF ( Zi = 1 )  THEN
            IF  ( Yi > U ) THEN Yi = U
            IF  ( Yi ≤ L ) THEN  Zi = 0

IF ( Yi < 0 ) THEN  Yi = 0
```

The result of applying the above algorithm to typical test data is shown in Figure 2.15. The effect of two increment and decrement values, with one larger than the other, ensures that targets are declared quickly when none are present in the azimuth window, but rapidly lost once first threshold crossings start to diminish. The resulting plot window is more accurately defined than would be achieved with a single increment value.

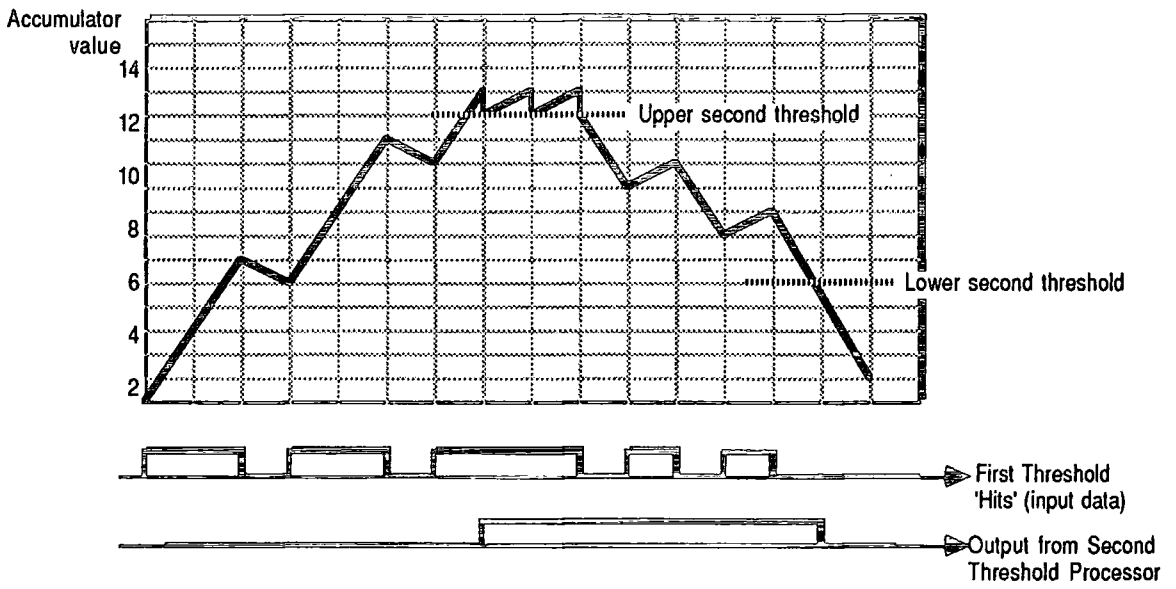


Fig. 2.15 : 'Marcoz & Galati' Second Threshold Processor

In the example given above, each vertical grid line represents one pulse repetition interval. A target is declared 7 p.r.i.s after the initial first threshold crossing, following 5 'hits' and 2 'misses'. The increment and decrement values (3 and 1 respectively) are then swapped and after a further 7 p.r.i.s the accumulator count falls below the second threshold and the target ceases to be declared.

2.5 Data-Rate Reduction Through Plot Forming

In general, it must be assumed that target declarations which have passed both first and second threshold processors have exhibited some degree of correlation in both range and azimuth directions and can therefore be deemed to be genuine reflectors. At this stage it is still not possible to evaluate their properties as desired targets, since correlating reflections could equally be due to bird flocks, land clutter or unfiltered moving clutter (precipitation, waves e.t.c.). Further processing should ideally evaluate the scan-to-scan correlation of plots as well as their tendencies to manoeuvre in acceptable trajectories, and this would typically require some form of computer-based filtering because of the complexity of the tasks involved. The data rate of most computers is not sufficient to support the potential output rate of a within-beam integrator which could, in extreme conditions, produce an output for every range cell. It is therefore necessary to find some method of reducing the data rate to avoid overloading the computer, and a solution is offered by considering the type of data which is processed by the software.

The computer is likely to be required to filter out all stationary plots from the incoming data stream, which will remove most types of land clutter, and then to track all remaining plots. For these functions, it would be simplest to carry out processing based on the co-ordinates of each plot, thereby simplifying the calculations involved in tracking manoeuvring targets, and this therefore provides an ideal format for data transfer from the detector to the computer. As genuine targets are likely to produce second threshold detections which extend over a number of range cells and pulse repetition intervals, a convenient method of data-rate reduction can be achieved by converting the range/azimuth target declarations from the detector into co-ordinates representing the centre of a plot. Additional logic must be included to ensure that plots are restricted to acceptable dimensions so that multiple adjacent targets are not lost.

2.5.1 Plot Forming Procedure

The plot forming function can be established using a number of methods, such as latching start and end azimuth readings as they are declared by a within-beam integrator and establishing their central point. A less complex method involves finding the geometric centroid of a window surrounding a plot which has been declared by the within-beam integrator. This is achieved by positioning a window of fixed range depth, but varying width, around an area of second-threshold detections, allowing the width to increase until either there is an absence of second-threshold 'hits' in the range direction or a pre-defined limit is reached. This limit is set such that the area of the window (in range cells & p.r.i.s) is approximately equal to the maximum size of likely targets.

Once closed, the geometric centroid is established by weighting the second-threshold detections contained in the plot window, then looking for their centre of gravity. To avoid loss of adjacent plots, windows are permitted to overlap to a certain limited extent.

Consider Figure 2.16: the hatched areas represent second threshold detections, and the window surrounding them is due to the left-most active cell. Following initiation, the

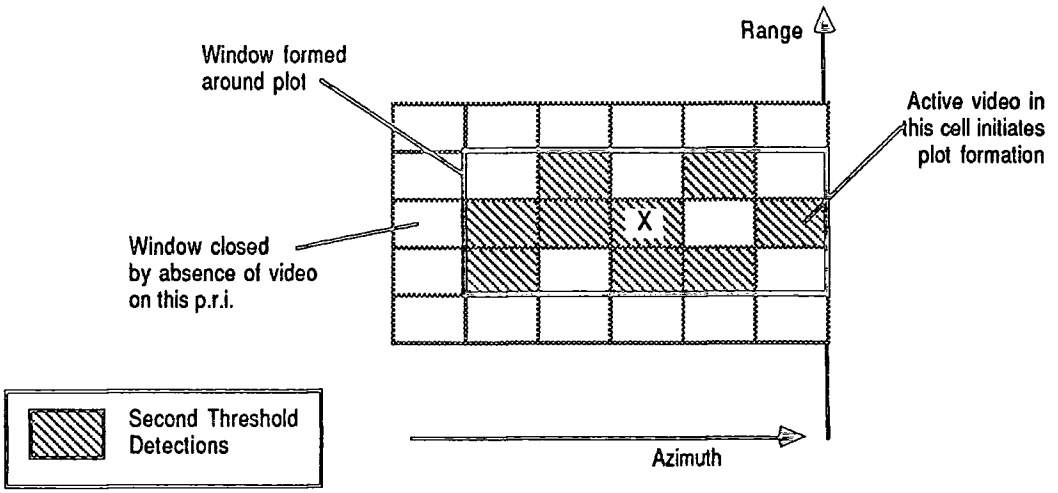


Fig. 2.16 : Evaluating the Centre of a Plot

window is established with a width of one cell and the initiating cell in it centre. Reference co-ordinates are noted for the azimuth and range values at this point ($\emptyset_e$, R_e respectively) as a basis for calculation of the plot centroid when it is closed. On subsequent p.r.i.s the window is widened to encompass further second threshold crossings until, after 5 p.r.i.s, there are no detections within the new range window and the plot is closed.

On each p.r.i., the new cells are all given weightings with respect to the reference co-ordinates and these are accumulated for use later. For every cell, the azimuth moment ($\Sigma\emptyset$) is calculated by multiplying its binary target status ($X_{\emptyset r}$) by the distance (in p.r.i.s) from the reference azimuth. Similarly, the range moment (ΣR) is evaluated from the product of target status and distance from the reference range. A count of the total number of active cells (i.e. $X_{\emptyset r} = 1$) is also accumulated (ΣN) and upon closure of the plot, the co-ordinates of the centre-of-gravity (R_c & $\emptyset_c$) are established by taking the accumulated moments and dividing by the number of active cells in the plot window.

Hence, if k is the number of range cells on either side of the centre of the window and m is the width of the window , then:

$$\sum R = \sum_{\theta=0}^{m} \left(\sum_{r=-k}^{+k} \left(X_{\theta r} * (r - r_c) \right) \right) \qquad \text{(accumulated range moment)}$$

$$\sum \Theta = \sum_{\theta=0}^{m} \left(\sum_{r=-k}^{+k} \left(X_{\theta r} * (\Theta - \Theta_c) \right) \right) \qquad \text{(accumulated azimuth moment)}$$

$$\sum N = \sum_{\theta=0}^{m} \left(\sum_{r=-k}^{+k} \left(X_{\theta r} \right) \right) \qquad \text{(accumulated active cell count)}$$

from which we can determine that:

$$\Theta_c = \Theta_c + \sum \Theta / \sum N \qquad \text{(centre of plot, bearing)}$$

$$R_c = R_c + \sum R / \sum N \qquad \text{(centre of plot, range)}$$

Application of this algorithm to the plot window shown in **Figure 2.16** would indicate the centre of plot as marked. A practical limit on the plot window size (for typically airborne targets) would be in the order of 3 to 5 range cells / p.r.i.s, although it must be remembered that plots are permitted to overlap by one (or two) cells in any direction. In practice, the algorithm is easy to implement and an example is given later in Section 3.5. With the advent of new signal processing techniques, the regularity of the processing lends itself to a highly compacted architecture [49] which will be referred to in greater detail in Chapter 8.

2.6 Summary

The importance of signal processing techniques to any radar system has been discussed in this Chapter, and a number of practical methods of implementation have also been shown. Reduction in the number of false targets being passed to the operator is of prime importance since it is difficult, even for a trained operator, to extract information from unfiltered data on a p.p.i. display. With computers needing access to the information that is obtained from the radars, it is equally essential to reduce the data rate from that of raw video down to a manageable level to avoid overloading, and the double threshold detector has been shown to perform this task with efficiency. The addition of a geometric centroiding processor provides target data in a more acceptable form to the computers, and this reduces the data rate still further without loss of sensitivity.

Variations in basic design of the CFAR processors are continuing to be developed, to counteract their traditional dependance on statistically distributed clutter, which rarely occurs in a practical environment. The statistical-rank and ordered statistic CFARs are comparatively new techniques, but are already being adopted due to their superior handling of clutter edges and multiple target situations.

2.6.1 Effectiveness of the OS CFAR Against E.C.M. Operation

The potential for 'programming' the threshold rank in the OS CFAR is likely to be of importance in an environment where electronic counter measures (E.C.M.) are in operation, as the rank can be chosen to ignore the high amplitudes and sharp edges associated with stretched-pulse or railing jamming. E.C.M is employed by targets which require some degree of invisibility to radar detection, and the techniques they use usually make some assumptions about the type of receiver which they are designed to deceive. In general, they work on a principle of attempting to raise the CFAR threshold artificially, so as to mask out genuine targets and achieve this by transmitting repeated pulses at intervals which match the detectors window size (railing jamming) or stretching and re-transmitting the original pulse (stretched-pulse jamming).

Use of a large window size should enable a rank value to be chosen which ignores the effects of railing jamming, whilst the stretched pulse technique requires a rapid response CFAR which can overcome clutter edge effects since a portion of the reflecting target will be visible for the time it takes to receive and re-transmit the stretched pulse.

3.0 Introduction

In the late 1970's, Admiralty requirements for a compact system, capable of fulfilling the radiolocation and tracking needs of the Royal Navy, prompted considerable work on the theory of clutter suppression and automatic tracking. During this time research was carried out by both MoD establishments and private companies and resulted in a number of papers on the subject of various tracking algorithms and, notably, a paper by Tunnicliffe of the Admiralty Surface Weapons Establishment (A.S.W.E.) entitled "A Simple Automatic Radar Track Extraction System" [33]. This paper was presented at the 1977 International Radar Conference, and in it he proposed a processor which could be interfaced to a number of different types of rotating surveillance radar and would accumulate data on surrounding movements, presenting this to the ship's data-gathering computers. The design challenge was quickly taken up and a specification produced for the Radar Automatic Track Extraction prototype, later to be simply referred to as RATES.

3.1.1 The RATES Specification.

The basic requirement for RATES was a system which could be installed in Royal Navy ships and would provide target track data for the Action Information Organisation (AIO) computers (which essentially functioned as a centralised data gathering system). The front end of RATES required an interface capable of matching almost any rotating surveillance radar (providing its functional parameters fell within the tolerances of Table 3.1), although 4 specific antennae were detailed for testing purposes.

Pulse Repetion Frequency:	256 384 768 1536 (Hz, ±10%)
Maximum Range:	235 data miles (P.R.F. not exceeding range capability)
Rotation Rate:	30 rpm maximum
Pulses / Beamwidth:	16 maximum
Range Resolution:	250ns max, & in multiples of same.
Range Cells / P.R.I.:	4096 max.
Interscan Dead Time:	200μs minimum

Table 3.1 : Radar Interfacing Constraints

The raw video would be digitised to a minimum of 8 bits accuracy and then CFAR processed, using a two stage threshold processor, resulting in binary target declarations which would then be extracted to produce plot co-ordinates. These could then be fed into a computer-based stationary plot filter and finally passed out to track association software as likely targets (Figure 3.1).

The output from RATES would ideally be a connection directly to the A.I.O., carrying the following information:

Confirmed track data (Output once per scan)
New track data
System control messages
Raw video data

A degree of feedback from the A.I.O. would also be required, most importantly the ability to initiate target tracking manually. The prototype would not need to provide all of these outputs however, although the design would need to be adaptable in terms of both hardware and software, so they could be added easily at a later date.

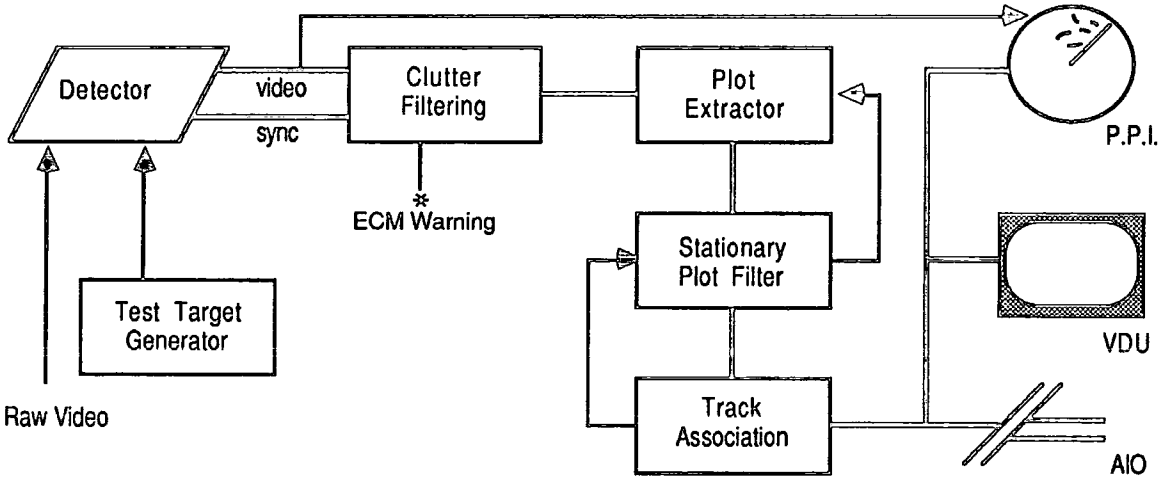


Fig. 3.1 : RATES Functions

Additional facilities required by the specification included a test-target generator which could simulate the operation of a radar in a programmable clutter environment, a simple electronic counter measures (E.C.M.) warning system and a comprehensive control panel so as to enable fine adjustment of parameters before choosing an optimum solution.

Several engineering parameters were provided for the RATES design, notably the power supply and size requirements. RATES was intended as a replacement for an early plot extractor, which lacked auto-tracking features, and it was important that the new RATES

system fitted into the space vacated by this machine. Accordingly, the designers were given the space of three 6' x 19" equipment racks, with a documented preference that the size be reduced as much as possible below this for a production version. The available power supply was to be a nominal 240 volts at 50Hz, to a maximum of 5KVA with a requirement that it be capable of operating from a 110 volt 60Hz supply if necessary.

3.1.2 The RATES Prototype

Given the specifications of RATES, it is intended to discuss the various sections of the system in detail, so as to describe the actual implementation of the prototype.

Technical design was undertaken by a major electronics company in the U.K., who were also responsible for the construction of two identical systems for testing and evaluation by the Admiralty. The design made use of late 1970's technology, using small/medium scale integration (S.S.I./M.S.I.) for the majority, with limited use of large scale integration (L.S.I.) components where practicable. The hardware was assembled using wire-wrap techniques on standard Eurocard prototyping boards, with external connections over 36 pin double-sided edge connectors. A total of 32 boards completed the system, most installed in one of the three 19" cabinets, along with the power supply. The remaining boards were fitted into a cabinet containing one of the Ferranti Argus computers, where they formed the other end of a data transfer link.

The 32 boards which made up the RATES system could be divided into functional groups - the number of boards per group varying between one and seven. **Figure 3.2** (overleaf) illustrates the complete prototype, in terms of the boards used to build it. The areas which dominate the system hardware are primarily those involved in interfacing - not only the F and H series boards, but also the Y boards which drive the P.P.I. display. It is worth noting that the latter were provided mainly for testing purposes, as a production system would probably require that video be sent directly over the A.I.O. interface. A brief description of the interface has been included, however, in a later section of this chapter.

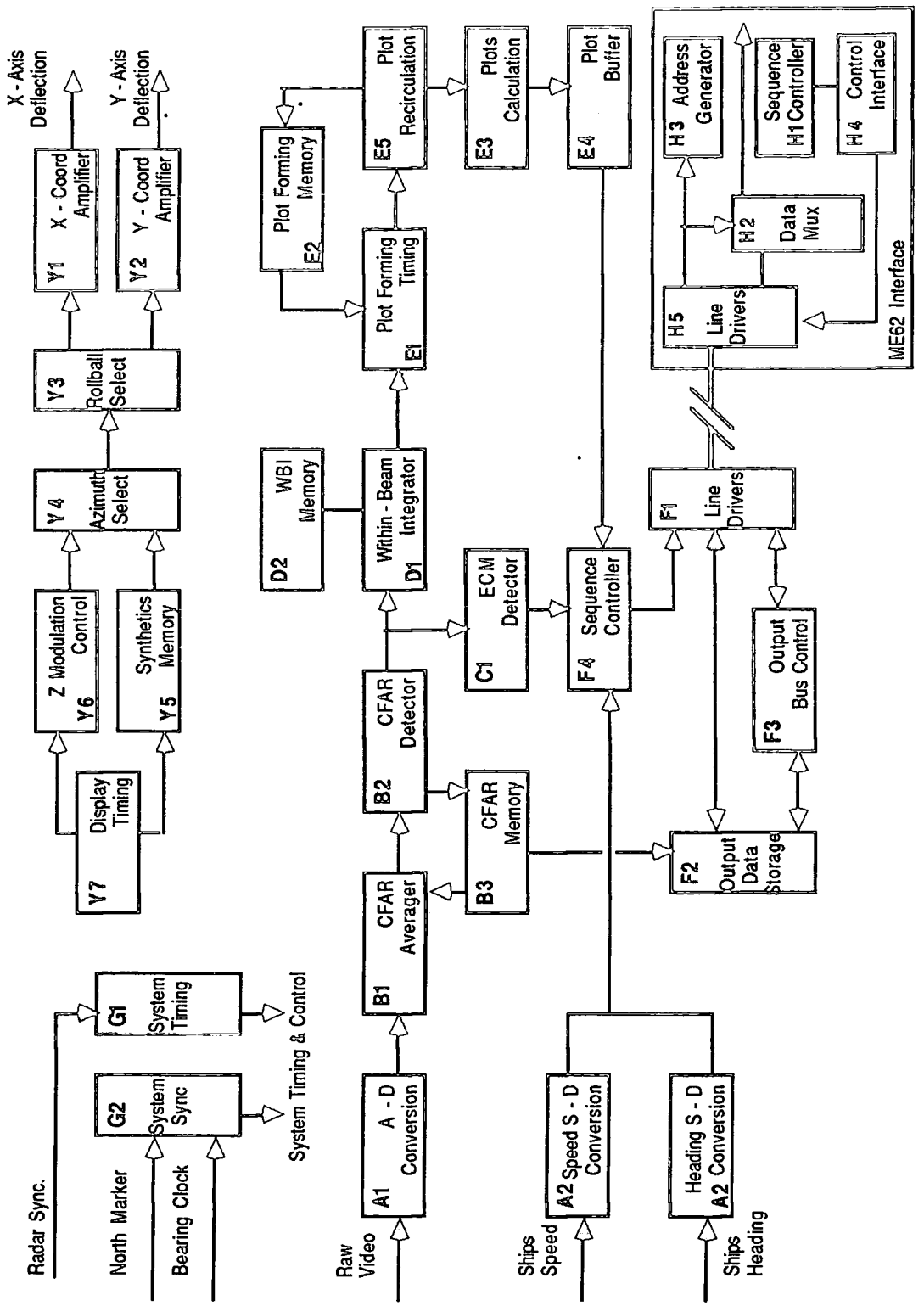


Fig. 3.2 : RATES Prototype Block Design

3.2.1 The Input Interface

The RATES input comprises 6 signals, 4 of which provide necessary radar data and the remainder giving details of the ship's movement (it must be remembered that RATES was designed for use on board a ship, which could conceivably be moving relative to identified targets). The radar data inputs are detailed below :-

(i) Logarithmic video signal (rate 20dB/V), to a maximum amplitude of 4.5V with a typical noise component of 0.24V.

(ii) Synchronisation signal - a positive 18V (±3V) pulse of 3μS duration, marking the start of each transmitted pulse. The signal would normally be carried by a 75Ω co-axial cable.

(iii) Bearing synchronisation signal - a 1:1 square wave generated as the antenna rotates. Normally 4096 pulses/rev, but sometimes 2048 or 1024 depending on radar type. As the speed of rotation of the antenna can be greatly affected by the attitude of the ship and weather conditions, this signal serves to synchronise the plot extractor with the true rotation of the antenna.

(iv) North marker - an active low signal, of period equal to one of the bearing clock cycles, transmitted when the beam is pointing in a due northerly direction. North marker and bearing clock signals are sent over an appropriately terminated twisted-pair cable.

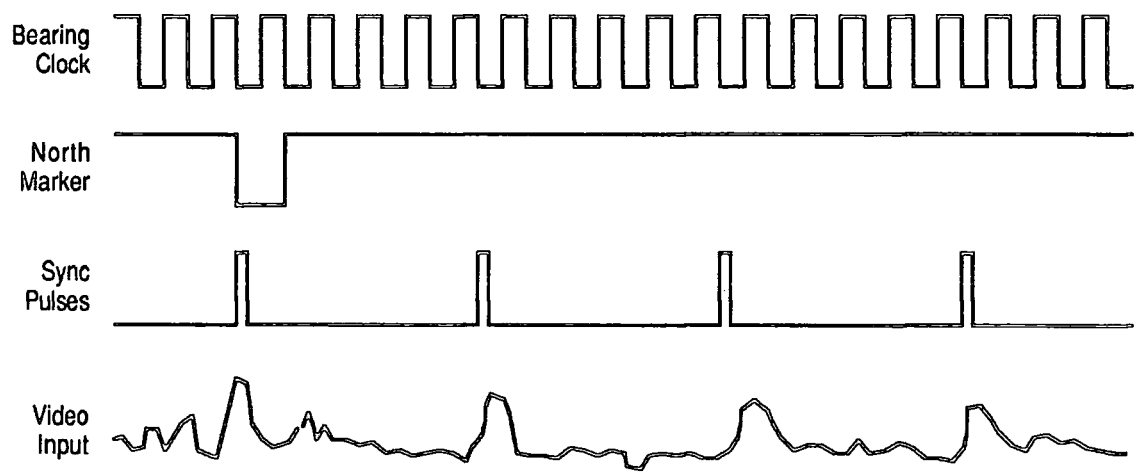


Fig. 3.3 : Relationship Between Primary Input Signals

The ship's motion inputs:

(v) Ship's Speed - 180° synchro output corresponds to 15 knots.

(vi) Ship's Heading - follows a 1:1 relationship with compass bearing.

Both of these are transmitted as 3-wire 115/90V 400Hz synchro signals, combined with appropriate reference data. To be of use to the plot extractor they have to be converted to digital form in the early stages of the interface.

3.2.2 Video Interface Design

The input interface is constructed on 3 main boards, which handle buffering and digital/analogue conversion of the input video and ship's motion signals. The remaining inputs are processed on the system timing boards (G1 and G2). Functional diagrams of interface boards A1 and A2 are shown in Figures 3.4a and 3.4b.

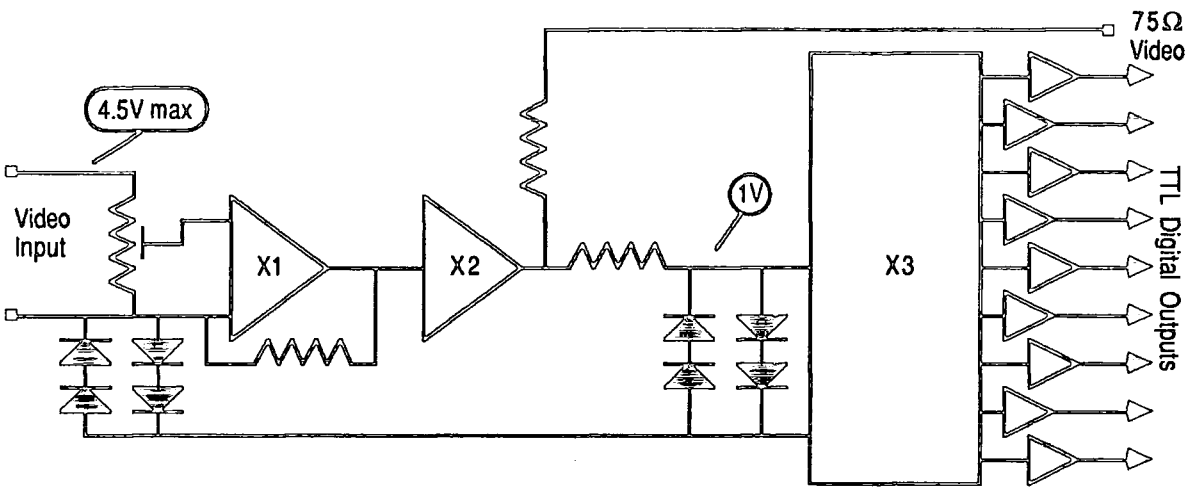


Fig. 3.4a : Video Interface Schematic

The video input circuits essentially isolate the detected video signal from the voltage ground of the plot extractor. The signal can be attenuated if required through use of the potential divider arrangement before it reaches the input of the unity gain buffer [X1]. This functions as a voltage to current converter, driving the current amplifier [X2] which with a gain of 1 acts as a second buffer, prior to a splitter which divides the signal to drive two 75Ω loads. One load is the input to the high-speed A/D converter [X3], a thick film hybrid component which supports a conversion rate of 12MHz and performs a single conversion in 150ns. The voltage at this point is diode limited to approximately 1V so as to prevent damage to X3, and it is intended that the potential divider be set such that an input of 4.5V produces a level of 1.0V here.

The output from the converter is an 8 bit byte at 10kΩ emitter coupled logic (E.C.L.) levels, plus a single 'conversion complete' signal. All 9 lines are buffered to convert them to standard TTL levels, prior to transmission to the next stage of processing.

The other output from the splitter is taken directly to the display interface, where it can be selected for output to the P.P.I. display.

3.2.3 The Synchro Interface

The synchro converters on boards A2a and A2b function in a similar manner to the video interface, with the exception that buffering of the input is performed on-chip by a hybrid synchro-digital converter [X4]. The synchro signal is received as 3 separate phase-encoded inputs and an accompanying reference. The transponders for ships motion and heading both convey angular information, which is transmitted to the interface along these 3 wires, each carrying a sine wave, the phase of which is displaced from the reference by a defined amount. The simple operation of the synchro - measuring 3 phase differences - can easily be integrated onto a single device and a digital conversion applied to the result, enabling an accurate value to be equated to the angular displacement of the transponder.

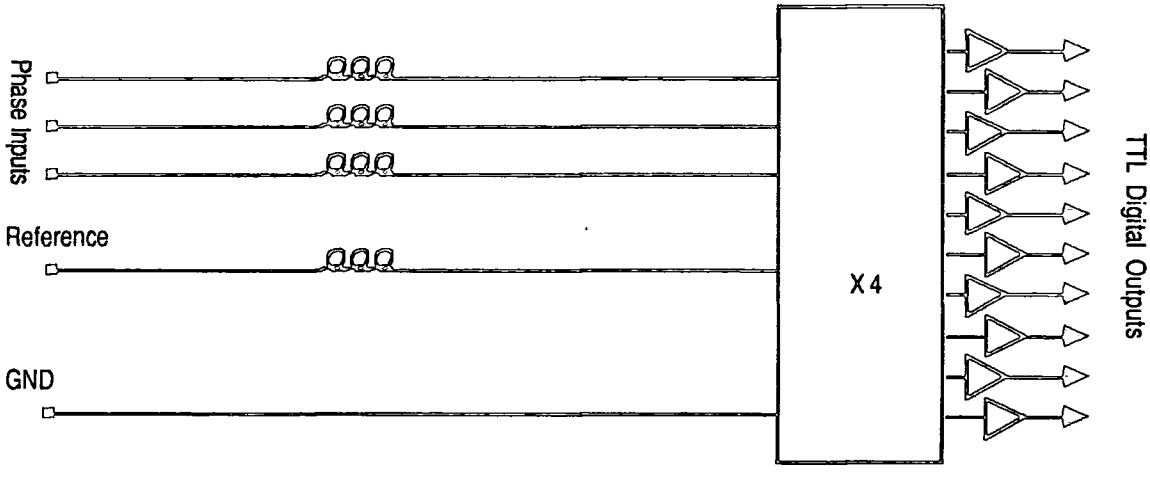


Fig. 3.4b : Synchro Interface Schematic

The output from the device [X4] is a 10 bit word, at TTL levels, and is buffered prior to transmission across the backplane bus. The synchro information is not used at all within the hardware plot extractor and only the upper 6 bits of the converted speed signal are passed on to the computer interface, thereby enabling a single 16 bit word to convey both speed and heading information to the computers.

3.3 The Constant False Alarm Processor

The converted video signal is processed by the constant false alarm rate circuitry on the three B series boards, which implement a cell-averaging, greatest-of (CAGO) CFAR as described in the previous chapter [2.3.2]. The block diagram of this section is shown in Figure 3.5 - its apparent simplicity disguises an interesting circuit design and, consequently, detailed schematics are also shown (Figures 3.6a & b).

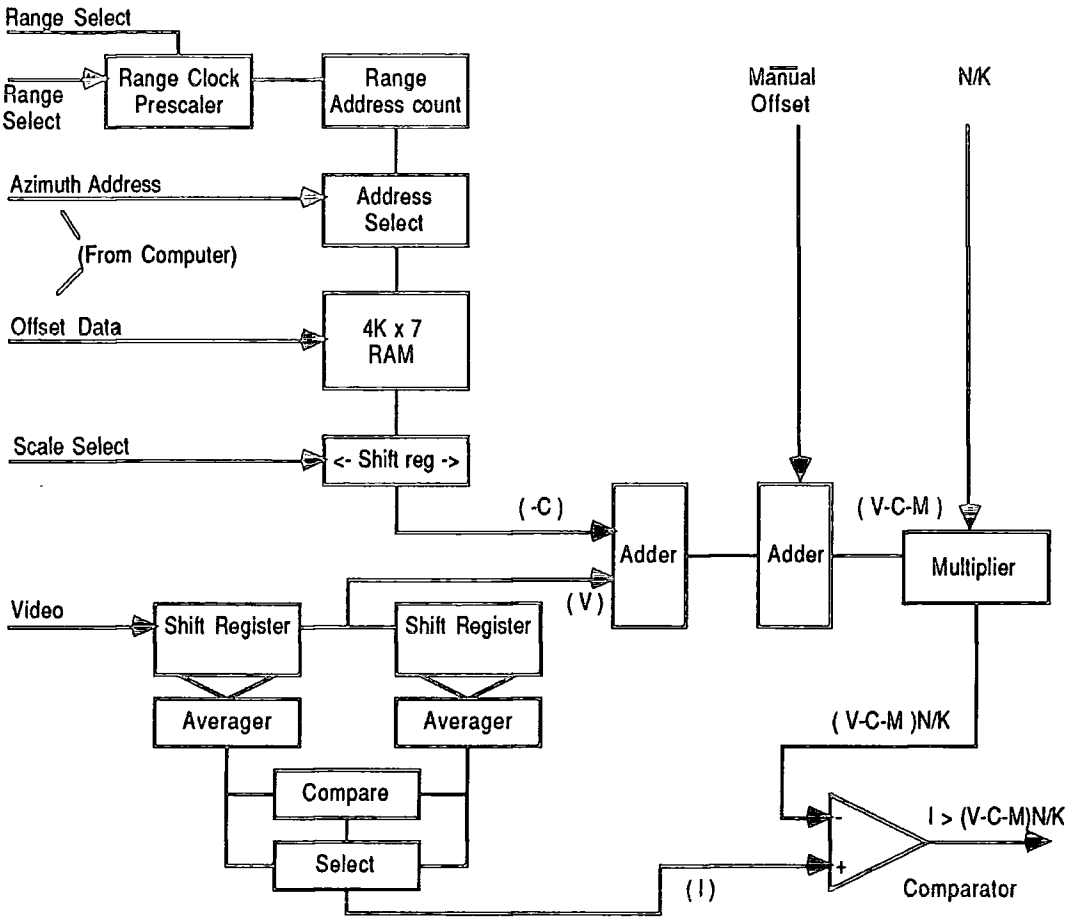


Fig. 3.5 : CFAR Processor

3.3.1 CFAR Averager

Board B1 holds the circuitry for the variable length shift registers which produce the pre- and post-range averages. By allowing the operator to select the number of cells used to derive the mean surrounding level, it is possible to tune the system to process a wide variety of target types and densities in differing clutter conditions. The number of cells on either side of a sample cell may be adjusted between 3 and 18 - and can be varied through use of a 16 byte RAM, addressed sequentially by a 4 bit counter, with its preset count value set according to the number of cells required. Thus, a preset value of 15 will result in a total delay of 3 clock cycles - 1 clock cycle in the RAM and 2 in adjacent latches. The input to the RAM is added to the previously accumulated sum in adder [X1] and the output of the RAM inverted and delayed by

one clock cycle before being added to this sum to form the new accumulated sum.

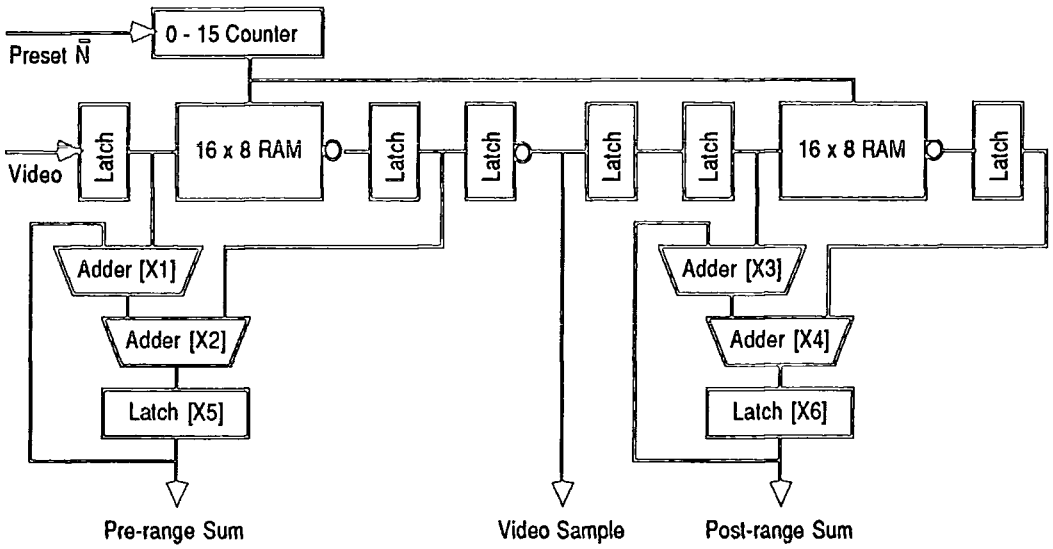


Fig. 3.6a : Function B1 Schematic

If sample cell is v[t] then, with a RAM delay of n cells we have:

- v[t+1]	input to adder X2
- v[t+2]	output from RAM
v[t+2+n]	input to RAM & adder X1

At time t, output from latch X5 is:

$$v[t+1+n] + (v[t] + v[t+1] + v[t+2] + \ldots v[t+n]) - v[t]$$

thus output from adder X1 is:

$$v[t+2+n] + (v[t+1+n] + (v[t+1] + v[t+2] + \ldots v[t+n]))$$

and output from adder X2 is:

$$v[t+2+n] + (v[t+1+n] + (v[t+1] + v[t+2] + \ldots v[t+n])) - v[t+1]$$

which may be rewritten as: $v[t+2] + \ldots v[t+1+n] + v[t+2+n]$

or:

$$\sum_{k=t+2}^{t+2+n} v[k]$$

therefore at time t, the output from the pre-range sum latch (X5) is:

$$\sum_{k=t+1}^{t+1+n} v[k]$$

which is the sum of the n cells preceding the sample, with a single guard cell.

Analysis of this output expression should indicate that the summing process takes place by way of an accumulator, to which is added the incoming new value while simultaneously subtracting the outgoing value which is delayed by (n+1) cycles. This is much faster than adding all (n) samples in RAM and proves easy to implement in hardware.

3.3.2 CFAR Memory

Board B3 stores the computer generated threshold levels which are used to offset those produced dynamically by the CFAR processor, and enables the system to mask-off areas of known high clutter level or electronic counter measure activity. It is capable of storing a total of 4096 x 7 bit values, which are updated by the computer every scan. The circuitry consists of address-select logic which controls access to the memory by plot-extractor or computer (the latter only having access during the interscan period). Clock signals, from the system timing circuits, are pre-scaled on B3 according to radar type, range setting and currently scanned azimuth. The output from the threshold RAM is then inverted and scaled by a factor of 0, 2, 4 or 8 times before being fed across a 9-bit bus to board B2.

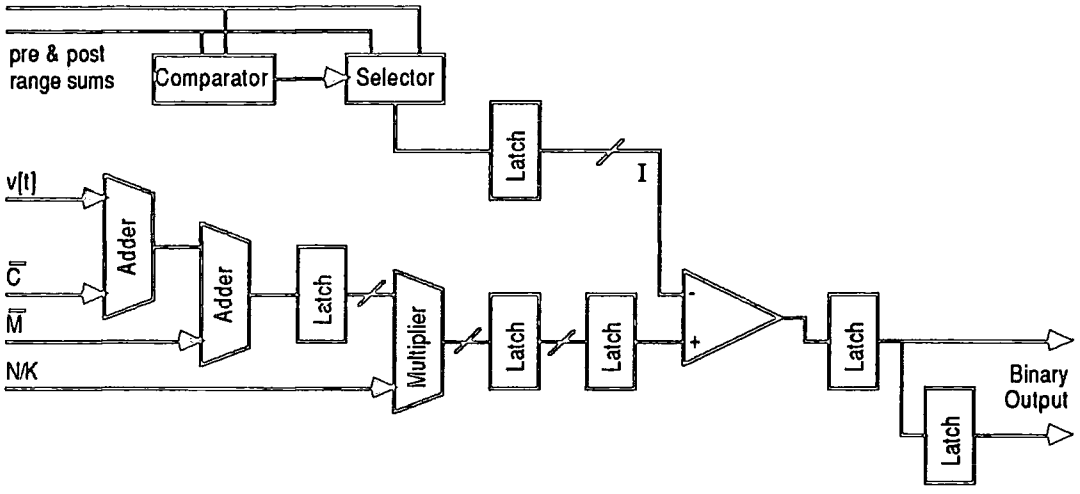


Fig. 3.6b : Schematic of the CFAR Detector

3.3.3 CFAR Detector

Board B2 carries the circuits which evaluate the binary target declarations from knowledge of the pre- and post-range sums. These two outputs are compared in an 8-bit wide comparator, and the output used to select the greater, which is then latched to prevent timing problems. The detector works on the CAGO principle whereby a target is declared when the following condition is satisfied:

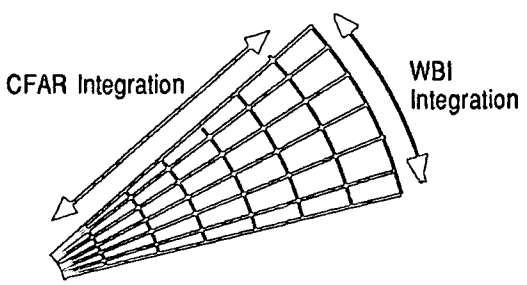
$$V > \frac{I}{N}.K + C + M \qquad \text{in the form:} \qquad \frac{N}{K}.(V - C - M) > I$$

The video sample (V) is added to the complements of the computer generated offset (C) (from B2) & the the manual offset (M), giving the result (V-C-M). This is retimed in a latch before passing to the multiplier. The expression (N/K) is evaluated by a PROM (on G1), and this too is applied to the multiplier, resulting in the expression (V - C - M)*N/K. After retiming, the result is applied to the final comparator which produces a binary '1' if it is found to be of greater magnitude than the greater of the pre- and post-range sums.

3.4 The Within Beam Integrator

The binary output from the constant false alarm rate circuit is retimed prior to input to the Within Beam Integrator (W.B.I.) section, firstly in a single latch, with a second output being delayed by a further clock cycle so as to be in synchronisation with the output of the W.B.I. processor. This feature greatly simplifies the design of the display processor, as detailed later.

The W.B.I. section works as the second half of a double threshold CFAR processor, and has been designed around the model proposed by Marcoz and Galati [47], described elsewhere in this thesis (Section 2.4). In contrast to the first threshold processor, described in Section 3.3, which integrates over range, the W.B.I. functions by accumulating each range cell over bearing (Fig. 3.7).



As with the first threshold integrator a sum is accumulated for every possible range cell value and a further set of binary target declarations produced whenever a preset threshold is exceeded.

Fig. 3.7 : CFAR & WBI Integration

Analysis of this mode of operation should indicate that an important feature of the W.B.I. processor is a high speed memory, equal in size to the number of range cells per sweep, storing the accumulated sum. The old value must be read from this memory at every range cell and the updated value rewritten before proceeding to the next range cell. A target is declared if an ascending accumulator value crosses an upper threshold, and remains in this state until the descending value crosses a lower threshold. To sharpen the boundary between clutter and a potential target, the ascending accumulator value increases by 'A' for every binary CFAR declaration and decreases by 1 for a miss. Similarly the descending accumulator value decreases by 'A' for every missed CFAR output and increases by 1 for a hit.

The following parameters are defined in the context of the W.B.I. processor:

A	Increment / decrement value for integrator (1...8)
LL	Lower level threshold value
UL	Upper threshold threshold (= LL + 1...32)
V_n	CFAR binary threshold for range cell 'n'.
I_n	Accumulator sum for range cell 'n'.
TGT_n	Target declared flag for range cell 'n'.

3.4.1 The W.B.I. Processor

The within beam integrator is implemented on two boards, with D1 carrying the main control and calculation circuits (Fig. 3.8a).

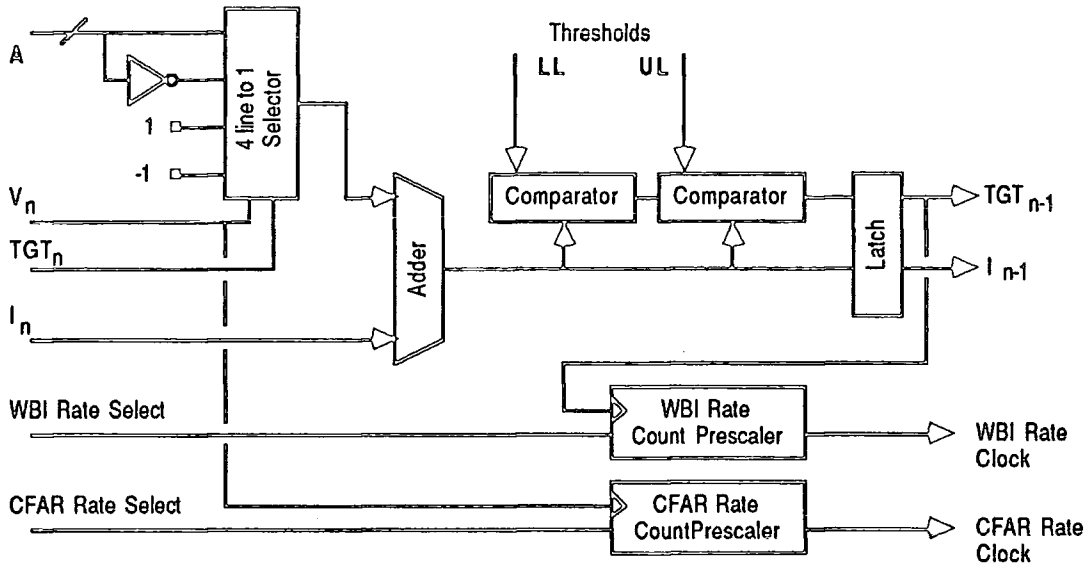


Fig. 3.8a : Within-Beam Integrator Schematic

The output from the first threshold processor (V_n) is used, in conjunction with the range cell target declaration from the previous sweep (TGT_n), to select the correct value of increment or decrement to be applied to the integrator ($+A_p$,$-A_p$,$+1_p$,-1). This is accomplished using a 4-line to 1-line by 4-bit wide decoder with its output connected to the adder. The resultant signal (I_n) is compared with upper and lower thresholds and the results of the test used to determine the state of the new range cell target declaration which is stored in the latch along with the accumulated sum prior to output to the W.B.I. memory.

The thresholds are read from the control panel in the form of a lower threshold level in the range (0...31) and an upper level offset, in the range (1...31), which is added to the lower level before presentation to the comparators. The maximum accumulator value is restricted to ($+63_{10}$), which corresponds to a six bit representation in memory, with the seventh bit holding the TGT flag. Additional gating is provided to prevent an accumulated value from either exceeding the upper threshold level or going negative, replacing the value presented to the latch by the upper level or 0 respectively.

The first and second threshold outputs are both fed to count pre-scalers on the same board, prior to output to the count-and-display circuits of the control panel, enabling the user to monitor the binary outputs from the CFAR and WBI processors. Upon reception of a north-marker signal from board G1, the outputs of these counters are latched into the display decoders and the counters reset in preparation for a new set of target declarations.

3.4.2 The W.B.I. Memory

As mentioned previously, an important feature of the W.B.I. circuits is a high speed memory, which ideally should be capable of a read-modify-write cycle in one range clock period. From the values given in Table 3.1, this would call for a complete processing cycle in 250ns. Whilst not particularly fast by the standards of the mid 1980's, this has evidently provided some problems at the time the unit was designed, and to avoid the necessity for a large number of high-speed bipolar memories, the circuit has been modified to adopt a less demanding read-write cycle. Referring to the schematic of Fig. 3.8b, the mode of operation of the W.B.I. memory controller should become clear.

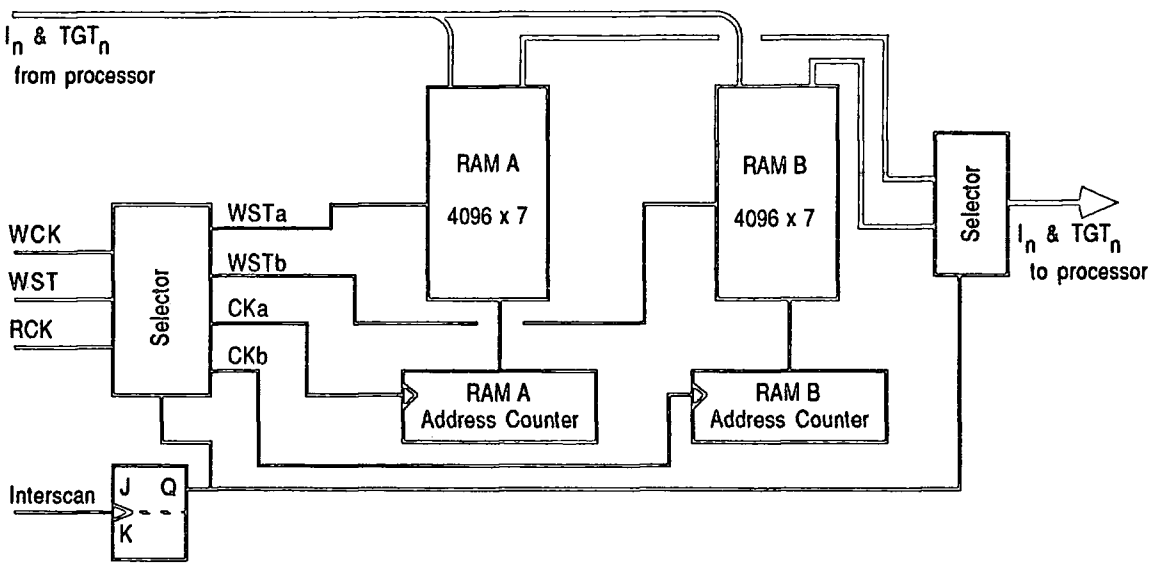


Fig. 3.8b : W.B.I. Memory Schematic

The circuit uses two banks of dynamic random access memory, each arranged to provide a 4096 x 7 bit store and each with its own address counter. Data from the board D1, consisting of accumulator values (I_n) and target declaration flag (TGT_n), is applied to the inputs of both RAM banks, but only one is enabled for writing on any given range scan. This is accomplished by arranging for the Interscan pulse to toggle a flip-flop, the output of which is fed to two data selectors, such that on the first range scan (for example) RAM 'A' would be selected for reading, with Counter 'A' being clocked by the pulse RCK. The output from this memory would also be selected for transmission to board D1 for processing by the W.B.I. At the same time, RAM 'B' would be enabled for writing, with the write enable strobe (WST) steered to the appropriate pins, and Counter 'B' clock by the pulse WCK.

At the end of a range scan a pulse toggles the flip-flop, and RAM 'B' is now used to read out the old accumulator values, with the new values being written to RAM 'A'.

Although dynamic RAMs are used for the memory, the update rate ensures that no special refresh circuits are needed to prevent loss of the memory's contents, thereby saving on the control circuits needed for this section.

3.5 The Plot Forming Processor

The plot forming section of the processor is the most complex of the RATES functions, and is implemented on 5 boards. It carries out the final processing of the signal prior to transmission to the filtering and tracking computers, and is therefore responsible for reducing the data rate to a level which had be handled by the computer, achieving this by reducing the binary targets declarations into a set of polar centre-of-plot coordinates. Four of the five boards hold circuitry for performing these calculations, with the fifth acting as a temporary store for accumulated coordinates.

The basic function of this unit is described in the previous chapter, however a summary is included here for convenience, using the terminology detailed below:

$\emptyset_{WBI}$	Current WBI azimuth (12 bits)
R_e	Reference range (12 bits)
$\emptyset_e$	Reference azimuth (12 bits)
$\emptyset_{pw}$	Current plot width (8 bits)
δR	Range moment (3 bits)
δN	Number of active cells this sweep (3 bits)
$\Sigma\emptyset_f$	Accumulated azimuth moment (15 bits)
ΣR_f	Accumulated range moment (8 bits)
ΣN_f	Accumulated active cell count (9 bits)

3.5.1 Principles of the Plot Forming Operation

The requirement of the plot forming processor is to establish the likely centre of a plot given a set of 'hits', from the within beam integrator, which define a likely plot area in terms of range and azimuth. The processor is expected to impose a maximum size on the plot, so as to enable it to resolve adjacent targets from what would otherwise appear as a single one. In practice, the maximum plot area is based on the targets to be tracked, and since RATES is intended primarily for monitoring aircraft movements, the individual plot area will tend to be small. Considering a typical range/bearing cross-section as shown in Figure 3.9, the principle of the plot forming process is described as follows:

The process is initiated by a logic '1' output from the W.B.I. circuits (cell A on Fig. 3.9), at which point the range and azimuth of this trigger cell are recorded (the reference co-ordinates). A delay is then introduced to allow the cell to ripple to the centre of the plot window, which in this example has a depth of three range cells. The central range cell is taken as a fulcrum for measuring the range moment of the plot, enabling the processor to establish the range with the highest average W.B.I. 'hit' incidence. For a 3-cell deep plot

window, moments take the values +2, 0 & -2 and hence the central cell does not contribute to the weightings at this point.

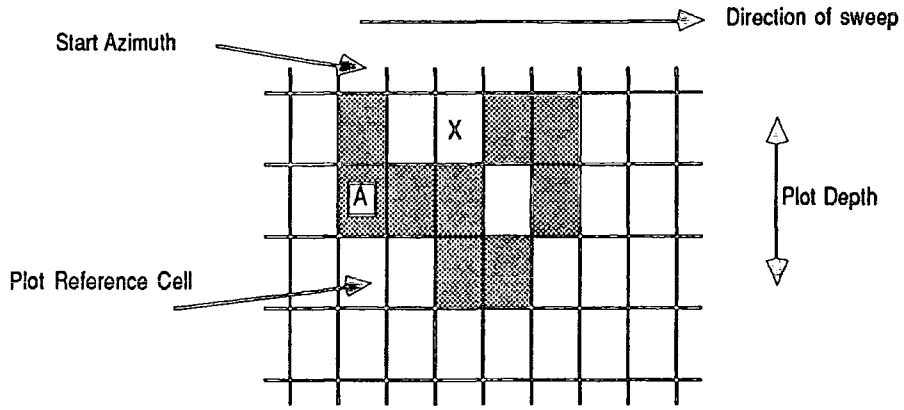


Fig. 3.9 : Example Plot Window

On the next azimuth bearing a new group of cells are brought into the plot window, where they contribute a range moment to the plot as described above. An azimuth moment is also calculated, being equal to the total number of 'active' range cells on that bearing multiplied by the distance (in bearing units) from the reference.The azimuth moment, number of active cells and range moment are accumulated over successive bearings until either the plot width exceeds the maximum permitted value, or there is an absence of W.B.I. video in the plot window, at which point the plot is closed and processing passes to the next stage of the extractor. The calculations performed on each bearing are summarised below:

∂R	- Range moment.
∂N	- Total number of hits on this bearing
$\Sigma N_{b+1} = \Sigma N_b + \partial N$	- Accumulated number of hits
$\Sigma R_{b+1} = \Sigma R_b + \partial R$	- Accumulated range moment
$\emptyset_{pw} = \emptyset_{wbi} - \emptyset_e$	- Current width of plot
$\Sigma \emptyset_{b+1} = \Sigma \emptyset_b + \emptyset_{pw} * \partial N$	- Accumulated bearing moment

Upon closure of the plot, the three accumulated values detailed above are combined with the reference co-ordinates to produce the centre-of-plot co-ordinates:

$\emptyset_c = \Sigma \emptyset_b / \Sigma N + \emptyset_e$	- azimuth centre of plot
$R_c = \Sigma R_b / \Sigma N + R_e$	- range centre of plot

For the example in Fig. 3.9, the centre of the plot can be calculated to be cell X.

3.5.2 Plot Forming Controller

The operation of the plot extractor is described in terms of the function of each board, so as to give a clearer picture of the way that data is processed. Figure 3.10a depicts the initial stage which processes data as it emerges from the within-beam integrator (board E1).

Binary target declarations are passed into the shift-register, which can be set to a width of 3 or 5 bits to accomodate different target depths, (this variable does not affect the general operation of the extractor and will be assumed to be set to 5). Upon reception of the first logic '1', the timing circuits activate the latches to hold the current antenna azimuth ($\emptyset_{wbi}$) and the current range which in turn become the plot's reference co-ordinates ($\emptyset_e$ and R_e). The range is derived from the counter on board E1, which receives a range clock from the system timing boards, and bearing derived from a separate counter on board G2 such that it lags behind the true antenna azimuth to allow for timing delays in the WBI and CFAR processors.

After a number of range clock pulses, the initial target declaration has moved to the central cell of the shift register, and at this point the adders calculate the total number of binary '1's in the shift register, and their moment about the central cell. These two "plot increments" (δN and δR respectively) are then output to the latches alonside the previously stored reference co-ordinates.

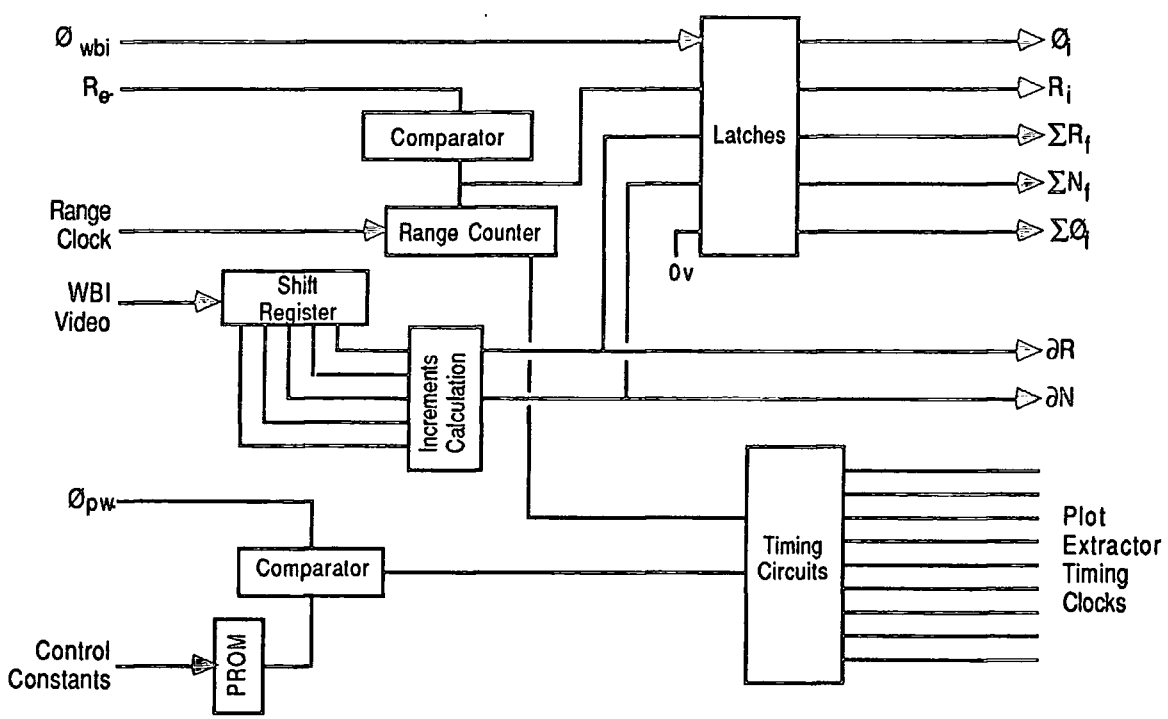


Fig. 3.10a : Plot Forming Controller

The initial target declaration sets a latch within the timing circuits, such that any further WBI 'hits' entering the shift register are unable to initiate a new plot before the first set of co-ordinates and plot increments have been sent to the next processing stage (board E5). This

limits the maximum possible overlap between plots to 2(1) cells for a 5(3) stage shift register.

At the end of the first scan, the Interscan pulse clears the range counter, and the timing circuits ensure that the reference co-ordinate details of the first plot are set up on the data bus, along with the total width of the plot ($\emptyset_{pw}$). When a target declaration from the WBI enters the shift register, the timing circuits initiate a comparison test between current range and the reference $\mathbb{R}_e$, and if they are equal (or within the permitted overlap) a latch is set to prevent initiation of a new plot. The adders calculate the new plot increments as usual, and these are then passed on to the next procesing stage.

Formation of the plot is completed when either the incoming WBI video is unable to trigger the timing control (i.e. absence of WBI 'hits'), or the plot width ($\emptyset_{pw}$) exceeds a preset maximum which is set by a combination of manual offset and radar-type setting. At this point, the timing circuits direct the plot-output stages to receive the data, rather than the plot re-circulation memory.

3.5.3 The Plot Forming Memory

Once a plot has been initiated, the data passes to the second stage on board E5 (**Figure 3.10b**), which serves as a storage area for the accumulation of plot information. The memory consists of a 256 x 56-bit RAM, arranged as two 128 x 56-bit blocks so as to facilitate the updating of plots, and permit the processing of up to 128 plots at any given time. A 56-bit data bus links the plot-forming memory with the remainder of the processing circuits, and access is controlled by the timing on board E1.

This configuration of RAM allows the processor to read data from the lower 128 words, calculate new offsets and then return updated plots to the upper 128 words. At the end of a sweep, the rôles are reversed and the upper block is used for reading. Additional advantages of this mode of operation are the ability to insert new items or delete completed plots without disturbing the ordering of the data, simply by adding an extra write pulse or reading an additional item.

Selection of the upper or lower half of memory is performed by a flip-flop, with its outputs connected to the most significant input to the address select decoder. The non-inverted output is linked with the write counter and the inverted with the read counter, thus ensuring that the two counters always address different sections of memory. The flip-flop is toggled to the opposite state upon reception of an Interscan pulse, which simultaneously resets the two address counters, ready for the next sweep.

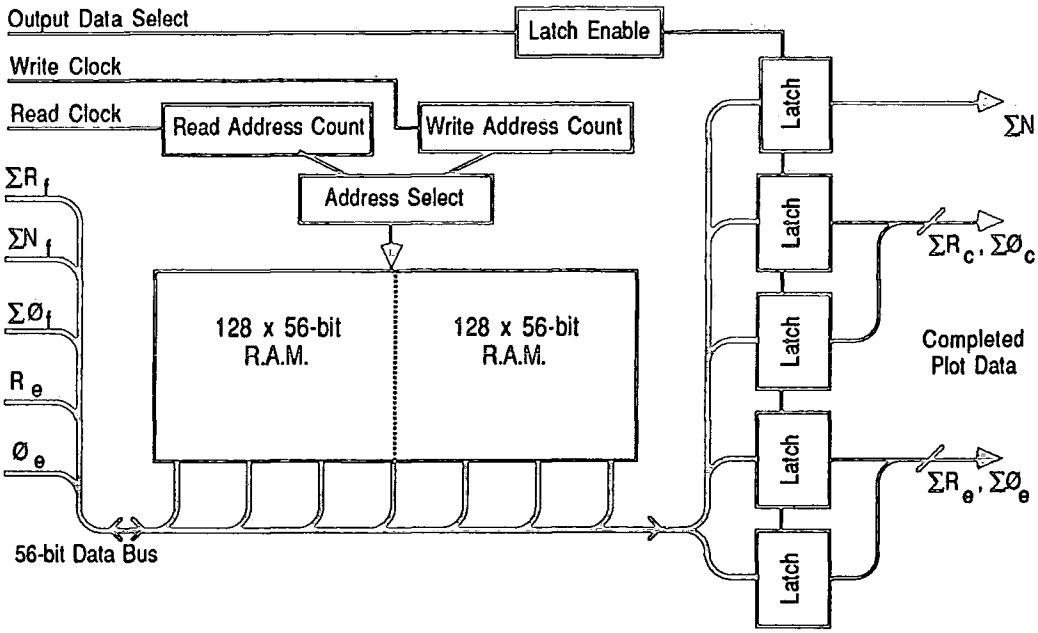


Fig. 3.10b : Plot Forming Memory

From reception of the interscan pulse, the operation can be described as follows: The first plot in range is addressed and read out to board **E2**, the read counter then being advanced to address the next plot. The plot accumulator on board **E2** handles the updating of the plot and the data is then sent back to **E5** to be written into memory immediately after the second plot is read by the accumulator. On the trailing edge of the write clock from **E1**, the write counter is incremented to address the next write address, and the unit reverts to read-mode. This process continues until all plots have been processed, which is signified by a reference range of 4095 being output from the plot memory. This value is initially generated by the interscan signal, which forces a memory-write cycle at the end of the sweep and thereby prevents the unit from processing garbage data from the unused portions of memory.

If **E1** initiates a new plot the data is passed back to the memory in exactly the same way as an established plot returning from the accumulator, with **E1** generating the appropriate timing signals to insert the new plot into the correct range order in memory. In this situation, the read-address counter is not incremented so the next plot in range order is still passed out to the accumulator/recirculator.

If a plot is completed, then the data returns from the accumulator/recirculator as before, but no write clock is generated. Instead, the timing circuits generate an output latch clock, which latches the data into the output registers. The write address therefore remains unchanged and the completed plot is effectively removed from memory. It should be noted at this point that the two reference co-ordinates are multiplexed together as are the range & azimuth sums - this simplies the next stage of processing on board **E3**.

3.5.4 The Plot Accumulator / Recirculator

The circuits on board E2 (Figure 3.10c) are responsible for the updating of existing plots with new increment data (∂R and ∂N) and then returning them to the memory on E5.

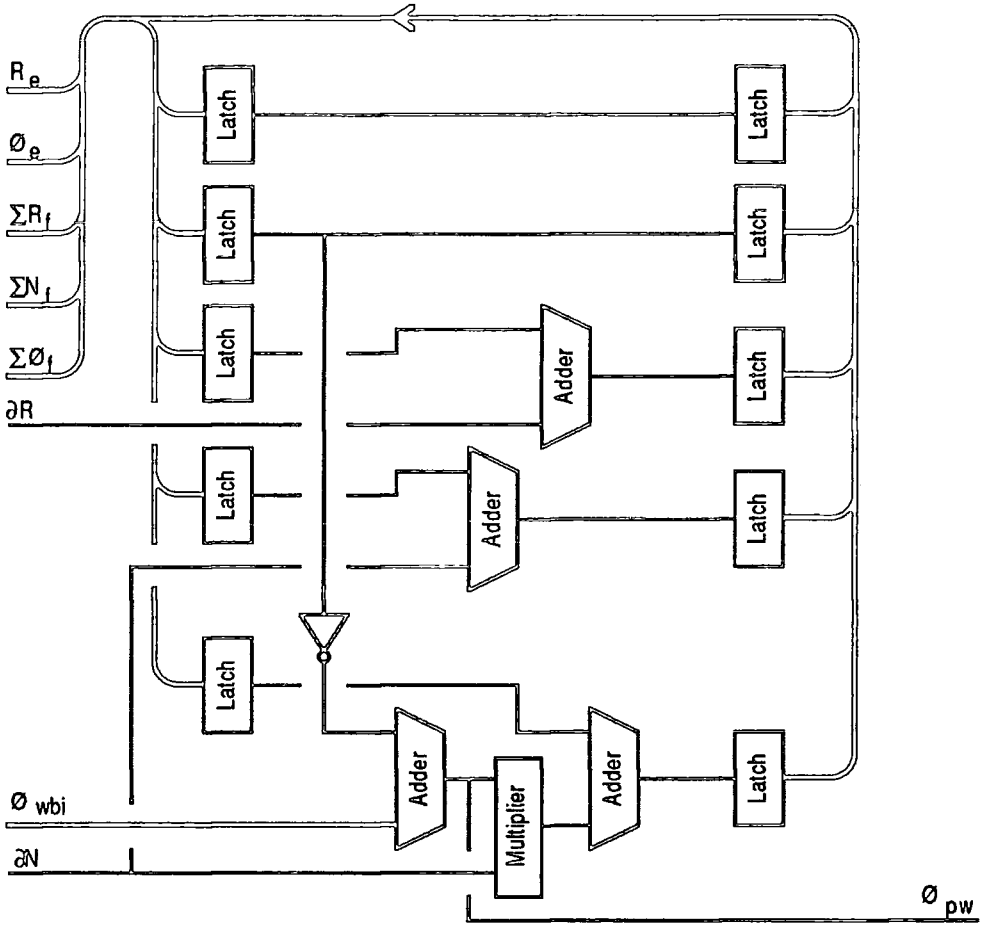


Fig. 3.10c : Plot Accumulator / Recirculator

Data transfer occurs across the 56-bit wide bi-directional data bus, which links the three main stages of the plot-forming circuits, with access controlled by the timing circuits on board E1.

At the start of the sweep (Section 3.5.3), the first plot in range is loaded into the latches ready for processing.The reference bearing ($Ø_e$) is negated and then added to the current bearing ($Ø_{wbi}$) to give the current plot width ($Ø_{pw}$). The plot increments (∂R & ∂N), from board E1, are then added to the accumulated range moment and active cell counts respectively (ΣR_f & ΣN_f). The incremental azimuth moment is then calculated by multiplying the number of active cells on that sweep (∂N) by the distance from the reference bearing ($Ø_{pw}$) and this is then added to the the accumulated azimuth moment ($\Sigma Ø_f$). The new plot data is then latched and transferred back to the plot memory as described in the previous section.

3.5.5 The Plot Calculator

In the final stage of plot forming, the plot calculator (board E3 and Figure 3.10d) evaluates the centre-of-plot co-ordinates from the accumulated sums and the reference data.

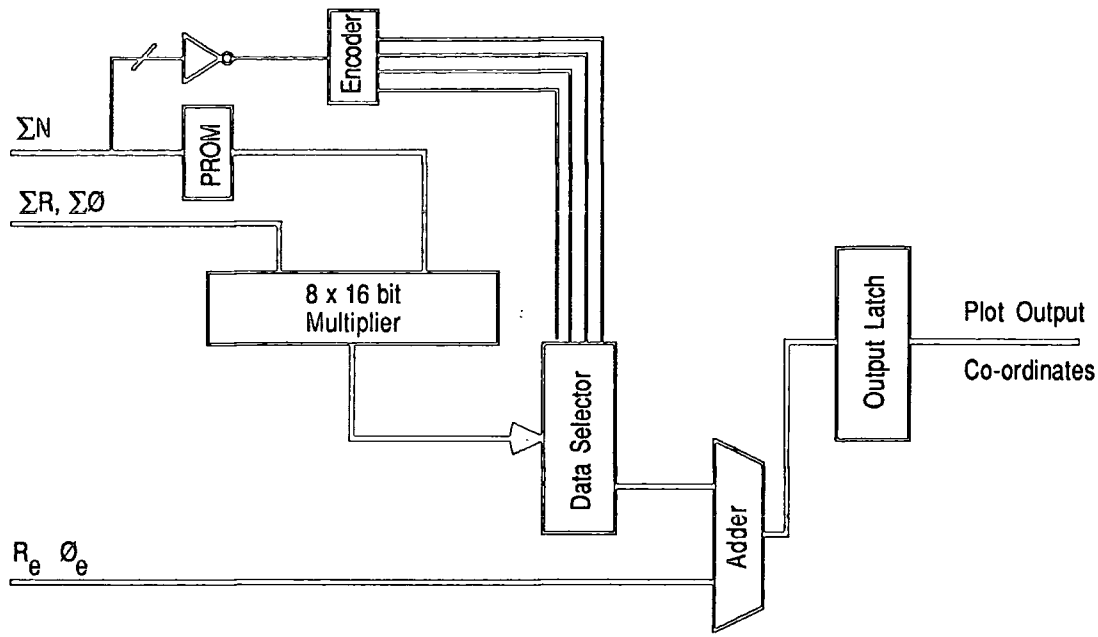


Fig. 3.10d : Plot Calculator Schematic

The PROM is used to calculate the reciprocal of the accumulated active cell count, by way of a look-up table, and this value is then applied to the multiplier. The bearing data is processed first, and the accumulated bearing moment ($\Sigma Ø$) is multiplied by the value $1/\Sigma N$, as described in Section 3.5.1.. The output from the multiplier is unscaled, and this feature is provided through the data-selector, which selects the outputs from the multiplier to produce a correctly scaled fixed-point result. This value is then added to the reference bearing ($Ø_e$) to give the azimuth centre of plot, which is then latched in the output register.

Under control from the plot timing, the range value is then processed in an identical manner: accumulated range moment (ΣR) is multiplied by $1/\Sigma N$, the output is scaled and added to reference range (R_e) and then latched in the output register. Further timing signals ensure that the final stage of the plot forming processor reads the azimuth co-ordinate before the range co-ordinate is latched.

3.5.6 Plot Output Buffer

The final stage of the plot forming process involves storage of the plot co-ordinates prior to their transmission to the filtering and tracking computers and consequently board E4 is essentially a large RAM (1024 x 16 bits) with associated address counters. The output of the buffer is fed, under control from the computer interface, across to the SPF computer and to

allow for the difference in data rates the RAM is configured as two separate banks, each with its own address counter. This circuit therefore emulates a large First-In-First-Out (FIFO) buffer, and additional circuits are included to handle events such as buffer overflow and data-ready strobe - Figure 3.10e gives the schematic of the complete plot buffer.

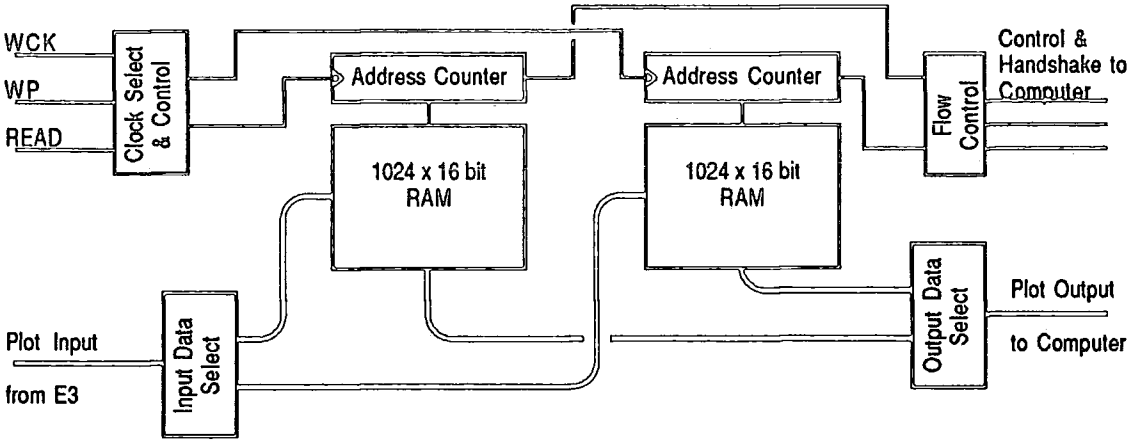


Fig. 3.10e : Plot Buffer Schematic

In operation, the two counters are both cleared by the master-reset which is generated by the system timing boards. As with the plot-forming memory, a flip-flop selects the bank of memory to be used initially for writing and the other bank is selected for read-out of the accumulated plot co-ordinates. Plot data is then sent from function **E3** and enters one RAM under control of the write clocks, which increment the appropriate counter after each data item is written. This process continues until the circuit receives the end-of-octant marker from the system timing boards, whereupon the rôles of the two memories are reversed, and the output handshake control takes over for transfer of data to the computer.

To transfer data to the computer the **Stop** flag is firstly removed and, providing there is data available for transfer (address counter>0), the **Octant Demand** flag is set. This signals the interface circuits to transmit the header words across the data bus, and then data is transferred from the RAM under control of the **read** pulses. The address counter is decremented after each value is transferred, and this process continues until the counter reaches a value of zero, at which point the **Stop** flag is set and the interface circuits terminate the data block. This RAM then remains inactive until the next octant trigger.

If a block of RAM becomes full before the next octant pulse is received, the control circuits raise the **Inter Octant Demand** flag, which stops further plots from being written and requests the interface to execute a transfer. If the interface is able to accept this request, data is then transferred in a similar manner to a normal octant transfer, except that no header information is transmitted. If unable to acknowledge, then an overflow status is indicated to the tracking computers, and plots are lost.

3.6 The System Control and Timing Functions

In order to maintain a degree of synchronisation between the various functions of the plot extractor, an essential feature is the provision of global clocks and timing signals from a central source. This feature is provided by the two G series boards, one of which handles range synchronisation functions whilst the other is concerned with azimuth processing.

3.6.1 Range Controller

Function G1 is responsible for the global timing at range clock level, and therefore all the main system clocks are derived from this (the range clock frequency being the highest in the system). A 16MHz generator is used to provide the base frequency with 2MHz, 1MHz, 500KHz and 250KHz being derived directly from this, through use of the divider network [X2]. The actual range clocks used in functions A1, B1, B2, C1 & D1 are derived from this master frequency through use of a programmable divider, since their frequency is dependant upon the radar type which has been selected by the operator. Programming is achieved through use of a PROM which supplies the preset-load value to the counter, depending upon the radar type. The output of the counter is synchronised to start with reception of the incoming radar sync. pulse (initially buffered in an opto-isolator) and re-timed it to a 1:1 mark:space ratio, before it is made available to the functions mentioned above.

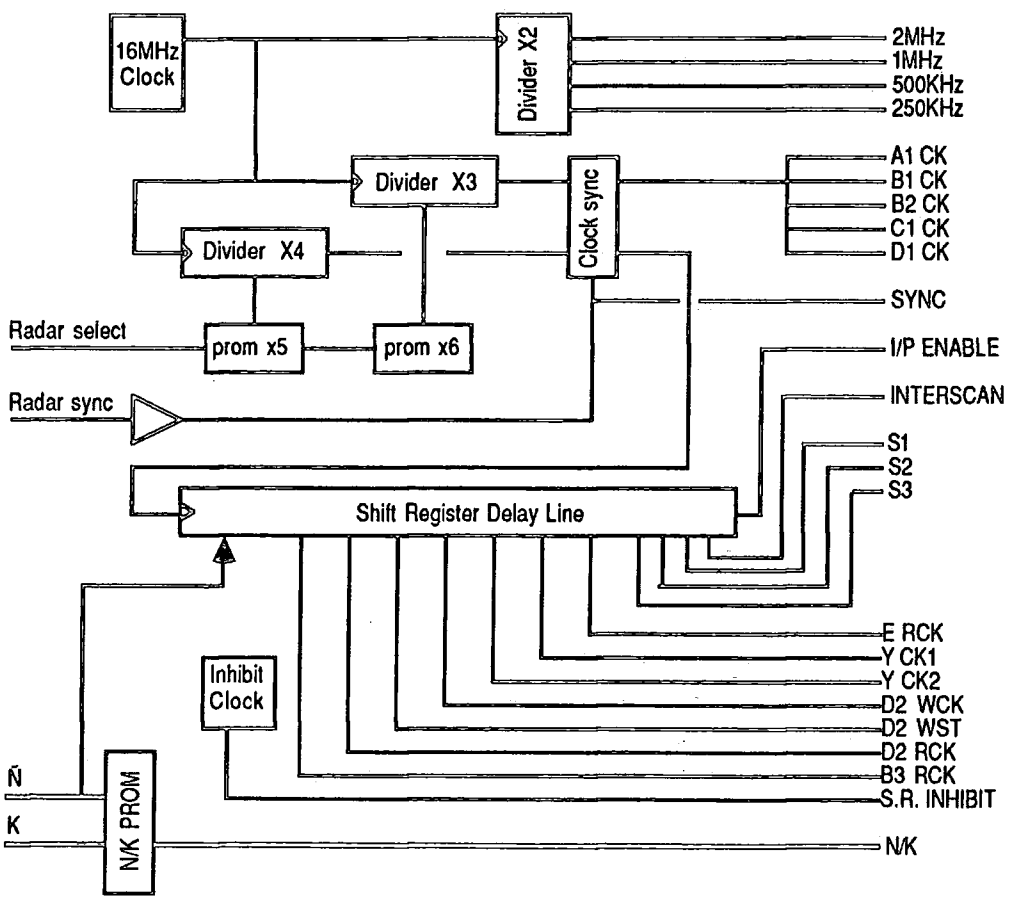


Fig. 3.11a : Range Controller Schematic

A second PROM is used to load a range counter with the number of cells per sweep for the selected radar type, and this is then used to time the sweep against the derived radar clock pulses. Through use of gating, a further series of range clocks are produced for distribution to the functions E1, Y6, D2 & B3. These are delayed in order to keep their destination functions synchronised with the video passing through them - the greatest delay occurs in the variable length shift register on function B1, so the delay line (the shift register array) is programmed to take account of this. The shift register also produces signals for Input Enable (to B1), Interscan, S1 S2 & S3 (to the display driver) and a short range inhibit for the E.C.M. function.

Also located on board G1 is a look-up PROM which decodes the value of N/K for use in the Within Beam Integrator (Section 3.4).

3.6.2 Azimuth Controller

The azimuth controller ensures that the plot extractor is kept in step with the true azimuth of the rotating antenna, since this rarely rotates at a fixed speed being susceptible to the effects of wind, precipitation and ships-movement. Synchronisation is achieved through use of two outputs from the antenna - Aerial North and Aerial Turning Clock (see 3.2), which are buffered and retimed before being used. **Figure 3.11b** shows the schematic of function G2.

The aerial turning clock is made up from 4096 pulses per revolution, and this is initially retimed so that transitions occur on an edge of the 1MHz system clock. Its output is then applied to a 12-bit counter, which produces the system azimuth count. To prevent problems with the azimuth changing during a sweep, such as may be caused by the antenna momentarily changing speed, the output from the counter is passed through a latch, which is clocked by the sync signal from board G1, before being transmitted to the rest of the plot extractor. The output from the latch is also taken to an adder which subtracts a value corresponding to the delay introduced by the W.B.I. circuits. This ensures that the targets declared by the within beam integrator are correctly positioned in azimuth. The delay approximates to (UL/A) pulse repetition intervals, and can therefore be evaluated as:

$\frac{UL}{A} \times \partial\emptyset$ angular units, where $\partial\emptyset$ is the change in bearing after each sweep.

One PROM is used to evaluate (UL/A) and a second to provide the increment ($\partial\emptyset$) which depends on the radar type selected- these are then applied to a multiplier and inverter to give the correct value for addition (i.e. subtraction) to the antenna bearing.

The true antenna bearing is partially decoded to provide the octant trigger, which occurs 11.25° after the octant boundary, making the computerised tracking of targets easier in the areas surrounding the octant boundaries. The octant value is similarly extracted at this point, and is delayed by one octant plus 11.25° to simplify the tracking software.

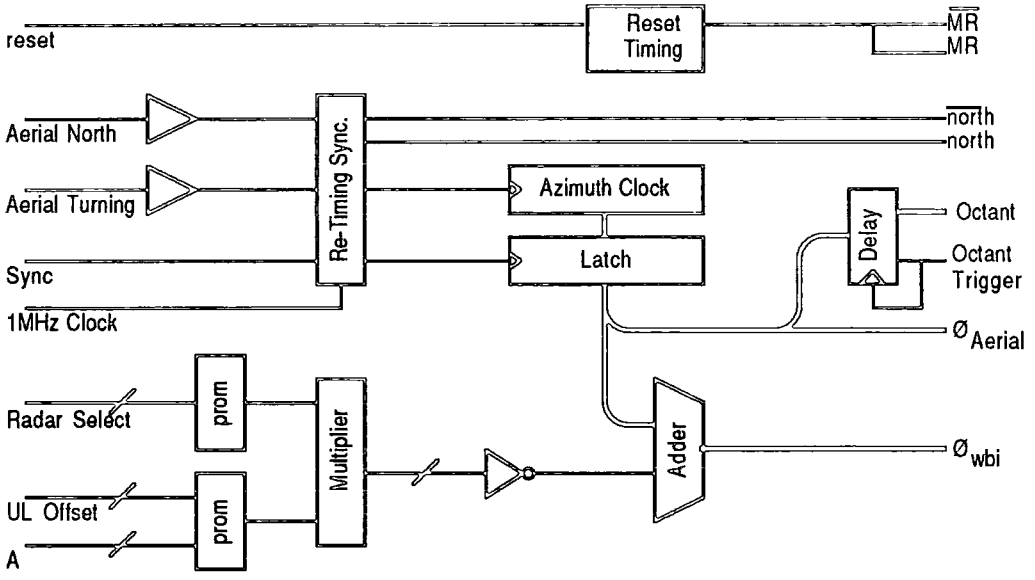


Fig. 3.11b : Azimuth Controller

The north marker signal is buffered and retimed to coincide with a falling edge of the aerial turning clock, and its output transmitted to the rest of the system. It is also used to reset the azimuth counter, so that bearing is synchronised with the north marker signal.

The azimuth controller also holds the circuit for the master reset, which is capable of being generated both at power-on and by way of a switch on the system control panel. The reset function is simply a monostable which ensures that adequate timing (2.5μs) is allowed for a complete reset.

3.7 The Interface to the Computer

This section makes up the link between the plot extractor hardware and the tracking computers, and in practice is divided into 2 parts - one based in the extractor itself and the other in the base of one of the computers. The section based in the extractor is made up from four F-series boards while that in the computer consists of five H series boards; both are necessarily complex - a requirement of the interface standard which applies to the link to the computer, even though the information content and features they provide are quite limited.

3.7.1 Component Sections of the Interface

For clarity, the following conventions will be observed in the description of the interface. The **Computer Interface** is made up from the four F-series functions and controls the link from the plot extractor hardware across to the computer. The **Plot Extractor Interface** consists of five H series boards and is responsible for the link from the computer's ME62 DMA highway to the level of the computer interface.

The sections communicate across a 25-bit bus, using the signals defined in **Figure 3.12a**, the **Computer Interface** using those in **Fig. 3.12b** to control access to the plot extractor. Note that in the context of the interface controllers, the word **output** implies a signal sent *from* the computer, while **input** describes one transmitted *to* the computer.

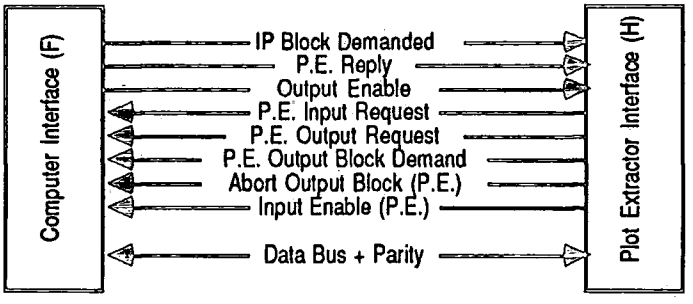


Fig. 3.12a : F-series <-> H-series Interface Connections

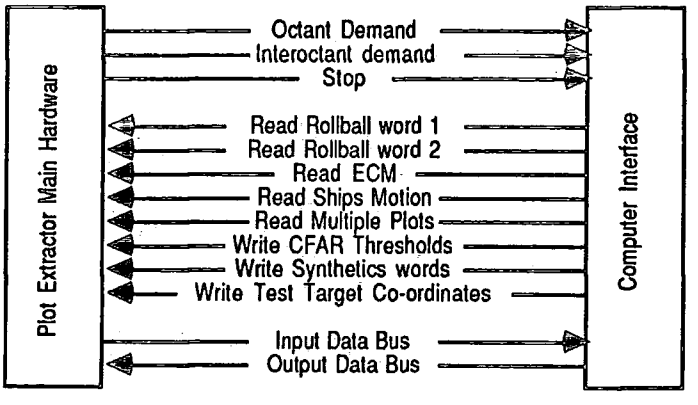


Fig. 3.12b : F-series to Plot Extractor Wiring

In addition to the control signals which cross the link between computer (F) and plot extractor (H) interface, a 17-bit wide bi-directional data bus is also to be found, permitting 16-bit data plus single bit parity transfer. The entire physical link is established using twisted-pair cable with suitable line drivers located in the F and H functions.

3.7.2 Operation of the Computer Interface

Plot co-ordinates are generated by the plot-forming processor, as described in Section 3.5.5, until either the end of an octant is reached or the plot buffer is full, whereupon the **Octant Demand** flag is raised causing the computer interface to set the **Input Block Demanded** line. The plot extractor interface replies by setting **P/E Input Request**, causing a status word to be read from the extractor. **P/E Reply** is then set by the computer interface and this latches the word into the Plot Extractor Interface buffer.

The process of:

P/E Input Request --> P/E Reply

is repeated, reading more status information words followed by the accumulated plot co-ordinates until function E4 raises **Plot Buffer Empty**, whereupon **P/E Input Block Demanded** is reset by the computer interface. The exact format of this block is shown in **Table 3.2a**. If the transfer was initiated by the plot buffer becoming full during an octant, the status words are not transmitted and the format follows that shown in **Table 3.2b**.

word	data
0	Data word count (max 1031)
1	Octant Identifier (id bits 0-2, Octant transfer bit 7)
2	Radar I.D. (bits 0-2 type, 3-4 C-Range 5-6 C-Scale, 7-8 Display Mag)
3	Roll ball X
4	Roll ball Y
5	ECM region 1 (bits 0-9 angle, 10-15 width)
6	ECM region 2 (" ")
7	Ships Motion (bits 0-9 heading, 10-15 speed)
8	1st Plot range
9	1st Plot bearing
10	2nd Plot range
11	2nd Plot bearing
...	

3.3a

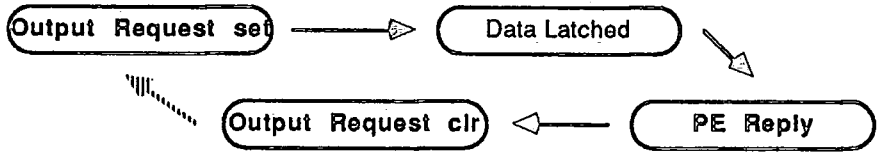
word	data
0	Data word count (max 1025)
1	Octant Identifier (id bits 0-2, Octant transfer bit 7)
2	1st Plot range
3	1st Plot bearing
4	2nd Plot range
5	2nd Plot bearing
...	...
1025	

3.3b

Table 3.2
Octant I/p Block Format (a)
Interoctant I/p Block Format (b)

Transfer of an input block from computer interface to plot extractor interface is usually followed by an output block transfer in the opposite direction. A similar method of handshaking is employed to perform this function, with data using the format in **Table 3.3**.

The Plot Extractor Interface starts by setting **P/E Output Block Demand** to which the Computer Interface replies with **Output Enable**. The data is then transferred by the sequence:



The first three words are stored in latches but the remainder are stored in a 1024x16-bit RAM in the computer interface because of the need to distribute the data amongst several functions during the present time period.

word	data
0	Word count (maximum 1027)
1	Number of synthetics marker words (max. 512)
2	Number of first threshold values (max. 506)
3	Octant identifier (bits 0-2)
4	1st Test Target range (bits 0-11)
5	1st Test Target bearing
6	2nd Test Target range
7	2nd Test Target bearing
8	3rd Test Target range
9	3rd Test Target bearing
10	1st Synthetics Marker word 1 (bits 0-9 range, 10-15 character code)
11	1st Synthetics Marker word 2 (bits 0-11 bearing)
12	2nd Synthetics Marker word 1
13	2nd Synthetics Marker word 2
...	...
522	1st Threshold value (bits 0-6 data, 7-15 address)
523	2nd Threshold value
...	

Table 3.3 : Output Block Format

The 506 CFAR thresholds are transferred in bursts of 32 words during interscan, synthetic markers in 8 bursts of 64 words during scan and test target co-ordinates in a single burst of 6 words. Control of this distribution is handled by the Computer Interface itself.

At the other end of the link, the Plot Extractor communicates with the computer using the resources of the ME62 Direct Memory Access (DMA) highway. Control signals for this connection are shown in **Figure 3.13.**

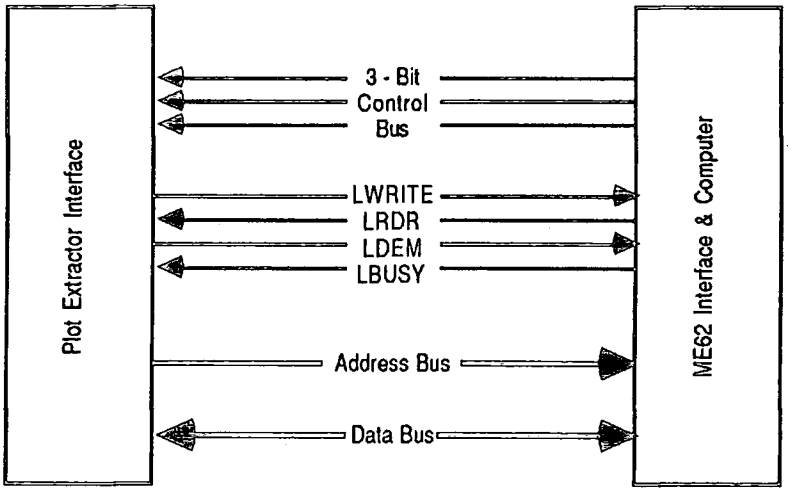


Fig. 3.13 : ME62 Interface Connections

3.7.3 Operation of the Plot Extractor Interface

Data flow for this part of the interface is under complete control of the computer which transmits control words across the 3-bit control bus. All transfers begin with an **Initialisation** sequence which resets the Plot Extractor Interface to a known state before an Input or Output transfer can take place. At the end of the transfer the link is cleared ready for the next initialisation sequence to take place.

Initialisation

For a transfer to take place the computer must write either a **Go-Input** or **Go-Output** control word which signals the Plot Extractor Interface to set either **Input Enable** or **Output Block Demanded** if a transfer is not already in progress.

Once this has been acknowledged, a start address is set up on the address highway, using hard-wired values in the Plot Extractor Interface, and **LDEM** is activated. The ME62 replies with **LBUSY** and then **LRDR** which clocks the address into a separate buffer in the Plot Extractor Interface and also into a start address store and address counter. **LRDR** and **LBUSY** are then resetin turn and the initialisation sequence is complete.

Output Transfer

Following the initialisation sequence, the Plot Extractor Interface sends the start address to the ME62 once more and the computer replies with the word count for the impending data transfer. **LRDR** this time latches the count into a word counter.

Words are then transferred one at a time to the Plot Extractor Interface using the **LRDR/LBUSY** latch sequence. After each transfer the data is passed on to the Computer Interface using the **P/E Output Request / P/E Reply** sequence and the word counter is decremented. This continues until the counter reaches zero whereupon the Interface proceeds to the **Link-Reset** routine.

Input Transfer

After the initialisation sequence, the Interface sets the **P/E Input Request** line to the Computer Interface, which replies with a pulse on **P/E Reply** and valid data on the highway. The pulse clears the Input Request line and simultaneously latches the data into a buffer, at which point the address counter in the Plot Extractor Interface is incremented so that it now accesses the appropriate storage location in the computer's memory. By setting **LWRITE** and then **LDEM** (which the ME62 responds to through **LBUSY**), the data is then transferred from the latch into the memory, ready for the next word transfer. The process continues until all data has been transferred, whereupon the Plot Extractor Interface clears **PE Input Block Demanded**, a signal for the Computer Interface to transmit the final data item, a word count, which is stored in the first location of the input data block (the actual data starts at the address following the word count).

Interface Reset

In the final sequence of a data transfer, the Plot Extractor Interface activates LWRITE and LDEM so as to write a status word to a preset location in the computer's memory. The word contains details of the previous transfer, coded in the following format (condition true if bit set to logic '1'):

Bit 0: Last transfer was an input block
1: Last transfer was an output block
3: Parity error detected in output block; block discarded by Interface
4: Parity error detected between P/E hardware and Plot Extractor Interface
5: Parity error in Start Address (Initialisation sequence); no data transferred

After completion of this sequence of events the Interface is ready for a further data transfer, under control of the host computer.

As with other sections, a brief description now follows concerning the hardware operation of each board, with the exception of the line drivers which are identical for both F and H functions. The details therefore commence with the most important function in the Computer Interface section: board F4.

3.7.4 The Sequence Controller

The heart of the Computer Interface is the sequence controller, which is responsible for generating the appropriate read and write signals to access both the plot extractor hardware and the bus linking to the Plot Extractor Interface. The schematic of this function is shown in Figure 3.14a and can be seen to consist primarily of a pair of 4-bit counters, which are activated following reception of either **PE Input Request** or **PE Output Request** from the Plot Extractor Interface.

In operation, the circuit is initialised by either **Octant Marker** or **Plot Buffer Full** (input request), or **PE Output Block Demanded** (output request), which starts the operation of counter A. The output of the counter is taken to a decoder which produces the reply signal **Input Block Demanded** (input block) or **Output Enable** (output block), and counter B is then incremented.

For an input block, a count of 1 in A activates counter B's decoder which enables the octant data onto the bus, while an output block results in a signal to F2 to latch the "No. of synthetics words" value into memory. Counter A then reaches a value of 2, which produces a pulse on the **P/E Reply** line. At a count of 3, counter B is disabled and a count of 4 causes A to be reset back to zero in readiness for the next initialisation (as above).

Upon receipt of the next P/E Input Request or P/E Output Request, the sequencer is activated as described above, but counter B is now incremented to a value of 2 which enables other data onto the input bus, or reads the "No. of threshold words" from the output bus. Subsequent requests read or write other data items, by activating different control lines dependant on the value in counter B.

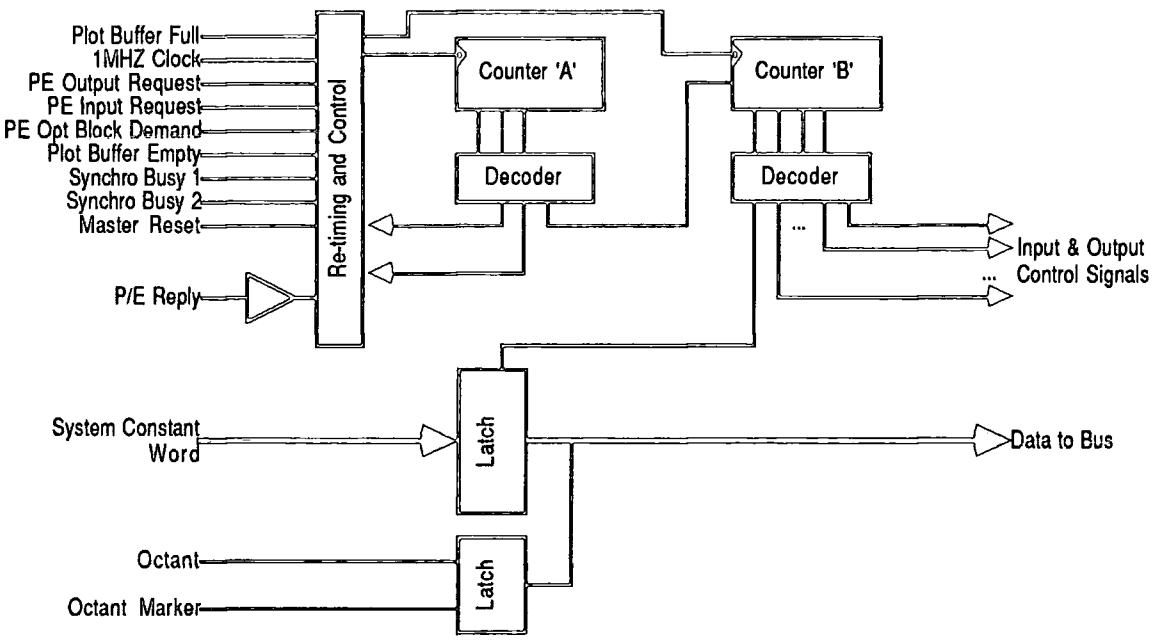


Fig. 3.14a : Sequence Controller Schematic

3.7.5 Output Data Storage

The sequence controller ensures that the correct control signals are generated to enable data to be read directly from the plot extractor hardware onto the input data bus, but transfer of output data has to be carried out in specific time periods, and it is therefore necessary to install a buffer between the bus and the hardware. Function F2 provides this feature under control of the sequencer on F4, and its schematic is shown in Figure 3.14b. The memory provides 1024 x 16-bits of storage and is capable of storing the maximum of 506 threshold values in addition to the 512 synthetics marker words and 6 test-target words.

In operation, the sequence controller ensures that the incoming data words are distributed appropriately: the "No. of synthetics marker words" is loaded into S Counter, "No. of threshold values" into the C Counter, "Octant identifier" into the octant latch and subsequent data words into the memory. The RAM is addressed by the Address Counter which is incremented after each word is written to memory, until all values have been read from the data bus.

Once transfer of the output block is complete, a comparator detects that the Address Counter has a value equal to the sum of "No. synthetics marker words" (S), "No. of threshold values" (C) and binary 5. This produces an output which instructs the sequencers on function

F3 to start re-distributing the data amongst the plot extractor hardware functions. As data is read from the memory onto the hardware bus, the Address Counter is decremented, producing comparator outputs at values of (S+5) and (5) until a value of zero is reached. These outputs are redirected to function F3 where they are used to control the destination of the data.

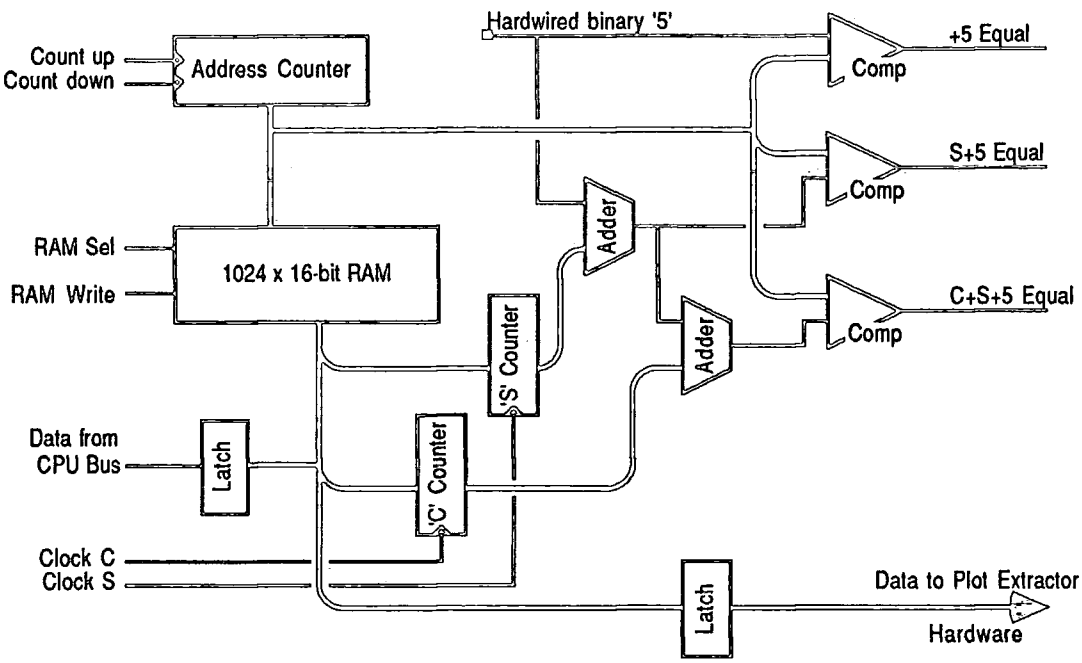


Fig. 3.14b : Output Data Storage Function

3.7.6 Output Bus Control

The third main function of the Computer Interface is responsible for re-distribution of data from the memory on board F2 to the plot extractor hardware. Like F4, the controller is essentially a sequencer producing timing signals for latching data into appropriate destinations, and its schematic is shown in Figure 3.14c.

As mentioned in Section 3.7.5, this section is triggered by the final data word being read into the memory on board F2, whereupon the control circuits are enabled, ready to transfer the threshold words to the CFAR memory. As this transfer can only take place during interscan periods, the control circuits are configured to enable the Threshold Burst Counter only when Interscan is true and thus data is moved in a maximum of 16 blocks of 32 words each.

When S+5 is activated by function F2, the control directs the data to the synthetics memory in the display driver. On this occasion, the transfer can only occur during scan periods, so the Synthetics Burst Counter is only enabled when Interscan is false and this takes 8 scan periods to transfer the 512 words to memory.

Following receipt of +5 from F2, the control directs the final 6 words to the Test Target generator, in a single block. The control circuits are then ready to process the next output data block from the computer.

Function F3 also holds the circuits for checking the parity of data which passes across the bus, using a standard ODD-parity generator and checker. In the event of a parity failure, an indicator is illuminated on the operator's console and a signal sent to the computer to communicate the error.

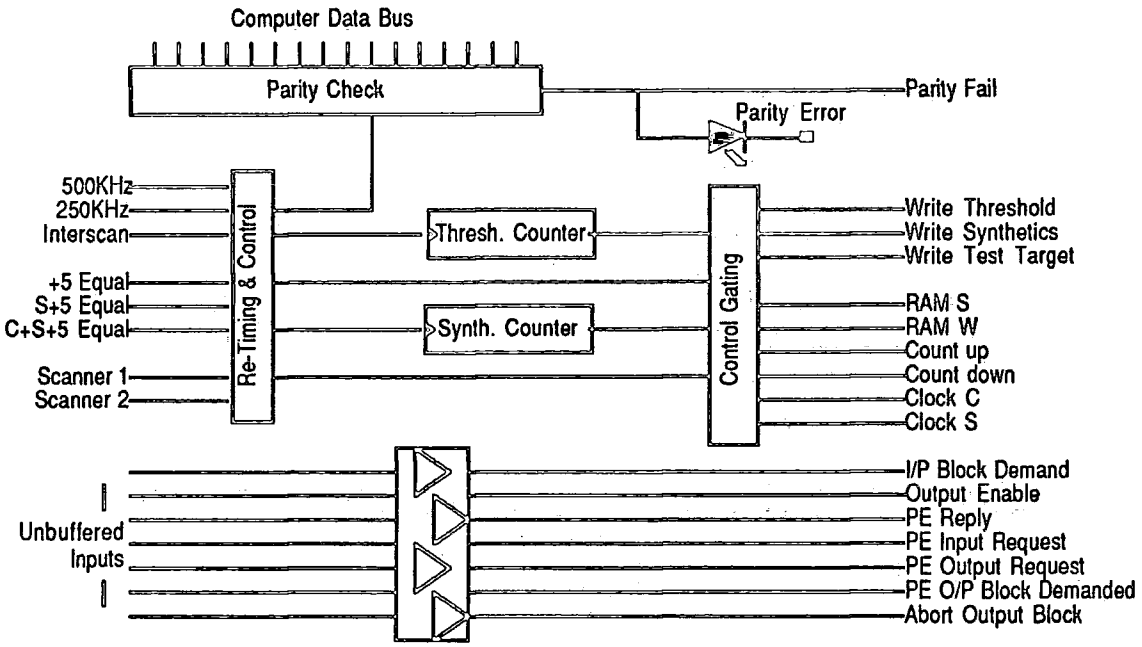


Fig. 3.14c : Output Bus Control Schematic

It should be noted that the buffers on F1 provide for the data bus, while those on F3 provide isolation for the *control* signals transmitted to and from the Plot Extractor Interface.

3.7.7 The Plot Extractor Sequence Controller

Function H1 in the Plot Extractor Interface performs in the same manner as the Computer Interface sequence controller (F4), but controls the link between the Interface data bus and the computer's own ME62 interface. It's basic operation sequence has already been described in Section 3.7.3, and the schematic of this circuit is illustrated in **Figure 3.15a.**

Function H1 is controlled from a 4-bit counter which has its outputs decoded to 8 possible operations:

1 Resets the control circuits to a known state
2 Sets P/E Input Request (for an input block)
3 Clocks the address counter on H3 (for an input block)
4 Sets "Data Valid" on H2 and H3
5 Activates signal LDEM to ME62 bus
6 Clocks the address counter on H3 (for an output block)
7 Sets P/E Output Request (for an output block)
8 Resets the sequencer

The counter is triggered by either "Go-Input" or "Go-Output" from the computer and then counts at 2MHz through the sequence given above.

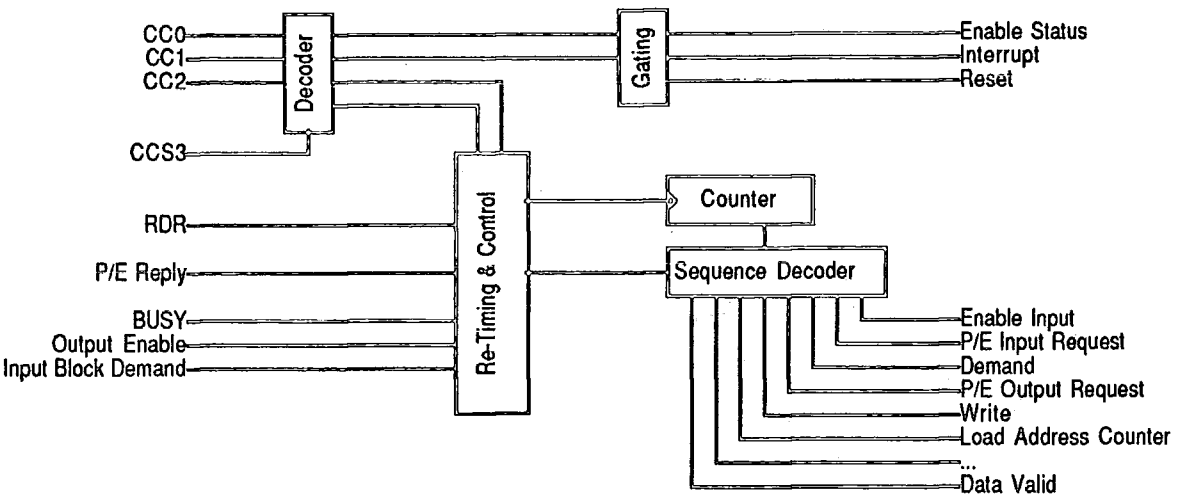


Fig. 3.15a : Plot Extractor Sequence Controller

3.7.8 Data Multiplexer

The Data Multiplexer schematic is shown in Figure 3.15b, and this function is held on board H2 of the Plot Extractor Interface. Its primary function is to multiplex data onto or off the ME62 data bus: Input blocks from the Computer Interface are checked for correct parity, and the value of the parity flag transferred to the computer, as part of the status word, at the end of the block. The value of the word counter is also transferred to the computer, just before the status word.

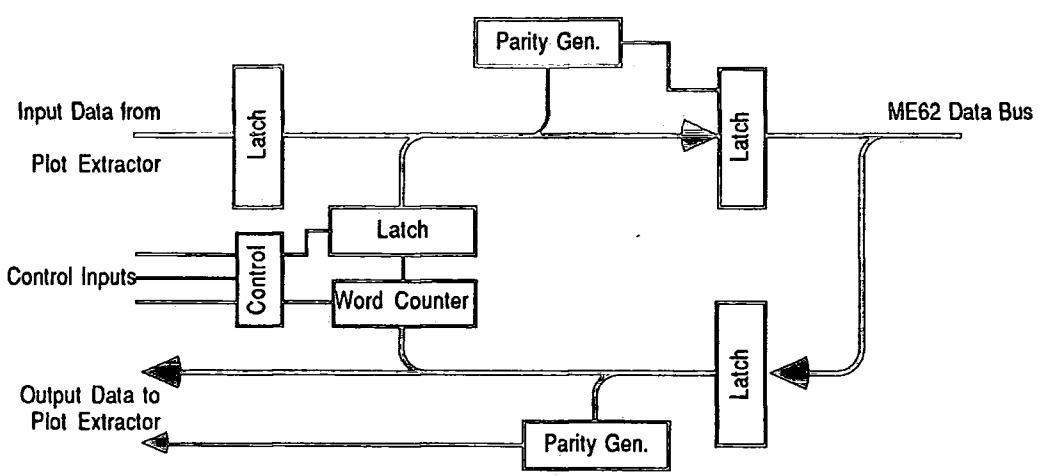


Fig. 3.15b : Data Multiplexer Schematic

For an output block, the data is first loaded into the latch under control of LRDR, and then on to the bus leading to the Computer Interface. Its parity is checked against that provided

by the ME62, and a single parity bit passed on to the Computer Interface. In the event of a parity failure, the appropriate signal (Read Start Address Fail or Abort Output Block) is generated by the circuits on this board. The output block also carries the word count from the computer, and this is loaded into the Word Counter where it is decremented after each output word transfer.

3.7.9 Address Generator

This function (H3) generates both hardwired and incrementing addresses for output to the ME62 address bus. Its schematic is shown in Figure 3.15c, and its operation can be described as follows:

The Address Counter is loaded from the data bus following an Output Data sequence, and this is then incremented during each word transfer to point to the next output word in computer memory.

For input transfers, the board also holds the Start Address latch, which stores the memory address of the first input word (for use end the end of the transfer to store the word count). Hardwired addresses are presented to the address bus through additional buffers, and include the Input Start Address, the Output Start Address and the Status Word Address.

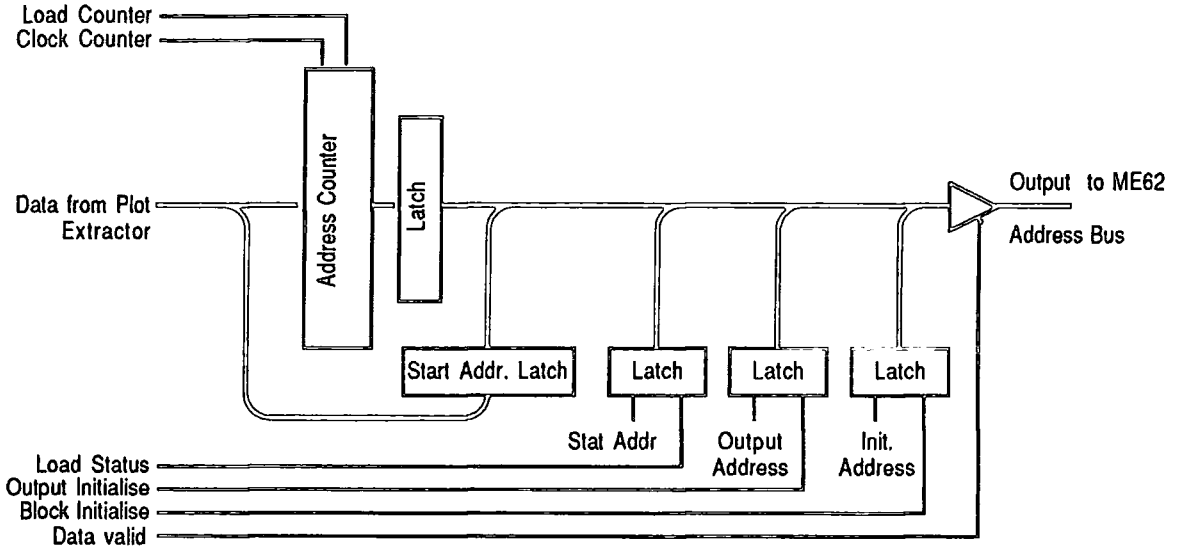


Fig. 3.15c : Address Generator Schematic

3.7.10 Control Interface

Function H4 of the Plot Extractor Interface provides the interfacing circuits for the control signals of the ME62 bus. All signals are tied to +5 Volts, through pull-up resistors, and those which are used for control of the Plot Extractor Interface are buffered prior to re-distribution amongst the other functions. As this function is relatively simple, its schematic has not been shown.

3.8 The Electronic Counter Measures Detector

Part of the RATES specification includes a requirement for some means of detecting the occurence of electronic counter measures (ECM) operation. In general, ECM manifests itself as a large area of a particularly high clutter level, and has the effect of masking anything which occurs at a greater range from the radar. The detection approach adopted for the RATES system looks for a high clutter output from the CFAR detector, which is sustained for a certain beamwidth and it produces co-ordinate details of this activity for transfer to the tracking computers. The schematic of function C1 is shown in Figure 3.16, and its operation can be described as follows:

The pre-range CFAR average, calculated on board B1 is taken by the ECM detector and compared with a level threshold which is set manually from the control panel. For every range cell where the background average exceeds the preset value, the **Down Counter** is decremented by 1 until it either reaches zero, or **Interscan** is activated. Once the end of the sweep occurs, the **Range Cell Threshold** is re-loaded into the counter, ready for the next sweep.

If the counter has reached a state of zero before the end of the sweep, the shift register loads a value of 1 from the left, otherwise a value of zero is loaded from the right. The 4-bit output of this device is masked with the **Spoke Threshold** and ECM activity declared if detected on 2, 3 or 4 successive sweeps. Once declared it remains active until a corresponding number of 'clear' sweeps have been detected.

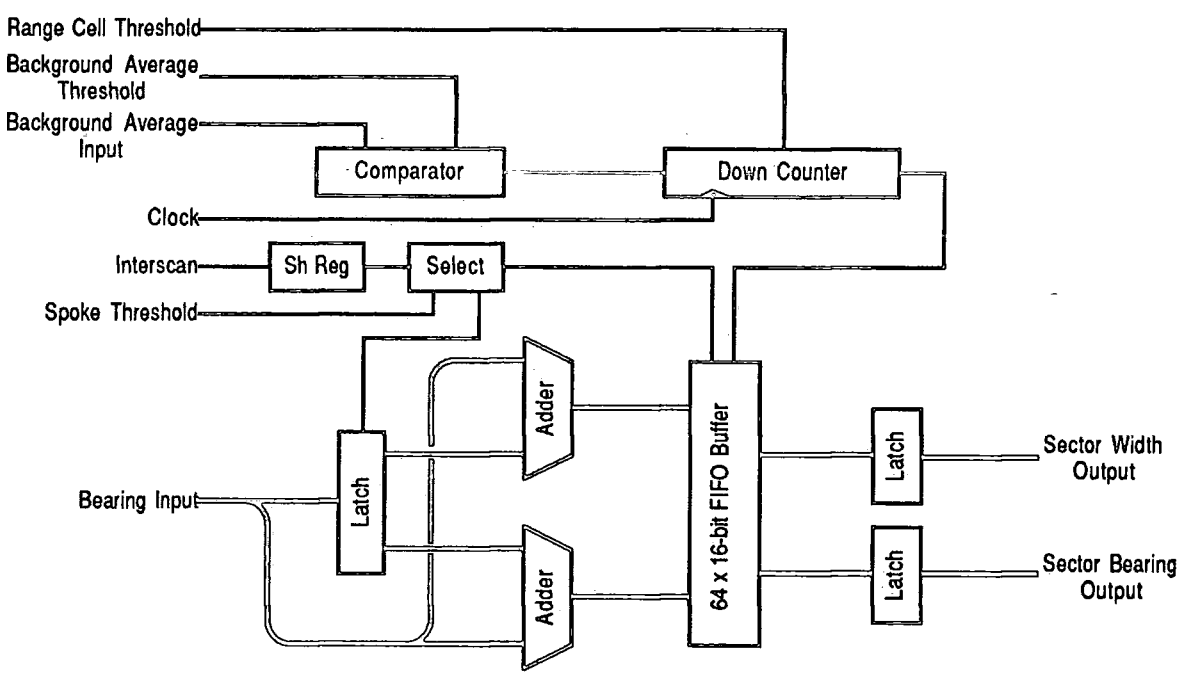


Fig. 3.16 : The ECM Detector

Initial detection of ECM latches the antenna bearing into registers, and this is subtracted from the current azimuth to give the width of the ECM sector. A second adder sums the opening and current azimuths, then divides by 2 to give the central azimuth for the sector. Once the sector is closed (either by close of activity, or a width of 11.25° being detected), the sector width and central bearing are loaded into the 64 by 16-bit FIFO buffer, to await transfer to the Computer Interface.

Although straightforward, this circuit provides the operator with the ability to detect the most common forms of electronic counter measures, such as may be generated by dispersed chaff, or a high amplitude transmission. It is not intended to discuss the ECM capabilities further, as there was no specified requirement for enhancements to the ECM detectors and no satisfactory means of testing existed.

3.9 The Display Driver

The display driver is included both for testing purposes and because the target system would probably require some form of local monitoring, even though not detailed in the original specification, and a detailed description of this function now follows.

3.9.1 Display Driver Options and Capabilities

The driver is designed to interface directly to the Hewlett Packard model HP1321A vector-graphic display, with the addition of TTL Blanking, 4-bit Z-Modulation and Long Persistance Phosphor options. It will, however, interface to other displays, with minor modifications to the original circuit (see section 4.2.1). This display requires the following input characteristics:

X-Deflection : 1.5V max. into 50Ω

Y-Deflection : 1.5V max. into 50Ω

Z-Modulation: analogue 1V max. into 50Ω

digital 4-bit binary, standard TTL levels.

With this type of display, the driver can operate in any one of four modes (Table 3.4) with each mode displaying three separate overlays. This is done by dividing each pulse repetition interval into 3 distinct display phases, and allowing the operator to have a degree of control over what is shown in each phase. The operator is also given control over the intensity of each display phase, to enable comparison of the video signals resulting from each section of the extractor and the whole display may be magnified by 1, 2 or 4 times if required.

Mode	Rea-time Sweep	Flyback Sweep	Flyback Vectored
A	Raw video	Compressed CFAR	Synthetics video
B	Raw video	Compressed WBI	Synthetics video
C	CFAR video	Compressed WBI	Synthetics video
D	WBI video	Compressed CFAR	Synthetics video

Table 3.4 : Permitted Display Modes

The video inputs to the display section are as follows:

1 Analogue video (from A1), 0-1V into 75Ω. Up to 5MHz bandwidth.

2 CFAR video (from B2). Bit-serial data, up to 4096 bits per p.r.i. plus strobe.

3 WBI video (from D1). Bit-serial data as for CFAR.

The driver is also capable of generating the synthetics marker characters shown in Table 3.5, from co-ordinates and character codes supplied by the tracking computer. The co-ordinates are supplied as two words, of which the first carries the 12-bit azimuth and second the 10-bit range and 6-bit character code. The Rollball Marker, however, is generated directly by the hardware which decodes the rollball inputs, and is always the last synthetic character to be drawn.

Tentative Track	0	
Confirmed Track	1	
Plot	2	
Rollball Marker	3	
SPF Entry	4	
ECM Azimuth	5	
Numbers 0-9	6...15	
Test Target Expected Position	17	
Unsmoothed Plot	18	
Friendly Airborne Target	22	
Unidentified Airborne Target	23	
Enemy Airborne Target	24	
Friendly Submerged Target	25	
Unidentified Submerged Target	26	
Enemy Submerged Target	27	

Table 3.5 : Synthetic Marker Characters

The display of each type of video is marked by a synchronisation signal generated on board G1, such that S1 indicates the start of the analogue video, S2 the CFAR video and S3 the WBI video. At the start of the first display phase, one of these video signals is directed to modulate the Z-axis of the display while the real-time sweep generator produces a ramp voltage which is decoded into X and Y co-ordinates to drive the display deflection circuits. During this phase, synthetics marker information is loaded into a RAM from the computer, ready for display during phase 3, and binary video for the second display phase is logically "OR-ed" in groups to compress it, then stored in another block of memory.

In the next display phase, the compressed data is read from RAM and used to modulate the display Z-axis while a sweep-generator produces a high speed range ramp to drive the display deflection.

The final display phase reads co-ordinates from the synthetics RAM, and these are selected as direct input to the polar-to-rectangular convertor for subsequent output to the display as starting co-ordinates for the markers. The markers are generated from vectors provided by a look-up PROM, and these are combined with the current position of the cursor to draw the required symbols.

The intensity of the each display phase is controlled by a separate digital modulation input to the display and a different control is activated for each individual phase. Additional protection against phosphor burning is provided by blanking the first 128 (programmable) range cells.

3.9.2 The X- and Y-Display Interfaces

The X and Y display drivers employ essentially the same techniques to provide the analogue signals for driving the P.P.I. display, and their combined schematic is shown in Figure 3.17a. In practice, the two boards are realised as functions Y1 and Y2, and constructed on custom printed-circuit boards to reduce high-frequency effects.

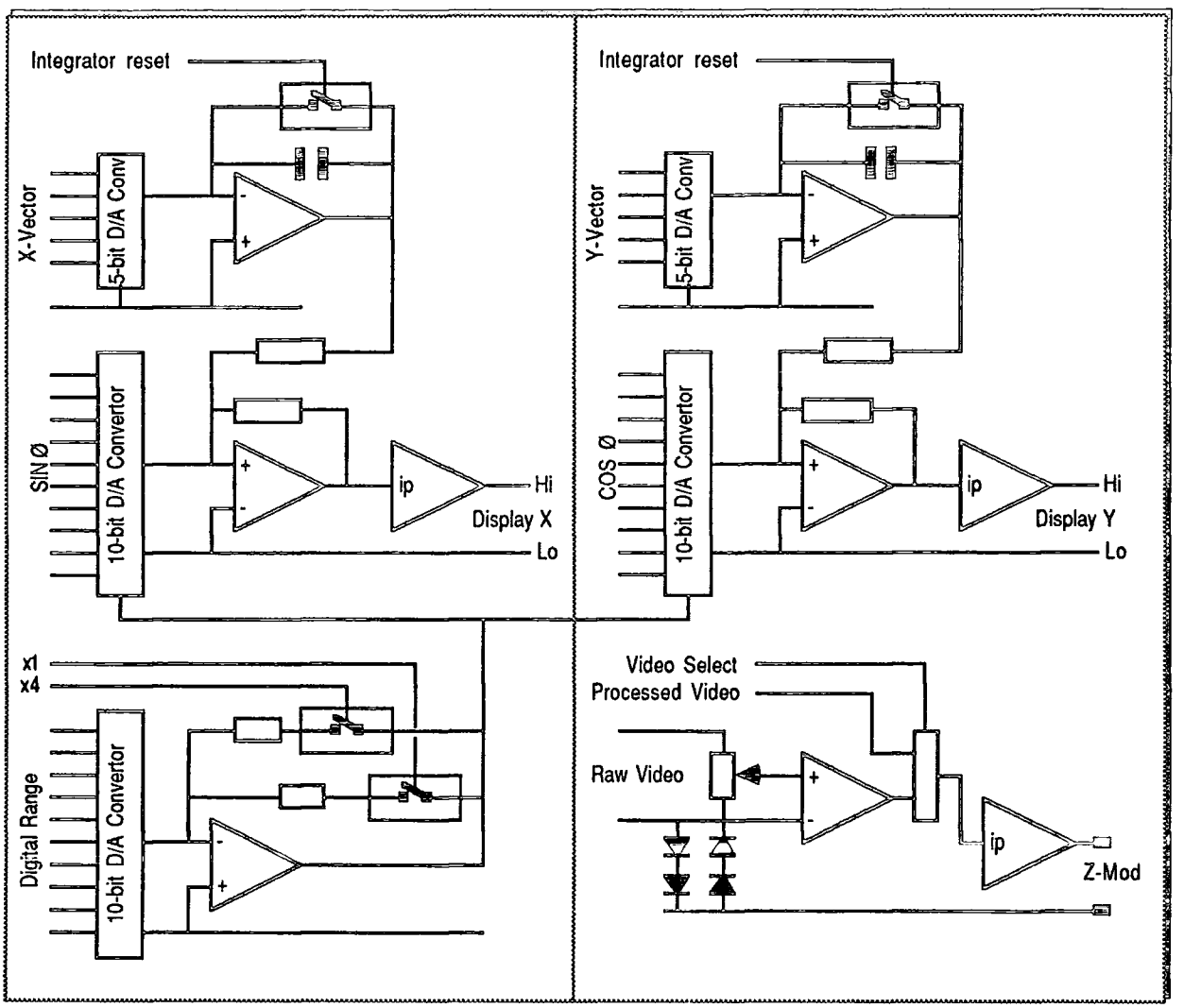


Fig. 3.17a : The X (left) and Y Display Drivers

Range information is input as a 10-bit digital signal, where it then passes into a D/A convertor with a current output. The current is amplified in the operational-amplifier which follows it and is output as a voltage. The gain in this stage is determined by the settings of the two switchable feedback resistors connected to the op-amp and allows the range output voltage to be divided by one, two or four to give X4, X2 and X1 magnification.

The X and Y deflection voltages are derived from polar co-ordinates using the expressions:

$$X = R * \sin(\emptyset) \qquad Y = R * \cos(\emptyset)$$

where R is the range and $\emptyset$ is the azimuth. The values of $\sin(\emptyset)$ and $\cos(\emptyset)$ are derived from function Y4 and input to Y1 and Y2 respectively as 10-bit +sign words, where they are converted in a pair of multiplying digital/analogue convertors to current outputs. The other input to the DACs is the range voltage derived from the circuit described above, and hence the outputs correspond to true X and Y co-ordinates.

These functions also convert the stroke signals from the synthetics marker PROMs on Y5 into analogue voltages which can be combined with the deflection signals. Stroke inputs are 4-bit plus sign, enabling a wide range of characters to be drawn, and are already formatted as X and Y increments. They are presented to 5-bit D/A convertors which convert them to currents, to be used as inputs to integrating operational amplifiers. The stroke signal for drawing each marker is therefore derived by zeroing the integrators, then summing the vectors which are output from Y5. The outputs from the integrators are combined with the outputs from the multiplying DACs in op-amps and then buffered in 50Ω driver amplifiers prior to transfer to the P.P.I. display.

Function Y2 also includes the Z-modulation buffer which takes input from one of three sources: raw analogue video, processed binary video or blanking signal, under control of a video-select signal on function Y7. The raw video is buffered in a unity-gain op-amp prior to selection and the desired output is then buffered in a 50Ω driver before output to the display.

All devices used in these functions are designed for high-speed operation to give the necessary response to binary deflection signals. Typical slew rate for the op-amps is in the order of 350V/μs (as compared to 0.5V/μs for standard devices) giving a frequency response of d.c to 400MHz.

3.9.3 The Rollball Integrator

Function Y3 contains the circuits for processing the data transmitted from the tracker ball, which consists of phase and quadrature signals for each of X and Y directions. The particular tracker ball in use on the RATES system generates 512 pulses per revolution from each of its outputs, and this must be integrated to establish the desired co-ordinates (Figure 3.17b). All inputs are divided by two in pre-scalers, then the phase input is used to decrement an 11-bit counter, while the quadrature input increments the counter. No end-stops are imposed, so the marker will appear to travel off the bottom of the screen and re-emerge at the top.

The X and Y counter outputs are passed to the computer interface through latches and also taken to data selectors which are under control of function Y7. The other inputs to the selectors are the values of sin(Ø) and cos(Ø), scaled to allow direct output to functions Y1 & Y2.

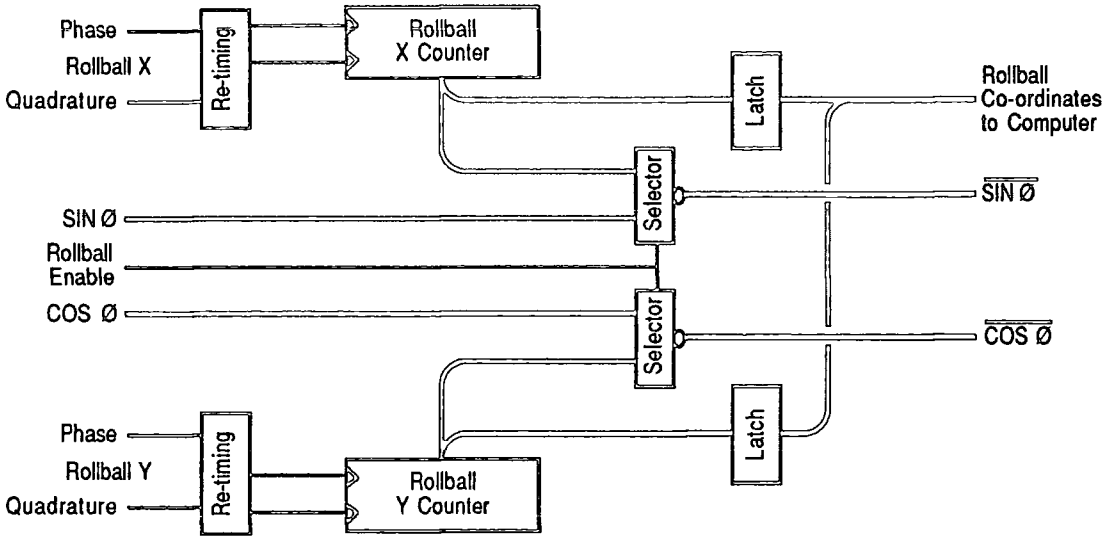


Fig. 3.17b : The Rollball Integrator

3.9.4 The Sine/Cosine Convertor

The co-ordinate system used throughout the plot extractor is polar, however this is unsuitable for the output devices which rely on Cartesian deflection co-ordinates. Function Y4 is therefore included to convert azimuth information to a more suitable format, and its schematic is shown in **Figure 3.17c.**

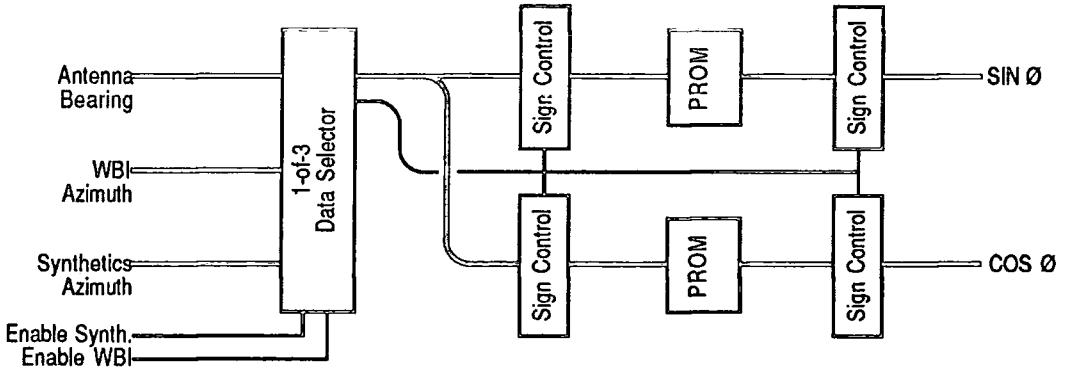


Fig. 3.17c : Sine/Cosine Convertor

The conversion to sin(Ø) and cos(Ø) is carried out using look-up PROMs which effect a single quadrant conversion, with sign control provided by additional gating. The 12-bit input is derived from either the real-time azimuth counter or delayed W.B.I. azimuth outputs on G2, or from the RAM holding the synthetics marker co-ordinates on function Y5.

3.9.5 The Synthetic Marker Generator

Function Y5, shown in schematic in Figure 3.17d, provides storage for the synthetic marker co-ordinates which are generated by the tracking computers. Two words are required for each marker, using the format described in Table 3.3, and the RAM is arranged to hold a maximum of 512 such words. Data is written during the scan period under control of function F3, and read out again during display phase 3. As each pair of words is read from the memory, the range data is fed to the data selectors on function Y6 and the azimuth data to the sine/cosine convertor on Y4.

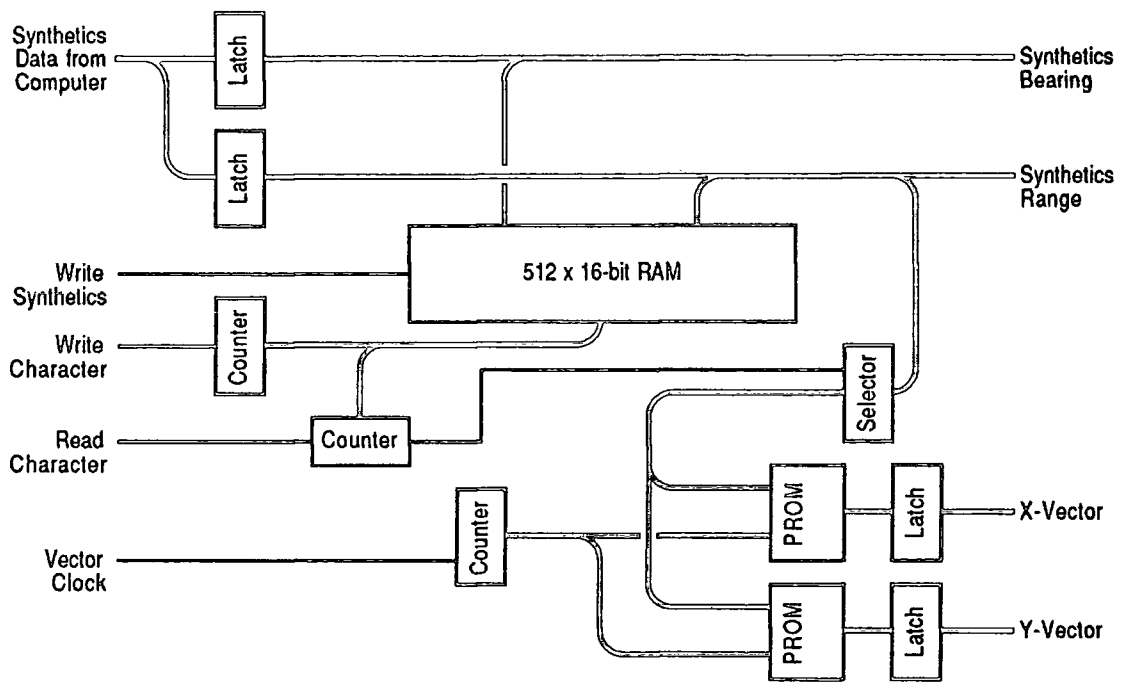


Fig. 3.17d : The Synthetics Generator

The character code, from the RAM, is fed into the X and Y synthetics PROMs where it is used to select one of the 32 markers using the upper 5 bits of each PROM's address bus. The lower 5 bits are connected to a counter which then steps sequentially through from 0 to 31, allowing a total of 32 movements to generate each synthetic marker. The outputs from the PROMs consist of 5-bits of vector information, which are fed directly to the X and Y display interfaces, a binary Z-modulation signal and a stop signal which simultaneously stops the stroke counter, resets the integrators on functions Y1 and Y2 and increments the marker address counter to point to the next synthetic marker in RAM.

In operation, not all of the markers are displayed during a single instance of phase 3, so the generation of markers is suspended until the next occurrence of phase 3. Once all 256 markers have been displayed, the address counter is incremented and points back at the first marker, ready to cycle through the sequence again.

3.9.6 The Range Controller

A special function is necessary to establish range control for the display interface because of the requirement for a full resolution scan in display phase 1 then a compressed scan in phase 2, and this is provided by the circuits on board Y6. The schematic for this function is shown in Figure 3.17e - the large number of oscillators is due to the need for compatability with all of the six radar types in any of the 4 display modes. Three data selectors are used to derive signals from the clocks - one for use in scan time, one during interscan and the third for the compressed scan in phase 2. The outputs from the selectors are directed, through appropriate gating, to an 8-bit range counter which is configured by additional circuits to function as a 10-bit counter for the duration of display phase 1.

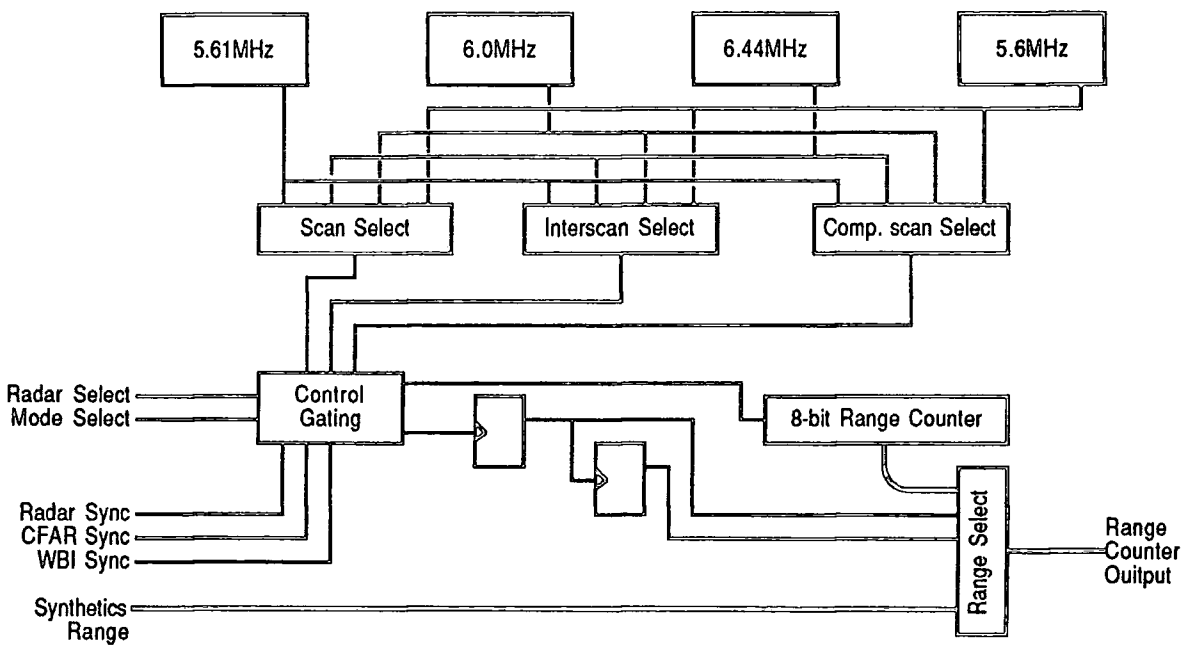


Fig. 3.17e : Range Controller Schematic

The output from the counter is directed to a data selector which, under control from Y5, selects either this or the synthetics range for output to the display interface, Y1. This board also contains the circuits for selecting the appropriate intensity level for the digital modulation of the P.P.I. display, based on the currently active display phase.

3.9.7 The Processed Video Controller

The last functional unit in the display interface is realised by the circuits on board Y7, the processed video controller (**Figure 3.17f**). In this unit, binary data from either the WBI or CFAR processors is read into the shift registers, during the first display phase, according to the display mode selected. The compressed video for phase 2 is derived by "OR-ing" 1, 2 or 4 of the shift register outputs, and the resulting video signal is stored in the RAM at the address determined by the address-counter. A second counter increments from zero during this period to provide additional timing for the display interface.

In display phase 2, the address-counter is used to read the stored video, and the output from the RAM is fed into a mixer which combines binary data from this and other sources to produce an analogue signal suitable for transmission to the Z-modulation driver on Y2. During phase 2, the second counter decrements to zero whereupon it generates an underflow condition which halts the second display phase and starts phase 3.

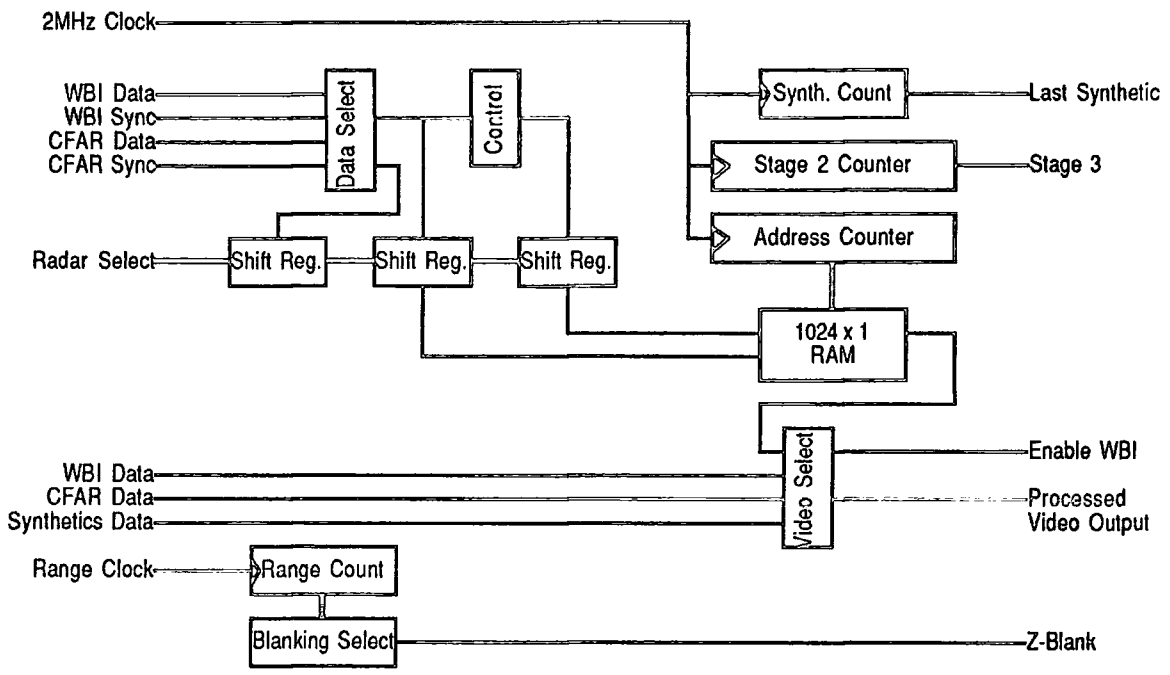


Fig. 3.17f : The Processed Video Controller

Synthetics display is timed in a third counter, and the signal **Last Synthetic** is produced when there is only sufficient time left to draw one marker before phase 1 recommences. The function also generates the video timing signals for the rest of the display interface, and the blanking signal for masking the display at short ranges.

3.10 The Control Unit

This feature of the RATES system needs only a brief mention to describe the facilities it offers to the operator. The intention in providing this facility was to enable those testing the system to choose the optimum parameters for normal operation; these would then be hardwired into the extractor to present a less complex control panel for use by the radar operators.

The parameters controlled from this function are given below and Figure 3.18 shows their typical layout, used in both prototype and Durham systems.

System Controls:

Master Reset	Generates a reset pulse for the system hardware
Radar Select	3-bit output of radar type (A...D)

CFAR Controls:

CFAR Scalar 'K'	6-bits, value 1 to 2 in 32 steps
CFAR Offset 'M'	6-bits, value 0 to 5.76dB in 64 steps
CFAR Cell Count 'N'	4-bits, sets number of cells used for background average (8...16)
Computer Range	2-bits, sets range of computer offset cells to 1, 2 or 4 mile range
Computer Scale	2-bits, sets weighting of computer offset to 22.5, 45 or 90dB

ECM Controls:

ECM Level Threshold	8-bits, value 0...255
ECM Cell Threshold	12-bits, value 0...4095
ECM Spoke Threshold	2-bits, sets spoke threshold to 2, 3 or 4

WBI Controls:

WBI Lower Level	5-bits, sets lower level threshold to 0...31 in 32 steps
WBI Upper Level	5-bits, sets upper level offset, in 32 steps
WBI Increment 'A'	4-bits, value 1...8, the integral step used in the WBI

Plot Forming Controls:

Plot Range Depth	1-bit, sets plot forming range depth to 3 or 5 cells
Plot Bearing Width	3-bits, sets plot forming bearing width to 3, 4 or 5 beamwidths

Display Controls:

Display Mode	4-bits, sets display combinations (see Table 3.4)
Display Magnification	2-bits, enables x1, x2 or x4 magnification of display
Intensity 1...3	4-bits, sets intensity of primary (secondary/tertiary) display phases
Roll-ball	X and Y phase & quadrature, for positioning of display marker

In addition to the switches on the control panel, the operator is given some visual indication of the status of the unit, through the CFAR and WBI rate indicators and a number of warning lights. The displays provide a count of the number of targets declared by both CFAR and WBI threshold processors and hence give a measure of the performance of the two functions.

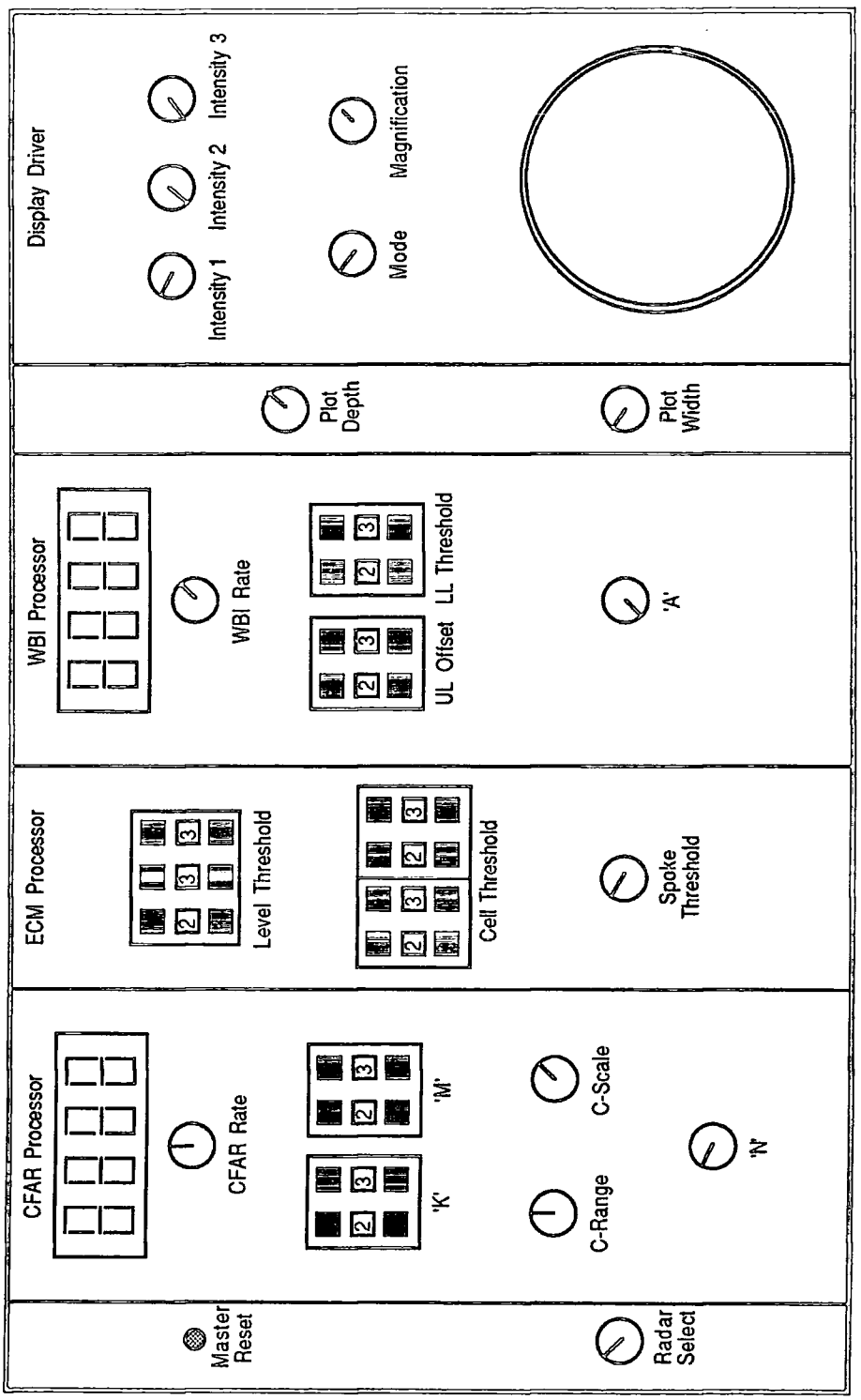


Fig. 3.18 : RATES Control Panel Layout

3.11 The Filtering and Tracking Computers

The most important requirement of the tracking computers is their ability to accept, as input, genuine plot data mixed with land and sea clutter and extract the true targets, measure their speed and heading and keep track of them from scan to scan. From this information, the computer generates a display of synthetic markers on the P.P.I. display, transmits target data across the A.I.O. and shows status information on a separate video monitor. All of this processing must take place in real time, so high-speed is an essential element in the choice of a suitable computer system.

3.11.1 The Prototype Hardware

When RATES was designed, there was little choice in a suitable computer for the prototype system: any contender had to be ruggedised to military specification (which excluded most microprocessors) and capable of high-speed execution. It also had to support the approved language: CORAL 66 and its associated run-time operating environment, MASCOT (Modular Approach to Software Construction, Operation and Testing) which provides high-level multi-tasking and process communication capabilities [34]. The prototype therefore made use of two Ferranti Argus 700G processors, one of which performed stationary-plot filtering while the other was used for target tracking. Due to difficulties surrounding data transfer between the computers, this configuration was later modified to use one as a fully operational system, while the other was used for software development. The H-series boards were installed in the base of the operational computer system, linking from the computer's ME62 interface bus to the hardware of the plot-extractor.

As the entire plot extractor was experimental, the computers were equiped with extra backup and output devices, which would not otherwise be supplied with a production system. This gave the user the option to get a hardcopy of track parameters as tracks were built-up, plot confirmed tracks on an X-Y plotter, and record tracking scenarios on floppy disc for later replay at a chosen speed. The computer displayed status information regarding data input and number of existing tracks, as well as specific information on selected tracks, on a video console attached to it. The system could be fine-tuned during operation by use of a separate terminal to change the tracking parameters.

3.11.2 Prototype Software

The original software (RATES1) was written for the two separate processors, but the difficulties in getting these to communicate successfully prompted a re-write to the standard of RATES2A - the current implementation [35]. This has been in operation for some time now, but difficulties arise concerning the amount of memory it requires, and the limitations imposed on the number of tracks that can be handled at any one time. The MASCOT activity diagram for the complete filtering and tracking software is shown in **Figure 3.19.**

3.11.3 Description of Software Function

The diagram shows the activities involved in the plot extractor software, linked by either pools (a global data area) or channels (a data stream). Activities are divided into two groups: the standard MASCOT input/output routines (in the section on the right of the diagram), and the RATES special functions. The standard functions handle interfacing with the disc drives, x-y plotter, printer and control terminal and are built into the MASCOT operating system. The special functions are described as follows:

DMA Control receives data from the plot extractor by way of the ME62 link, and function H. It then passes the data on to the tracking software, under control from the **DMA Commands Pool**.

The **SPF & Tracking** activity forms the main part of the software, and can be subdivided into a further three sections. The first reads the data from the DMA interface, and resets the work areas for a new set of calculations. The second section checks all incoming data to see if it associates with an existing plot. Plots that do not associate have a confidence count decremented while new data is used to initiate a stationary plot. All other plots have ships-motion correction applied to them.

The tracking section then takes over and identifies all plots which should be active in the current octant. New data which associates with a non-moving track is discarded, while other data is assumed to come from a manoeuvring track, so the nearest target to each is selected. The system checks for possible ambiguities in its choices for track association then calculates the smoothed position of each plot, using a simplified Kalman filter [49] algorithm. Any tracks which do not associate, along with unassociated plots from the previous scan are used to form tentative tracks, providing that the clutter level in their region is low.

The **Recording** section takes confirmed track data from the tracking software and then transfers it on to the line printer.

The **Command Interpreter** accepts data from the control console and thus enables the operator to have some degree of control over the parameters used in the various activities. The operator is also able to select a target using the rollball marker, then instruct the computer to initiate a track for the selected item.

The **A.I.O Interface** provides communication with another computer, under control of the SPF & Tracking Section. Data may also be routed to the monitor terminal through the **VDU Manager**, and this also displays the level of processing activity which is calculated by the **Loading Calculation** activity. Finally, the tracking section produces data for transfer back to the extractor, under control of the **Output Controller**. This includes Test Target data, Clutter Map Threshold Levels and P.P.I. Synthetic Display Markers.

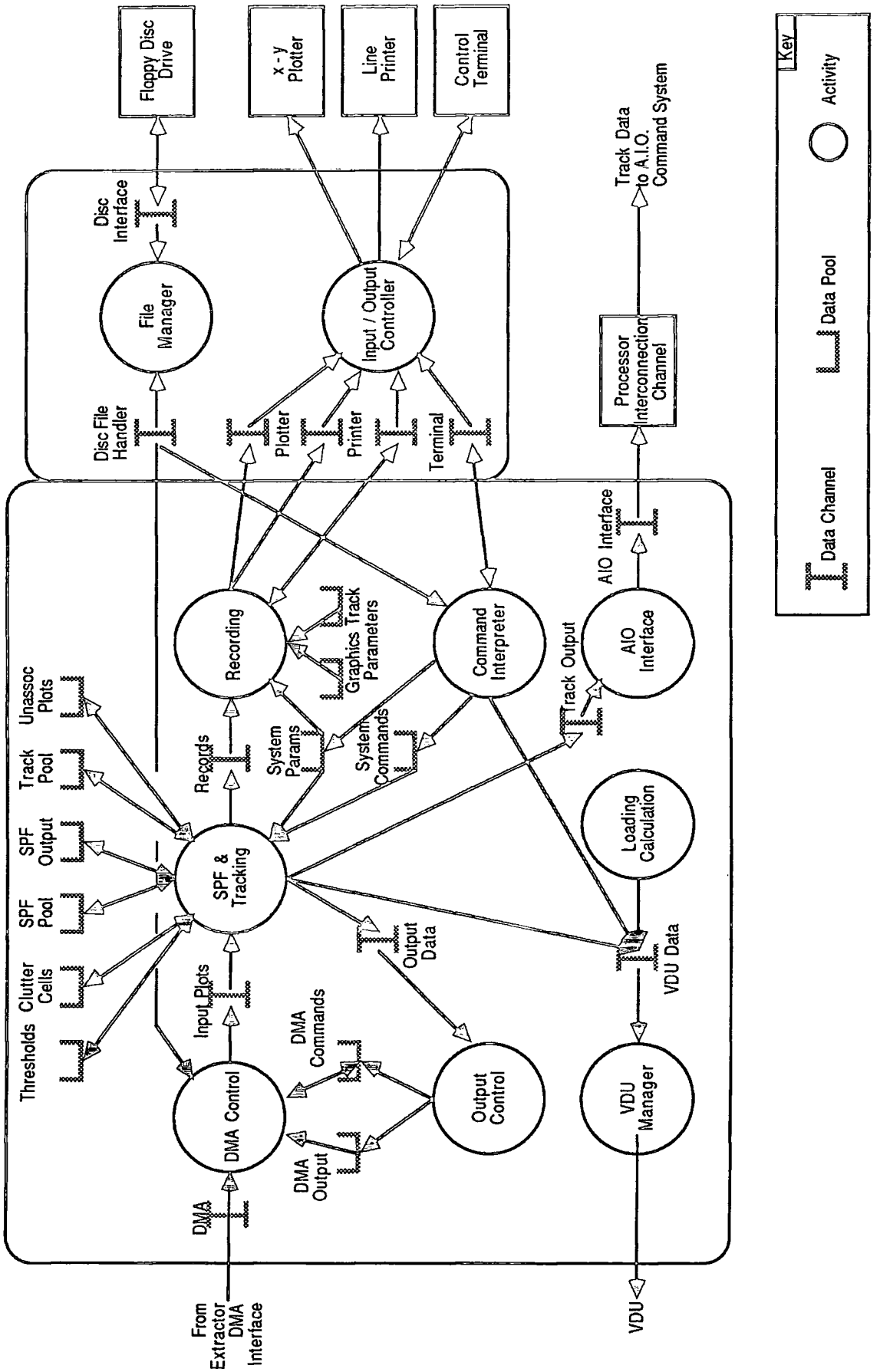


Fig. 3.19 : RATES Software, Mascot Description

4.0 Introduction

The prototype RATES unit was commissioned in the late 1970s' with the aim of demonstrating the viability and cost-effectiveness of an automatic radar track extraction system. Using the technology that was readily available at the time, namely small and medium scale integrated circuits, the prototype was assembled with little attention paid to the system's commercial potential or to the problems that could be incurred if a commercial design were to be attempted. Once finally operational, the constraints of the original design became clear, and a move was then made to redevelop the deficiencies of prototype using a technology that could be associated more in keeping with the 1980's.

The Durham University RATES Development System was introduced, not only as a practical demonstration of what could be achieved by adapting the prototype to the latest technology, but also to indicate possible avenues of research using different architectures. In this chapter, a number of possible alternatives to the existing MSI / SSI circuit architecture are proposed, with consideration given to existing VLSI circuits and components.

4.1 Deficiencies in the RATES Prototype

To justify any study of new approaches to the design and construction of the RATES prototype, it is necessary to discuss some of the failings of the original design, some of which may be explained by reference to the previous chapter:

- Use of MSI and SSI logic, whilst offering high speed processing, gives a poor performance/size ratio.

- Power supply requirements are excessive for the type of processing involved in RATES , and would present difficulties in a shipboard system.

- Computer hardware is now outdated and could be changed to use smaller & faster processors.

- Communications between plot extractor and tracking computer could be simplified without loss of performance and would probably result in a higher bandwidth.

- Use of wire-wrap construction techniques, although adequate for prototyping purposes, is not suitable for a commercial design. Also, the individual functions (A1,A2,B1...B3, etc) do not lend themselves to printed-circuit board construction as they are functionally too small to warrant this design approach.

In addition to the points mentioned above there have been a number of difficulties encountered with the tracking software, which may be explained by deficiencies in the design of the interface to the plot extractor hardware:

- Tendancy to lose track of targets which enter regions of high clutter density, cross octant boundaries or rapidly change speed (e.g. helicopters).

- Lack of reliability when faced with overload conditions (software tends to'hang' and does not recover).

- Clutter map badly implemented, resulting in excess of targets at >90% of effective range.

- Target tracking is abandoned if the target passes over the antenna.

- No checking on realistic target speeds

- Software code poorly optimised resulting in comparatively slow execution.

There is clearly scope for improvement in the design of the prototype, however any enhancements should go beyond the point of simply upgrading components. The function of the various modules in the system is not questioned, as they represent standard processing techniques for this type of radar, but the architecture used to implement them could be improved. The prototype made use of MSI and SSI circuits to realise each module, and any new design should consider the use of VLSI techniques in their place, with the aim of improving performance on a smaller scale and at lower cost.

The current design is too bulky to be of use within the confines of a ship, where the environment is more likely to favour a compact system which is still easy to maintain. The prototype system had a weight limit of 800Kg imposed upon it for this reason, and this has resulted in a design which is comparatively limited in its capabilities because of the weight penalties that would be incurred by increased functionality. One of the priorities in a new design should therefore be in reducing the physical size of the system, concentrating initially on those modules which represent the worst performance/size ratio - the computer and its interface.

4.2 Introduction to the Durham RATES

As a guide to the type of technology that was available, it should be noted that work on the project began in September 1982, with a brief to study the applications of new semiconductor products to the existing RATES prototype. An important consideration in achieving this was to gain familiarisation with the existing system and a duplicate set of wire-wrap boards was therefore provided in July '83, covering all of the RATES functions with the exception of the computer and its local interface (**Function H**), and the ship's motion convertors (**A2a/b**).

4.2.1 Plot Extractor Hardware

With the aid of the existing wiring diagrams, a suitable casing was constructed to hold the boards while preserving the grouping that was introduced in the prototype. Power supplies were wired using a heavy braided conductor, to reduce the effects of voltage drop due to the high currents that were likely to be drawn by the boards and with 43-way double sided edge connectors to interface to each board, a backplane was built-up using wire-wrap techniques, following the details on the wiring diagrams. The initial development system is illustrated in **Plate 4.1**: it should be noted that the objective at this stage was simply to produce a system which would emulate the CFAR, WBI and plot forming functions of the prototype. Care was taken to make it as easy to modify as possible as no ideas had been formulated regarding possible enhancements to the prototype.

A monochrome Kratos P.P.I. display was obtained for use with the development system, however this did not have the same characteristics as the H.P. display used in the prototype, and it was therefore necessary to design and construct an interface which could be installed between the output drivers (**Y1** & **Y2**) and the p.p.i.. Some extra features were included in the driver to enable it to interface to another type of display (a Kratos 2-colour phosphor Penetron) which was used as a backup in the event of the first display failing. The operating parameters for the two displays are given below, and the complete schematic for the interface is illustrated in **Figure 4.1**.

Display	Display Type	Blanking Input	Deflection Inputs	Video Input	Colour Select
Kratos	Monochrome p.p.i. vector display	+ 5V to display	1V / 1° deflection 0.56°/µs max. 1KΩ impedance	+1V for max. Intensity. 50Ω impedance	N/A
Kratos CM317FPT	Two colour (red, green) p.p.i. vector display	- 2V to display	IV / 1° deflection 0.56°/µs max. 1KΩ impedance	+3V for max. intensity. 0V blanks 2KΩ impedance	Logic '0' selects one of 4 icolours (red, amber,yellow,green)

The deflection sensitivity of the colour display was adjustable, so no further interfacing was included to compensate for the difference between the two displays, however it

was necessary to derive the video enable signals and a suitable offset for the colour video input. Neither display featured digital modulation and this has been derived by converting the 4-bit signal to an analogue voltage which is used to offset the incoming video level. With a maximum input for full intensity of +3V for the colour display, it was found necessary to provide a second offset which is added to the intensity modulation signal. Switches are provided on the board to enable rapid configuration for either display.

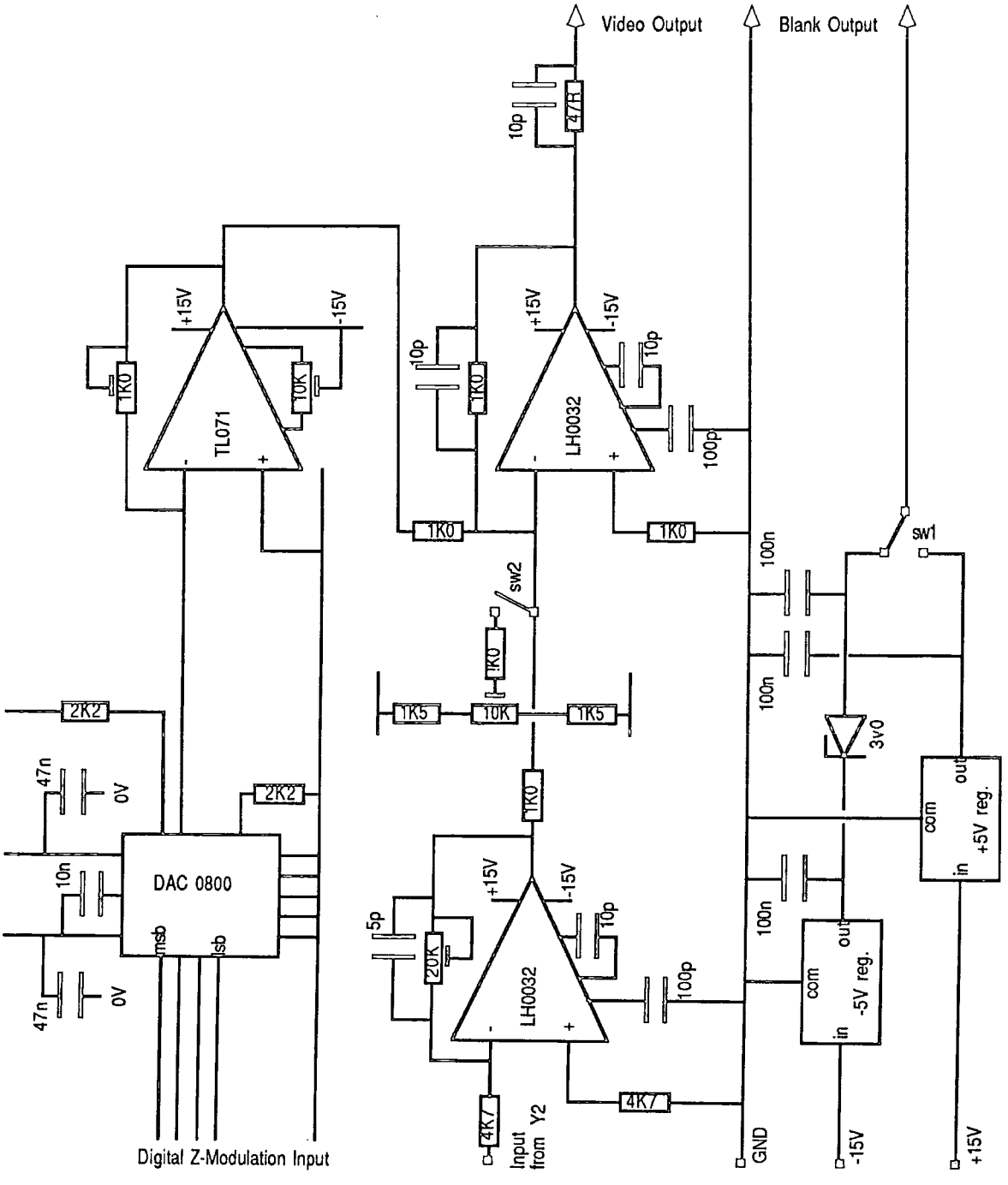


Fig. 4.1 : Schematic of the PPI Display Interface

The video input to this interface is taken from a slightly modified version of **Function Y2** - as 50Ω buffering is not necessary prior to the interface, a signal is taken directly from the output of the final operational amplifier (see **Figure 3.17a**). The output from the interface is designed to drive a 50Ω load directly fromthe operational amplifier, and no further buffering is included at this stage.

Power was supplied to the system by a smaller unit than that specified in the prototype: +5V at 20 Amp and 10 Amp, ±15V at 5 Amp and -5V at 1 Amp. This was found to be satisfactory and showed no evidence of voltage drop or overloading. A control panel was constructed in an identical manner to the prototype unit described in Chapter 3, and linked to the plot extractor by three 50-way ribbon cables. The complete display and control panel is illustrated in **Plate 4.2.**

4.2.2 Initial Test Results

Following initial testing of the completed development system, using the test target generator (**Chapter 7**) that was part of the original design, it became clear that the prototype wiring diagrams were extremely inaccurate. Checking the wiring with the circuit diagrams of the prototype indictated a total of 53 missed or incorrect wiring connections, which have now been documented and are available as a set of amendments to the original diagrams.

Further tests using the now modified plot extractor proved successful, and demonstrated the clutter-filtering properties of the CFAR and WBI sections. The system was noted to compare well with the prototype, which had also been seen in operation demonstrating these features. With no computer processor attached, the plot forming and tracking facilities could not be studied.

Plate 4.1

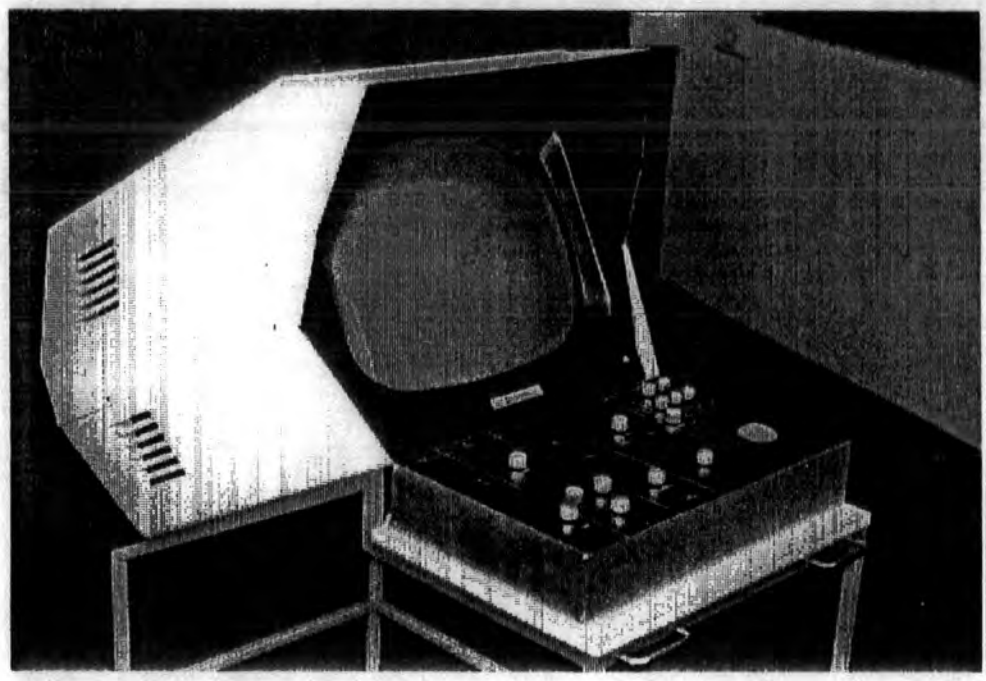


Plate 4.2

4.3 Digital Design Approach

Having gained familiarity with the prototype system, it is appropriate to suggest some areas where improvements could be made to the basic design, in particular with respect to replacing MSI / SSI parts with some sort of functionally equivalent LSI alternative. Since the prototype was designed there have been significant advances in such areas as microprocessors, bit-slice processors and logic arrays, all of which could be used to some effect in improving the RATES design. There has also been increased scope for custom design of integrated circuits and in some cases this can prove cost effective even with low production runs

Some aspects of improving the RATES prototype have already been proposed in a study, by Ferranti Computer Systems P.L.C., of the potential for VLSI in plot extractors [51], however this was concerned largely with the applications of Uncommitted Logic Arrays (ULAs) in a general purpose plot extractor.Whilst some of the findings of this report remain valid, it should be noted that several aspects have changed in recent years and it is therefore important to reconsider all possible semiconductor technologies for use within the RATES system.

4.3.1 Applicable Semiconductor Technologies

Ferranti's recommendations regarding applications of LSI / VLSI technology to plot extraction were the results of a proposal submitted in late 1981, and much of the information pertaining to this could be regarded as out of date by 1984, when studies began on alternatives to the architecture of the Durham system. Since then, a number of advances have been made and the following examples can be considered to be typical of available devices:

- Microprocessors (8 - 32 bit, clock rates to 21MHz, 32-bit register addition 0.2μs, 16x16-bit multiply 3μs).

- Bit Slice Processors (up to 16-bit slices, clock rates 12MHz, a.l.u. addition 0.1μs, multiplication 45ns).

- LSI and VLSI devices, various functions e.g. high speed, high capacity RAM; digital correlators (64 x 1 bit @ 15MHz).

- Custom LSI / VLSI logic, which is no longer restricted to the larger semiconductor manufacturers. Fairly reliable devices can be obtained with 5MHz clock rate, NMOS fabrication.

- Uncommitted Logic Arrays, also known as ASIC's, are now offered by most semiconductor manufacturers, often with local design facilities that can be run on the more advanced personal computers. Most use 1.5...2 micron gate length and require manufacture by the supplier. Typical gate delay 0.65ns, register clock rates to 170MHz.

- Field Programmable Logic, including PROMs (40ns access time), PALs (15ns delay per gate) and EEPLAs. Design tools available to run on personal computers. Programming by user through use of a special unit.

4.3.2 Parameters Applicable to Determining a VLSI Solution

In order to determine the suitability of various semiconductor processes to replacing the RATES functions, it is necessary to establish some criteria which represent a worst-case plot extractor. By adopting this approach, it is possible to retain the general purpose nature of the prototype, and should enable a final design to be easily adapted for use with any monostatic surveillance radar simply by changing the phase and frequency of the timing clocks. A summary of the RATES parameters outlined earlier in Table 2.1, together with their most important derivatives, is given below:

◦ P.R.F	256Hz ... 1600Hz
◦ Pulse Length	0.08μs ... 2μs
◦ Beamwidth	0.75° ... 2.3°
◦ Rotation Rate	7.5 r.p.m. ... 24 r.p.m.
◦ Scan Period	366μs ... 3.072ms
◦ Effective Range	30 miles ... 250 miles
◦ Number of hits / beamwidth	5...14
◦ Range clock frequency	500Khz ... 10MHz
◦ Number of range cells	256...4575

The latter 2 parameters are based on a range sampling frequency equal to (1/pulse length), however it should be noted that for long pulse lengths it is common to sample at 2 or 4 times the implied frequency. The lower bound for the number of range cells per sweep is therefore likely to be much higher (around 512 ... 1024). It should also be noted that RATES imposes a maximum of 4096 range cells per sweep resulting in a loss of processing at higher ranges, hence the effective "RATES Range" figure quoted in Table 2.1.

To allow for such a large variation in the number of range cells which must be processed in one second, a set of parameters is chosen to represent the worst case situation:

- Range Clock Frequency 4MHz
- Number of Range Cells 4096
- Pulses / beamwidth 14
- Rotation Rate 24 r.p.m.

4.3.3 RATES Functions Suitable for VLSI Implementation

Most of the functions described in Chapter 3 are suitable for implementation in LSI or VLSI form to some degree, although the ideal solution is dependent on the parameters outlined above. For the purposes of this thesis, the RATES functions will be considered as follows:

- CFAR Processor
- WBI (Marcoz and Galati) Processor
- Plot Forming Unit
- Filtering and Tracking Hardware / Software

In the Durham implementation, the display driver has also been considered to a limited extent, although the unit is not one of the most important. Supplementary functions such as ECM detector, timing and test target generation are not considered as being of sufficient importance to warrant further study or redevelopment.

4.4 A Solution in Custom VLSI?

In recent years, semiconductor design facilities have improved to the stage where a new design can be produced reliably and cheaply, even where low volumes are involved. For this reason, it can often prove highly cost effective to design circuit functions directly onto silicon, especially where the circuit forms a simple 'black box', i.e. single input and output plus a limited number of control pins.

4.4.1 Durham CAD Facilities

Durham University is fortunate in having access to the semiconductor design and manufacturing facilities provided by the SERC, as well as its own clean room where small numbers of chips can be prepared on-site, and it therefore seemed logical to investigate the potential of a custom VLSI plot extractor. Design exercises are traditionally undertaken by students reading for the M.Eng. in Microelectronics, using CAD facilities on the SERC's mainframes in the Rutherford Laboratories. A number of terminals are located in Durham, permitting on-line design and simulation using a variety of packages including HI-LO, GAELIC and SPICE. Designs are based on an NMOS substrate, using standard 2 micron rules, and when complete are sent to the SERC manufacturing facilities in Edinburgh where a small number of chips are produced.

To keep costs low, a number of different designs are combined onto a single substrate and separated later for packaging, however the expense involved in each design restricts the number of chips that can be produced during the year. For this reason, and also because of the difficulties experienced by some M.Eng students in choosing a suitable design, it was decided to offer specifications for a number of potential plot extractor-related functions to the course students. This had the added advantage of permitting time to be spent investigating other aspects of the RATES system while the designs were being prepared.

4.4.2 Functions Chosen for VLSI Implementation

The main constraint on any functions that were to be chosen for implementation in VLSI, was the design time available - a mere 12 weeks from specification to completed design. It was clearly not advisable to extend a design beyond this period by transfering the project to another student, and so a small function had to be chosen, obeying a simple algorithm that could easily be converted to a hardware implementation. It practice, it was decided inadvisable to attempt a complete plot extractor design on a single chip, as the cost benefits would almost certainly be lost through its lack of flexibility - an important factor in considering the adaptability and lifetime of the product. Maintenance, too, would be hampered by a single chip design that would necessitate discarding any defective product.

During a three year period of study, a total of three designs were submitted for implementation in VLSI. Complete reports of these implementations have not been published outside the University.

4.4.3 CFAR Processor

The first design was an attempt at the implementation of a basic CFAR processor algorithm, simplified to reduce design time. To achieve this simplification, the window size was restricted to a fixed value of 5 cells, thereby removing the need for a variable-length shift register. It was accepted from the start that the limitations of the design process would render full-speed operation impossible, so it was decided to simply demonstrate the potential of a single chip CFAR albeit running at a lower clock rate.

In practice the time limit proved restrictive and it was not possible to achieve more than a small part of the overall design - a parallel adder for summing the shift register contents.

4.4.4 Polar-to-Rectangular Coordinate Convertor

This design, which was attempted by another M.Eng student in the same year as that described above, was the only consideration given to a redesign of part of the display driver, where it was hoped that it would be used to replace most of functions Y3, Y4 and Y5. As with the first design, it was accepted that the operational speed would be below that required in practice but it was hoped that the circuit would be able to demonstrate conversion of the range and azimuth data to X and Y digital co-ordinates. It was also hoped that this approach might illustrate the potential for using raster-scan displays in place of the vector drawing P.P.I. displays, thereby giving the advantages of reduced power consumption, reduced size and improved visibility.

4.4.5 The W.B.I. Integrator

This was the most promising of the three designs attempted, taking advantage of a new process that was introduced after the first two, and offered more reliable operation at higher speeds than was previously possible. The design was a complete implementation of a Marcoz and Galati within-beam integrator, excepting the RAM used for storing the counter values. Input parameters were identical to those in the RATES prototype (Section 3.4) with the intention of reducing the WBI functions to a two-chip solution.

Regretably, a minor design error prevented more than superficial testing of the production devices. Indications suggested that correction of this fault would probably result in successful operation, but at the expense of a further chip run.

4.4.6 Summary of the Custom VLSI Approach

It was unfortunate that the design time available for these exercises was so limited, as none of the designs attempted by the M.Eng. students was found to work in practice. It was concluded that the lack of success in this approach confirmed earlier doubts about the viability of custom chip design, although the WBI processor showed potential and could possibly be made to work with further modifications. It is, however, doubtful whether the cost of such an implementation could be justified when compared to the likely cost of, for example, a ULA solution to the same problem.

In rejecting the VLSI solution, at least to the Durham system, the following points were also noted:

- The SERC process used to manufacture the chips is not wholly reliable.
- The earlier process could not support clock frequencies greater than 1MHz for the first and second designs, and the new process had not been tested sufficiently at the time of the third design.
- The process was only suitable for small runs of NMOS I.C.s. For use in a military environment, the design would have to be re-engineered for production using Ferranti's bipolar process.
- The simulation facilities available on the SERC computers were unable to test for all conditions and hence there was always a possibility of design errors being overlooked.

4.5 Programmable Logic in RATES

The term 'Programmable Logic' can now be used to refer to a large number of different families of semiconductors, although in this instance it will be taken to mean the two groups mentioned previously: 'Field Programmable devices' and 'Uncommitted Logic Arrays'.

In general, thc family of field programmable devices includes Programmable Logic Arrays (PLAs), Electrically Eraseable Programmable Logic Arrays (EEPLAs) and Programmable Read Only Memories (PROMs). The latter devices are already used to within RATES, mostly as look-up tables for converting manually-set parameters to some derivative.

EEPLAs and PLAs are essentially similar - the former retaining the ability to be programmed to some logical configuration then electrically erased again, much in the same way as the more traditional E^2PROMs.

4.5.1 Programmable Logic Arrays

PLAs have been introduced since the production of the RATES prototype, and they are now widely available as standard MSI logic devices. Each device consists of an array of (typically) 8 OR-gates, with the inputs to the gates being made up from the wire-ANDed inputs to the device. The device's outputs are taken from the OR-gates either directly or through a D-type register. The wire-AND function is performed through a set of fuse links which may be 'blown' to achieve different logic combinations.They are programmed using a variation on the technique employed for PROMs, but this can be done without recourse to the manufacturer. The devices are produced in their own series (c.f the 74xxx TTL series), with only minor variations in programming requirements between manufacturers. In considering PLAs for use in a design, the following guidelines should be considered:

- PLAs are intended primarily for replacing small - medium size *combinational* logic designs. They are not suitable for multi-function logic designs, although they may be used to implement small registered-state machines.

- Device delay is typically 12...30ns, for a non-registered PLA, whilst registered devices support a clock frequency of up to 22MHz.

- Maximum input/output support is 22 inputs and 10 outputs. The maximum package size is 24 pin (the PAL22V10 family) and this capacity is achieved by the use of bidirectional connections (all outputs are usually bidirectional).

- Most PLA designs use only 50% of the possible combinational terms because of the restrictions on I/O and outputs often end up being used simply because they were available to the designer, without consideration of more economical alternatives.

- Devices are programmed using either a standard language, boolean expression (relating outputs to inputs) or using a fuse map (the lowest level).

Taking account of these guidelines, it can be deduced that PLAs are not suitable for completely replacing a module in RATES (and certainly not with a single array), but there is potential for using them to replace some of the more laborious combinational logic. In general, a PLA will replace between 5 & 12 SSI/MSI functions (with certain restrictions). In a later chapter, PLAs have been used to advantage in the designs suggested for a new interface to the tracking computer and one design is based entirely around a PLA state machine.

Considering the individual RATES functions, there is little scope for an advantageous PLA design as other architectures could provide more suitable solutions. The WBI function, however, is essentially logic-based and could probably be reduced to a single-board function using memory, counters and a controlling PLA. This concept has been discussed in greater detail in Ferranti's report, but it is considered that current ULA devices would permit complete replacement of the WBI using a single device... possibly a more cost-effective solution in a production situation.

4.5.2 Uncommitted Logic Arrays

ULAs and ASICs currently show the greatest potential for implementing a single-board plot extractor, having advanced considerably (technologically speaking) in recent years. The number of manufacturers now providing facilities for designing and producing logic arrays has increased considerably from those originally suggested by Ferranti, and typical gate parameters have been similarly improved.

Current ULAs are mostly based around a 1.5 micron gate length and implemented in CMOS. They achieve very high gate speeds, similar to those of 10kΩ ECL (e.g. 0.65 ns through a 2 input NAND gate). In contrast to Ferranti's report of a maximum of only 4000 - 6000 gate devices by 1983, manufacturers are now offering ULAs with 41,000 gates per chip. As with all gate-arrays, however, only about 30% of the total gate count is available for use whilst the remainder is lost through implementation of bus structures and interconnections. Packaging is unlikely to restrict designs as pin counts of up to 257 are permitted in current devices.

Programming is usually performed by the manufacturer, using designs prepared by the customer. Some manufacturers (e.g. **Xilinx**) are also producing low-density arrays (2000 gates) which may be programmed in a similar manner to PLAs, but the design facilities for these is essentially the same as for a large PLA design and are based around the use of boolean expressions or fuse maps.

In this country, facilities for ULA design are offered by a large number of manufacturers. LSI Logic Corp. in particular offer a comprehensive set of guidelines to complement their extensive range of gate arrays, many of which include pre-defined macro-cells for use within a design, including functions such as memory (18...36K bits) or bit-slice processors (AMD2901). For these devices, programming is carried out through the concept of functional modules, most of which can be related directly to their 74xx series TTL equivalents. Using LSI Logic's guidelines, it is possible to produce estimates for the number of ULA gates required to implement a particular function, for example:

RATES WBI Module:

DEVICE	# Used	GATES / DEV.	TOTAL
7400	1	4	4
7404	3	4	8
7406	1	6	6
7474	2	15	30
7485	4	44	176
74109	1	21	21
74153	10	14	140
74158	1	11	11
74161	6	60	360
74174	1	37	37
74240	1	26	26
74273	1	50	50
74283	2	59	118
74390	6	86	516
		Total:	1503

The WBI memory is not included in the above gate count, but could be implemented either externally or as a part of a gate array mega-function. For the former case, the total count is within the capabilities of both low-range ULAs and the Xilinx user-programmable arrays.

This example clearly illustrates the potential of gate arrays for logic replacement. The result is cost-effective as development times can be significantly reduced if a manufacturer's design macro-library is used and there is no redesign of the existing circuits. A study of the CFAR and Plot Forming modules produces similarly encouraging results:

The CFAR module requires, in addition to a large number of registers, a 4096 x 7-bit memory for the computer threshold data and two 8-bit multipliers to calculate the threshold value. The latter is realised using four 2901-type ALUs, each of which can be built from approximately 800 gates. Excluding the memory, which must be supplied as a mega-function, the estimated gate count for the CFAR is only 7225 gates, which can be easily achieved using an array such as LSI Logic's LSA 1502.

The plot forming module uses a total of 41 74LS374 registers, which account for a large percentage of its gate requirements, and it also involves six ALU modules. It also requires a 128 x 56 bit memory (7168 bits) which can be implemented as a mega-function. As the total requirement (9241 gates + memory) is actually less than that of the CFAR, it is estimated that this module could be realised using a device such as LSI Logic's LSA 1501.

It is clear from these results that the technology exists to implement the hardware of the RATES plot extractor using little more than 3 gate arrays, although it is doubtful whether this would represent the optimum solution. Whilst compact, the design would have no flexibility outside that permitted by the use of the existing control panel, and future enhancements would be impossible without redesigning the appropriate array. For this reason, the use of gate arrays is not recommended for CFAR and plot forming units, although it is considered that they represent the best solution to the problem of a VLSI within-beam integrator (when compared with alternatives discussed later).

4.6 Microprocessors in Radar Signal Processing

Historically, the plot extraction process in a radar has been performed by a dedicated hardware unit, tailored for the task in hand, with little flexibility or potential for redesign to suit another radar. One of the most important developments in VLSI techniques over the past few years, however, has been the growth of programmable processors and studies have already been carried out into applications of these devices to radar signal processing. To date, designs have been hampered by the slow clock rates which processors can sustain and many of the "microprocessor" designs thus proposed have turned out to be microcoded units based around a core of bit-slice processors, (simply termed microprocessors by virtue of their programmability from higher level instructions).

The simplest means of introducing microprocessors, to take advantage of their low cost and flexibility, is to adopt the parallel architecture approach, similar to that developed by Polise & Moskovitz [52] and more recently by Loppnow [53]. With this type of 'bit-slice-based' architecture, a set of identical parallel processors act on the incoming data, processing it bit-wise to achieve the desired result, and achieving high throughput by way of their limited instruction capability. The cost of a complete unit produced by these methods may be lower than a discrete logic solution and flexibility similarly enhanced, but in terms of size and power consumption there is little difference between the two. In addition to this, the development times for a complex bit-slice implementation can often outweigh any advantages that may be gained from its programmability.

It is suggested that an ideal, cost-effective programmable solution can only be easily achieved through a "single chip", serial data stream architecture using either a traditional microprocessor or one of the specialised digital signal processors that have been introduced in recent years. It is therefore proposed to discuss the potential of current microprocessor techniques in implementing replacement modules within RATES.

4.6.1 Summary of Available Microprocessors

In his study of "Radar Data Processing Using Microcomputers", Ebert [31] drew a number of conclusions regarding the type of processor suitable for implementing various sections of an automatic tracking system. His solution used a combination of a special-purpose bit-slice extractor and a multi-board microcomputer for post-processing, but was restricted by the type of devices available at that time. He concluded that plot-extraction could not be performed by standard microprocessors due to speed limitations, but post-extraction processing was within the scope of any microprocessor satisfying the following criteria:

- RAM expandable to a minimum of 64K bytes
- 16-bit data word & CPU
- Large number of instructions (>60)
- Efficient instructions (bit, byte & word processing, multiply, divide)
- Low execution time
- Versatile input/output interface
- Versatile and efficient software support
- Design based on source code-compatible minicomputer

At the time of his study, Ebert noted that very few devices satisfied these criteria, however current technology is such that new microprocessors are being announced which far exceed these specifications. Many new devices are being introduced as this report is written, and hence the objective has been to illustrate the potential of existing microprocessors, with the assumption that future devices will offer improved performance.

4.6.2 Classification of Microprocessors

Early microprocessors tended to offer a standard internal architecture, with a large instruction set that typically combined basic ALU operations with a variety of data addressing modes. In recent years, more advanced processors have appeared which retain the complex addressing capability but expand the instruction set to encompass higher-level operations. The execution speed of these devices, however, has only marginally improved and for certain applications the older devices can offer a better performance than more modern processors. This is particularly true in the case of signal processing, where the complexities of the 16-bit machine's instruction set are unnecessary, and has resulted in the introduction of optimised devices, with a small instruction set and fast execution speed. The use of these devices in the RATES system has been evaluated and the conclusions are discussed later.

Traditional microprocessors can be subdivided according to a variety of parameters, such as register structure, addressing capability & architecture and availability of support circuits to enhance the microprocessor's performance. Using Ebert's criteria as a minimum, a selection of devices that are likely to be suitable for use in the RATES system is shown in **Table 4.1.**

Part Number & Manufacturer	No. Registers & width	Data Width/ Multiplexed?	Addressing Range & Architecture	Maximum Clock Rate	Additional Details
MC 68000 Motorola	8 Data 8 Address 2 other (32 bit)	16-bits Non-multiplexed	4M Bytes Linear	12 MHz add : multiply :	Asynchronous data bus. Can use 6800 peripherals.
8086 Intel	4 Data 6 Address 4 other (16 bit)	16-bits Multiplexed	1M Byte Segmented	10MHz add: multiply:	Operates in segmented or 8080 mode. I/O instructions using ports.
32016 Nat. Semi.	1 ALU 8 Other (32 bit)	16-bits Non-multiplexed	16M Bytes Linear	10MHz add : multiply	Requires special clock signals. Pre-fetch queue & co-processor interface
99105 Texas Inst.	(16 bit)	16-bits Multiplexed	256K Bytes Linear	6MHz add: multiply	Stack based machine, using different code & data areas.
uPD70116 NEC	4 Data 6 Adress 4 other (16 bit)	16-bits Multiplexed	1M Bytes Segmented	10MHz	Superset of 8086 with BCD, bit, multiply & divide operations

Table 4.1 : Summary of 16-bit Microprocessors

A study of the above details should reveal that, although representing "state-of- the-art" technology, the processing speed of these devices is low by comparison with bit-slice programmable parts. This is primarily due to the provision of large register sets which can be regarded as multiple ALU configurations, programmed using high level op-codes. Most of these processors are based on a nano-programmed instruction set, which is sequenced using the external clock with no internal timing generation, and hence many clock cycles are required to execute a single instruction. It is therefore not surprising that a 12MHz Motorola 68000 processor takes twice as long as a 6MHz, 8-bit 6502 to execute an 8-bit Add Immediate instruction. Fortunately, this observation is not upheld for 16 or 32 bit operations, and the 68000 remains a superior processor by virtue of its extensive addressing modes and high-level instructions.

4.6.3 Plot Extraction Using Microprocessors

In considering the suitability of conventional microprocessors for use in the RATES plot extractor hardware, it is necessary to take account of the typical operations required by each of the 3 main sections described in the previous chapter (CFAR, WBI and Plot Forming).

The cell-averaging CFAR requires two additions and two subtractions followed by a multiplication and a further two additions in order to calculate the average of two windows surrounding a test cell. Using the optimum processor from Table 4.1, this would suggest a total of 100 clock cycles, taking 8μs at 12MHz or 4.5μs at 21MHz, but not allowing for data-access overheads. Calculations in the CFAR must be completed within one range period, resulting in a maximum of 2μs (for radar type 'A') processing time. As it is likely that radars with such a long pulse width will be sampled at a frequency somewhat higher than this, the use of conventional microprocessors in this application would not support the required data rate, and some alternative solution must be found.

An alternative design could be used, based on a rank statistic such as that suggested by Rohling [41]. By basing this around a content-addressable data manager, there is no longer a requirement to continuously sum the cells in the window, however this would only marginally reduce the processing time, which is bound by the multiplication operation (70 clock cycles).

In spite of the considerable reduction in bandwidth achieved through use of the constant false alarm rate processor, the within-beam integrator still requires a worst-case throughput of one detection per range cell, as it is impossible to average the detection requirements over a longer period of time. A typical sequence of operations to calculate increments and test against thresholds may require as many as 17 op-codes and would suggest an instruction cycle of some 15ns if the faster radar sampling rates were to be accommodated. This is clearly impractical, as no devices exist with this specification (at present). A logical solution would seem to be offered by handling a number of detections in parallel, across a 16 or 32-bit data bus, but in practice this does not achieve any significant saving due to the complexity of the comparison operations which must be carried out.

The same restrictions apply to the plot forming process, which must be able to cope with worst-case conditions of one detection every range cell. The only difference between this an other processes is that an output need only occur (at worst) once every two range cells, due to the restrictions placed on new plots.

There is clearly potential for using programmable processors in an automatic track extraction system, if only to increase the flexibility of the design. Current microprocessors are still unable to handle the data rates present in the plot extractor front-end, although the rates are more acceptable when considering the output from the plot-forming section. It therefore

seems logical to consider their use in additional filtering and tracking operations, an aspect which is discussed in detail in Chapter 6.

The need for fast programmable processors is still apparent in many applications, and an alternative to the traditional bit-slice processor has therefore been developed. The Digital Signal Processor (DSP) has progressed from being a spin-off from research into speech processing and is now available in one form or another from most major semiconductor manufacturers. The potential of these devices in radar signal processing warrants a separate discussion in Chapter 5.

4.6.4 Bit Slice Processor Solution

The bit-slice processor has been available for many years, with only minor additions to the family during this time. They were originally intended for designing fast computer systems which could be hardware-optimised for the application in hand, and were therefore programmed at a very low level by comparison with modern microprocessors . The slices were typically produced in 4-bit data widths, but could be expanded to permit the use of any required value. Clock speeds were usually fast (10MHz minimum), but it must be remembered that many cycles were required to achieve the same results as a single microprocessor operation. Bit-slice programming is a complex and error-prone procedure which can easily absorb a large part of a product development cycle.

In contrast, modern bit-slice processors offer a slightly faster clock speed, but a considerable increase in data width (32-bit) and instruction complexity. For some devices there is now little difference between a slice and a microprocessor, although much of the programming complexity remains.

For a plot extraction system, the speeds available from current slices could easily handle the data rates required, even at the front end of the processor in the CFAR section. With the width of slices increasing this would be relatively easy to implement in hardware terms, however the software complexity would require several man-years work (judging from Durham experience with previous designs) and would be unlikely to result in a cost-effective final design.

4.7 A Summary of some other VLSI Devices

In addition to the devices discussed in previous sections, a number of other VLSI components have become available and can be considered for inclusion within the design of the plot extractor. The manufacturers of these devices are prominent in the field of radar systems, and it is therefore considered useful to draw attention to them. As these are simply suggestions for possible future use, no detailed designs have been provided. They are summarised below:

- **Plessey PDSP16330,** (Pythagoras Processor):

  This device converts cartesian data to polar format at 10MHz clock rate, using the conventional Pythagoras algorithm:

$$Z = \sqrt{X^2 + Y^2} \qquad \text{and} \qquad \emptyset = \text{ARCTAN}(\frac{X}{Y})$$

  The input data can be accepted in 2's complement or sign magnitude format, and the magnitude output can be scaled in powers of 2. The phase output represents a full 360°, thereby removing any ambiguities

- **Plessey PDSP16401,** (2-Dimensional Edge Detector):

  This may be used to determine the presence, direction and gradient magnitude of an edge within a 3x3 cell frame. The ability to sense and track a change in signal amplitude is considered to be of possible use in a variation on the conventional CFAR.

- **Marconi MA7170,** (Bit-Slice Correlator):

  A digital correlator which gives 64 stage correlation (Z_k) for a 1-bit reference (a_i) and 4-bit data sequence (X_i), at a rate of 10MHz:

$$Z_k = \sum_{i=0}^{63} a_i \cdot X_{k+i}$$

  It can be extended in both width and length to allow for longer 'windows' and a wider data path. This device was predicted in Ferranti's report, and several CFAR designs suggested using it, but it was not released until late 1986.

5.0 Introduction

The potential for using software-programmable devices in radar signal processing has long been a subject of study, but until recently has been confined to theoretical proposals or reduced-speed demonstrations. The main restriction in the past has been the comparatively low clock rate that could be sustained, combined with restrictions on the width of the data path that was supported. Recent technological advances have resulted in faster components based around a different architecture and reduced instruction set, and aimed specifically at traditional analogue signal processing applications. It is therefore appropriate to consider the possible uses of these 'digital signal processors' in RATES, with a view to their use in the plot extraction hardware.

5.1 Digital Signal Processors

The introduction of microprocessors has revolutionised circuit design over the past decade, but in spite of further advances in their design, they remain slow by comparison with hardwired logic. This restricts their use in one of the most rapidly expanding areas of interest, namely digital signal processing, where the advantages to be gained over analogue techniques have already been demonstrated in devices such as Compact Disc Players or Digital Audio Tape. An ideal solution to the problem would be a device which offered the speed of a bit-slice processor, but the instruction set of a conventional microprocessor, however one of the many reasons for the slow operation of a the latter is the vast instruction set that is a feature of this type of device. A compromise must therefore be reached between a high-level instruction set and a fast execution cycle, and the resulting Reduced Instruction Set Computer (RISC) architecture is a feature of all current digital signal processors (DSPs).

5.1.1 Early DSP Devices

The first DSP device was introduced by NEC in the early 1980's, and incorporated a 16 x 16 bit multiplier on a programmable processor. This basic design was substantially improved upon by Texas Instruments TMS32010 processor, which appeared shortly afterwards as a spin-off from T.I.'s research into speech synthesis techniques. The device has remained popular by virtue of its advanced design, including a Harvard address architecture - separate program and data areas - which is now a standard feature on DSPs.

A block diagram of the TMS32010 is shown in **Figure 5.1**; it incorporates a fast pipelined multiplier and allows for limited off-chip memory expansion. Most instructions,

including multiply, execute in a single instruction cycle which corresponds to 160ns for the fastest device currently available, and is achieved through use of a 25MHz clock, divided down internally. The features of this device can be summarised as follows:

- 16-bit instruction / data word
- 32-bit ALU / accumulator performing basic arithmetic and logical operations
- 0 to 16 bit barrel shifter
- 1.5K words of on-chip program ROM
- 144 words of on-chip data RAM
- 4K words of off-chip expansion memory
- 8 input & 8 output channels
- 6 megabytes/sec I/O to external devices across 16-bit data bus

The TMS 32010 supports 60 instructions, using a variety of direct and indirect addressing modes including such features as post-increment and post-decrement addressing. A limited number of basic instructions are pipelined together as additional operations, but in general any load operation occupies one cycle so the device is only capable of high speed processing in tasks which involve only a small number of data transfers.

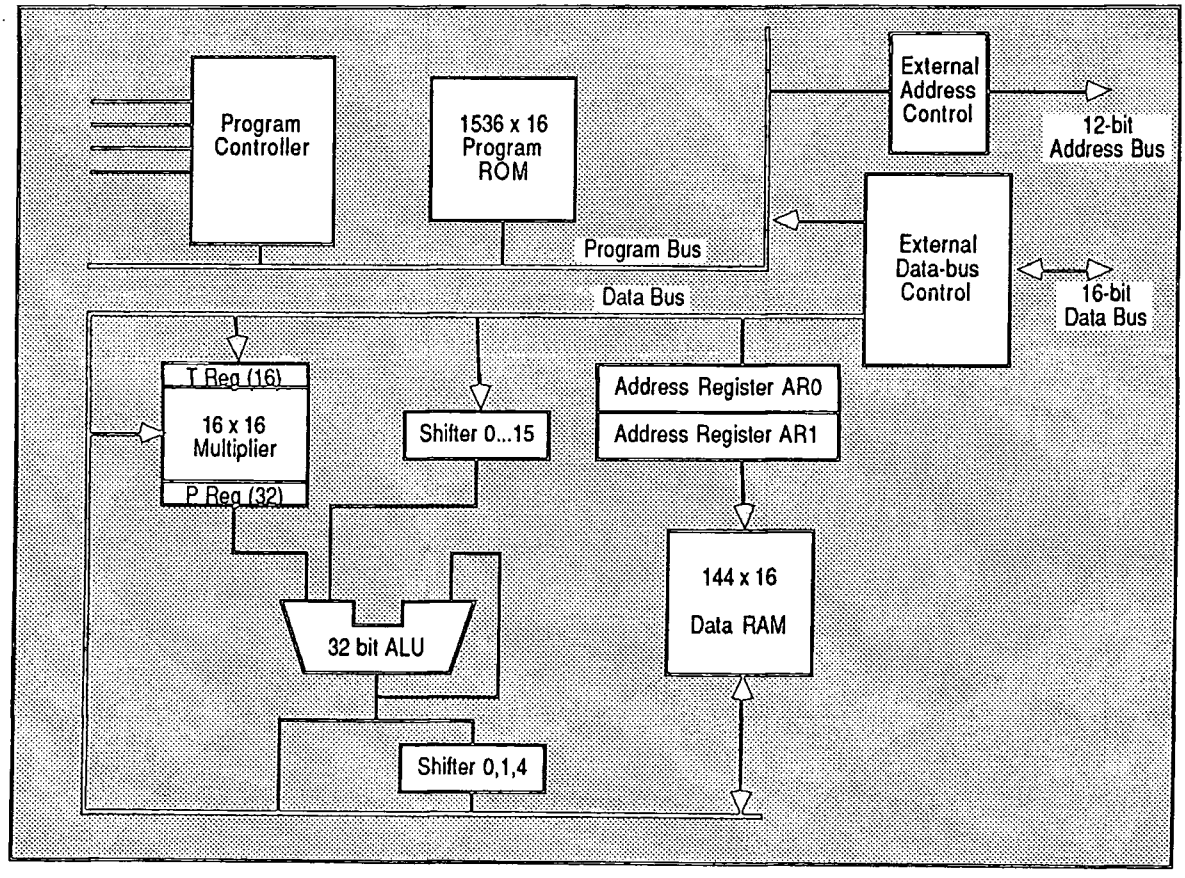


Fig. 5.1 : Block Diagram of TMS 32010 DSP

5.1.2 'Third Generation' DSPs

The TMS 32010 can be described as a second generation digital signal processor, by virtue of its basic features, and it is therefore logical to note the introduction, in recent years , of third generation DSP devices. These typically offer enhanced address generation, twin data RAM areas, pre-programmed ROM look-up tables and faster execution times. The following are examples of third generation DSPs:

- **Hitachi HD61810**

  This device is unusual when compared to other members of the DSP family for a number of reasons: It is (at present) the only DSP which performs floating-point arithmetic by default, and includes instructions to allow conversion between fixed and floating-point data formats.

  It has an advanced address manipulation capability, but offers no facilities for RAM expansion beyond its internal provisions, which would prove limiting for most radar signal processing applications. It is also the slowest of the DSPs with a 250ns instruction cycle, although this is predictable when considering floating-point operations.

- **Fujitsu MB8764**

  The Fujitsu processor allows limited parallel processing by permitting simultaneous loading of data registers from the two RAM areas, but its address generation techniques are complex and any advantage that would be gained from the fast (100ns) instruction cycle could easily be lost in handling the peripheral requirements of an I/O bound task. The device's unusual pipelined architecture requires that both data and addresses are generated one instruction before they are needed and this results in a lower *true* speed of operation.

Due to the slow execution speeds and complexities of these third generation DSPs it is suggested that some alternative be found for a DSP plot extractor implementation. Fortunately, recent 'fourth generation' devices have appeared, offering an ideal solution to the problem.

5.1.3 The Motorola 56000

The most impressive DSP specification is currently offered by a new product from Motorola, the DSP 56000, which can be considered as a fourth generation device due to its high-level instruction set, advanced parallel programming features and microprocessor-like design. A block diagram of the complete processor is shown in Figure 5.2.

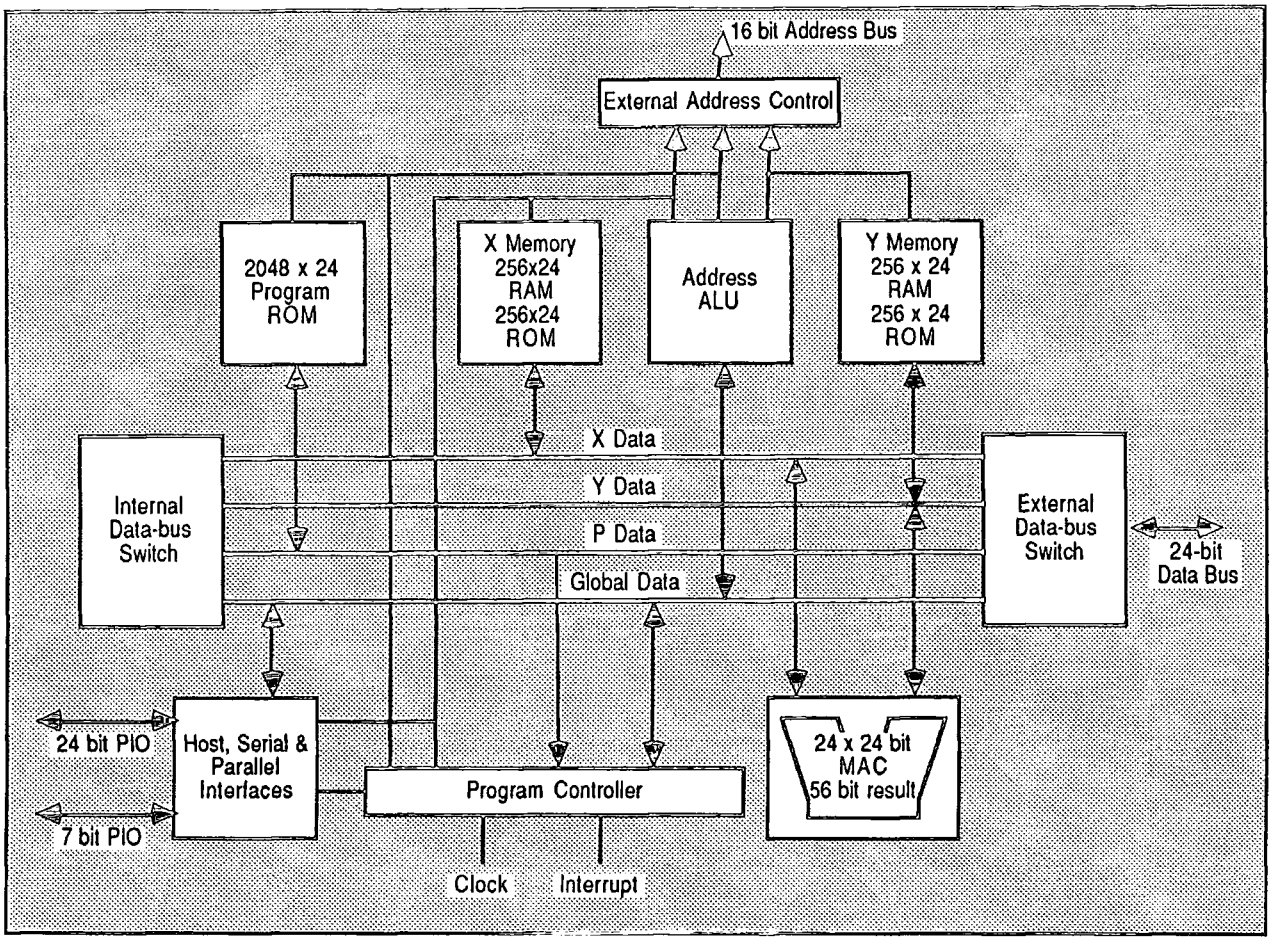


Fig. 5.2 : Block Diagram of Motorola DSP 56000

The DSP 56000 offers a number of advanced features which have not previously been considered within the scope of digital signal procesing, including the 4 separate data buses and three concurrently operating execution units - address ALU, data ALU and program controller. The device is intended to interface easily with a host processor and can be configured to use a variety of standard peripherals. It allows memory expansion to 64K words for each of the two data memory blocks, and access can be achieved in a single cycle provided that both X and Y blocks do not need to access the external memory simultaneously.

A total of 8 address registers are available on-chip, allowing for indirect addressing in a variety of modes: postincrement by 1 ... N, postdecrement by 1 ... N, predecrement by 1, offset

by ± N or no modification on access. Each address register (R_0...R_7) has an accompanying offset register (N_0...N_7) and modulus register (M_0...M_7), allowing address arithmetic to other than modulo 32768 which makes it ideal for queue or buffer processing. The remaining properties of this device are summarised below:

- 100ns instruction cycle, most operations execute in a single cycle
- 24-bit instruction / data word
- 24 x 24 single cycle multiplier / accumulator with 56-bit result
- Twin 56-bit accumulators (A & B), may be split into 2 x 24-bit + 1 x 8-bit (A0...A2).
- Two 48-bit (or four 24-bit) data registers (X0,X1,Y0,Y1)
- 15 level stack allowing nested hardware DO Loops
- Two independent 256 x 24-bit data RAMs
- Two independent 256 x 24-bit data ROMs
- 2K x 24-bit program ROM
- 24 Programmable I/O pins which may be otherwise configured as:
  - 8-bit host microcomputer interface
  - Serial comms interface with baud rate generator
  - Synchronous serial interface with clock generator
- Off-chip memory expansion: 2 x 64K x 24-bit data, 64K x 24 program

Through use of the separate internal data buses and concurrently operating execution units, the processor can be programmed to execute most register-based arithmetic and logical functions concurrently with data moves, provided that there is no conflict for any data bus. If, through bad programming, the software requires two accesses to the same bus the machine will take an extra cycle to complete the operation. It is therefore important to ensure that no contentions will arise as a result of concurrent data operations.

When programming the 56000, the assembler expects four fields to specify the operation, e.g.:

```
MAC     x1,y0,a     x:(R0)+,x1     y:-R4,Y1     ;Comment field
```

This specifies a multiply-accumulate operation in which the contents of 24-bit register X1 are multiplied by those of register Y0, & the result is added to the contents of accumulator A. The contents of the X memory space addressed by R0 are simultaneously loaded across the X-Data bus, as the new value of register X1, and R0 is incremented. Address register R4 is decremented then used as a pointer to a location in Y memory space which supplies the new value for 24-bit register Y1.

To conclude discussion of current DSP devices, the properties of four typical processors are summarised below in Table 5.1:

Device & Manufacturer	Cycle Length	Number of Registers	Register Width	Data Bus Width	Instruction Width	Data Memory	Program Memory	Memory Expansion Program	Memory Expansion Data	Other Features
TMS32010 T.I.	160ns	2 data 2 addr 1 accum	16 bit 16 bit 32 bit	16 bit	16 bit	144x16 RAM	none	4Kx16	4Kx16	Barrel Shifter 2's complement arithmetic. 8 I/O ports. Interupts.
HD61810 Hitachi	250ns	6 data 6 addr 2 accum	16 bit 16 bit 20 bit	16 bit	22 bit	200x16 RAM 128x16 ROM	512x22 ROM	none	none	Serial I/O. Floating Point arithmetic. All instructions single cycle.
DSP56000 Motorola	97.5ns	6 data 24 addr 2 accum	24 bit 24 bit 56 bit	24 bit	24 bit	2x256x24 RAM 2x256x24 ROM	2Kx24	64Kx24	128Kx24	2 Data Shifters High level programming. DMA,,UART & CODEC.
MB 8764 Fujitsu	100ns	4 data 6 addr 1 accum	16 bit 16 bit 26 bit	16 bit	24 bit	2x128x16 RAM	1Kx24	none	1Kx16	16 bit parallel interface. Divide instruction. Variety of address modes.

Table 5.1 : Digital Signal Processors

Two other 'fourth-generation' DSP's which have been announced recently and may represent a significant challenge to the Motorola 56000 devices include:

- The **ADSP-2100** from Analog Devices is based around a similar architecture to the 56000 and implements concurrent operations and single-cycle execution for all operations.

- The **ZR36010** (also known as the VSP-325), from Zoran Corp., is a more unusual DSP in that it performs complex arithmetic on arrays of data. It is primarily intended for use in video-picture processing, but its advanced instruction set implements a large number of operations of use in general signal processing. It is due to be released in sample form in the first quarter of 1988.

5.2 DSP Design of a CFAR Processor

A study of Table 4.1 should confirm Ebert's observations that none of the traditional microprocessors is capable of supporting the throughput required by the hardware front-end. Given that the time available for processing each incoming word from the detector is equal to the range cell sampling rate (≈ pulse width), and that processing must be repeated for every range cell, this would allow only 250ns...2μs, depending on radar type. It is likely that radars with a long pulse length will be sampled at a greater rate than $1/\tau$ to improve accuracy, and hence design around an assumed long pulse length would be unwise.

A CFAR solution using digital signal processors would seem more likely, due to their shorter cycle times and high-speed data transfer capability. The clutter map in a DSP implementation could be stored on the DSP device itself, the restriction being that only a small part of the map could be downloaded at any one time. This would have the advantage of simplifying map address calculation, but would require careful synchronisation between the source of the clutter-map and the device performing the downloading operation. Most digital signal processors support direct memory access (DMA) and it would therefore seem logical to use this approach to load the data into the DSP memory, although it would be equally possible to load data under control of the DSP provided this were carried out during the interscan period when there is no other input to the device.

5.2.1 A DSP Cell-Averaging CFAR

The simplest design approach is offered by considering the basic cell averaging CFAR processor, which calculates the average signal level in a pre-determined number of "range windows" on either side of a test cell. The average is then scaled and offset by both a manual quantity and a computer-generated "clutter-map" value, before comparison with the level in the test cell. If the test value exceeds the scaled average, a target is declared in that range cell.

The sliding-window can be regarded as a form of circular buffer of length 2N + 3, including 2 guard cells, and where N is the number of window cells on either side of the test cell. By splitting this into 2 separate buffers (**Figure 5.3**), it should be clear that an adapted modulo-addressing technique permits a simple program design, which would be ideally suited to implementation with a Motorola DSP56000.

The optimum configuration for the CFAR processor memory should arrange the clutter map in external addressing space, to take advantage of fastest download rates from the host computer, however there is little advantage to be gained from implementing the map on-chip as the software will become unnecessarily complex. The five input parameters (N, Scalar, Manual, C-Scale and C-Range) can all be loaded into the registers which comprise the DSP56000's host computer interface, thereby simplifying the data access requirements. Should

any of these parameters be changed during operation, it will be the responsibility of the host to ensure that the new value is passed on to the signal processor. If this approach is adopted it should be noted that the host data registers reside in X-memory space, which could prove restrictive.

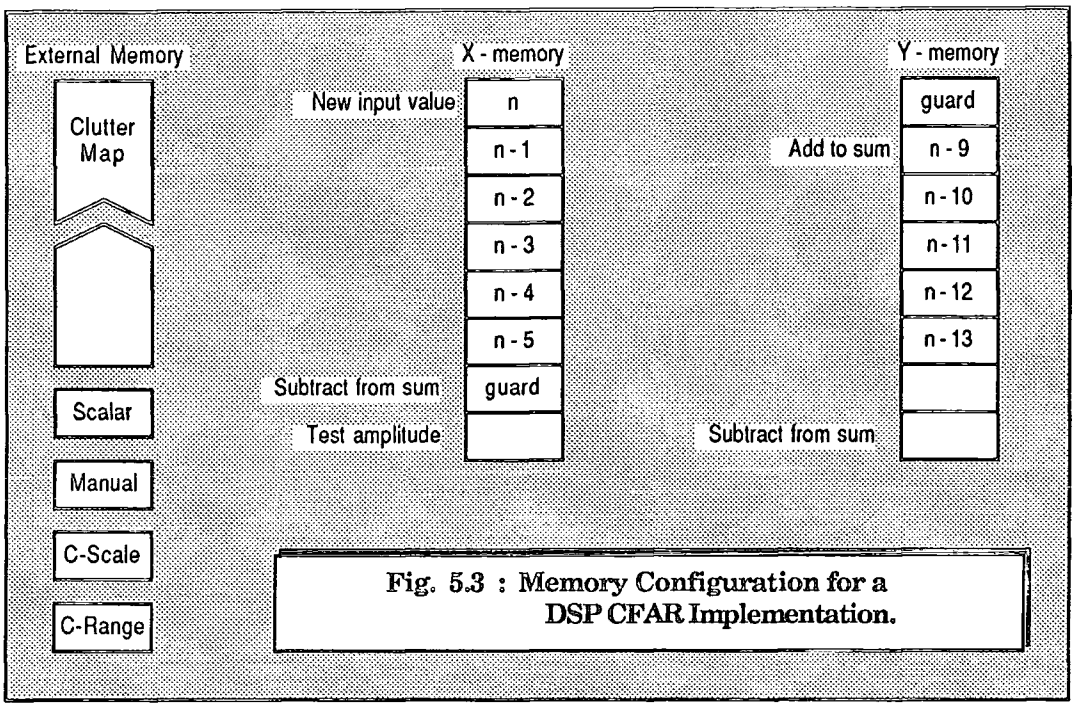


Fig. 5.3 : Memory Configuration for a DSP CFAR Implementation.

Using the memory configuration of Figure 5.3 permits optimised access to the two "windows" through parallel loading of registers. All that is needed is to initialise the modulo-modifying registers to accept the appropriate buffer size. The sequence of operations (which must be repeated for each range cell) then becomes:

- Load new video sample into buffer and add to accumulator
- Subtract pre-test guard cell from accumulator
- Add previous post-test guard cell value to accumulator
- Subtract previous end-window cell from accumulator
- Multiply accumulated value by scalar
- Divide accumulated value by number of cells in window
- Add manual offset
- Add clutter map value
- Compare with test cell - declare target if (test) > (average)

It is fortunate that the nature of these operations enables a high degree of parallellism to be programmed into the design. A suggested program to perform this sequence of operations is given below:

```
START:
        SUB     Y0, B       X:input, X1     Y:(R4)+, Y1     ;B is <0 if target present. X1 holds new
                                                            ;input data (n). (n-9) to Y1.
        ADD     X0, A       X1, (R1)+       Y0, Y:(R4)+     ;A always holds accumulator. Add new data,
                                                            ;X1 to (n), old test value to (n-8).
        ADD     Y1, A       X:(R1)+, X0     Y:(R4)-, Y1     ;Accumulate (n-9). Test value to X0.
                                                            ;(n-15) to Y1.
        SUB     Y1, A       X:(R1)-, X1     Y:(R5)+, Y1     ;Subtract (n-15). (n-6) to X1. Clutter
                                                            ;map value to Y1.
        SUB     X1, A       X:c_scale, X1                   ;Subtract (n-6). C-Scale value to X1.
        REP     #1          X:manual, B     B, Y:output     ;Manual offset to B, previous B to output
        MAC     X1, Y1, B   X:n_over_k, X1  A0, Y1          ;C-Scale * Clutter-value + Manual -> B
        MOVEP   X0, Y0                                      ;Shift test cell to Y memory region
        JMP     START
```

This program could implement a complete first threshold CFAR processor in nine instruction cycles, representing a total of 877.5ns at an internal clock rate of 10.25 MIPS. This cycle time would permit a throughput of 1.24 million words per second, although it is possible that an improved memory architecture could be found which would allow faster operation. A short initialisation program (shown below) would also have to be run, in order to configure the registers in the DSP56000, although this stage would not be time-critical as no significant data would be flowing through the system.

```
INITIALISATION:
        MOVE    #8, M1                  ;Load X-memory modulus register
        MOVE    #8, M4                  ;Load Y-memory modulus register
        MOVE    #x_block, R1            ;Load R1 with address of (n)
        MOVE    #y_block, R4            ;Load R4 with address of (n-9)
        JMP     START                   ;Go execute main program loop
```

It should be noted that the choice of address registers is important, as the DSP 56000 requires that they be divided into two "files" for X and Y memories in order that parallel

addressing functions in single-cycle mode. In the program shown above, the memory is configured for use as a 6-stage sliding window (6-bits on either side of the test cell) with two guard cells added to improve performance. Register R1 is initialised to point to the n^{th} video input, and after executing the main program loop is left pointing to the new video input cell. After 8 times round the loop, the modulo-8 address arithmetic ensures that the register now points back to the original input cell thereby maintaining a circular buffer configuration.

The speed of currently-available devices would allow processing of long pulse-length signals, however it is not fast enough to permit second-threshold calculations to be performed by the same device. Although it is likely that a faster version of the DSP 56000 will be introduced, offering a greater throughput through a higher clock speed, the best alternative is to look for a different CFAR architecture.

5.2.2 A DSP Rank-Statistic CFAR

A cell-averaging (greatest-of) CFAR DSP wastes much time on loading and storing the incoming data, with comparatively little time spent on processing. If a different approach (such as the rank based detector proposed by Rohling and discussed in an earlier chapter) is adopted a better solution can be achieved by off-loading the storage task to a separate VLSI device, and then processing the stored data using a DSP.

This solution is achieved by loading the incoming data directly into a device such as the AMD 95C85, a content addressable data manager (CADM) which sorts the data to form an array of ordered samples. The configuration of the device would then permit a particular sample to be extracted by the DSP, multiplied by a scaling factor and then added to a manual offset. The signal processor could then select the appropriate cell from a clutter map and add this to the accumulated value, before comparing the sum with the chosen input cell to produce a single binary target indication.

The use of the CADM allows for future expansion to greater data width, as the device can be programmed for between 1 and 255-bit width, although its input/output port is fixed at 8-bits. All of the signal processors described earlier have a data width of 16-bits or more, and hence there would be little difficulty in reconfiguring these devices to a wider data path.

5.2.2.1 The AMD 95C85

This AMD 95C85 is a "Content Addressable Data Manager" which consists of 1K x 8-bits of on-chip RAM which can be addressed either directly (as in a conventional RAM) or by content, and hence the device is capable of hardware-sorting any data that is written to it. The device is seen as an index file with variable length key and pointer fields, which may be programmed for any data width up to 255 bytes. Items written to the file may be either sorted on

entry or in a batch mode, but this process is effectively transparent to the programmer. The CADM has a 16-element instruction set which enables selection of a variety of operating modes.

5.2.2.2 Using the CADM

The CADM would typically be used as an external peripheral to a digital signal processor such as the DSP56000, described in the previous section, in order to reduce problems of hardware design. A suitable sequence of operations for this configuration is given below. Note that it is still necessary to involve a circular buffer design in the DSP in order to extract entries from the CADM after they are out of window range, but the buffer does not need to be split as guard cells are not used.

- Set CADM to search for outgoing window value & pop off stack
- Set CADM for on-line sort & load new video sample into memory / buffer
- Set CADM for direct access and read "K"th value
- Multiply this value by scalar
- Add manual offset
- Add clutter map value
- Compare with test cell - declare target if (test) > (average)

As with the previous design, the parallel loading capabilities of the signal processor can be used in order to speed execution. The complete program for implementing a DSP rank-based CFAR is given below, with suitable initialisation code for the processor and CADM coming before the main program loop.

```
INIT:
        MOVE                    X:manual, B     #LAS-K, Y:cadm      ;Set CADM for Rnd access at Kth
                                                                    ;address & load manual offset into B.
        MOVE                    X:c_scale, X0   Y:(R5)+, Y0         ;C-scale to X0, computer offset to Y0
        MAC     X0, Y0, B       X:scalar, X0    Y:cadm, Y0          ;Scalar to X0, rank value to Y0
        MAC     X0, Y0, B                       #STK+FND, Y:cadm    ;Set CADM to stack mode & locate next
                                                                    ;operand
        CMP     A, B                            A, Y:cadm           ;Compare result with test value
```

It should be clear from the above program that use of the DSP is restricted largely to a series of data moves, and hence the processing speed of this device is lost. An improvement could be achieved through use of a combined DSP / CADM / custom logic solution, where the data load and control sequences to the CADM were generated by some programmable logic (**Figure 5.4**). This could achieve a reduction to only 5 cycles for the complete CFAR processing operation, which would permit a data throughput of approx. 2 MHz - sufficient for most surveillance radar pulse lengths.

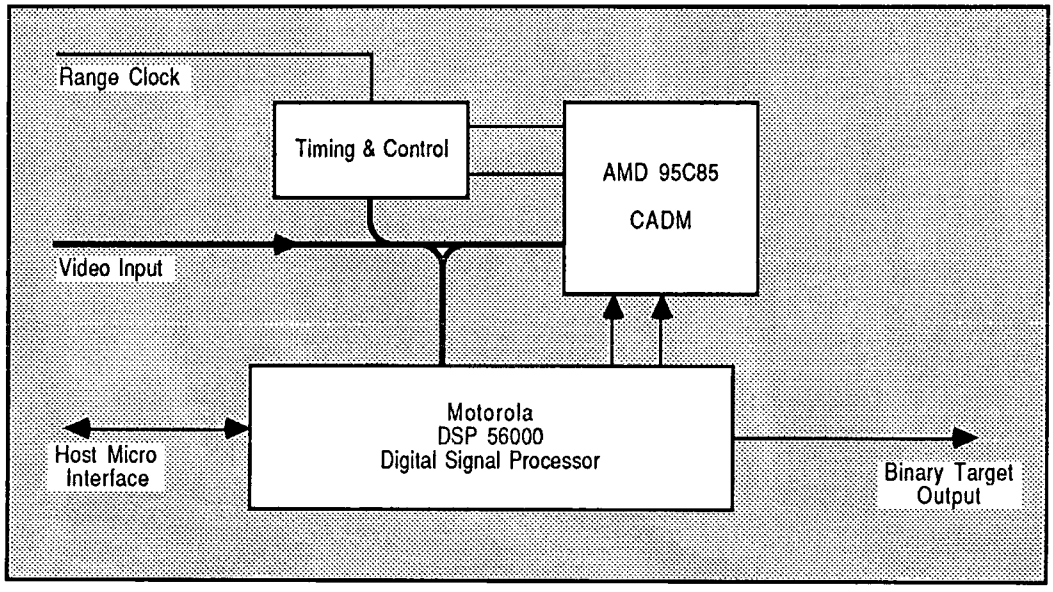


Fig. 5.4 : Configuration of DSP and CADM

5.3 The Within-Beam Integrator

The combination of a within-beam integrator with a constant false alarm rate processor would be an ideal objective for a DSP plot extractor, however the two previous examples have shown that in spite of their short cycle lengths, these devices are still not fast enough to accomplish both tasks in the allotted time period (one range clock cycle). An obvious solution would be to divide the task between a number of signal processors, however it is unlikely that this approach could be justified on grounds of cost, and much of the device's processing power would be wasted in the WBI section.

The basic sequence of operations required by a programmable within-beam integrator can be established from discussions in the previous chapter, however it should be clear from this that no advantage is to be gained by using a DSP due to the narrow width of the WBI data bus. The DSP 56000's 24-bit data bus would be largely redundant as only 5-bits are required by the current WBI configuration, and there would be little advantage in increasing this figure. The input to this section is only a single bit, and again the DSP 56000 is not suited to this type of data. In general, high speed operation would not be possible due to the number of conditional instructions and the restrictions of the DSP 56000 instruction set. As the requirement to address a 4K-nibble threshold memory would discount the use of all but the Motorola device, it is suggested that some alternative method of implementing the within-beam integrator function be adopted, such as the ULA solution proposed in the previous chapter.

5.4 Plot Forming Processing

Although constrained by the need to complete a processing cycle on at least every other range cell period, the extent of the calculations involved in the plot forming process make it an ideal target for DSP implementation. As with the within-beam integrator, difficulties arise due to the single-bit format of the input data and any solution will probably require a compromise between complex hardware and flexible (but slower) firmware.

The basic plot forming process generates pairs of 16-bit co-ordinates, based on the centre of gravity of a group of binary cell "hits". As plots must be integrated both in range and azimuth the present design is configured to store a minimum of 128 active plots at any one time, although a larger number would be more acceptable. A total of five 16-bit values must be stored for each plot, requiring a minimum of 128 * 5 (= 640) words of storage. This is unfortunately out of range of the Hitachi DSP, which would have otherwise been the most suitable for the design due to its versatile floating-point arithmetic operations. From the choice of remaining devices the DSP 56000, with its parallel loading features and expandable external memory, seems to provide the best solution to the problem, assuming the X and Y memories are chosen so as to give optimum concurrent access without extra wait states. Using Motorola DSP instructions, a sample program for the plot forming operation, is given below:

```
START:
        MOVE                    X:theta, X1     Y:(R4)+, Y1     ;x1=Øae, y1=Øe
        SUB     X1, Y1, B       X:(R1)+, A      Y:(R4)+, Y0     ;x1=Re, b=Øpw, Y0=Re
        CMP     #dw, B          X:num, X1                       ;X1=∂N
        JGE     Close

        ADD     X1, A           X:(R1)+, X0     Y1, Y:(R5)+     ;store Øe , x0=ΣØ
        MUL     X1, B           X:(R1)+, X1     Y0, Y:(R5)+     ;store Re, x1=ΣR
        ADD     X0, B           A, X:(R2)+      Y:weight, Y0    ;store ΣN
        ADD     Y0, X1          B, X:(R2)+                      ;store ΣØ
        MOVE                    X1, X:(R2)+                     ;store ΣR
        JMP     Start
;
; Plot closure sequence
;
CLOSE:
        LUA     A0, N7                          Y1, A           ;ΣN to n7,
        MOVE                    X:(R1)+, X0     Y:(R7+N7), Y1   ;x1=ΣØ, Y0=1/ΣN,
        MAC     X0, Y1, A       X:(R1)+, X0     Y0, B
        MAC     X0, Y1, B       A, X:bearing
        MOVE                    B, X:range
        JMP     Start
```

The potential of a DSP plot extractor is illustrated by this program, which executes a complete plot forming sequence (regardless of whether the plot is completed or re-circulated) in 10 cycles. It is likely that the code could be optimised still further, but in its present state is capable of handling most plots, which could be output directly to the host processor DMA interface on-chip. A study of the code should indicate that the principle of operation is similar to that of the RATES hardware: data is read in from one buffer and re-written (if the plot is not closed) to a different area. Some code would therefore be needed to switch the buffers at the end of each scan, but this would not be time-critical due to the length of the interscan period.

5.5 Summary of DSP Applications

Digital signal processors represent the most viable solution to the problem of implementing a programmable plot extractor, and their performance is likely to be enhanced still further over the next few years. The Motorola DSP 56000 has been released recently and, whilst not offering comparable performance with the existing RATES hardware in terms of speed, shows considerable potential as the basis of a feasability study into future DSP designs. Texas Instruments have also announced their successor to the TMS320 processor, with cycle times reported to be equal to those of the Motorola device, but no further information is available at the time of study. General Intrument Corp. have also released CMOS varients on the TMS 32010 with on-chip EEPROM.

In the short-term, a solution may be best achieved through use of a combined programmable logic / digital signal processor approach, possibly using DSPs to realise the CFAR and plot forming functions, with the remainder provided by the hardware. In time, it is likely that faster devices will be made available which would be capable of implementing a complete hardware plot extractor on a single chip.

Finally, it should be noted that the programs described in this section have been based on advance information supplied by Motorola, without access to full programming details. There is therefore no certainty that programs will execute correctly, particularly when considering concurrent data movements, for which rules regarding legal operations have not been clarified.

6.0 Introduction

The greatest potential for improving the design of the prototype system can be found in the tracking computers. The current design has resulted in inefficient software and a complex interface which is totally unnecessary when compared with modern design techniques. A radical redesign of the tracking system is therefore proposed, using an efficient interface design and optimised software. This chapter sets out to illustrate that a more compact interface with the plot extractor hardware can be achieved. The design is supported by examples of software modules which illustrate the advantages of a low-level approach to the problem.

6.1 The RATES Prototype Tracking System

The RATES prototype is designed around a hardware front-end processor feeding plot data into a computer, which then performs further filtering and tracking operations before producing target-movement predictions. These operations are described by the MASCOT connectivity diagram in Chapter 3 (Figure 3.19), although it should be noted that this includes a number of features which have been added to help assess the performance of the prototype.

The data received by the computer consists of co-ordinates based on the centre-of-gravity of filtered plots. At this stage, many of these will be due to false alarms (an average of at least eight per revolution) but must be checked to see if they could correspond to any existing moving targets. Those which fail to associate with existing targets must be removed.

As RATES is intended primarily for target tracking, the next stage is to filter-out stationary plots, to avoid wasting processing time while attempting to follow fixed targets. All remaining plots must therefore be either tentative or confirmed tracks and any remaining processing time can be devoted to calculation of target velocity, heading and expected position.

The RATES design provides the user with full manual control, to enable operator-assisted initiation of target tracking and to adjust threshold levels to compensate for variations in clutter regions which have been detected by the experienced operator. It is also possible to select any confirmed track and obtain status information, which is then displayed on the operator's console.

The additional prototype functions include a means of recording plot data to removable magnetic media, and allowing it to be replayed at a later date at the same speed, or faster (or slower) if required. In this mode, the software is capable of storing plot data to an additional hardcopy device, such as a printer, or drawing the tracks on a suitable plotter.

An early version of the prototype used twin Ferranti Argus 700 computers to carry out filtering and target tracking, but following difficulties with resource sharing this was abandoned in favour of a single machine. In terms of physical size, the Argus accounts for approximately half of the total volume and it would therefore seem logical to study the feasibility of replacing this machine with a single board solution, using more modern technology.

6.1.1 Advantages of a Microprocessor-based Tracking System

The redesign of the RATES tracking computer, adopting current semiconductor technology, would offer a number of advantages, due in part to the high-speed operation of existing microprocessors and peripherals.

The main interface to the plot extractor hardware, which currently accounts for almost one third of the total number of circuit boards, could benefit from being completely re-designed. The basic function of the interface is to control DMA transfer to and from the Argus, but a new design would permit the use of existing specialised VLSI controllers, especially those related to the host microprocessor family.

The use of surface-mount and leadless chip-carrier (LCC) semiconductor technology would permit a single processor board solution, which could be installed alongside the existing plot extractor hardware and would require less power than the Ferranti Argus. If recommendations regarding the hardware redesign are adopted, this would increase the likelihood of a successful 2-board solution to the problem of automatic track extraction.

The current tracking software is written in Coral 66, but has already been shown to contain a number of bugs and general shortcomings. The MASCOT design approach to the software has been undertaken so as to retain compatability with similar systems, but does not offer any advantages in terms of speed of execution. Similarly, the MASCOT approach results in a less optimised code, which could be substantially improved by redesigning around a specific processor and hardware configuration. It is probable that optimised software, written in assembler or a medium/high level language, would result in a substantial increase in the performance of the RATES tracking potential.

6.1.2 Determination of a Suitable Microprocessor Solution

Ebert's microprocessor criteria (Section 4.6) are aimed primarily at automatic tracking and filtering functions, and can therefore be used as a basis for determining suitable devices to replace the existing computer hardware. Table 4.1 listed a small sample of devices which satisfy these criteria, providing a high level of performance which is likely to be exceeded by future product releases from their manufacturers. Using the product details alone, it is impossible to recommend a suitable replacement however, and it is necessary to take other factors into account and compare these with the Ferranti Argus specification, before a final choice can be made.

6.1.3 The Ferranti Argus 700

The Ferranti Argus is essentially a multi-board computer which can be easily expanded to suit differing applications, whilst retaining a ruggedised construction, suitable for military use. It supports an instruction set which is similar to those of the processors in Table 4.1, permitting bit-wise, shift, arithmetic (*/+-), logical (AND, OR) and comparison operations. Typical instruction execution time provides a useful benchmark for comparison with other processors and some examples are given in Table 6.1, against two of the devices from Table 4.1 (based on fastest available clock rate).

At this point the NEC and T.I. microprocessors have been disregarded as the former is essentially an enhanced version of the Intel 8086, but is not so well supported. The T.I. device is considered unsuitable due to its stack-orientated processing which is more appropriate to a mainframe-style application than the register-based RATES software. Appropriate details of the National Semiconductor product have not been forthcoming.

Processor	Register Multiply	Register Divide	Register ADD	ADD Immediate	Register AND	Register CMP	Subroutine RET
Ferranti Argus	7.1µs	7.8µs	1.85µs	2.65µs	1.85µs	1.85µs	3.0µs
MC 68000 Motorola	<70 (cycles) <5.8µs	<140 <11µs	4 0.3µs	8 0.6µs	4 0.3µs	4 0.3µs	16 1.3µs
8086 Intel	118 (cycles) 11.8µs	162 16.2µs	3 0.3µs	4 0.4µs	3 0.3µs	3 0.3µs	16 - 26 1.6 - 2.6µs

Table 6.1 : Comparison of Processor Execution Times

The above table clearly illustrates the dramatic improvement in speed that can be achieved using modern microprocessors, when compared with the Ferranti Argus. The fast

multiply and divide times of the Argus are exceptional and can probably be attributed to a hardware multiplier built into the machine's CPU. These functions are usually implemented in microcode on traditional microprocessors.

Working from these figures, it is suggested that a microprocessor replacement for the Ferranti Argus will result an improved tracking performance due to the increased speed of operation. Not surprisingly, this in turn will be dependent on a number of other factors such as choice of programming language, efficiency of compiled code and construction of a suitable interface to the plot extractor hardware, so attention must be paid to these areas in order to maintain the advantages offered by the technology.

6.1.4 Choice of Microprocessor

If the table of sample execution speeds were used to select a suitable microprocessor, the advantage clearly lies with the Motorola 68000. There are, however, other factors which support this device as an ideal choice:

- The MC 68000 offers the largest number of registers and addressing modes, thereby increasing its flexibility.
- Although on only a 16-bit data bus, the internal width is 32-bit and long-word operations can be performed with little overhead.
- Data and address buses are not multiplexed, so address translation and interfacing is more straightforward.
- Programming is simplified by the continuous linear address space
- The asynchronous bus will readily interface to a variety of devices, and may prove to be an important factor in a direct interface with the RATES hardware.

The Motorola product range is second-sourced by a number of other manufacturers, including Hitachi, Thomson-CSF and Mostek and volume-supply difficulties are therefore unlikely to arise. Motorola offer potential for future upgrades, and it can be argued that their products are an indication of future trends in other manufacturer's technology. All 68000 family microprocessors are fully code compatible and hence a solution developed using the 'bottom-of-the-range' 68000 will be fully transportable to future products including :

68010 - Available, but only adding a virtual memory capability to the 68000.
68020 - Also available: 32 bit, 20Mhz processor, high performance capability.
68030 - Announced & quoted as offering higher performance than a top-range VAX.

As reports have suggested that the 68020 outperforms the comparable offering from its main competitor (the Intel 80286) it would therefore seem wise to concentrate on use of Motorola's

products for a long-term solution to the automatic tracking computer.

From a more personal point of view, there is already considerable experience in the use of the MC 68000 in Durham, as this device had been chosen for a number of earlier projects. A cross-assembler is also readily available, based on the Northumbrian Universities Multiple Access Computer (NUMAC) installation at Durham, which can be used to download compiled code across a serial link.

6.1.5 A Note on Other Processors

Digital signal processors have not been considered for use as replacements for the RATES tracking computers. On the basis of Ebert's criteria, they are precluded by virtue of their limited addressing capability and instruction sets. Until recently, they could only be programmed in assembly language, and the complexity of DSP assembler would render this an unsatisfactory solution, as there would be a noticible increase in the required development time.

Another family of processors which has not been used in this design is the Inmos Transputer range. These devices (discussed in a later chapter) offer sophisticated inter-processor communications which make them ideal for array-orientated processing, but unsuitable for the sequential program design of the automatic track extraction software. In this latter type of application, recent studies have noted that the Motorola 68020/68881 combination exhibits a performance increase of 2.5 times over the top-of-the-range Inmos T800 transputer.

6.2 Development System Computer Hardware

It has been mentioned that one of the factors supporting the choice of the Motorola 68000 processor was a degree of experience in earlier work involving this device, which had resulted in a number of development boards becoming available for use in the RATES project. Manufactured by Motorola, the boards offered space for additional user-designed circuits and could be customised as required. It was decided that this would enable identification of the ideal hardware configuration, with the aim of locating a specific single-board commercially available design, that would function with little or no modification in a production system.

6.2.1 The Motorola MEX 68 KDM

The board, the MEX 68 KDM, was originally designed for use with early production prototypes of the XCS 68000 chip, and was supplied with 16K-words of RAM and a small monitor program in ROM. The board was fitted with two asynchronous RS-232 ports, two 6821 parallel ports and a counter/timer chip. Based on Motorola's Exorciser bus (an 8-bit bus designed for the 6800 processor), the KDM board could easily be expanded to address external memory or devices, with a maximum addressing range of 128K bytes. A small monitor program was supplied with the board, capable of basic terminal control, memory modification and upload/download to a host computer using the Motorola standard 'S' record format. A block diagram of the complete board is shown in **Figure 6.1.**

The restrictions of the KDM board meant that a number of modifications had to be made before it could be used alongside the RATES hardware. In a production system it is unlikely that these modifications would be applicable, as suitable hardware of the required specification would be more readily available.

There was insufficient RAM to hold both the code and data for the tracking software (based on the data storage requirements of the prototype software), and so the 16 RAM chips (type 4116) were replaced with semi- pin-compatible upgrades, using Cowan's report [55] as a reference. After modification, the board held a total of 128K bytes of RAM, which was deemed to be sufficient for an initial implementation of the RATES software.

The Motorola monitor (MACSbug), which was supplied with the KDM boards, was well documented but had clearly been based on an early 8-bit operating system, and took account of the most common non-functioning instructions in the XCS68000. As a result it was very slow and made little use of the extensive loop-handling and bit manipulation features of the MC68000. Whilst this in itself was not a problem to the RATES software, the inability to load Motorola 'S2' records (those with a 3-byte address) meant there was little compatibility with existing cross-compilers. It was therefore decided to implement a modified proprietary

monitor in place of MACSbug, taking advantage of its use of similar device addresses. A small number of modifications were made, and the replacement EPROMs installed on the board in place of the original chips.

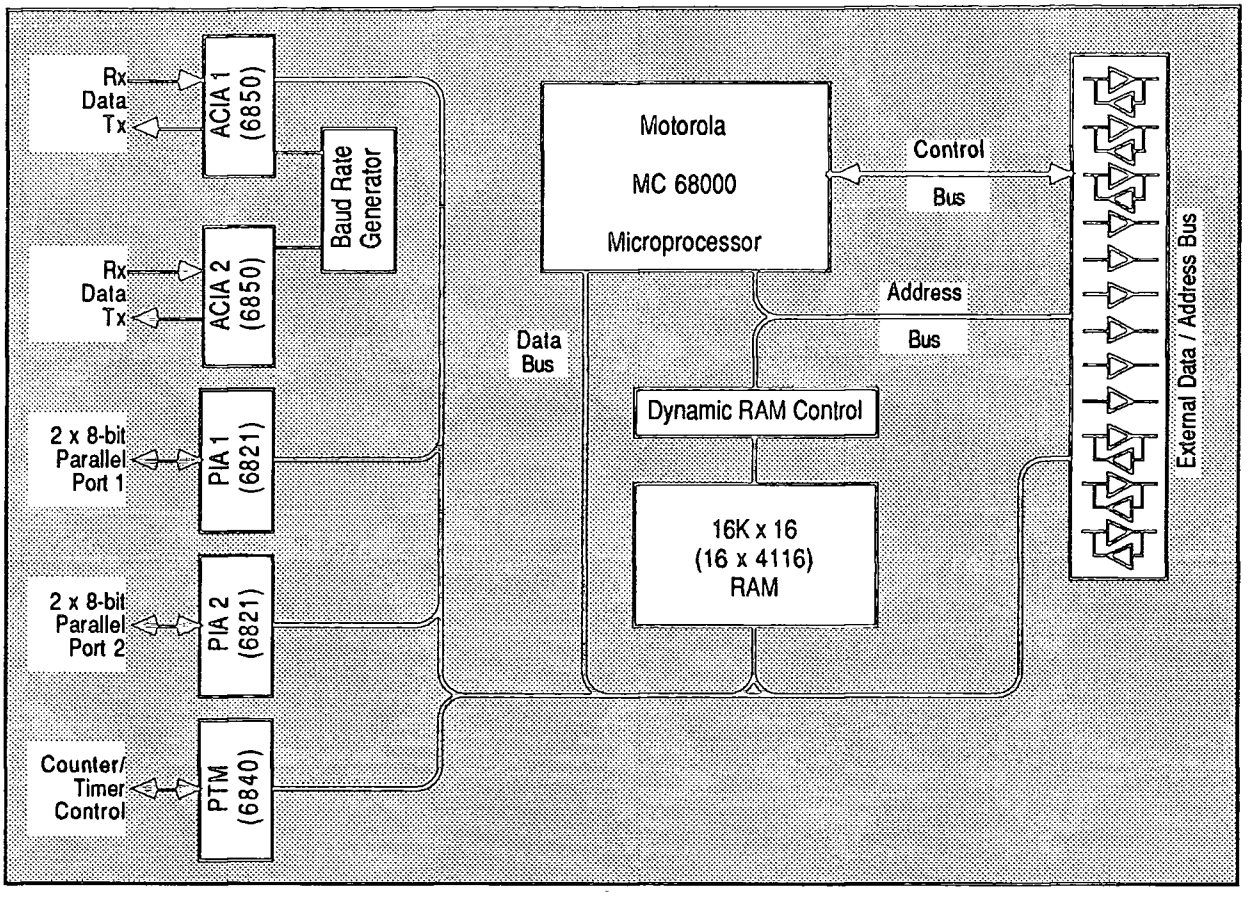


Fig. 6.1 : MEX 68 KDM Development Board Schematic

The development board interface was comprehensive, and required little modification to enable it to drive external peripherals. Three pins were designated as available for additional bus signals, and one of these was therefore used for address line A17, enabling use of a full 64K bytes of external storage which did not conflict with the internal port addresses. The complete pin-designation diagram is given in **Appendix B** together with a description of the associated pin functions.

6.2.2 Development Sequence

In the timescales available for research, it is unlikely that a complete implementation of the RATES system with all of the features of the prototype could be produced. A list of development priorities has therefore been established, with the aim of offering an increasing degree of user-visible functionality, whilst at the same time providing some means of assessing the processor loading factor. This list also takes account of the constraints imposed by component availability, notably the floating-point hardware to assist the MC68000, which was not available at the time of work on the hardware.

Stage 1 : Establish suitable interface to plot extractor hardware
Stage 2 : User keyboard interface - establishes early control of system for operator.
Stage 3 : Output interface software - confirm functioning output to plot extractor
Stage 4 : Control of plot input & miscellaneous information input from extractor
Stage 5 : Display of input plot data as synthetics characters
Stage 6 : Filtering of stationary plot information
Stage 7 : Update of moving track data
Stage 8 : Update unassociated plot data
Stage 9 : User interface - confirmed track enquiries
Stage 10: Prototyping and "debug" facilities, e.g. hardcopy output, record / replay.

The progression of development through these stages is evident from the discussions which follow.

6.3 The Computer Interface

As stated earlier in this chapter, one of the prime aims of a replacement tracking computer should be a redesigned interface to the plot extractor hardware. The current design comprises one quarter of the total number of boards in the system, and is largely dedicated to the specific requirements of the Ferranti Argus's direct memory access (DMA) interface. It also takes account of the possibility that the computer may not be situated near to the extractor due to space considerations, and provides for full buffering and error detection across a parallel link.

A new interface can adopt one of several formats:

- Retain the ME62 interface structure, but adapt the MC68000 board to work with the existing Function H boards.

- Redesign the ME62 interface using VLSI components, to produce a more compact solution along the same lines as the original.

- Adopt a memory-mapped solution, which places the input/output sections of the plot extractor hardware in the memory space of the new processor and thereby dispenses with the need for RATES functions F and H.

The first of these options is clearly unsatisfactory, as it results in little change to the existing hardware, and will only reduce the physical size of the final solution because of the difference in size between the Argus processor and the MC68000. This option would require additional circuits to interface from the existing Argus bus signals to those applicable to the MC68000's bus structure. The advantage of this solution would be the ability to locate the processor remotely from the plot extractor hardware, but with the ultimate aim of a single (or twin) - board RATES system, this would have little practical advantage.

Similarly, the second option does little to address the problem of the physical size of the existing interface, and whilst redesigning the functions F and H may result in two single replacement boards, the only true advantage gained is that of the remotely-located processor board. If this were a requirement, then this option would be sensible, as it would enable a more optimised interface to the MC68000 bus to be incorporated.

For the type of application currently under study, the third option represents the best solution as it would completely remove the need for any of the data acquistion and re-distribution functions contained on boards F and H. The processor board, however, does need to be located close to the rest of the hardware, to avoid problems with system noise.

6.3.1 Application of Memory Mapping to RATES

Early microprocessor applications communicated with external devices through "ports", which typically provided a buffered interface through to the data bus. Each external device required its own port, and each port was given its own address, which could (in some processors) be accessed through special I/O instructions. Communication through these ports often took longer than a similar memory-access cycle, but avoided the problems of buffering and possible damage to the processor. With improvements in semiconductor technology, the sink current of most integrated circuits has dropped and it is now possible to drive a large number of devices, even from an unbuffered processor bus. This, coupled with the increased addressing range of most processors, has made memory-mapping an ideal solution to the control of external peripherals.

Through memory mapping, each I/O function of the plot extractor hardware is assigned a unique address (or range of addresses) within the computer's memory space. The processor can then read from or write to the function as if it were a conventional area of memory, without the need for specialised instructions or parallel ports adapters. The Motorola processor is particularly suitable for this technique due to its linear address space, which gives access to any memory location without the need to predefine a 64K byte working segment. The asynchronous data bus is another useful feature, which results in the processor being compatible with any speed of peripheral device. In practice, the MC68000 begins a read or write cycle to a peripheral in the same manner as synchronous processor, but then enters a continuous "wait" state until the peripheral acknowledges with a cycle-completed signal (DTACK). This allows the designer to choose the optimum settling time for the peripheral response, possibly allowing extra time for noisy or heavily-loaded data bus conditions, which would otherwise produce unpredictable results with a synchronous processor.

One of the disadvantages of using memory mapping is the need for a relatively close proximity to the associated peripherals, in order to reduce the effects of noise on the bus. As a consequence, the RATES system has been re-housed in a modified cabinet which permits the KDM board to be inserted alongside the remaining boards, in spite of its occupying a double-height slot. The board's edge connector is wired into the remaining plot extractor hardware through a special interface, which is described in detail later. The immediate advantage to be gained from this approach is that the ME62 computer interface is no longer needed, as there is no significant distance between processor and peripherals. Similarly, there is no need for any logic to pack and unpack the data for transmission across the bus, as all of these functions can now be undertaken by the MC68000 itself. The prototype functions F1...F4 and H1...H5 are therefore redundant, and the total number of boards in the system can be reduced from 36 to 29, *including* the processor and interface.

6.3.2 Plot Extractor Interface Functions

There are a total of 8 functions in RATES which require some form of input/output with the tracking computer. These are detailed below, along with two extra functions which are part of a later release of the RATES software. Complete data formats are given in Figure 6.2.

Constant False Alarm Rate processor - (Output)

Consists of 7-bit threshold and 9-bit partial address, with the remainder of the address supplied directly by function G. Data transfers therefore have to be synchronised to the correct octant and must occur during the interscan period.

Synthetics Display Characters - (Output)

256 x 2 words with first word carrying 3-bit octant and 9-bit bearing information, and second word carrying 10-bit range and 6-bit character code. Transfers must take place during the scan period as synthetics are displayed during interscan.

Test Target Co-ordinates - (Output)

3 x 2 words, each encoded to 12-bits accuracy. First word is target range, second is target bearing. Transfers take place during the interscan period.

Electronic Counter-Measures Report - (Input)

32 x 2 words, first of which is the bearing of the ECM sector, and followed by the width of the sector.

Rollball Co-ordinates - (Input)

Read as 2 separate words from different locations. One is the X-coordinate, the other the Y-coordinate. Can be read at any time.

Ships Motion Data - (Input)

A single 16-bit word, with the ship's heading encoded in the lower 10 bits and ship's speed in the upper 6 bits.

VDU Status Monitor - (Input/Output) - Version 3

Five separately-addressable registers which control the operation and display of the status VDU which was added as part of software release version 3. This addition is discussed later in Section 6.6.3.

Number of Plot Input Words - (Input) - Version 3

A single word which represents the number of plot words transferred to RAM.

Constants Word - (Input)

This is a single word which holds information regarding the settings of user-selectable options, and the status of the plot extractor cycle. The following 9 parameters can be determined from this word:

- Radar Select, C-Range, C-Scale, Magnification : All reflect the state of these switches on the control panel. In later versions of the software, Radar Select is bi-directional, and overrides the setting on the console.
- Octant : Is taken from Function G2 and indicates the currently scanned WBI octant (which may not be the same as that being scanned by the CFAR).
- Interscan : Active low, indicates an interscan period is in progress.
- North : Active low, indicates a north-marker cycle.
- Octant Demand : Active low, request from the plot forming hardware to transfer data into the tracking processor.
- Stop : An active high signal which indicates that all available plots have been read from the plot extractor interface.

Extracted Plot Data - (Input)

This is an output from plot extractor function E4, and consists of pairs of words with the first carrying the plot bearing, and the second the plot range. Data input to the computer continues until the Stop flag, in the constants word, is raised. In later versions of the software, this feature is removed, as a more sophisticated DMA technique is employed.

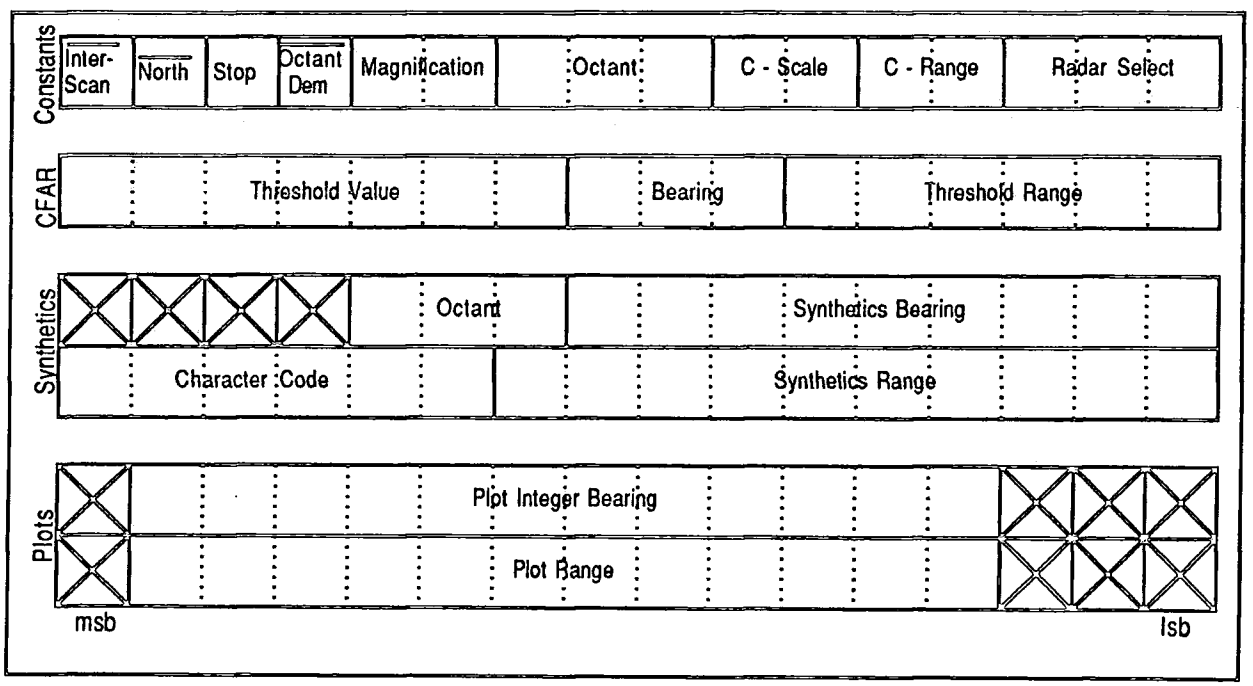


Fig. 6.2 : Format of System Data

6.3.3 Address Allocation

Use of memory mapping offers a significant improvement in efficiency for some of the RATES data transfers. For example, the clutter map was originally transferred as a 16-bit word, made up from a 7-bit threshold and 9-bit partial address. The most significant 3-bits of the address was supplied by the current octant value (from function G), requiring that clutter-map data transfers be synchronised to the correct octant. The mapping technique allows the CFAR clutter-map to be treated as a section of the processor's own memory space, with the threshold address being supplied directly from the KDM's address bus, independently of the currently scanned octant. Data transfers must still occur during the interscan period and some care is needed to avoid conflict with the CFAR's own access cycles. The control processor therefore monitors the state of the INTERSCAN signal which is obtained as the most significant bit of the Constants Word.

Although similar to the above case, the synthetics character memory, ECM report buffer and test-target latches are not treated as blocks of memory, as their current design would require extensive modification to allow true memory mapping. For a production system, however, these changes would be acceptable and could be implemented with little difficulty.

In designing the address map for the plot extractor, the aim was to simplify the interface through minimum address translation. The map for the development board is given below, using address space from 00000_{16} to $3FFFF_{16}$:

$00000	$003FF	MC68000 Exception Vectors (in RAM)
$00400	$006FF	MACSbug Vectors (in RAM)
$00700	$1FFFF	User RAM (128K bytes)
$20000	$21FFF	MACSbug Monitor Program (ROM)
$22000	$2FFFF	EPROM Expansion Space
$30000	$3FEFF	Spare
$3FF00	$3FFFF	On-board Peripherals

To allocate a total of 11 peripheral addresses, with an approximately even spread of read-only and write-only locations, the most economical translation is obtained using a 32K-byte block, and addresses spaced at 4K byte intervals. Allowing for possible future EPROM expansion, the best solution for the development system is to use the block of memory from $30000 to $37FFF, and to translate only the upper 6 address bits, plus A1 and WRITE. This also allows for possible future peripheral expansion into the $38000 - $3FEFF region. The resulting address map is given in **Table 6.2.**

ADDRESS	PLOT EXTRACTOR FUNCTION	STATUS
30000 - 30FFF	CFAR Clutter Map	Write only
31000 - 31FFF	"	"
32xxx	Synthetics Display Characters	Write only
32xxx	ECM Report Buffer	Read only
33000 - 33004	VDU Status Monitor	Read/write
34000	Rollball X Coordinate	Read only
34002	Rollball Y Coordinate	Read only
35000	Constants Word	Read/write
35002	No. Input Plots	Read only
36xxx	Test Target Coordinates	Write only
36xxx	Ships Motion Word	Read only
37000	Plot Input Buffer	Read only

Table 6.2 : Plot Extractor Peripherals Map

6.3.4 The Interface to the Plot Extractor

The memory map allows all of the plot extractor functions to be accessed, simply by reading from, or writing to, a specific memory location and thereby dispenses with functions F and H, which were responsible for this action in the prototype. The design of the original hardware incorporates low-power tri-state latches for most of the addressable functions, so these be used to interface directly to the (buffered) input/output bus of the KDM board. Interfacing is, in most cases, reduced to simply providing an enable signal for these latches. The only complication that arises is in ensuring that those functions which latch output data from the computer are synchronised to the appropriate data strobes. The complete interface circuit is built on a single board and included in the plot extractor as Function K1. A schematic diagram of the board is shown in Figure 6.3.

The eighteenth address line from the KDM board is brought out to the interface on one of the unused pins (PWR FAIL), as it is not normally available for external use. This is combined with the 5 other most significant address lines, and used as inputs to a device mapping PROM. The PROM allows the exact configuration of peripheral addresses to be changed at a later date, without the need to rewire the interface, or the connections to it. The remaining two address inputs to the PROM are provided by address line A1 and the Read/Write strobe line, allowing differentiation between a read-only and write-only location.

When correctly addressed, and accompanied by an address select strobe, a device-select code is output from the PROM, and used as direct input to a 3-to-8 line decoder, which provides the appropriate enable signal for the various plot extractor functions. The most

significant bit of the output code is used to enable the decoder and is always set if the device address is legal - thus there is no output from the decoder for an invalid device address. All devices which can be written-to have their enable signals gated with the upper and lower data strobes, since these define the timing for the write cycle. Once the strobes go high, the enable signal is removed and the new data latched in the plot extractor hardware.

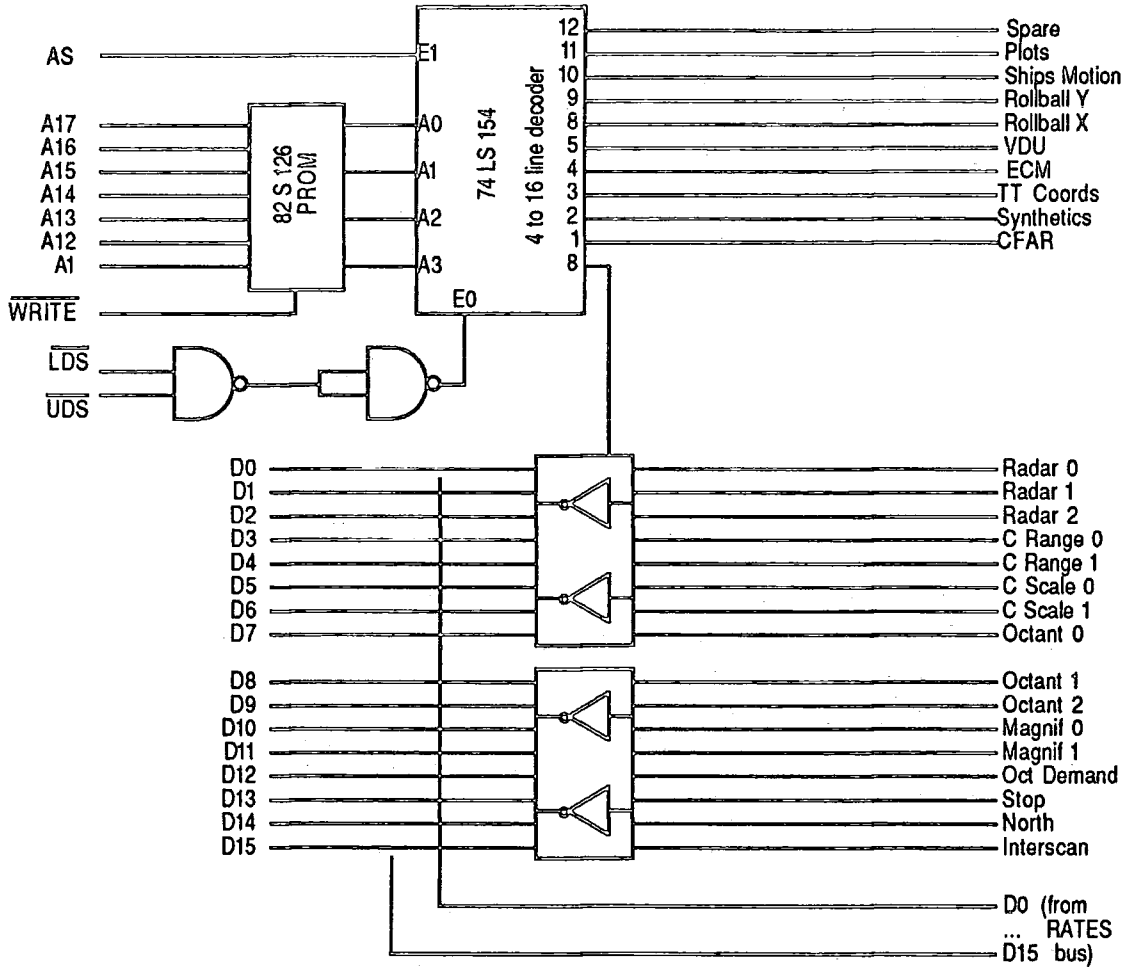


Fig. 6.3 : Computer Interface (K1) Schematic

The outputs from the 3-to-8 line decoder are enabled by the AS strobe, which ensures that data is only passed to the computer bus when a valid address is present on the address bus. This strobe also controls the tri-state outputs of the data buffers.

The data-acknowledge timing for this interface (DTACK) is derived from the computer board and produces a cycle time which is the same as the main processor memory. None of the interface devices has been found to give rise to errors using this timing cycle, which is illustrated in **Figure 6.4.**

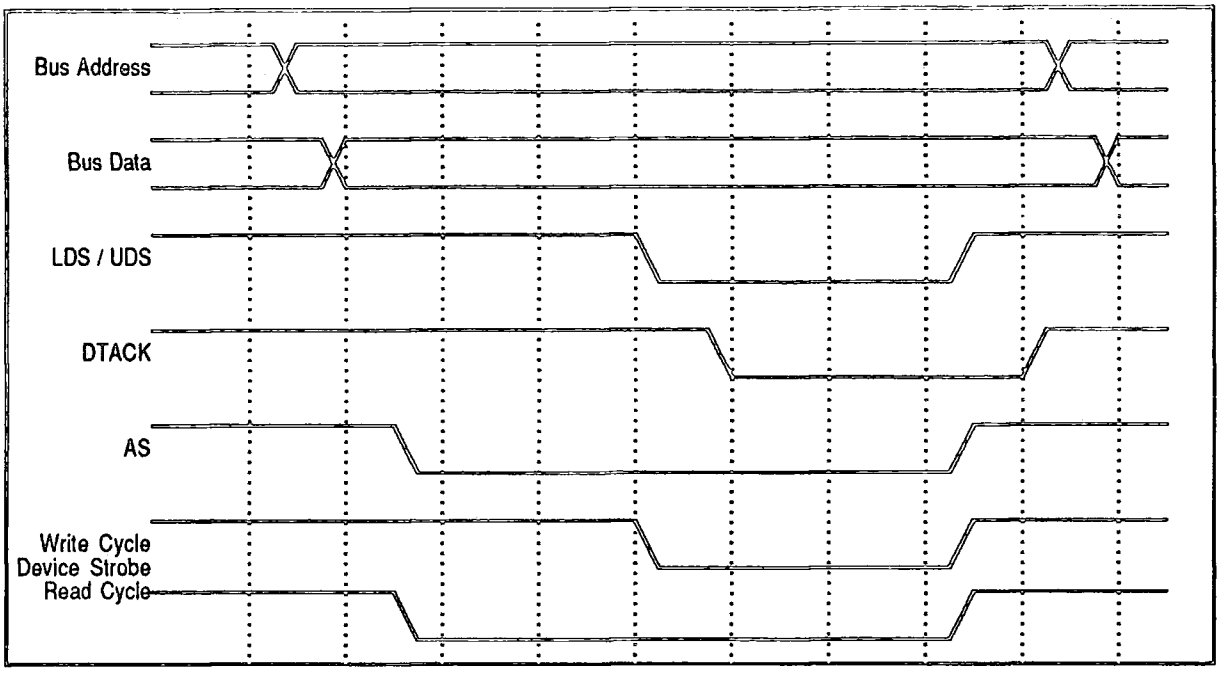


Fig. 6.4 : Plot Extractor Interface Timing

For the majority of the RATES devices, the interfacing procedure is relatively straightforward: they may be read or written at any time regardless of the state of the plot extractor cycle. For certain devices, however, access is limited to either a scan or interscan period to prevent conflict with internal cycles. During the operation of the plot extractor hardware, the following transfers must take place as indicated:

Scan Period:

Synthetics character data.

Interscan Period:

CFAR threshold data

Test target coordinates.

In addition to these restrictions, it should be noted that plots can only be transferred following receipt of an **Octant Demand** signal, or in overload conditions an **Interoctant Demand**. This limitation is due to the peculiar double-buffering arrangement of the plot extractor interface, and results in a somewhat inefficient operation of the plot output process.

In practice, the synchronisation of the data transfers to the plot extractor hardware is left to the software in the tracking computers, whilst correct device activation and cycle timing remains under the control of interface function K1.

The PROM on this board (type 82S129, 256x4-bit) is programmed with a code which corresponds to the device which must be activated at a given address. For the 11 devices listed, the codes are given below, in Table 6.3.

CFAR	01	SYNTHETICS	02
TEST TARGET	03	ECM	04
VDU MONITOR	05	CONSTANTS	06
No WORDS	07	ROLLBALL X	08
ROLLBALL Y	09	SHIPS MOTION	0A
INPUT PLOTS	0B	(INVALID ADDRESS)	00

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
60	1	1	1	1	2	0	5	5	0	0	0	0	3	0	0	0	writeable devices
70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
A0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
B0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
C0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
D0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
E0	0	0	0	0	4	0	5	5	8	9	6	7	A	0	0	0	readable devices
F0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Table 6.3 : Interface PROM Programming Codes

It should be remembered that the above table decodes addresses based on the upper six most significant address lines from the processor, the least significant address line (A1) and the write status strobe. The use of the PROM allows future devices to be added simply through reprogramming, without the need to rewire any of the interface. It is analagous to the address decoding PROMs on the MEX 68 KDM board, which were also reprogrammed to take account of the changes in use of the 68000 address space (Cowan, [55]).

6.3.5 The Plot Output Interface

The interface linking the plot extractor hardware to the tracking computer presents the greatest potential for bottlenecks during processing, as the original design has made a number of assumptions concerning the volume of plot data passing through this point. Function E4 has been designed around a maximum of 512 plots per octant, with provision made for occasional overflow conditions, (which cause an interoctant transfer to be initiated).

Plot data is collected by function E4 in one of two FIFO buffers, and when one buffer becomes full they are switched and the computer requested to handle a transfer into its memory. The buffer switch is essential to avoid loss of plot data while the computer responds to the request. Whilst this remains a viable option for the redesigned tracking system, the inefficiency of the double-buffer configuration draws the conclusion that some improved interface should be attainable.

Considering the operation of the plot extractor (functions **E1...E3, E5**), it has been noted from observations of the prototype system that there is little difference between the 3 and 5 cell plot depth options during plot forming. The latter allows for a guard zone of 2 range cells before another plot can be started, and it can therefore be established that there must be a minimum delay of 2 range cell periods between output plots from function **E3**.

Taking the worst-case parameters defined in Chapter 4, this would imply that the maximum rate of plot transfer *directly from the plot forming section* would be one plot every three range cells, corresponding to a frequency of 1.3MHz. With typical memory access times of only 250ns, a viable solution would be to transfer data directly from function **E3** into the tracking computer memory. To this effect, the following two solutions are proposed:

- Design a customised interface which permits DMA transfer between the plot forming processor and the tracking computer, using discrete logic or semi-custom devices.

- Implement a general purpose interface, using a proprietary DMA controller and associated control circuitry.

A third option, to implement a DMA interface between the existing function **E4** and the computer shall not be considered, as it does little to alleviate the problem of slow data transfer rates. The other two options both have advantages, and are now discussed in detail.

6.4 A Custom DMA Interface

A custom DMA interface represents, potentially, one of the most economical solutions to the problem of transferring data from the plot extractor hardware to the tracking computer. In theory, it would allow a dedicated link between the output of function E3 and the memory on the processor board, and could be optimised for a fast response. The scale and complexity of the design is determined by the timing requirements of both sides of the interface: should these be too complex, it is unlikely that the best solution can be achieved through custom logic.

6.4.1 Interface Timing Considerations

The interface timing is dictated largely by the 68000's bus request sequence, but must be initially under control of the plot extractor hardware. The plot forming controller E1 generates timing signals which are distributed to the other functions in this group. Of these the signal WP is the most important, causing a word to be written from the plot calculator to the buffer. It has a worst-case period of 1 range-clock pulse, and it is therefore necessary to achieve a response from the processor during this time. Fortunately, the plot extractor guarantees that no plots will overlap by less than 2 range periods and hence the following conditions apply:

Maximum period to alert processor to transfer: 250ns
" " to transfer plot output: 500ns

A study of the MC 68000 timing diagrams will indicate that the worst-case response to a bus request is during a processor bus cycle. In these conditions, the processor still responds in 3 processor clock cycles which, for an 12MHz part running at full speed, is equal to the worst-case WP period.

A complete timing diagram for the bus arbitration and output sequence is given in Figure 6.5 - the exact sequence is as follows:

1) Plot forming hardware issues WP pulse, approx. 250ns duration, active low.
2) Bus Request (BR) issued from interface to processor, to acquire control of bus.
3) Processor responds with Bus Grant (BG), acknowledging availability of bus.
4) After short period of time, interface replies with Bus Drive (BD) & takes control of bus.
5) On rising edge of Plot Latch Clock (PLC), address is latched onto data bus, address counters are incremented.
6) Data and write strobes go low, as PLC goes high to allow bus to settle.
7) On rising edge of STROBES, azimuth data is latched into memory on processor board.
8) Range Select (RS) goes high, latching range data into plot forming output buffer.
9) PLC goes high again, putting new address onto address bus, and incrementing address counters.

10) After 1 clock period, STROBES go high latching range data into processor board memory.
11) After 1 further clock period, Bus Drive goes high and bus is released back to processor.
12) Interface cycles, waiting for second WP pulse. Once this goes high, the interface is effectively reset back to the initial state, waiting for an active low pulse on WP.

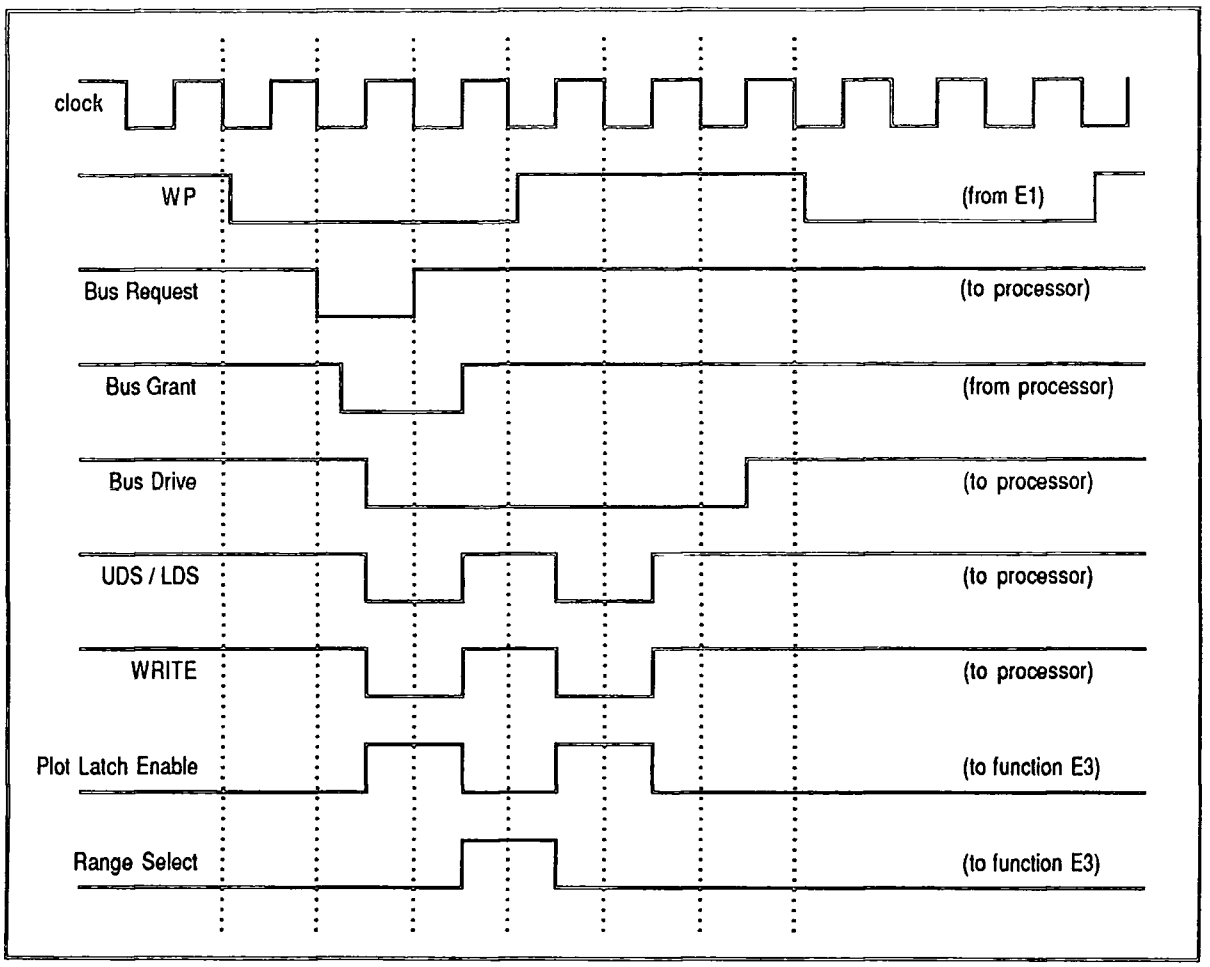


Fig. 6.5 : Interface Timing Requirements

It should be clear from these details, that timing is critical for this process to succeed: If the interface timing clock is too fast, the memory cycle (address setup to write strobe) may be too short and data lost. If the clock is too slow, the interface may not be fast enough to cope with the incoming plot rate. The speed of response of the plot forming hardware is not a limiting factor however:- it uses low-power Schottky devices, with a typical response time of 10's of nanoseconds.

6.4.2 Design of a DMA State Machine

The implication that the DMA interface for the plot forming function is based around a specific set of timing parameters, rather than simple boolean expressions, draws the conclusion that the interface design can best be achieved using a state machine representation. Using this technique, the DMA process is described as a sequence of operations, with specific outputs at each stage, and a set of defined inputs which cause the process to advance to the next stage.

The previous section has already discussed the inputs and outputs relevant to the operation of the interface, and by equating similar outputs this may be summarised as follows:

INPUTS	OUTPUTS
WP (Write Pulse)	BR (Bus Request)
BG (Bus Grant)	BD (Bus Drive)
	STROBE (UDS/LDS & WRITE)
	PLC (Plot Latch Clock)
	RS (Range Select)

The state machine is described by allocating an initial state (1) then defining the input conditions which cause a switch to state 2, and the corresponding change in output as a result. The DMA interface is described in full, using this technique, in **Figure 6.6.**

The initial state numbers are allocated randomly from 1, although they may not represent the ideal choice for a hardware design and may be changed later. The state changes are described as cause and effect, for example the expression:

0X / 010

linking state 1 to state 2 implies that while the machine is in state 1, an input of **01** or **00** will cause a change to state 2, and the output will then become **010.**

For the DMA interface, the input is specified in the form:

WP. BG /

and output as:

BR. BD. DS. RS. PLC

An X in the input condition implies that a binary value of 1 or 0 from this signal will satisfy the part-expression.

A study of the state machine description will indicate that it entirely satisfies the conditions expressed in the timing diagram (Figure 6.5).

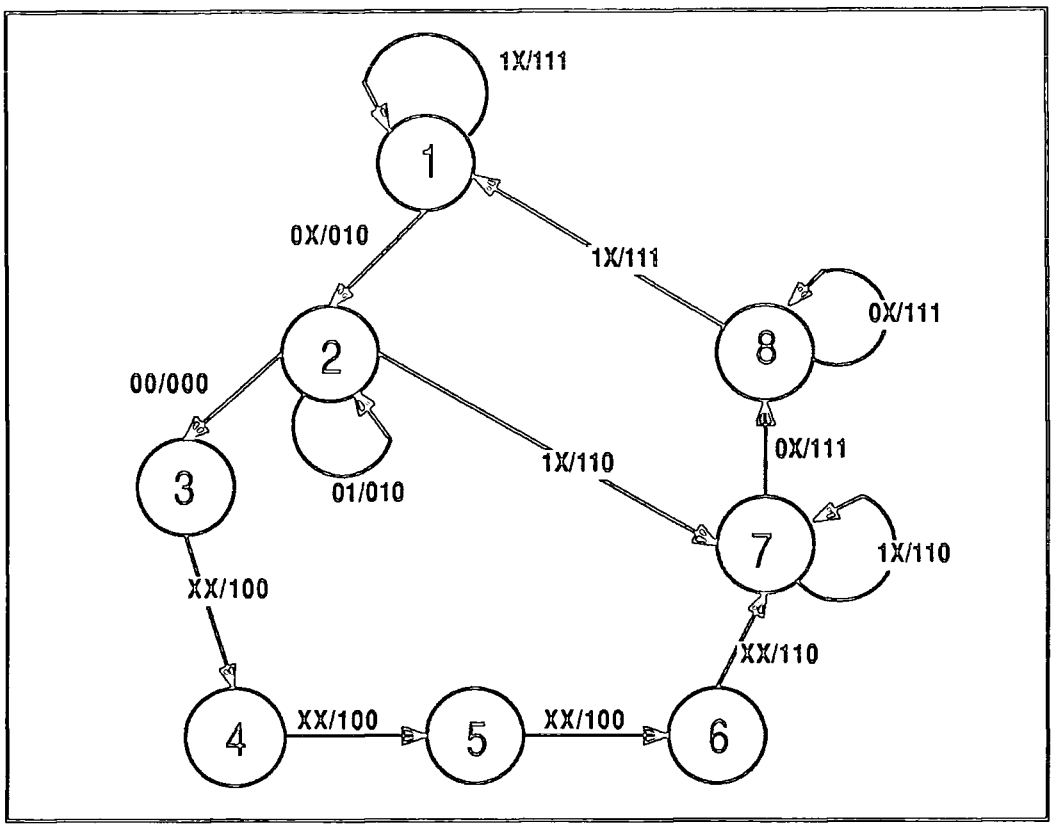


Fig. 6.6 : DMA Interface State Machine

The DMA interface requires only 8 unique states to realise its cycle, and it is therefore possible to represent this machine using a 3-bit binary counter. Whilst the entire design could be achieved using small-scale integrated logic, the best solution to a practical implementation is to design around a conventional programmable-logic array (PLA). This solves the problems of an otherwise complex gate-logic design, and allows other features to be included to improve the overall operation of the interface.

Current PLA devices all conform to a set of standard definitions, regardless of manufacturer, and the one which represents the best solution to the DMA interfacing problem is the 16R4 (16 I/O pins, 4 D-type latches). The registers are totally independent, and it is necessary to include programming logic for the counter which describes the internal states. To optimise the device for high-speed glitch-free operation, the counter should work around a modified Gray code and the following sequence is suggested (with reference to Fig. 6.6):

011 (1), 001 (2), 000 (3), 010 (4), 110 (5), 100 (6), 101 (7), 111 (8)

The sequence is somewhat arbitrary, and is chosen after a certain amount of trial-and-error in establishing the solution which results in the simplest set of boolean expressions. The state diagram can now be converted to tabular format, expressing next-state and output as a function of present state and input. The result is given below in Table 6.4:

Present State Q0 Q1 Q2	Input 0 0	0 1	1 0	1 1
0 1 1	001 / 01100	001 / 01100	011 / 11100	011 / 11100
0 0 1	000 / 00100	001 / 01100	101 / 11100	101 / 11100
0 0 0	010 / 10001	010 / 10001	010 / 10001	010 / 10001
0 1 0	110 / 10110	110 / 10110	110 / 10110	110 / 10110
1 1 0	100 / 10001	100 / 10001	100 / 10001	100 / 10001
1 0 0	101 / 10100	101 / 10100	101 / 10100	101 / 10100
1 0 1	111 / 11100	111 / 11100	101 / 11100	101 / 11100
1 1 1	111 / 11100	111 / 11100	011 / 11100	011 / 11100

Input is: WP. BG
Output is: BR. BD. DS. RS. PLC

Key: NEXT STATE / OUTPUT

Table 6.4 : DMA Interface State Table Description

This state table allows a set of descriptions to be drawn-up, defining the output signals and the inputs to the D-type latches in terms of the current value of the state counter and the input signals to the PLA. Although not strictly necessary for a PLA design, these can then be simplified using Karnaugh Map techniques. The results are programmed into a PLA compiler / programmer using a standard set of programming rules.

This stage of design requires several important points to be noted:

1) The device is programmed in terms of its output and input *pins*.

2) Registered gates are programmed in terms of the state of their output pins, following the next *positive* clock-edge input.

3) Outputs are inverted.

6.4.3 DMA Hardware Design and Programming

The complete program takes account of these rules, and a listing is given below. The format is a standard PLA programming language and can be interpreted by reference to any suitable manual. To simplify programming, the pins are assigned mnemonic names on the 5th (pins 1 to 10) and 6th (pins 11 to 20) lines of the program.

```
PAL16R4
RATES Plot extractor interface PLA, implementing DMA functions
R.R.S. ..... 9/8/84

CLK BG WP RCO CK2 FULL POR AS P9 GND
OE RS STR BR Q2 Q1 Q0 PLC BD VCC

/Q0:=   /Q0 * /Q1 +                              ; REGISTER OUTPUT 0
        /Q2 * Q1 +
        /Q0 * Q * BG +
        /Q0 * Q1 * Q2 * /BG * WP +
        /POR +
        /Q0 * Q1 * Q2 * /AS

/Q1:=   /Q0 * /Q2 * /WP +                        ; REGISTER OUTPUT 1
        /Q0 * /Q1 * WP +
        /Q0 * /Q1 * Q2 * /RCO +
        /Q0 * /Q1 * Q2 * FULL +
        Q0 * Q2 +
        /POR

/Q2:=   POR * Q0 * /Q1 +                         ; REGISTER OUTPUT 2
        POR * /Q0 * /Q2 * /WP +
        POR * /Q0 * Q1 * WP +
        POR * Q0 * /Q2

/BR:=   /Q0 * /Q1 * Q2 * /WP * /FULL * RCO +     ; BUS REQUEST
        /Q0 * Q1 * Q2

IF (VCC) BD =   Q0 * Q1 * Q2 +                   ; BUS DRIVE
                Q0 * /Q1 * Q2 +
                Q0 * /Q1 * /Q2 +
                Q0 * Q1 * /Q2

IF (VCC) PLC =  /Q0 +                            ; PLOT LATCH CLOCK
                Q0 * /Q1 * Q2 +
                Q0 * Q1 * /Q2

IF (VCC) STR =  Q0 * Q1 * Q2 +                   ; STROBE (write / enable)
                Q0 * /Q1 * /Q2

IF (VCC) RS =   /Q0 +                            ; RANGE SELECT
                /Q2 +
                Q0 * Q1 * Q2

DESCRIPTION
P2 is bus grant; P3 is write pulse; P4 is buffer full (active high)
P5 is same as CLK; P6 is buffer lock (active low); P7 is power-on reset
P8 is address strobe; P9 is unused; P10 is GND; P11 is output enable (low)
P12 is range select; P13 is strobe; P14 is bus request;
P15-P17 are state counter outputs (not normally used)
P18 is plot latch clock; P19 is bus drive; P20 is VCC.
```

It should be noted that a number of additional inputs have been included to ensure that the state counter starts in a pre-defined state (power-on reset), the number of plots that can be transferred is limited to the maximum address range of the address counter (RCO) and the DMA is stopped at the end of an octant while the word count is read (FULL). These could have been defined in the original state diagram, but their limited effect means that it is simpler to add them by a process of inspection and deduction.

The programmed PLA is included into the complete interface circuit, as shown in Figure 6.7. This is based around the general interface shown earlier (Figure 6.3), but includes the address generation and DMA control circuits as well.

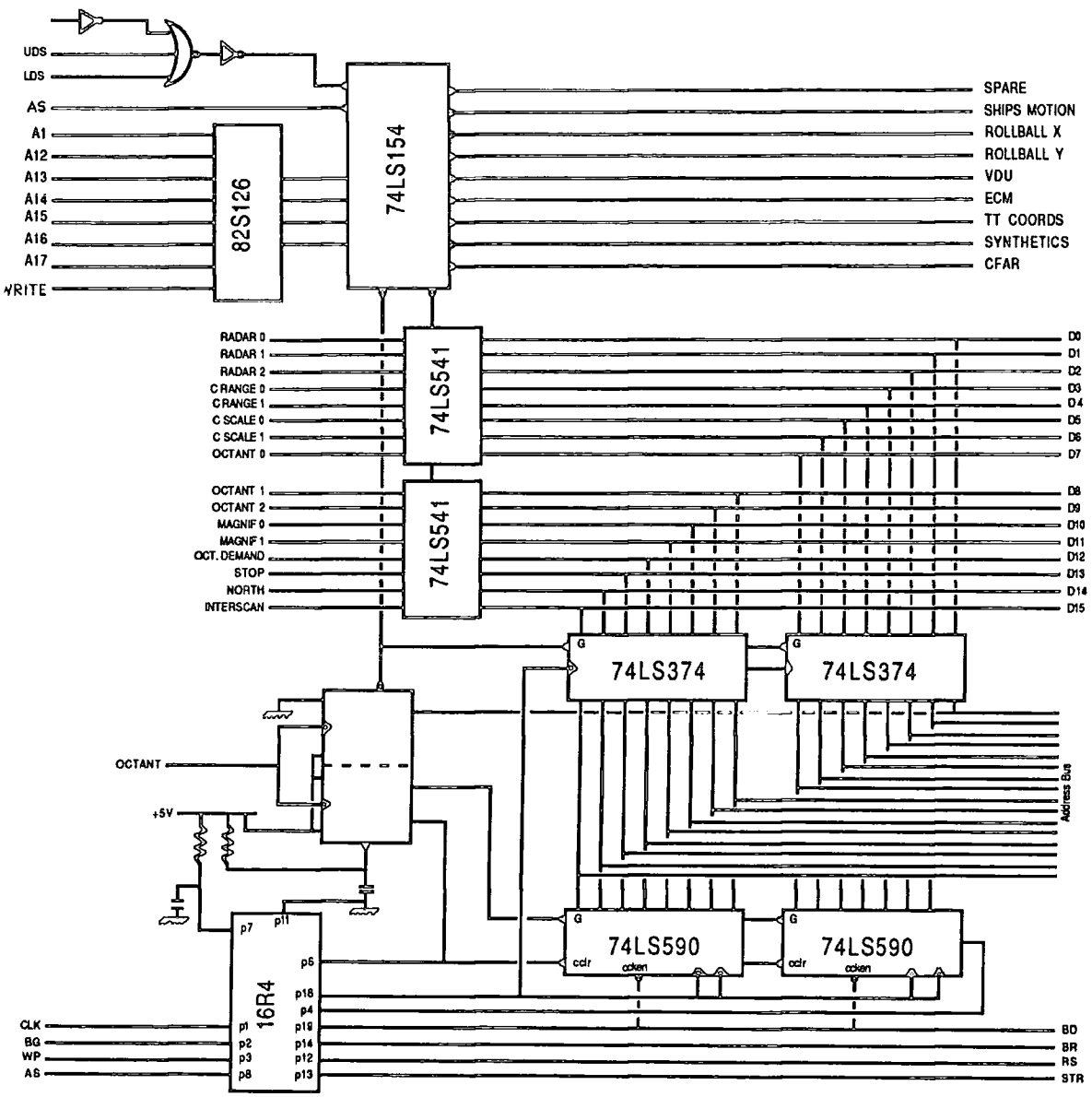


Fig. 6.7 : Custom DMA Interface Schematic

The interface can be connected directly in place of the original function E4, and has been extensively tested using a logic analyser to detect unexpected logic states, such as may arise from plot overload conditions. The circuit gives rise to the timing diagram shown in Figure 6.8, which may be explained as follows:

State 1:

Default (inactive) state: waiting for write pulse from function E1, following from completion of a plot forming operation.

State 3:

Interface has received write pulse from E1, and provided that the previous octant's word count has been read (RCO is low) and the address counter is not full (FULL is low), the machine has proceeded to state 3 and Bus Request has been issued.

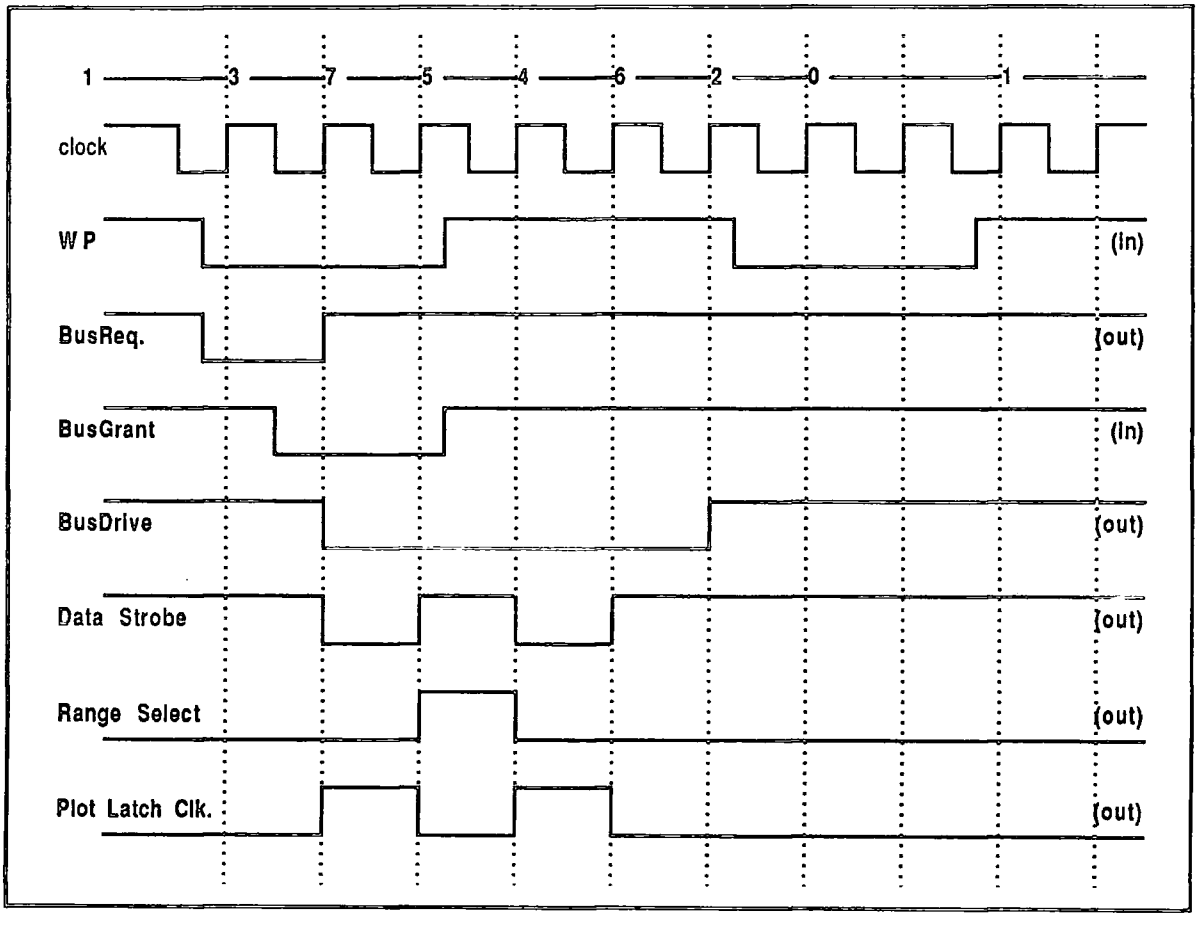


Fig. 6.8 : PLA Timing Cycle

State 7:

Host processor has responded with Bus Grant, and the transfer cycle is started. If WP goes high

before BG is received, the transfer cycle is aborted. In state 7, the PLA activates the Bus Drive signal, which simultaneously connects the data strobes onto the processor bus, and takes the memory write signal low. The first address is output to the global address bus, and this counter value latched into the word count buffer (rising edge of PLC). The data strobes go low and after a single cycle settling time, the data is latched into the processor memory on the rising edge of the strobe signal. The address counters are incremented to the next address.

State 5:

The sequencer issues a pulse on Range Select, to latch the range data on function E3 into the output latch.

State 4:

As in state 7, the next address word is output to the address bus and the data written on the rising edge of the data strobes. The address counters are incremented to the next address.

State 6:

The Bus Drive and memory write signals remain low for a final cycle, to allow the bus to settle.

State 2:

Bus Drive goes high, and the bus is released back to the host processor. The cycle is not complete, as the interface operates considerably faster than the original plot extractor timing sequence and there is a second write pulse due from function E1.

State 0:

WP is received from E1, and the interface enters a loop, awaiting the completion of this cycle.

State 1:

The interface is reset to its original state, awaiting a write pulse following completion of a plot forming operation.

If the counters reach their maximum permissible value during data transfer, the ripple carry output is fed back into the PLA, and further transfers are inhibited as the interface is held in state 1. This condition is reset by reading the data-word count buffer, which also resets the address counters to zero. Successive octants transfer data to one of two areas of memory, depending on the value of the address line A13, which is toggled upon receipt of the Octant signal from timing function G2.

6.4.4 Interface Testing and Reliability

The tests on this interface have shown that the design is generally reliable in terms of transferring data out of the plot extractor hardware but, as may be expected from the above description of operation, shows failings when transferring to a heavily loaded host microprocessor. This may be explained by one of two (independent) reasons:

1) Under certain conditions, the MC68000 does not respond with an inactive address strobe (AS) within the permitted time period - the sequencer therefore assumes an abandoned request and the data is lost. To allow for this possibility would require more logic states and would take the design beyond the complexity of current 20-pin PLA devices.

2) The design of the DMA unit blocks data transfer for a short period at the end of an 'octant', whilst the old word count is read from the latches by the host microprocessor. Again this reduces the complexity of the interface, and in practice should only lead to loss of plot data within the guard-ring surrounding the antenna. If the processor is heavily loaded, however, this operation may be delayed and local data may be lost before the DMA interface recovers.

With this design, it should also be noted that the unit is unable to respond to inter-octant transfers, as buffer-switching is under the control of the **Octant Request** input to the J-K bistable.

6.5 Interface Design using Proprietary Devices

The interface described in the previous section was an early attempt to solve the problems of direct data transfer into the MC68000's address space. Its advantages of low cost and high speed operation, coupled with the lack of availablity (at the time) of any suitable alternative indicated that this form of DMA transfer remained the optimum solution to the problem, in spite of the restrictions of flexibility and the limitations discussed earlier.

With the introduction of a proprietary DMA controller, the Motorola 68450, it seemed logical to study the applications of such a device in replacing the custom DMA interface, with a view to enhancing the performance and removing the limitations of the earlier design.

6.5.1 The Motorola 68450

The Motorola 68450 is intended as a flexible solution to traditional DMA interfacing problems, and has been designed to afford a maximum of functionality from a single chip. With more high speed devices becoming available, requiring rapid block transfer into processor memory, the limits of a single DMA channel are obvious and this device therefore offers four such channels. Each may be programmed using on-chip registers, allowing a large number of operating modes and ensuring compatibility with earlier 6800 devices - an important point which enhances Motorola's standing through its provision of "future-proof" devices.

The MC68450 is designed around a 16-bit bus, which carries the multiplexed data and upper address buses. It is capable of addressing memory in the same range as the MC68000 itself, and can accomplish either memory↔memory or memory↔device operations using 8, 16 or 32 bit operands. The device can be configured to start transfer following an external signal, or under software control, and the transfer can then take place using cycle-steal techniques or by complete bus control. Additionally, the source and destination addresses can be programmed to increment or decrement independently during transfer operations.

The large number of operating modes suggests that this device will offer the ideal solution to the problems of DMA transfer from plot extractor to tracking computer. A study of the appropriate data sheets for the MC68450 will indicate a wide variety of timing cycles, and it is therefore important to choose the right operating conditions for the device to ensure successful operation with the timing requirements discused in Section 6.4.1. The timing diagram for the interface to Function E3 (Figure 6.4) still applies, as the same signals can be used to generate and control the transfer request (i.e. requests will be initiated through WP from Function E1). The transfer of both range and azimuth data must be completed within a worst-case cycle time of 500ns, which corresponds to 6 cycles at 12MHz clock rate. The fastest transfer mode available with the MC68450 permits 4 cycle operation, but will only allow a

transfer using 16-bit operands (not long word) in this mode. The complete plot transfer must therefore take place as two separate single word operations, each initiated by its own WP write pulse.

At present, the only commercially-available version of the MC68450 runs at a maximum of 8MHz and it is therefore impossible to demonstrate full speed operation under worst-case conditions. The plot extractor interface can, however, be designed around any of the slower radars to illustrate the potential of this device should faster components be released onto the market. With this in mind, the interface has been constructed and tested up to maximum plot transfer rates:

6.5.2 Hardware Integration and Operation

The MC68450 has been designed to work with an MC68000 host processor using a minimum of additional interfacing circuitry. The majority of the interfacing connections are either tri-state or bi-directional and can therefore be wired directly to the counterparts on the KDM development board, and for this reason the DMA controller and associated components have been assembled on the prototyping area that is provided on this board. The device has been wired on the unbuffered side of the bus, with address and control lines connected to the outputs of the appropriate processor signal buffers. The data lines are connected in place of the (removed) parallel i/o chips. With this configuration it is important that the direction of the complete bus is not reversed when the DMA controller is bus master. Additional integrated circuits are included to separate the multiplexed address and data buses, but subsidiary logic makes use of spare gates in the existing memory expansion circuit as much as possible. The complete circuit (including memory expansion logic) uses a total of 7 integrated circuits and is illustrated in **Figure 6.9**.

The device is capable of being triggered by an active signal on any of the REQ inputs, but is configured to use only channel 0 for the plot forming interface. All other request inputs are tied high through resistors. The three lines carrying bus exception codes (BEC0...BEC2) are derived from the modified control interface (Function K1), which normally holds these inputs high unless some form of error occurs during data transfer. Peripheral control lines (PCL0...PCL2) are not used, as the interface is under control of the output from ACK0 which goes low for the duration of the data transfer. This signal is used to control the direction of certain KDM data-bus signals without reversing the entire bus, as would have happened prior to modifying the KDM board. The system clock is derived from the same source as the host processor, such that both parts operate at a nominal 8MHz.

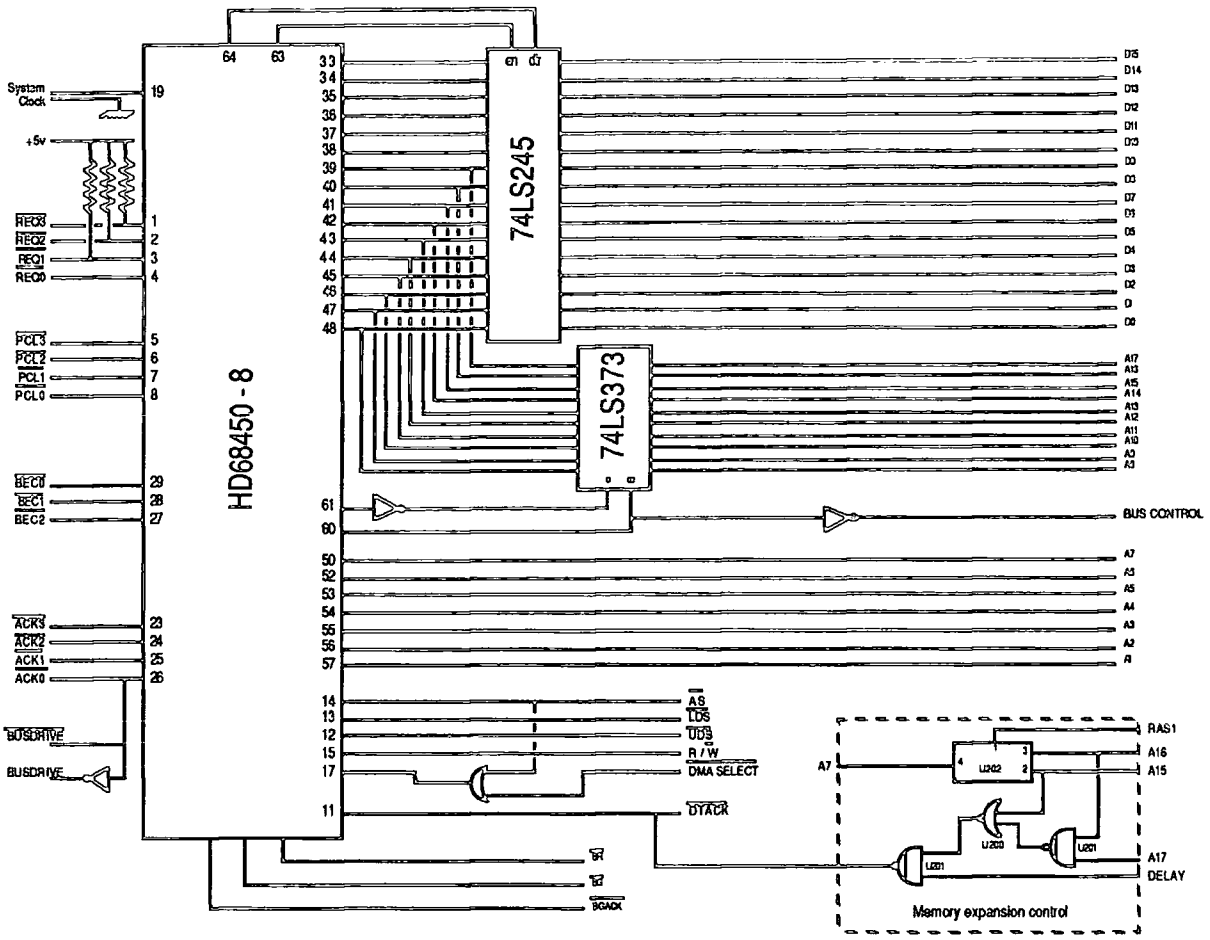


Fig. 6.9 : 68450-based DMA Controller Schematic

The signal, BUS CONTROL, is used to force the KDM board address and control buffers into a high impedance state whenever the DMA controller is bus master. The lower 8 DMA address bits are bidirectional and can therefore connected directly to the outputs of the corresponding processor signals.

The remaining part of the interface is a modified version of that used in the custom-designed Function K1. The sequencing logic in the PLA is no longer required, however a PLA design is used to simplify the enable logic for the various interface functions. The word counters are also retained to provide a measure of the accuracy of the DMA controller's transfer mechanism, as they will count every pulse on WP, regardless of whether a DMA transfer takes place. The schematic of this circuit is shown in Figure 6.10.

The circuit can be split into two parts : the address mapping control and the DMA transfer logic. The 82S126 is programmed as described in Section 6.3.4 (Table 6.3), with the

exception that the address translation for input plots is no longer required. The PROM produces mapping codes that are used as selection inputs to the 4-to-16 line decoder (74LS154). The decoder is enabled by a combination of an active address strobe (AS) and an active input from either of the data strobes. The logic for this operation is included in the PLA, making use of spare output capacity within this device to avoid the need for extra SSI devices.

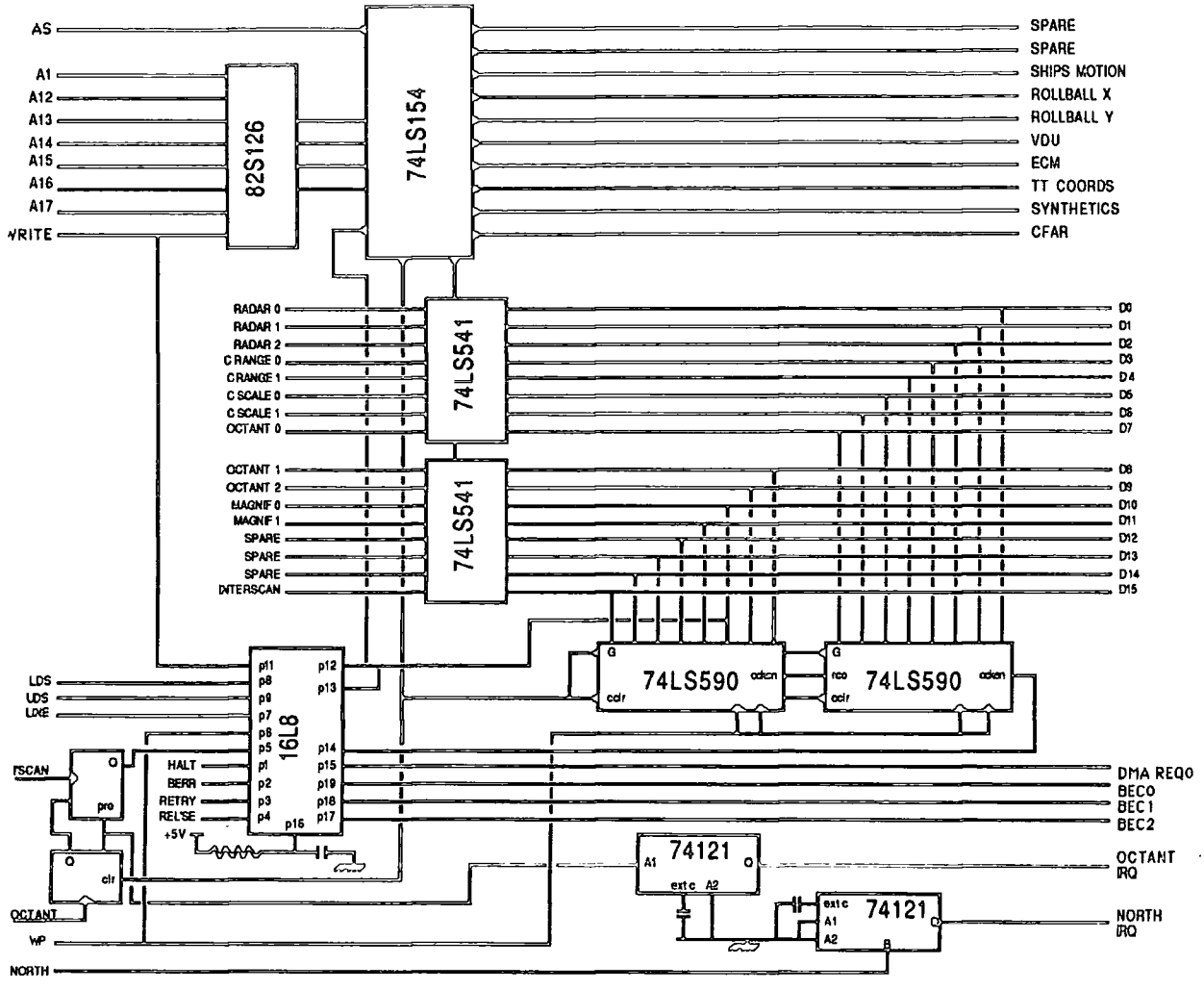


Fig. 6.10 : Function K1 : DMA <-> Plot Forming Interface

The constants word can still be enabled onto the data bus although the **Stop** signal is no longer required. **Octant Demand** and **North** signals are also ignored, as this modified interface is more suited to interupt-driven timing control in place of the polling techniques used in the **Function E4**-based design. Both signals are re-timed to a fixed period instead of being dependent on the rate of the range clock.

The DMA control side of the interface is still based around input pulses from **Function E1** which are used to request a DMA transfer to take place. The additional logic in the PLA is

used to limit transfers to certain conditions and prevent overflow of the processor memory. Following a power-on condition, the interface looks for an active low signal on WP. When this is received, the word counter is incremented and a request signal generated on DMA_Request. The DMA controller responds by transferring a single word from Function E3 to the processor memory. The plot forming controller generates the plot latch signal to latch the azimuth data in the output buffer, then sets WP low again to transfer the second word.

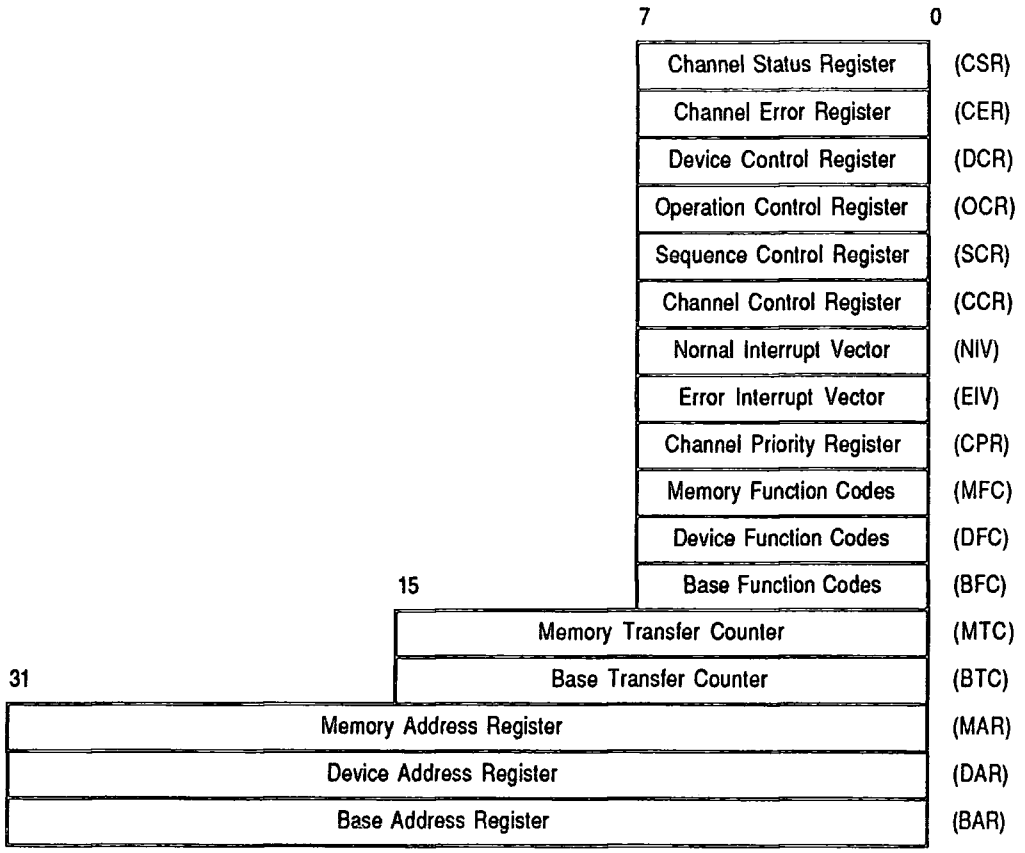
This process continues until either 2048 words have been transferred in a single octant (in which case further DMA_Requests are inhibited), or Octant Trigger is activated by G1. At this point, Octant_IRQ is activated and pin 5 of the PLA goes high inhibiting further DMA transfers. The interrupt causes the processor to reprogram the DMA controller with a new base transfer address, then to read the word count from the interface. The read signal clears the Octant Trigger latch and pin 5 of the PLA is taken low again following receipt of the next interscan signal. This process ensures that the interface should be reset in only one whole transfer period, with minimum loss of data. The LINE signal allows the entire DMA interface to be disabled under external control.

The PLA also includes logic to encode a number of potential bus error conditions onto the three bus error control lines (BEC0...BEC2), although only the reset option (from power-on-reset) is used in this implementation. Programming details are given below:

```
PAL16L8
RATES Plot extractor interface PLA controlling 68450 DMA.
R.R.S. ..... 3/1/85

HLT BER RTR REL INH WPI LIN LDS UDS GND
WRT FUL DEN CEN REQ POR BE2 BE1 BE0 VCC

IF (VCC) DEN =  /LDS * /UDS                    ; Decoder enable signal

IF (VCC) CEN =  INH + FUL + LIN                ; Word counter enable

IF (VCC) REQ =  WPI + INH + FUL + LIN          ; DMA request signal

IF (VCC) BE2 =  POR * /HLT +                   ; BEC2
                POR * /BER +
                POR * /RTR +
                POR * REL

IF (VCC) BE1 =  POR * /HLT +                   ; BEC1
                POR * BER * RTR

IF (VCC) BE0 =  POR * HLT * /BER +             ; BEC0
                POR * HLT * RTR

DESCRIPTION
P1 is Bus Halt; P2 is Bus Error; P3 is Bus Retry; P4 is Bus Release
P5 is Transfer Inhibit; P6 is WP; P7 is LINE; P8 is Lower Data Strobe
P9 is Upper Data Strobe; P10 is GND; P11 is Write; P12 is Max. Word Count
P13 is Decoder Enable; P14 is Counter Enable; P15 is DMA Request
P16 is Power-on Reset; P17 is BEC2; P18 is BEC1; P19 is BEC0; P20 is VCC.
.....
```

6.5.3 DMA Programming Considerations

For successful operation, the DMA controller requires careful programming of each of its control registers. For each DMA channel there are 12 8-bit, 2 16-bit and 3 32-bit registers which allow programming of a variety of options. For full details of all registers it is important to study the appropriate manufacturer's technical literature, however brief details are given below to justify programming methods. The registers are selected by supplying an 8-bit address and activating the chip-select input.



For identification of parameters, the DMA nominally transfers data between a device and a block of memory. There is no restriction, however, on the device simply being another block of memory. The start-transfer memory address is programmed into the 32-bit Memory Address Register (MAR). If data is to be transferred from an *addressable* device (rather than one with an acknowledge line), the corresponding device address is programmed into the Device Address Register (DAR). The base address register is only used in chained operations, which are not applicable to this design. The plot forming interface is not addressable, however all other devices in the plot extractor are addressable.

A count of the total number of words to be transferred is programmed into the Memory Transfer Counter (MTC), which is then decremented after each word transfer. When it reaches zero no further transfers are permitted on that channel. For plot transfers, this is loaded with a value of 2048, permitting a maximum of 1024 plots to be transferred per octant.

The Device Control Register (DCR) is programmed with information about the device, and the type of DMA operations to use. The plot interface is treated as a 16-bit device with Acknowledge. The peripheral control line is unused and left as a status input. The channel is configured to run in cycle steal mode without hold, thereby ensuring that transfers do not affect other processing. The remaining information about the transfer is programmed into the Operation Control Register (OCR), which allows definition of a device to memory transfer using 16-bit operands, initiated by an active signal on the REQ line. The Sequence Control Register (SCR) is programmed to allow the memory address to count up by the appropriate amount following each transfer.

The other registers in the device do not play an important part in the plot interface, with the exception of the Channel Control Register, which is used to start or suspend data transfers. Transfers commence when bit 7 of the register is set, and thereafter continue according to the rules programmed into the other registers.

The speed of operation of the 68450 makes it ideal for performing a number of memory transfer operations. In particular, the device is also used to clear blocks of memory by defining the "device" as a single word in memory set to zero, then setting the appropriate counter value and running the channel as a 16-bit 68000 compatible interface with maximum rate auto-request in burst mode. Large blocks of memory can be cleared in this manner in a fraction of the time that the host processor would take. A number of other techniques are also used in the experimental RATES software which is described in the following section.

6.6 RATES Software Design

To reduce development timescales, the Durham RATES software has been based around that used in the prototype design. The important difference in the new system is the lack of a multi-tasking operating environment and the need to address the interface functions at a low level. The use of a high level language for programming is dependent upon a suitable compiler for the MC68000, and equally importantly one which has been designed around the KDM development board. Careful research revealed only a PASCAL compiler, which was restricted to integer arithmetic and produced stack-orientated code. For such a time-critical system, the coding needs to make full use of the registers in the MC68000, and it has therefore been necessary to adopt MC68000 assembler as the programming language.

Only basic concepts have been adapted from the prototype design, as a direct translation from CORAL to assembler would not produce a highly optimised code, which is part of the advantage of using a low-level programming language. The disadvantage of this approach is the increase in the development timescale, due to the problems of debugging and testing at such a low level. The result of this has been a breakdown of the system software into manageable modules, each of which can be seen and tested to have results on the overall operation.

6.6.1 A Note on Release Versions

The software release versions have been based around the modules mentioned above, with significant enhancements forming a major release. Version 3.4 source code is listed in Appendix C.

The first release version of the software established operator control, implementing input and output of system commands via a control terminal, using the command sequences permitted in the prototype system.

The second release established the interface with the plot forming function and the remainder of the plot extractor functions. It allowed output of certain synthetic characters and basic control of plot extractor functions.

The third (and currently the latest) release implements elementary filtering of plot data, and display of general plot statistics on the plot extractor status display. For reasons given later, there has been no further development of the RATES software, although the results clearly illustrate the potential of the microprocessor-based approach.

6.6.2 The User Interface

It was decided from an early stage of development that a useful starting point for the RATES demonstration software would be some form of user interface, ideally duplicating the functions available in the prototype system. The interface would allow operator control of the main tracking parameters, and would also be responsible for control of data flow between the plot extractor hardware and the tracking computer. In establishing the control features at an early stage, it should be possible to build up various modules as required.

The low level operation of the redesigned tracking computer requires considerable care in the design of the interface to various functions, in order to minimise the time spent on non-effective processing operations. For this reason, the optimum solution is to use interrupts to control routines which arise from an external stimulus - the user interface routines are only called when a character is input from the console. To further increase efficiency, the interface operates in a buffered mode and data is only passed on for interpretation when a line terminating character is received.

The flowchart describing the user interface function is shown in **Figure 6.11**, and illustrates the 3 main modules this can be subdivided in to.

Routine **ReadChar** is called whenever a character is received from the operator console. It reads the character and carries out one of several actions as a result. ASCII printable characters are inserted directly into the input buffer and also echoed to the operator's output display. Backspace (Ctrl-H) and delete-line (Ctrl-X) characters are interpreted as would be expected, erasing the previous character or the entire buffer. Only a line-terminating character has any other effect, causing routine **Interpret** to be called to process the complete buffer.

The buffer is initially scanned to isolate one of the 21 available commands. The commands are hardcoded as string constants, and the parser locates one of them by matching as many characters as are typed with the nearest lexical equivalent. For example, the 'SetValue' command can be chosen simply with keyletter 'S', whilst the third test target requires the full command name 'TT3'. The index value of the chosen command is used as an offset to a table of function addresses, from which the routine appropriate to that command is called.

Many of the user control routines simply set or clear bits in a variable, turning display and I/O options on or off. Some routines affect the tracking parameters, and require additional parameters which are read from the input buffer by the routine concerned. If a command cannot be deciphered, or if there is an error in the supplied parameters, the command is ignored and an error message raised. If a system parameter is changed with a resultant effect

on other parameters, the routines DeriveFix and DerivePars are called to recalculate the variables. This is the worst-case action resulting from a command string - under most conditions the main processing functions will be resumed with little noticible effect.

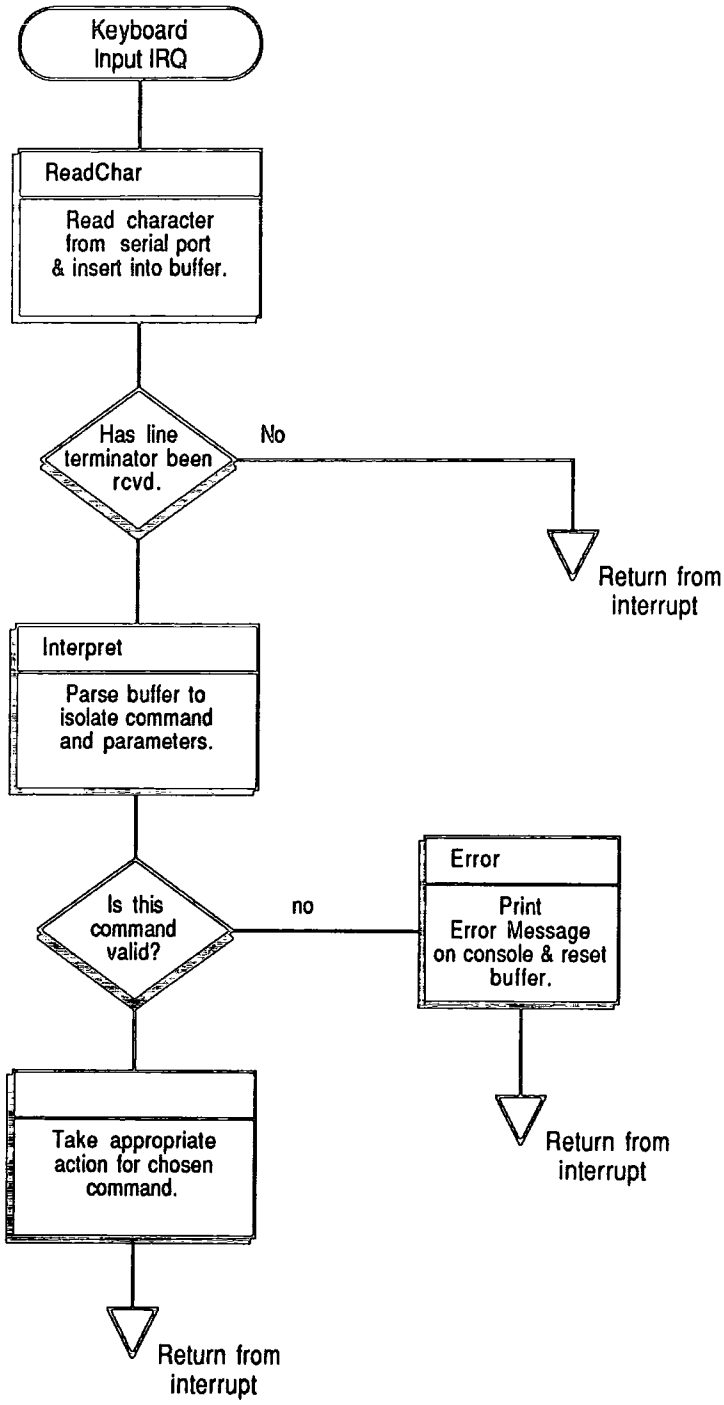


Fig. 6.11 : User Interface Flow Diagram

6.6.3 Status Monitoring and Display

Within the concept of a software development environment, the monitoring of system status and performance is an important requirement in order to assess the potential performance under a full range of operating conditions. The prototype system uses a conventional C.R.T. display, offering a 25 line by 80 column format. The large physical size of the display is unnecessary considering that its only real purpose is to assist in monitoring the performance and would therefore be largely ignored under normal operating conditions. The development system addresses this problem by replacing the C.R.T. with a liquid crystal (L.C.D.) display of similar display area.

The use of an LCD display represents a low-cost solution, which compares favourably with a less robust CRT (similar cost) and a plasma display (very high cost). The display allows the added flexibility of mixed graphics and text as a possible future option to assist performance measurement. Its disadvantage is poor contrast and reliance upon critical lighting conditions to maintain visibility, although this problem is largely removed when considering the use of the backlit super-twist LCD displays that are now available.

The development system makes use of the Hitachi LM225 display, which is addressed as two separate displays each of 25 line x 40 character format. Each side of the display has its own controller (a Hitachi 61830), wired such that the left-hand display can be addressed as a low-order byte and the right as a high-order byte. Careful design of template formats can therefore make use of word-transfers to perform simultaneous updates on both sides of the display. The basic template used by both prototype and development systems is shown in Figure 6.12 and (for the current release) only makes use of the LCD's text mode.

Although the screen is fully addressable, the speed of update and possibility of modifying the hardware at a later date to allow the DMA controller to perform updates has resulted in the use of a processor-RAM based template to hold the display image prior to transfer. ASCII data for display is written directly into this template (which is initially downloaded from the system constants area). Routine VduMapUp ensures that the upper and lower byte distinction is maintained, especially when copying a string that has been derived from an integer parameter by function IntToAsc.

When all required parameters have been converted to their ASCII equivalents, a second routine (VDUOut) is called to transfer the contents of the template to the display-controller memory. This is accomplished in a block of 640 words with an initial 8 word header. The speed of update results in little or no 'flicker' being visible to the user.

The VDU is initialised by routine InitVDU, which copies a set of parameters from the system constants area to the controller status registers during initialisation of the tracking software. This routine also copies the initial template from the system constants area into the local (dynamic) template store. The location of the fields within the template is stored as a set of offsets at address 'VduOffset'. Even offsets correspond to a field on the right of the display, while odd offsets indicate a field on the left.

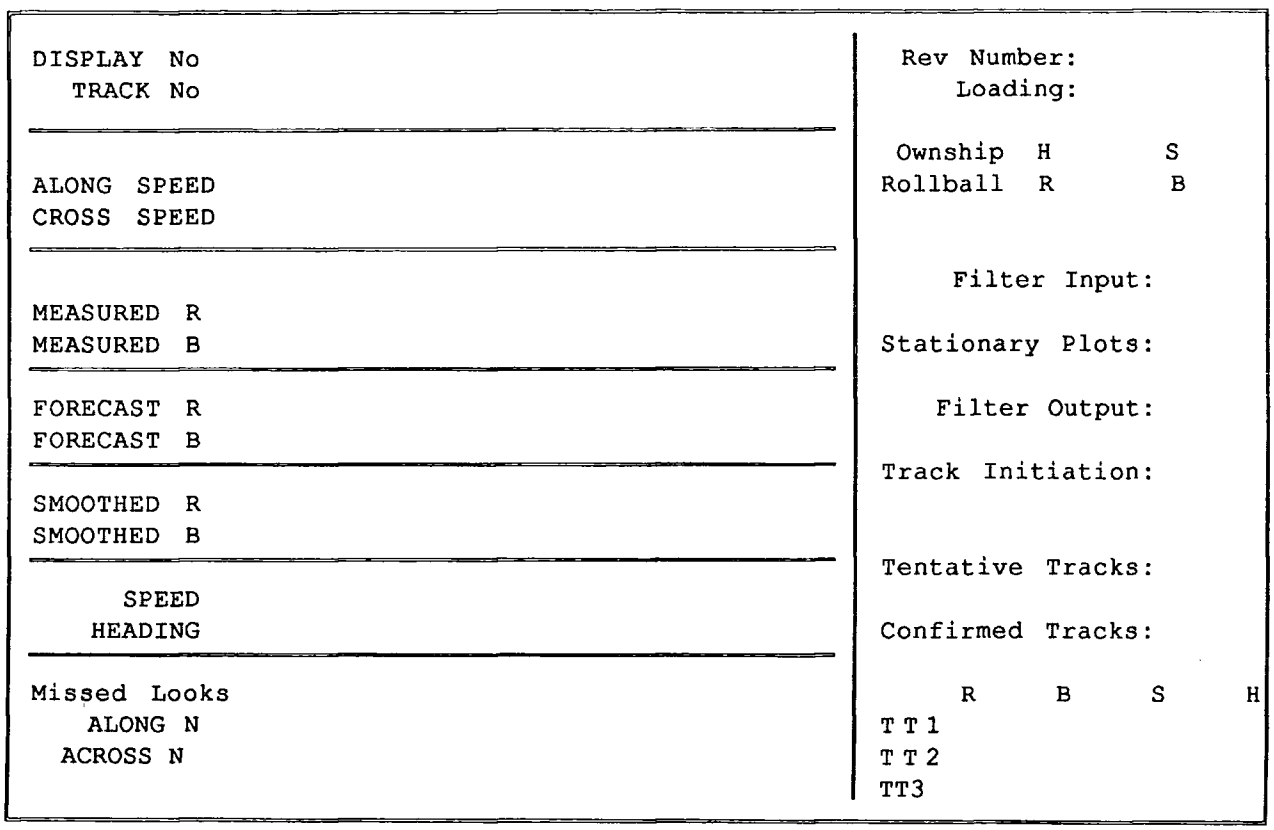


Fig. 6.12 : Status Display Template

The current version of the software (3.4) only makes use of the right-hand display, as the tracking module has not yet been implemented. This portion of the display gives sufficient information to determine some performance characteristics for the system, based on the plot counts and loading factor. Values for these are updated upon receipt of the north marker signal and remain on static display for the duration of one revolution.

6.6.4 Interfacing to the Plot Extractor

The software interface to the plot extractor can be broken down into a number of subsets according to the time-dependence of certain functions. For example, output from the tracking computer to the synthetics character memory can only take place during the scan period as these characters are written to the display during interscan. The table given below illustrates the time dependencies of the various functions:

Parameter	Transfer Details	Time Slot
Test Target	6 words output	interscan
CFAR thresholds	512 bytes output	interscan
Synthetics data	512 words output	scan
Completed plots	2 words / plot input	scan (by default)
Constants word	1 word input	not time critical
Ships motion data	1 word input	" " "

Data transfer is based around a 16-bit data bus and hence there is no time advantage to be gained through byte transfer (CFAR thresholds) - this will be accomplished in the same period as a similar word transfer. The transfer could (with additional logic) be undertaken by one of the channels in the DMA controller, however the small size of the data blocks indicates that the transfer could be accomplished by the host processor with only limited affect on the available processing time per octant. Unfortunately, transfer rates are not fast enough to support a complete block move during either one scan or one interscan period, and hence it is necessary to split the operation so as to handle a number of smaller blocks. Transfers can be controlled by monitoring the status of the scan input, which is conveyed as bit 15 of the Constants word.

Considering the requirements for each transfer operation:

Interscan Transfer:	(CFAR thresholds)		Cycles
LOOP	MOVE.B	(Table_address)+, D0	10
	MOVE.W	D0, (Device_address)+	8
	DBRA	Dn, LOOP	14
			=32
Scan Transfer:	(Synthetics data)		
LOOP	MOVE.W	(Table_address)+, $Dev_addr	20
	DBRA	Dn, LOOP	14
			=34

Each loop must be repeated 512 times, giving a total of 16384 cycles for an interscan transfer and 17408 cycles for a scan transfer.

For an 8 MHz processor, this corresponds to times of 2048 and 2176µs respectively (1365 & 1450µs for 12MHz). Scan and interscan times, according to radar type, are given below in Table 6.5. Based on these times, it is possible to calculate the optimum transfer block size for a given radar, or alternatively to choose a suitable worst-case block size. The results (for an 8MHz CPU) are also given in Table 6.5. If a 12MHz processor is used, the compromise block sizes may be doubled, with the number of transfer blocks correspondingly halved. It should be noted that these calculations assume an access delay of 150 - 200ns; longer delays will require a decrease in the transfer block size.

Radar Type	Interscan Period	Scan Period	Interscan Byte Txfr	Scan Word Txfr	Compromise Block Transfer
A	205µs	1097µs	16 x 32	4 x 128	16b x 32w
B	790µs	512µs	4 x 128	8 x 64	8b x 64w
C	936µs	366µs	4 x 128	8 x 64	8b x 64w
D	834µs	3072µs	4 x 128	1 x 512	4b x 128w

Table 6.5 : Scan & Interscan Timing

In practice, the RATES software assumes the worst case value for the block size, resulting in a data transfer of 16 blocks x 32 words. At the end of each block, the processor uses a polling technique to test the status of the Interscan signal so as to maintain synchronisation with the plot extractor hardware.

Synchronisation of data transfers in the prototype system was largely undertaken by the interface hardware, but relied on data being transferred to holding buffers a set period in advance. The low level interface adopted in the new hardware permits a more rapid response and transfers can now be accomplished immediately after scanning has been completed for a given octant. To assist in timing control, the hardware produces the **Octant Demand** and **Aerial North** signals, which are passed to the tracking computer as interrupts.

One of the main failings of the prototype version was its inability to handle more than 4096 input plots per revolution. This was due to the input plot processing sequence being incomplete when the next Octant Demand signal was received, and resulted in the processor "crashing" as its working data space was overwritten by incoming data. The new system avoids this problem by forcing termination of octant processing as soon as an Octant Demand

signal is received (Appendix C.19). The stack pointer is simultaneously restored to an initial state, ensuring integrity of the processing environment.

The important timing characteristics of the system can be detailed with reference to Figure 6.13 and the flow charts in Figure 6.14.

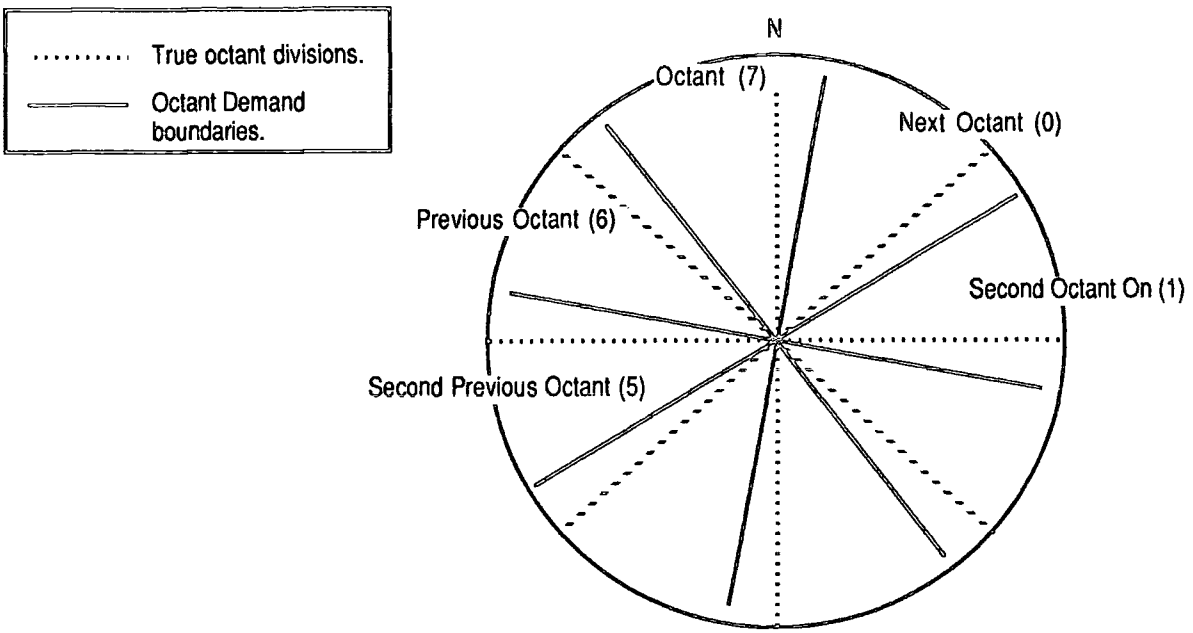


Fig. 6.13 : Octant Timing Control

Only the north marker is output on its true boundary : the Octant Demand signal is actually delayed by approx. 120ms from each octant boundary to prevent conflicts with north marker processing. The octant output from the Constants word lags behind the scanned octant number by one, so as to simplify processing in the prototype system. This feature is no longer necessary in the development system as the interrupt-driven octant processing is capable of reliably maintaining its own octant count, however it is retained to assist synchronisation with the system hardware.

When started, the system waits for the octant output from the timing circuits to reach a count of 5 (the penultimate octant) before enabling north marker and keyboard interrupts to the processor. The first north marker to be received after this causes the system to enter a "warm-up" state, where data is received from the plot extractor hardware, but there is no corresponding output until a certain number of aerial revolutions have elapsed. The software from this point is entirely interrupt-driven, with all other processing functions being called as subroutines from one of the interrupt handlers.

The keyboard interrupt processing has already been discussed (Section 6.6.2), and the flow diagrams for the other two functions are shown below in Figure 6.14.

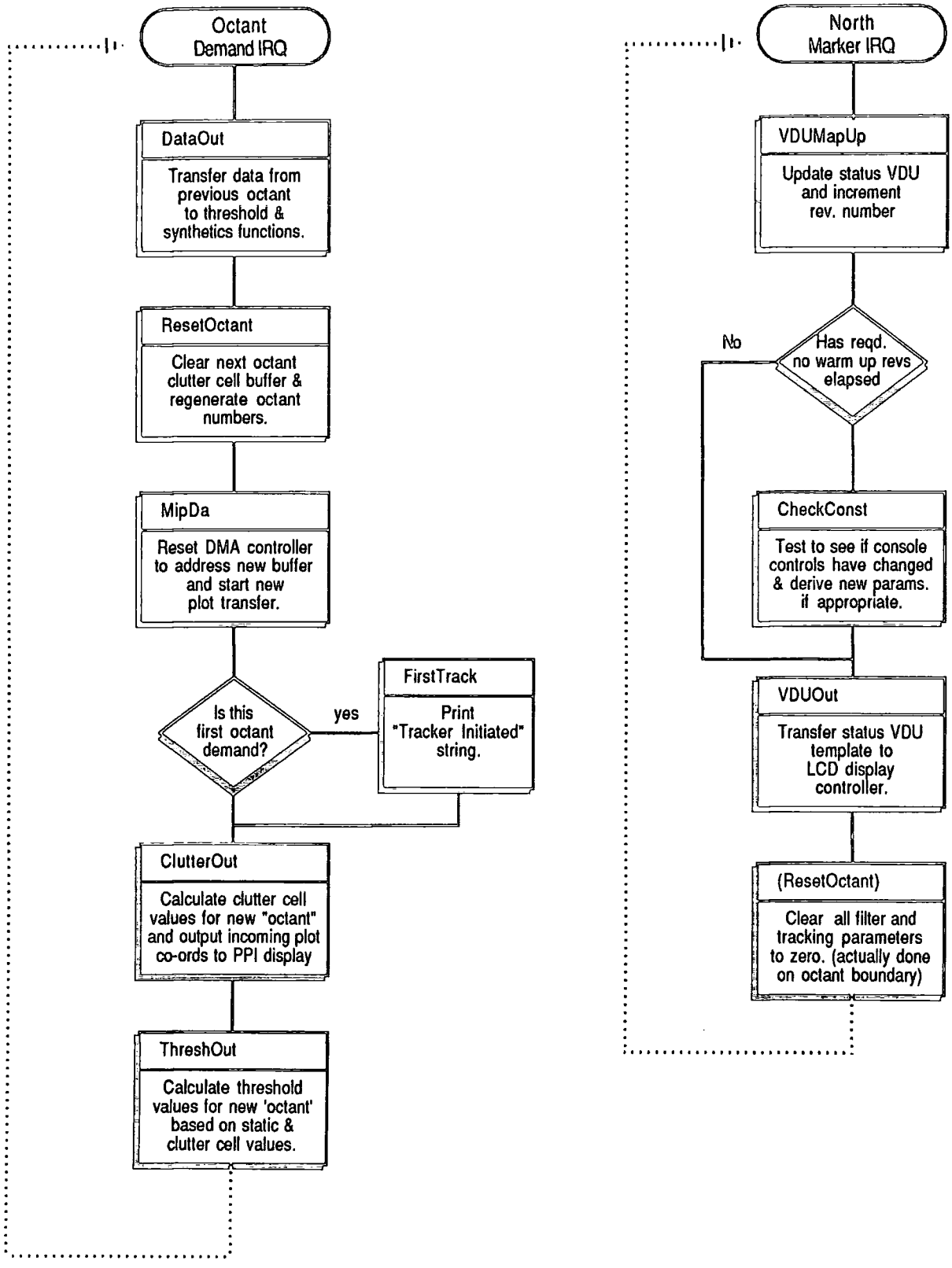


Fig. 6.14 : Octant and North Marker Processing

North marker processing initially calls a subroutine to update the LCD status display

(VDUMapUp), taking values which have been accumulated in the status table and converting them to their ASCII string representations. The strings are copied directly into a RAM-based template which contains all static text for the display. This greatly simplifies the transfer to the VDU controllers, as there is no need to specifically address areas of the display during the update procedure.

The interrupt subroutine then calls another routine which checks to see if any of the console switches, which affect the software operation, has been changed. If necessary the derived system variables are then re-calculated.

The final operation is to transfer the data from the VDU template to the display controllers, and this is accomplished by a simple peripheral data transfer to left and right display memories (640 data words plus configuration information). The north marker processing is thus completed, and the processor freed to continue any tracking calculations.

Approximately 120ms after each true octant boundary, the timing functions generate an octant-demand signal. The first instance of this signal causes a message "Tracker Initiated" to be printed on the console, but after this the octant boundaries are totally transparent to the user.

The first function of this interrupt handler (DataOut) is to transfer the newly calculated threshold values and synthetics characters to the plot extractor hardware, using the blocked- transfer techniques discussed earlier. It should be noted that these values are obtained from the second-previous-octant relative to that which caused the interrupt - the previous octant's data is now considered complete and available for filtering and plot association. In preparation for this all appropriate clutter cell values are reset and (if this is the north-most octant) the revolution status counters are cleared.

Once these operations have been completed it is possible to restart the DMA transfers from the plot forming circuits, and function MipDa does this by setting a new base transfer address and word count, then enabling the DMA channel.

Data is transferred from the plot forming function using a cycle-steal mode, and is therefore transparent to the processor operation. The software is able to make use of the full power of the processor from this point, to carry out all necessary filtering and tracking operations. Subroutine ClutterOut is called to generate the clutter maps from the density of plots in a given area, optionally generating synthetics characters for each of the incoming plots if required. From the clutter maps, function ThreshOut derives the new threshold levels for output to the CFAR processor, according to the settings of the C-Scale and C-Range switches on the operator's console. A constant radius around the radar is set to maximum threshold to mask-out the local echoes, whilst other areas can either be under manual or automatic control. The computer-generated offset involves relatively little processing but helps considerably in maintaining a constant false alarm rate in areas of particularly high clutter.

6.7 Summary and Recommendations

The functions available in version 3.4 of the development software have been chosen to give an indication of the performance of a basic tracking computer, and some indication of its potential as a replacement for the original Ferranti Argus system. Development of the software has not been continued towards the stage of a complete tracking system for a number of reasons, in spite of the success of the software implementation to date.

It must be considered that a successful tracking computer implementation should primarily be cost-effective, but must also take account of maintenance and future enhancement. If the development time for release 3.4 of the Durham system were extrapolated to determine the likely development period for a complete system, it is unlikely that the implementation could be justified. The use of assembly language for system programming also decreases the portability of the solution, and increases the problems of maintenance and enhancement.

In considering the possible options for further development work, it is concluded that the optimum path lies in the production of a *Unix*-like kernel structure, composed of low-level (assembly language) device drivers and a high-level controlling application. The RATES 3.4 software (with the exception of the threshold and clutter generators) forms the basis of a device-driver suite. The filtering & tracking software forms the controlling application, and it is recommended that this be written in a high level language to reduce the development time to an acceptable level.

Since the commencement of development work on the RATES software, the number of langauge compilers for the 68000 series processors has increased, and it would now be most realistic to consider the use of a language such as 'C' for the application level software. The requirement that military software systems make use of the ADA language could also be considered, increasing the flexibility of the 'kernel' approach.

There are other advantages to be gained from the use of a high-level language for application development in this system. The low-level drivers which make up the majority of the RATES 3.4 software modules are essentially I/O bound, with their speed restricted by the access times of the peripherals they communicate with. Conversely, the application software is computationally bound and is therefore affected by directly by the speed of the processor and (in this system) its ability to perform fractional arithmetic. The use of differing precisions for the system variables (see DeriveFix) is an attempt to reduce the time spent in performing more complex calculations, but introduces a number of problems when considering the design of portable software.

The ideal solution is achieved through use of standard variable types (e.g. Float, Integer, Character) and provision of a high speed calculations module to handle the more complex of these types - notably the floating point values. Since the development of RATES 3.4 software, Motorola have released a floating-point co-processor (the MC68881) which provides an extension to the standard MC68000 instruction set allowing manipulation of floating point operands at a speed many times that of similar software routines. The device is fully compatible with the MC68020 processor (which offers faster operation than its predecessors but maintains code compatibility).

It is evident from these discussions that the recommended solution to future software development will take account of the following points:

- Use of low-level assembly language for the 'device-drivers', including the user interface and communications with the plot extractor hardware.

- Development of the filtering and tracking software in a high level language. From studies of commercial software tendancies, it is recommended that the 'C' programming language be used, with ADA as the second choice. This is based on the following observations:

  - 'C' is the main language available for the *Unix* operating system, which is the current MOD & Government -preferred O/S.

  - 'C' is the most rapidly developed language, and new compilers are issued more quickly in the light of hardware developments than other languages (eg PASCAL, CORAL etc). This would be important if the use of the new MC68030 processor and MC68882 co-processors were to be considered.

  - ADA is given secondary preference as the 'MOD-recommended' language for new software projects. In view of its complexity, size and overheads it is not a prime choice for development as many of its features are unnecessary in a system of this type.

- Modern compilers allow different optimisation strategies, and it is probable that fast execution could be available as one option.

- The use of a high level language will greatly simplify enhancements and maintenance due to its increased readability, and its tendancy towards a more modular approach. This will also also promote a proven testing strategy, resulting in a more reliable product.

7.0 Introduction

The RATES development system has been tested using a number of techniques, but there has not been any attempt to provide a statistical analysis of the performance of the complete plot extractor. The test methods have involved visual comparison of results obtained using the prototype and similar data obtained from the development system, and also an analysis of the speed of the new processor and its interface.

This chapter describes the techniques used to generate the test data and gives the results of testing the new hardware and software. The successful operation of the new system is clearly illustrated and some predictions are given for operation of a complete tracking system.

7.1 Testing the Plot Extractor

The objective of the Durham RATES development system was to derive a practical implementation of a plot extractor, which could be shown to remove clutter from a genuine input signal, rather than a simulation based on hypothetical approximations to input signals. With this in mind, testing has been accomplished using data from one of three sources:

1) Real-time, Radar Data
2) Video-recorded Radar Data
3) RATES Test Target Generator (Function Z).

The test generator and video recorder both provide data compatible with a particular set of radar characteristics, so that tests cannot be influenced by choice of antenna and timing parameters. The radar in question is a 2-dimensional surveillance type, intended primarily for marine applications. Its parameters are summarised in Table 7.1. Function Z is capable of simulating the other radar types, however the prominence of the chosen test radar in likely field applications has resulted in much of the testing being based around this unit. Unless otherwise mentioned, all test results have been achieved using timing parameters applicable to this radar.

P.R.F.	Pulse Length	Maximum Range	RATES Range	Beamwidth	Rotation Rate
768Hz	2 μs	96 miles	90 miles	1.2°	15 r.p.m.
Video Signal :	Log @ 20dB/V or Linear, 4.5V max., noise 0.24V rms				
Turning clock :	4096 pulses / rev, 1:1 ratio				
Sync. pulse :	15V ± 3V, 3μs width at start of each p.r.i.				
North marker :	Logic '0' at due north, 1ms duration				

Table 7.1 : Summary of Test Radar Parameters

7.2 Testing Using Recorded Radar Data

Much of the system testing has been accomplished using data from a test radar which was recorded onto a standard video recorder, giving the advantages of a repeatable input signal which accurately reproduced a real environment. The video recorder used for testing is a Sony model EV-310CE reel-to-reel unit, which is used in conjunction with a Thorn-EMI model RR100 radar data recording adapter. The video recorder uses one inch tape and is able to record one video channel and two audio channels on this, giving approximately 70 minutes recording time from a 750m tape. The data adapter takes the 4 outputs from the radar receiver and merges them to enable them to be stored onto the recorder. The turning clock is input either as a 115/90V synchro signal or a 50Ω analogue signal and is combined with the north marker before being recorded on the first audio channel. Video data is merged with the synchronisation signal and these are recorded onto the video channel. The second audio channel is made available for recording ships heading and speed information, but these features have not been used during system testing.

7.2.1 Hardware Interface to the Video Player

On replay, the data is separated but north marker and turning clock signals need re-timing and buffering before they can be applied to the RATES plot extractor. The turning clock emerges from the RR100 as a 2048 pulses/rev signal and this frequency needs to be doubled before it can be used. The RATES prototype used the spare processing capacity of one of the tracking computers to achieve this task, however the development system, with only one processor, was forced to use the hardware converter shown in Figure 7.1 to produce the signal. Additional buffering is supplied because the 50Ω inputs on Function G2 require a 12mA input current, which is beyond the drive capability of the TTL monostables.

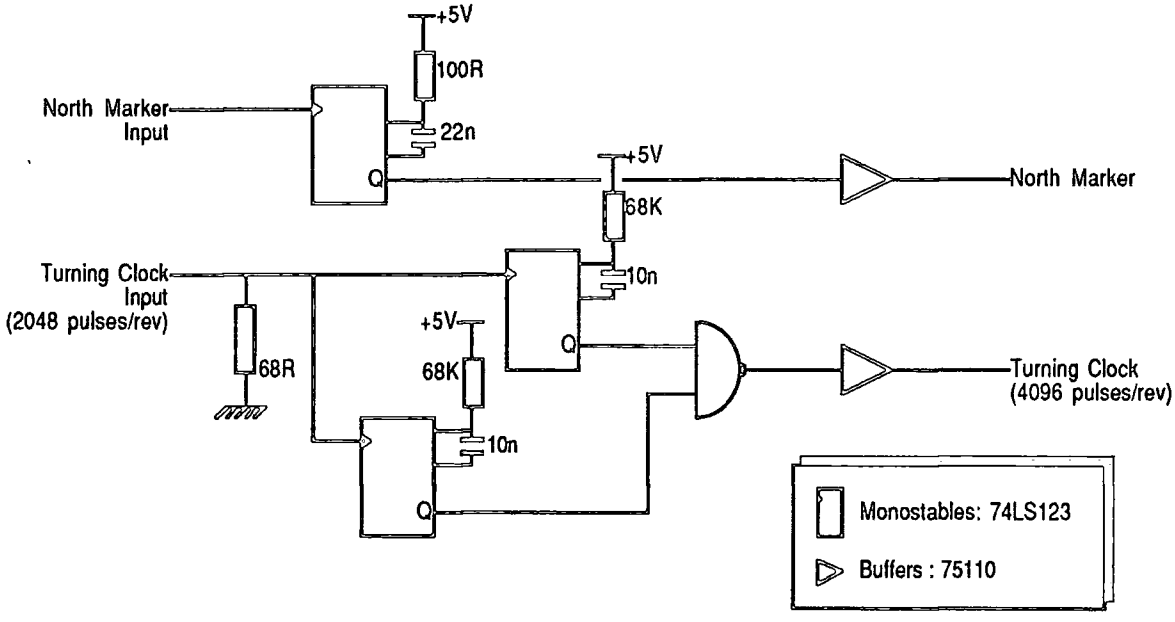


Fig. 7.1 : Radar Recorder Interface Schematic

The north marker signal is retimed in a monostable using a 100Ω resistor / 22nF capacitor combination (approx. 1ms) and then buffered in part of a 75110 line driver. The aerial turning clock is input as a 500Hz 1:1 signal and is presented to two monostables, one positive edge triggered and the other triggering on a negative edge. Both monostables generate a pulse of 0.5ms duration, and when combined in a NAND-gate, this results in the required 1KHz turning clock.

7.2.2 Recorded Radar Scenarios

Two scenarios were available for testing purposes, one from a land-based installation at Portsmouth and the other from a radar based on a ship which was involved in exercises off the coast of N.E. Scotland. Both tapes last for approximately 1 hour, but the quality of recording is such that breaks occur quite frequently during this period. Use of the tape counter has enabled testing to commence at a pre-determined point, so maintaining consistency between input data sets.

The Portsmouth tape features little meteorological clutter, but by nature of its location shows considerable effects from land clutter. The Isle of Wight can be seen clearly just south of the centre of the display, with Southampton Water to the west. Helicopter tracks are present around Portsmouth and to the north-east of the display are tracks from Gatwick airport. This scenario provides an excellent situation for target tracking as aircraft are visible on take-off and landing flight-paths around the airport. The effects of sea clutter are largely masked by the CFAR processor and most detections of shipping are largely removed by the stationary plot filters.

The tape from the ship was used initially for the RATES prototype performance evaluation, and there is already documentation on the results of these tests. Unfortunately, the development system (with version 3.4 software) has been unable to provide significant results from this tape, and data plots have not been included in the set of results.

7.3 The RATES Test Target Generator

The test target generator was designed as one of the functions to be included in the prototype system, although it is capable of acting as a stand-alone unit since all timing signals are generated independently of those on Function G. The test generator provides the user with the following components, which may be merged or rejected as required:

1) Random noise signal
2) Random clutter
3) Beam-correlated clutter
4) Fixed Targets, number appearing can be pre-programmed
5) Moving Targets, 3 available at any one time

As it has not already been covered in Chapter 3, it is proposed to discuss the implementation of Function Z, with attention given to each of the four boards which make up this unit.

7.3.1 Timing and Interface

Board Z1 carries the circuits for controlling the timing parameters of the test target generator, and its schematic is shown below in **Figure 7.2a**. An 8MHz oscillator generates the master frequency from which all other signals are derived, through use of programmable dividers. This is initially divided by 163 in X2, to give a 50KHz reference which is further divided in X3 to give either 256, 768 or 1536 Hz for use as a range synchronising clock. When active, this signal also enables the 3μs monostable (X4) which outputs a **radar sync** pulse to the plot extractor.

The 50KHz signal is also divided down in X5 to produce the azimuth clock and **aerial turning** signals. The latter is buffered in a standard RS232 driver, before transmission to the plot extractor. The azimuth clock is fed into a 12-bit counter to give the antenna azimuth count, which is then incremented by 8 before distribution to the remainder of the test target generator. The carry output of the counter is also buffered in the line driver and transmitted to the extractor as the **north marker** signal.

The range clock is derived from the 8MHz oscillator, divided down in programmable counter, X6. This is then distributed to the other functions which make up the test target generator.

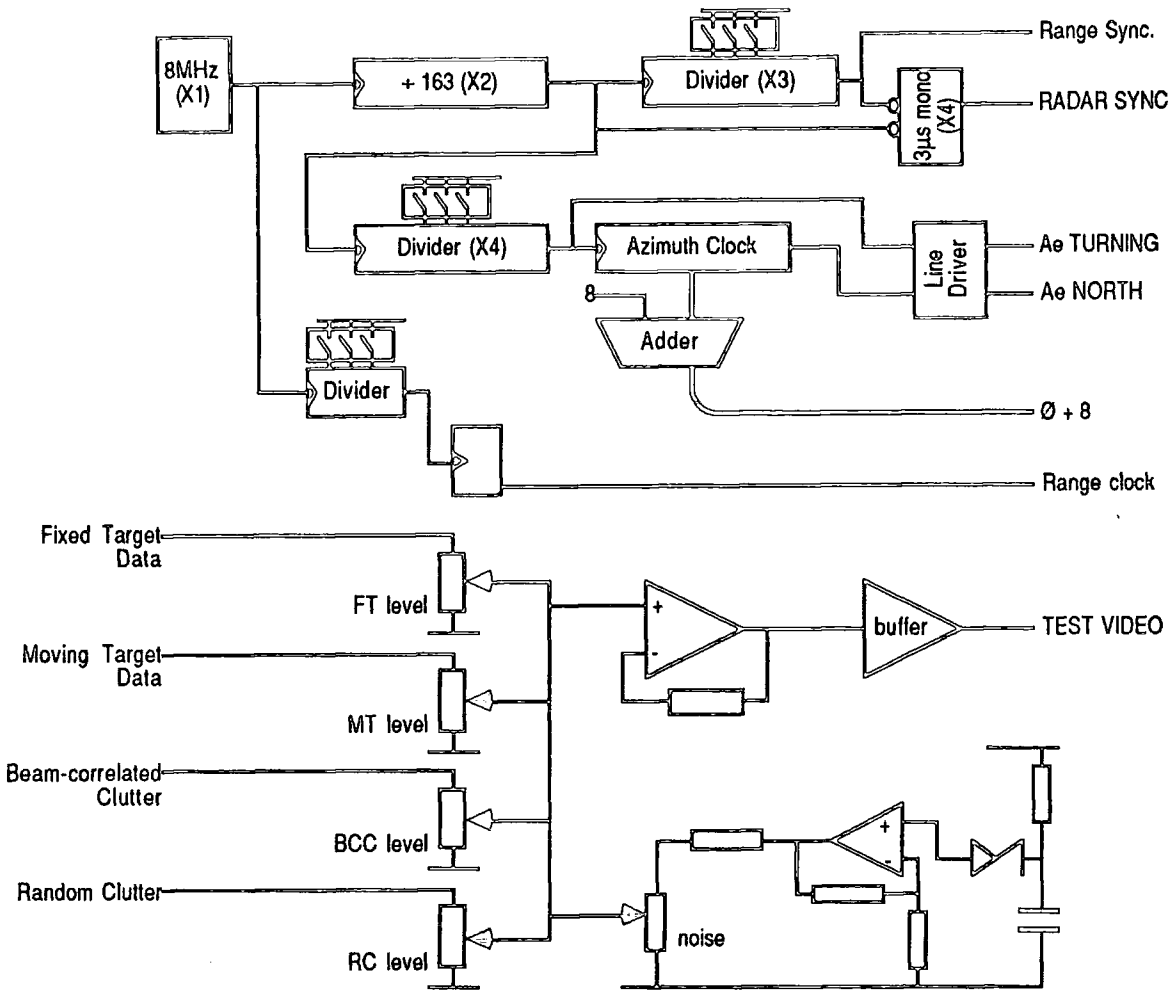


Fig. 7.2a : Test Target Interface & Timing

Function Z1 also contains the mixing circuits for the 5 video inputs: fixed & moving target data (from Z2) and beam-correlated & random clutter (from Z4) are buffered then mixed in a series of preset resistors. Random white noise is generated by a Z5J noise diode, then amplified before joining the mixer. The composite signal is passed through a high-speed operational amplifier then buffered in a line driver before being transmitted to the plot extractor as test video.

7.3.2 Clutter Generation

Random and beam-correlated clutter is generated by function Z4, and the schematic of this circuit is shown in Figure 7.2b. The range clock from Z1 is used to advance a 22-bit shift register, which together with a feedback network forms a random bit generator. The output of this is sampled at a rate which can be programmed between (range clock / 2) and (range clock / 256) to provide a variable random clutter density. When an active sample is detected, it is latched in a second register which is clocked at half the range clock frequency,

and the output from this is taken to function Z1. Any clutter will therefore last for a duration of two range cells.

The output from the random bit generator is divided by 16 and then again by between 4 and 256 to provide a clock for a second random generator. The output from this is sampled in a similar manner to the random clutter generator, to provide clutter hits of two range cells duration, and then presented to the beam-correlated clutter section.

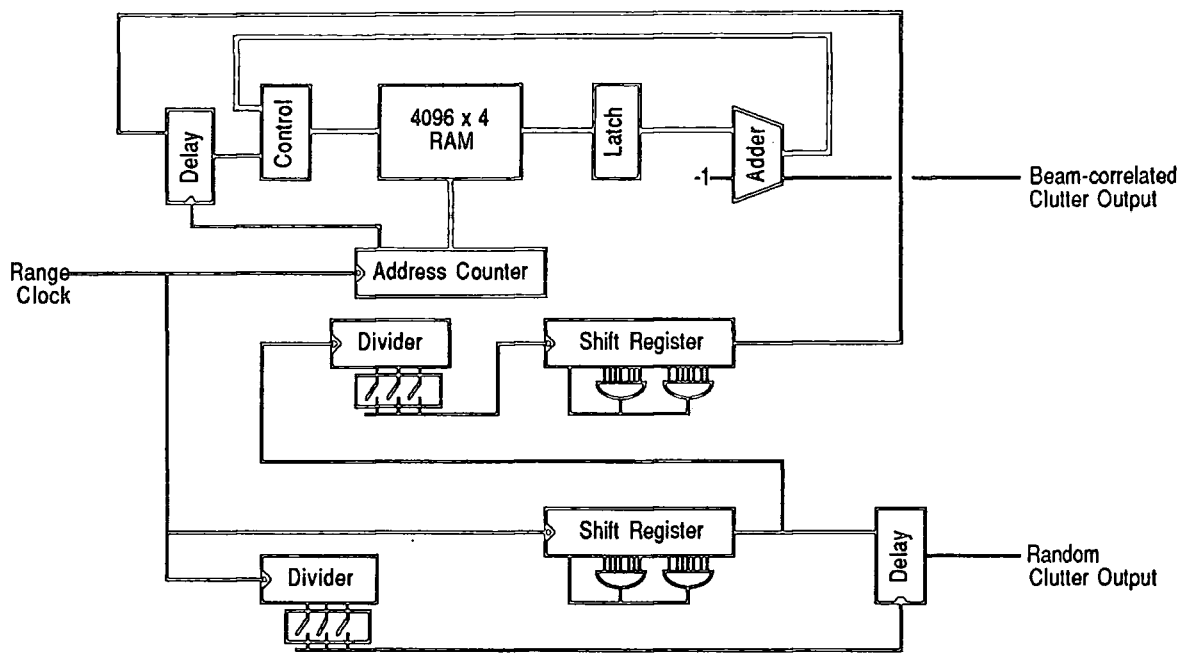


Fig. 7.2b : Test Clutter Generator

An important feature of this circuit is a 4096 x 4-bit RAM, which is addressed by a counter clocked by the range clock. Each nibble therefore corresponds to a unique range cell, between zero and the maximum range count for a sweep, and on every range count the corresponding data is read-out, latched and then re-input. Each nibble starts with a value of zero, and when an output from the random bit generator is detected, this forces a decimal 15 to be loaded into the RAM at that range location. On subsequent bearings this count value is re-addressed, latched then decremented and re-input. Clutter is declared for as long as the value remains positive, but as soon as a negative value is reached, this forces a zero to be reloaded and the clutter region to cease. An random area of beam-correlated clutter is therefore described, having a depth of 2 range cells and width of 15 pulse-repetition intervals.

7.3.3 Fixed and Moving Target Generators

The fixed and moving target generators are realised by functions Z2 and Z3, and their combined schematic is shown below in Figure 7.2c. The fixed target generator produces hits at regular range and azimuth intervals, giving rise to a spokewheel display of targets. Range ordered hits are produced by dividing the range clock by between 2 and 4096, in powers of two, to generate a trigger for the range co-ordinates of the target. Similarly, the azimuth clock is divided to allow targets to be generated from once every 32 turning-clock pulses to once per revolution. In a manner similar to the clutter generator, the range pulse is stretched to cover two range cells and the azimuth pulse is used to load a 4-bit counter which proceeds to count for the next 15 bearing clock pulses before being halted. The range and azimuth data is gated to detect coincidence and the output is a set of fixed targets, 2 cells deep and 16 cells wide.

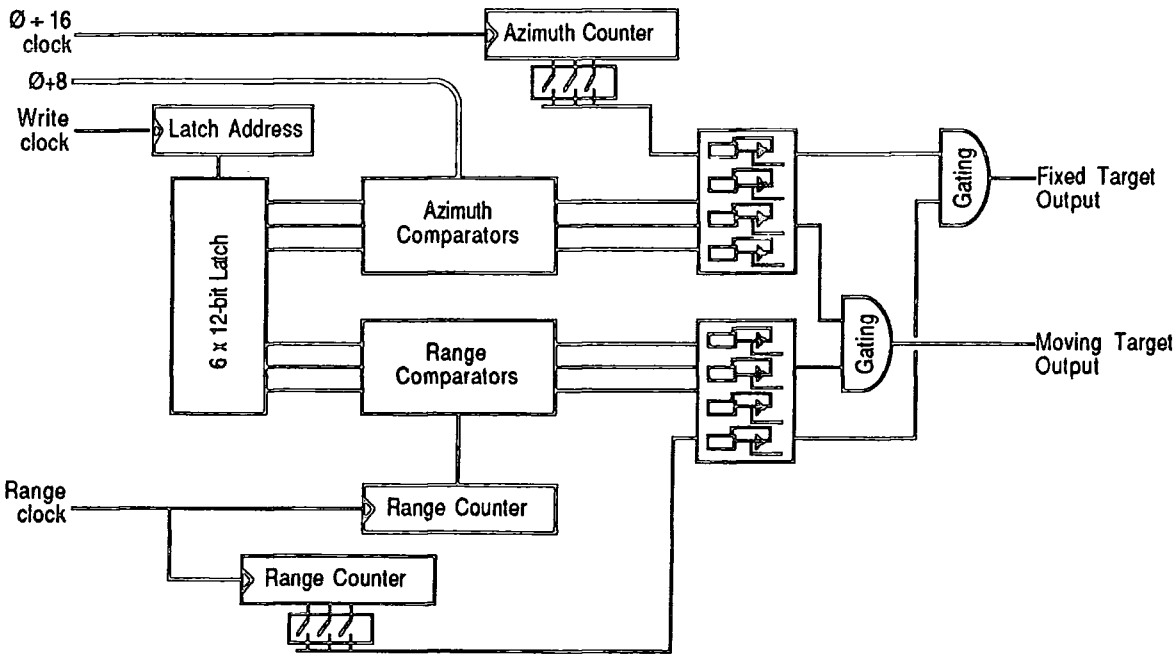


Fig. 7.2c : Fixed and Moving Target Generators

Up to 3 moving targets can be displayed from the test target generator, but the responsibility for calculating their positions rests with a separate processor. The hardware latches range and bearing co-ordinates as they are generated by the computer, and then tests for coincidence with the currently scanned range and azimuth. Once triggered, the range depth is stretched to 2 cells and, like the fixed target generator, a 4-bit counter is loaded to extend the target width to 16 cells. The range and azimuth declarations are then merged to provide a single target plot.

In the RATES prototype, the test targets are generated by the tracking computers, however this has the disadvantage of increasing the load on the tracking software and makes the test target generator dependent on the plot extractor hardware. With no tracking computers in the early development system, a requirement arose to generate a full range of test targets using function Z, and it was therefore necessary to derive an interface and software to enable targets to be generated on a general purpose microcomputer.

7.3.4 Target Generation Software

The software for the moving target generator is designed to run on a Cifer model 2684 microcomputer, using the CP/M-80 operating system. The machine is equipped with an 8-bit parallel output port, which is ideal for transferring data to the test target generator since no feedback is required. The strobe signals associated with the port's handshaking are used to clock the latch address counter to point to the next byte store.

RATES 992 TEST TARGET GENERATOR

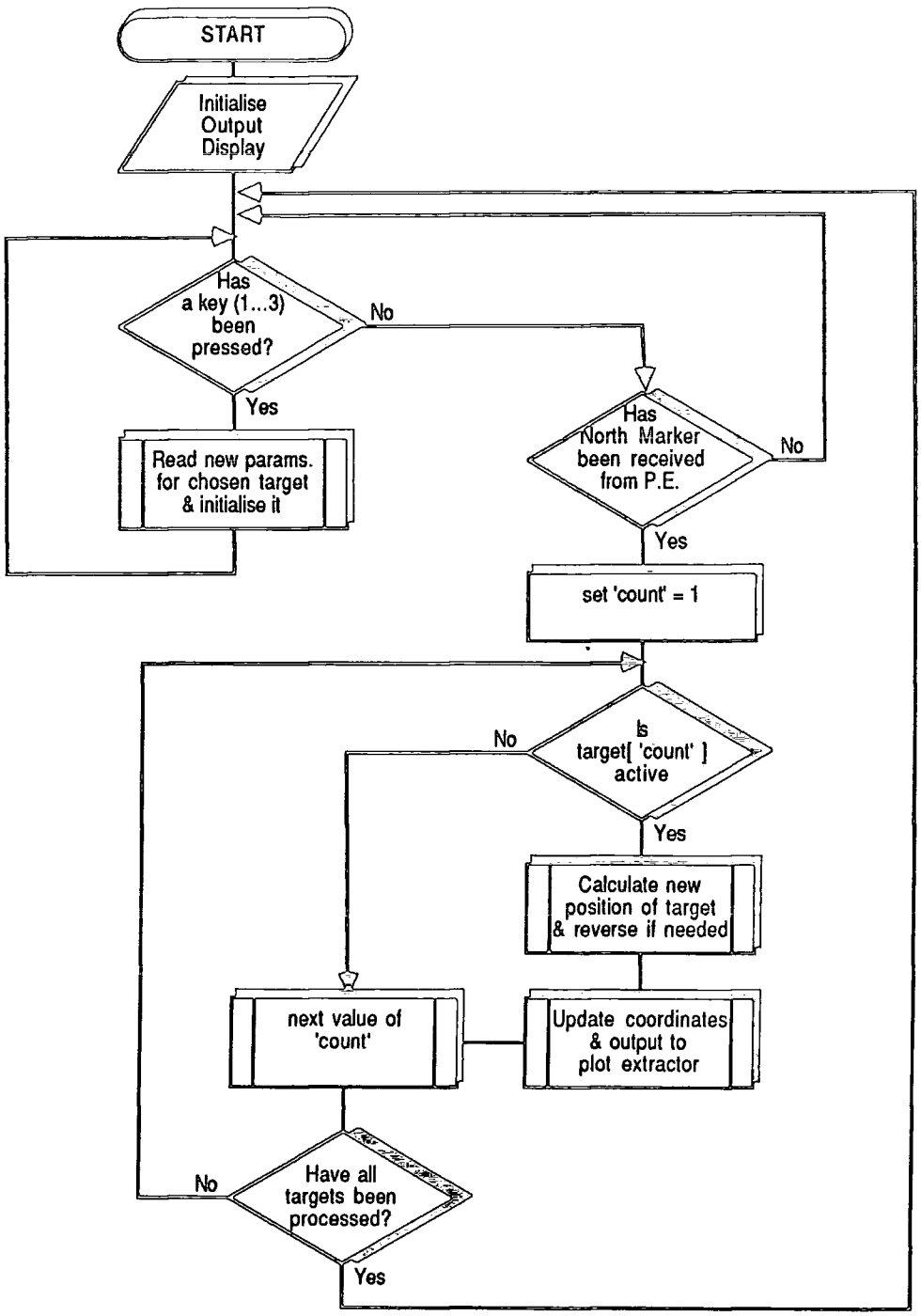
NUMBER	RANGE	BEARING	SPEED	HEADING
TT1	_			
TT2	120	46	300	22
TT3				

Fig. 7.3 : Moving Target Generator Screen Display

The program is intended to be simple to use, and presents the operator with a clear display, making use of the Cifer's enhanced display facilities (**Figure 7.3**). To activate or change a test target, the operator keys the number (1...3) of the desired target and is then prompted to select, in sequence, values for initial range, bearing, speed and heading. A range of zero cancels generation of that target, any other invalid inputs are rejected. The program can be terminated by keying an escape in place of a target identity code. The program is written in ADA, and a full listing is supplied in **Appendix D**, however it is proposed to discuss the basic algorithm in the section which follows.

7.3.5 Target Generation Software Algorithms

The complete operation of the test target generator software can best be described through use of a flowchart, as shown below. For clarity, the visual-display routines are not included: their operation simply achieves the type of presentation shown in Figure 7.3.



Following initialisation of the screen, subroutine "SET_CONSTANTS" is called to initialise the trigonometric functions look-up table, which holds values of sin(Ø) between in the range -1...+1. Program control is then transferred into an infinite loop, where the first

action is to test for keyboard input ("GET_TARGET_NUMBER"), using an operating system direct input request. The 'Esc' key causes the program to terminate, while any input in the range '1'...'3' transfers control to "READ_DATA" and then "SET_TARGET". Other inputs are ignored.

"READ_DATA" sequentially reads integer values for each of the target's initial parameters: range, bearing, speed and heading. If a range of zero is chosen, generation of that target ceases and no further parameters are requested.

"SET_TARGET" is then called to calculate the new target's component vectors in the directions along and across the chosen line of azimuth. Range is converted to plot extractor range-units, bearing to angular units and speed to 'range-units travelled in one scan'. Knowing the difference (β) between target heading and initial bearing, the along and cross vectors are easily derived from sin(β) and cos(β) (Figure 7.4).

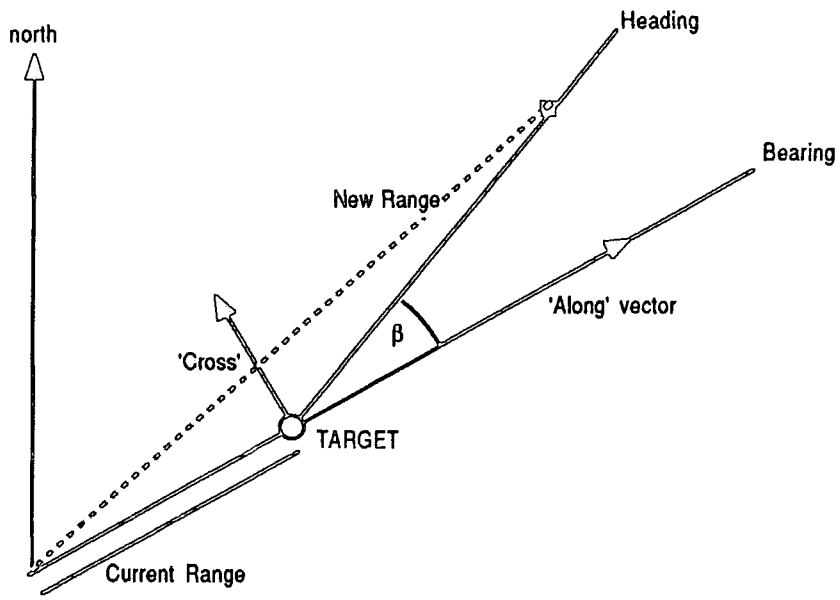


Fig. 7.4 : Calculation of Test Target Co-ordinates

Once the 'North Marker' signal from the plot extractor has been detected (by monitoring the output port handshaking lines), the routine "OUTPUT_TARGETS" is called to update each active target's position. The new range is given by Pythagoras' theorem, with the 'cross' vector forming one side of the triangle and the sum of current range and 'along' vector forming the other. Through use of a polynomial approximation, a value is also obtained for the change in bearing and hence the new azimuth value. Data is written to the plot extractor as 12 6-bit words, range first followed by bearing, for each of the three targets.

7.4 Test Scenarios and Results

Two sets of results have been derived as a result of testing the Durham RATES hardware and version 3.4 software. The main plot extractor hardware is functionally similar to that of the prototype, despite the considerable reduction in the number of boards which make up the complete system. This has been achieved through careful redesign of the interface to the tracking computer and subsequent modifications to the plot forming hardware. The tracking computer has undergone the most radical change within the system and no longer occupies a 19" cabinet adjoining the rest of the hardware. The ability to install the processor board within the main card rack has led to the success of the interface and permitted the introduction of faster DMA techniques. This, in turn, has resulted in a higher performance than could be achieved using the original Ferranti Argus and ME62 interface.

7.4.1 Study of Processor Utilisation

The main test results have been compiled from studies of the processor, whilst running the version 3.4 software. Unlike the prototype, the new design does not take up valuable processing time during DMA transfers from the plot forming buffer (E4) to the computer memory. The prototype system performed transfers in burst mode, during which time it copied a block of up to 1024 words into processor RAM. The development system makes use of the MC68450's cycle-steal mode of operation to transfer words during periods when the bus is inactive, and hence the transfers are totally transparent to both host processor and speed of operation.

To maintain system operation within reasonable levels, the maximum number of plots which can be transferred during a single octant has been both programmed and hardwired (on K1) to 1024. This allows a total of 2048 words to be transferred during this period - twice the capacity of the prototype system and with no possibility of program 'crash' due to the end-of-octant recovery techniques mentioned in the previous chapter. The new hardware does not permit inter-octant transfers, however the overload condition is less likely to occur due to the higher normal plot limit. There were indications that the prototype system could not successfully handle these transfers without substantial loss of data.

Provisional test results indicate that the limit on the number of plot transfers could be raised to accommodate larger numbers of input plots during each octant, although discussions with potential buyers of such a system have indicated that an upper level of around 8000 input plots (including stationary targets and angels) is considered acceptable. Without implementing a complete system, it would be difficult to establish more accurate figures for the performance of the new design, although certain enhancements could be introduced to further improve the speed and capacity of the tracking computers.

7.4.2 Comparison of Processor Types

To establish a suitable basis for evaluating the performance of the tracking computers, the RATES 3.4 software should be considered as a set of functions. The user interface functions are independent of the plot input, filtering and display routines and these are not included in the calculations. The approximate execution time for each function has been calculated, based on the assumption of similar source code regardless of processor type.

For the Ferranti Argus, processor times are based around those given in Table 6.1. The times used for the Motorola 68000 and Intel 8086 calculations are based on clock rates of 8MHz and 6MHz respectively. It has been assumed that 68000 and 8086 systems do not make use of hardware co-processors to perform arithmetic functions.

A study of the RATES software will indicate that the main processing is a function of the **Octant Demand** interrupt. In the verion 3.4 software, only one main function is dependent upon the number of input plots, and hence the relationship between total execution time and number of plots is linear. The processor loading factor is expressed as a percentage based on the total processing time against the total available processing time. The statistics do not take account of bus access times, which are likely to be similar for all processors. For the Argus, the time to transfer the data block from the plot forming hardware into the computer is not included either.

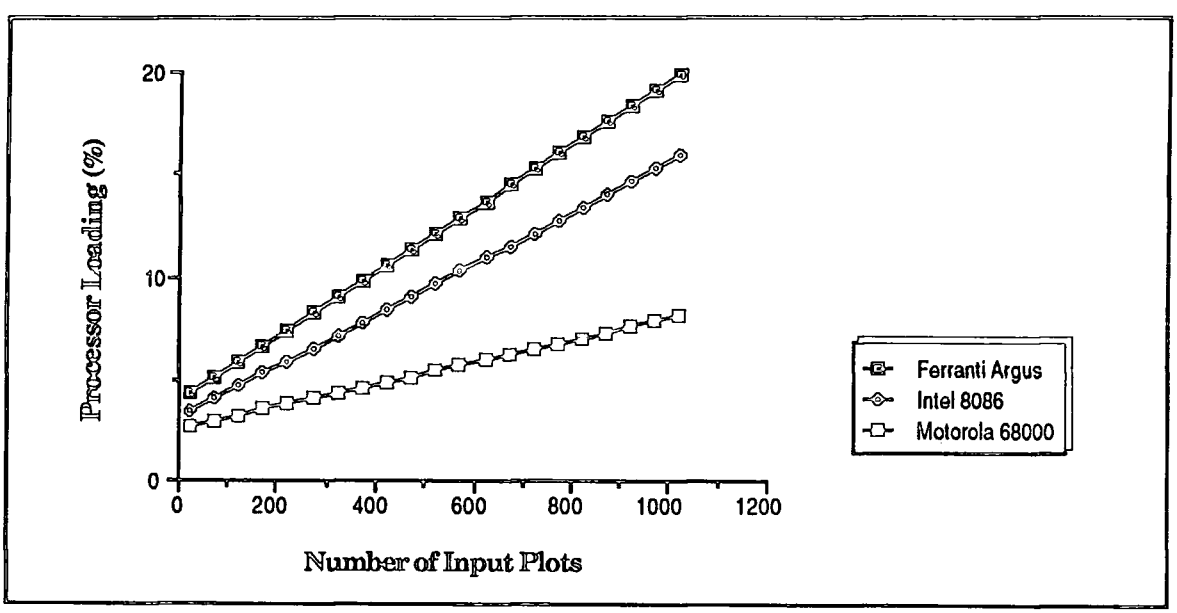


Fig. 7.5 : Loading Factor vs Input Plots for Different Processors

Figure 7.5 gives the results of processor loading against number of input plots for each of three processor types. These preliminary results indicate the success of the Motorola 68000 approach. Not only is the zero-input plot processor overhead significantly less than that of the

other processors, but the rate of increase in processing time is substantially less than its competitors. The functions called in this software release make use of all major processor operations and it is therefore considered that the use of the MC68000 could lead to a significant improvement in performance over the Argus.

It is particularly important to note from these results that the input plot limit imposed on the prototype system (512 input plots) still results in a processing time well in excess of that required by the 68000 to process 1024 input plots. This gives further weight to the suggestion that the maximum input plot threshold could be raised, although this could result in a more substantial increase in time spent in target tracking. Tracking procedures are unlikely to involve a simple relationship to the number of input plots, as a small increase in input plots may require a large number of association tests to be performed.

The relative performance of the three processor types is summarised in **Figure 7.6**, which compares the average times for each to process data in a single octant.

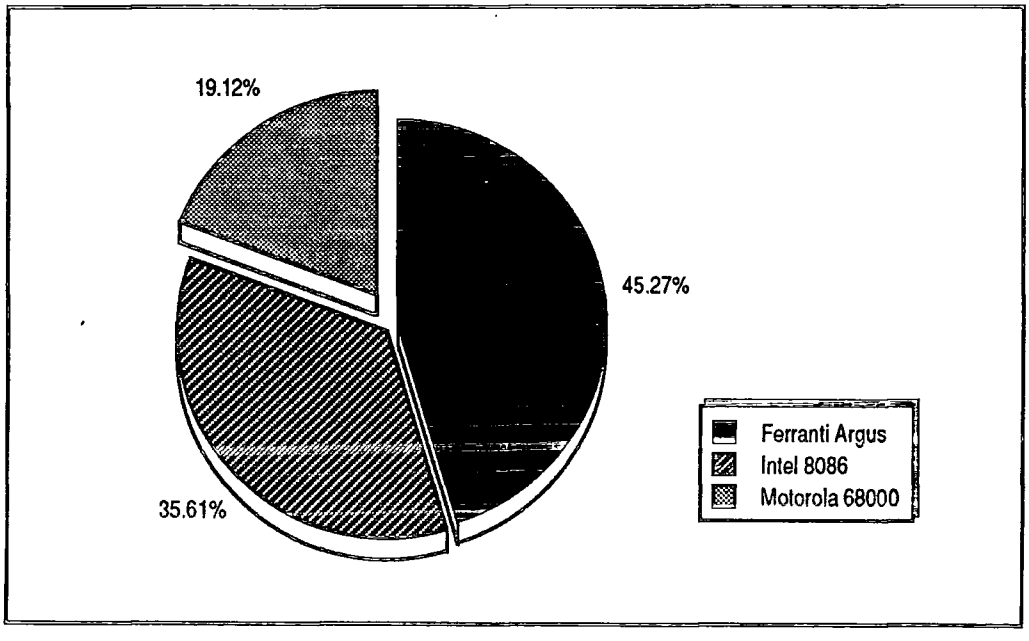


Fig. 7.6 : Comparison of Mean Processing Times

From a comparison of processor execution times for typical instructions, the performance graphs shown above are somewhat predictable. The Argus however is noticibly faster than the other two processors when performing multiplication or division operations. In a computationally intensive program, featuring a large number of these operations, it is possible that the performance of the 68000 will be degraded to below that of the Argus and for this reason the redesign of the computer hardware, using Motorola 68010 or 68020 parts is strongly recommended; (this observation is partially confirmed by the results in Fig. 7.8).

7.4.3 Overall Performance of the MC68000

Figures quoted in the previous section have assumed the use of the Type 'A' radar, which has average processing requirements. This has a rotation rate of 15 rpm, and therefore completes one octant every 0.5 secs. From Table 2.1, the rotation rates of the different types of radar give rise to a range of loading factors for the processor, which are summarised in Figure 7.7.

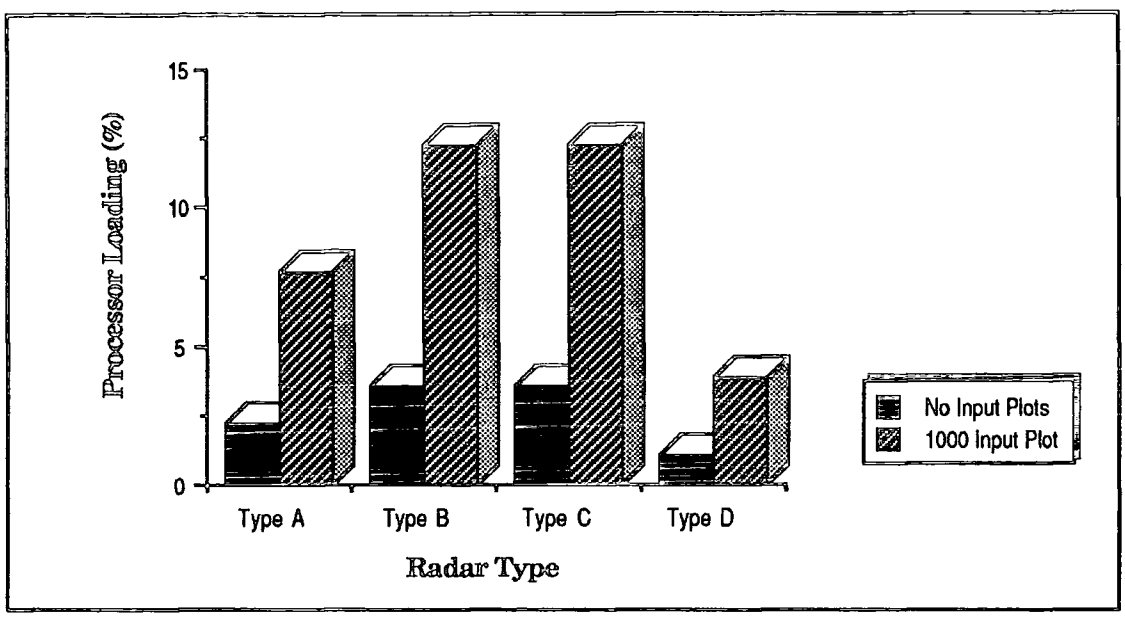


Fig. 7.7 : Processor Loading for Different Radar Types

It is still interesting to note that the worst-case loading, which results from 1000 input plots and a 24 rpm radar, is still lower than the processor loading for an Argus with only 512 input plots.

The final confirmation of the performance increase that could be achieved through use of the new design of tracking computer and interface is given below in **Figure 7.8**. This illustrates a predicted performance for each of three processor types, derived from a study of the execution times of one of the existing computationally-demanding functions (**DerivePar**).

This estimation is based on the assumption of a certain minimum number of complex operations on each input plot. While not based on genuine execution times, it is hoped that this will give some indication of more realistic performance figures for the Motorola processor.

The graph illustrates the improvements that can be achieved through use of this processor, even with more computationally intensive tasks. The threshold for processor loading occurs around the 750-input plots per octant mark (for the Ferranti Argus) whilst the

68000 does not reach the threshold until after the limit of 1000 input plots. It is suggested that use of a suitable co-processor could further improve these results, as many of the present processing bottlenecks are due to difficulties in handling a mixture of binary fractional arithmetic formats. The present system employs both 16-bit and 10-bit fractions when dealing with 16-bit word lengths and some care is needed to maintain the correct scaling of the variables. A more adaptable system would make use of a standard format (eg single precision floating point) and then use hardware to alleviate the extra processing requirements.

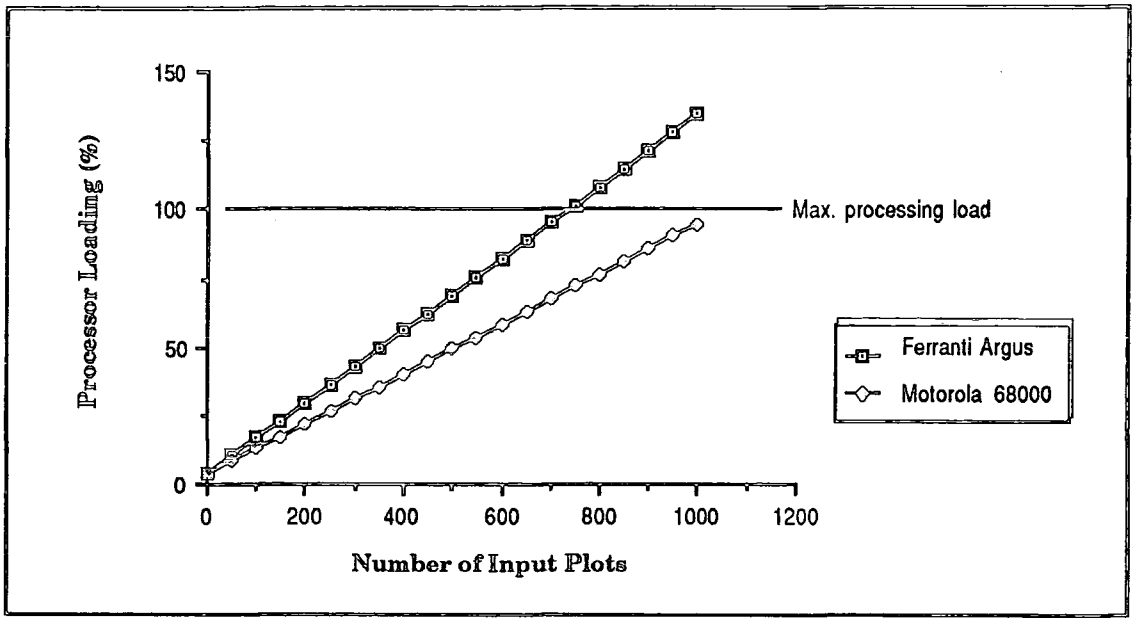


Fig. 7.8 : Overall Performance for Heavy Processor Loading

The results discussed in previous examples can be confirmed by monitoring the processor loading factor which is displayed on the status VDU. This display also shows the number of input plots for a given octant, and can therefore be used to produce practical confirmation of the graphical results illustrated in this section.

7.5 Interface Performance

The performance of the new interface is difficult to assess quantitatively due to the random nature of the input data which may be applied to it. For the purposes of testing its qualitative performance, two sources have been used to generate the input to the plot extractor hardware. Plots are formed in function E, then transferred across the DMA interface to the processor memory. The co-ordinates are initially used to calculate PPI display data and clutter-cell updates, then are output across the second serial interface to a minicomputer.

The data is received as <bearing> <range> pairs, with bearing being represented by a number in the range 0 ... 32768, and range as a number between 0 and (ideally) 4096. In practice, the range often exceeds 4096, as this figure is the limit of RATES range, rather than display range which is some 8% more. From this point the data can be processed as required, without the requirement of programming in assembler.

The test target generator (discussed earlier) is used to provide an output of 2 fixed targets per octant, each on the true octant boundary. The signals are mixed with a high level of random clutter and a lower level of beam-correlated clutter, most of which is filtered-out by the CFAR and WBI processors. The resulting plots are illustrated in **Figure 7.8.**

By applying a small amount of filtering to the data, it is possible to remove all out-of-range plots (i.e. range > RATES range). If a confidence count is applied to each incoming plot and attempts are made to associate it with existing plots, those which exhibit a high confidence count after a certain period may be deemed stationary. Removing all plots which do not reach the required count leaves a selection of 'stationary-target' co-ordinates, as illustrated in **Figure 7.9.**

The Portsmouth recorded scenario is also used as an input to the system, illustrating the interface performance with more realistic data. The tape section is too short to show the existence of tracks, but successfully illustrates the effects of beam-correlated clutter, interference and ground effects. The plot data is shown in **Figure 7.10.**

When the stationary-target filtering process is applied to the data, the results are not so well defined as the previous example. **Figure 7.11** shows the stationary plots remaining after a confidence count of 2 is applied. The ground effects over the period are too random to convey a definite pattern (e.g. The Isle of Wight) and the only significant features which emerge are the interference due to another radar and a minor helicopter track (005°). An alternative form of processing would be to filter all stationary plots, leaving only targets with a mid-range consitency count (i.e. 'moving targets'). The results from this are poor and no significant features can be identified.

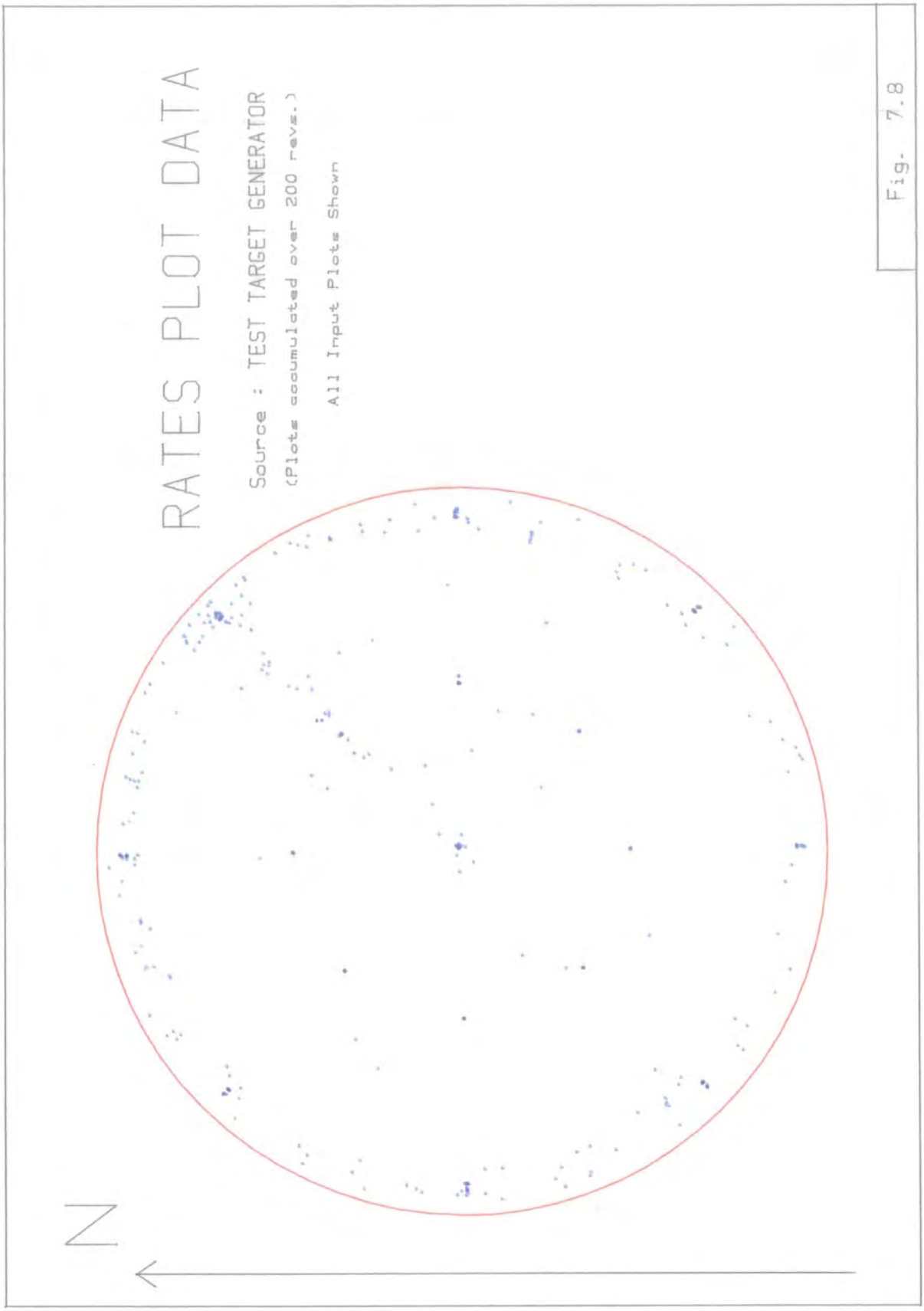

RATES PLOT DATA
Source : TEST TARGET GENERATOR
(Plots accumulated over 200 revs.)
All Input Plots Shown
N

Fig. 7.8

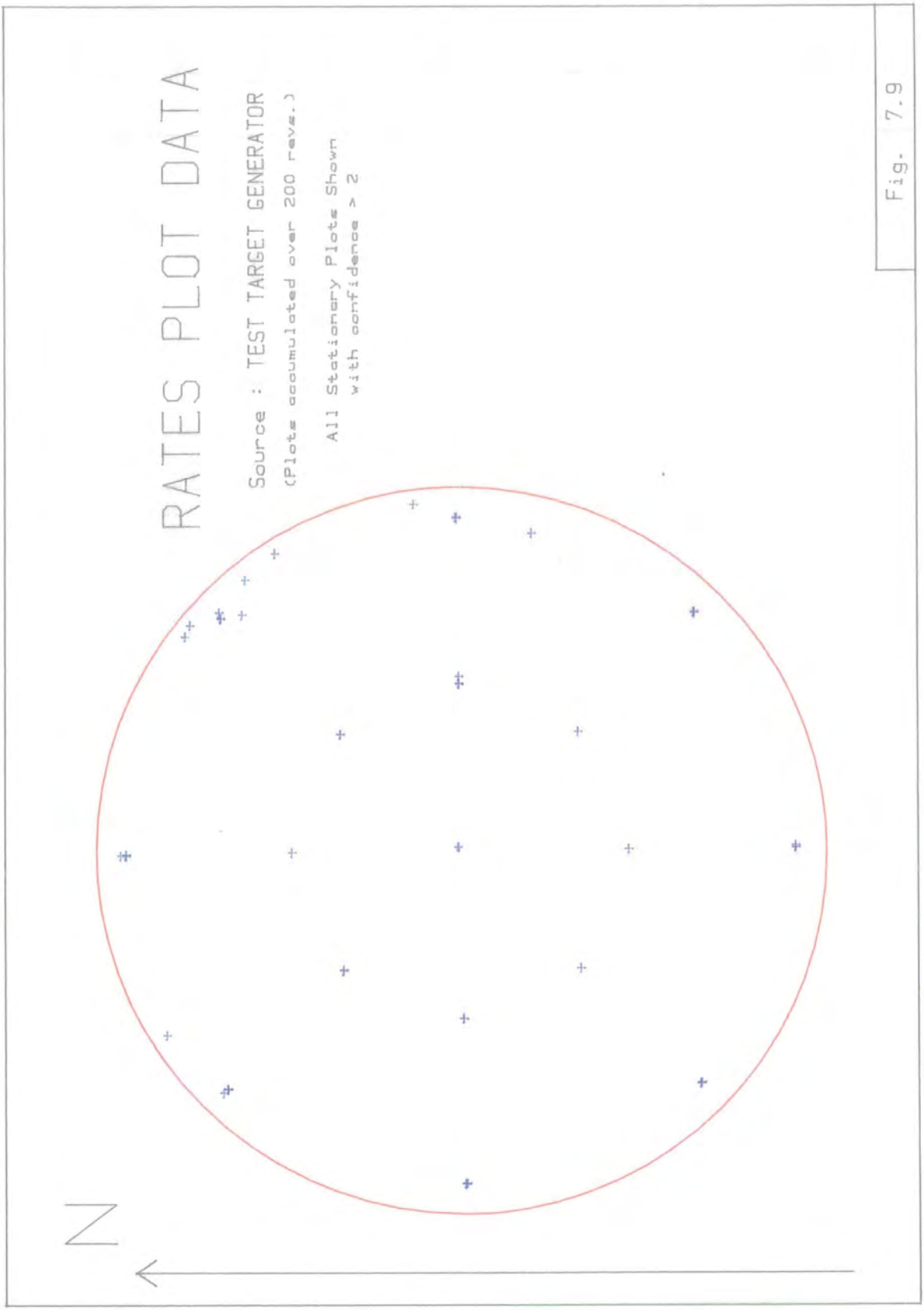
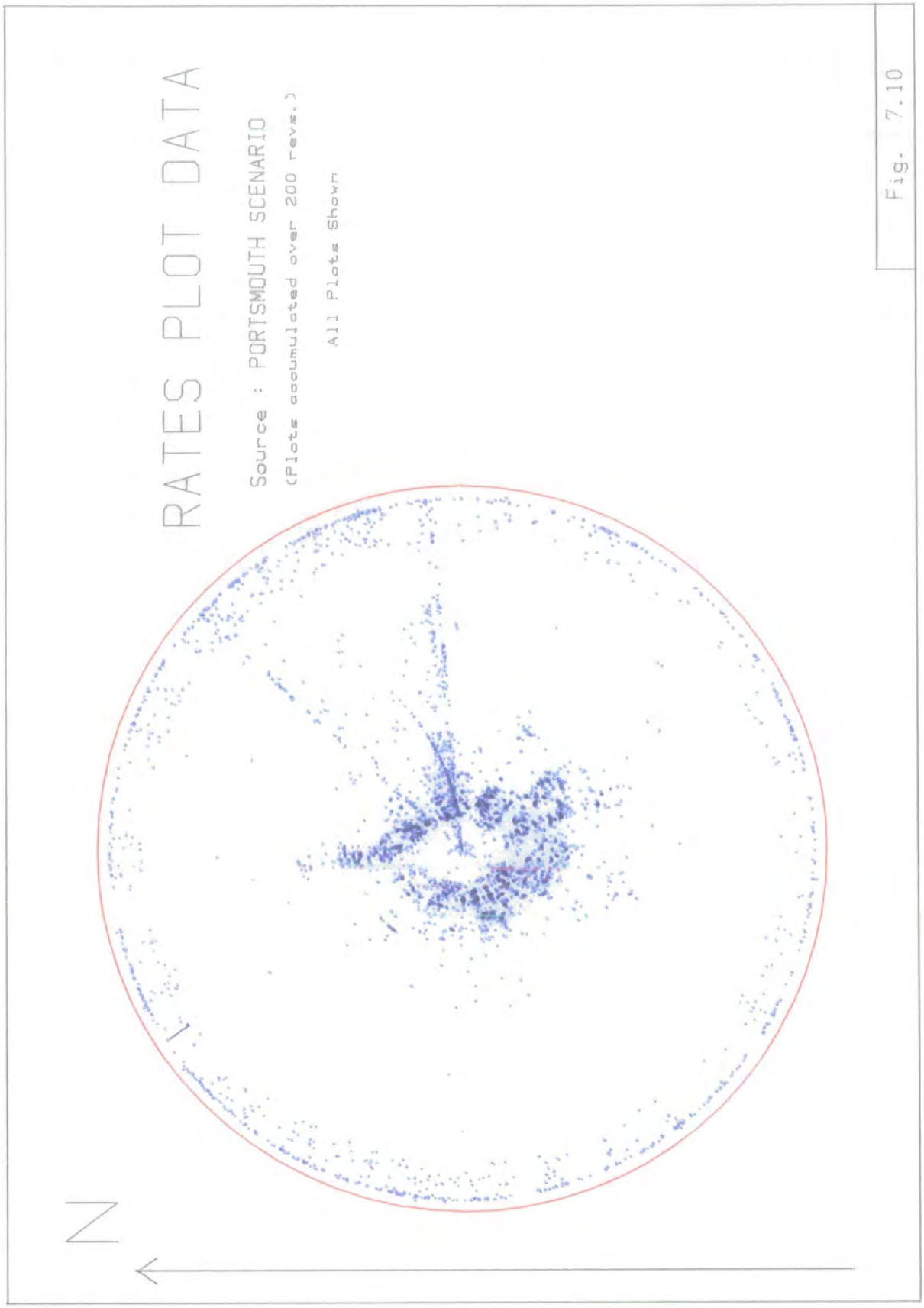
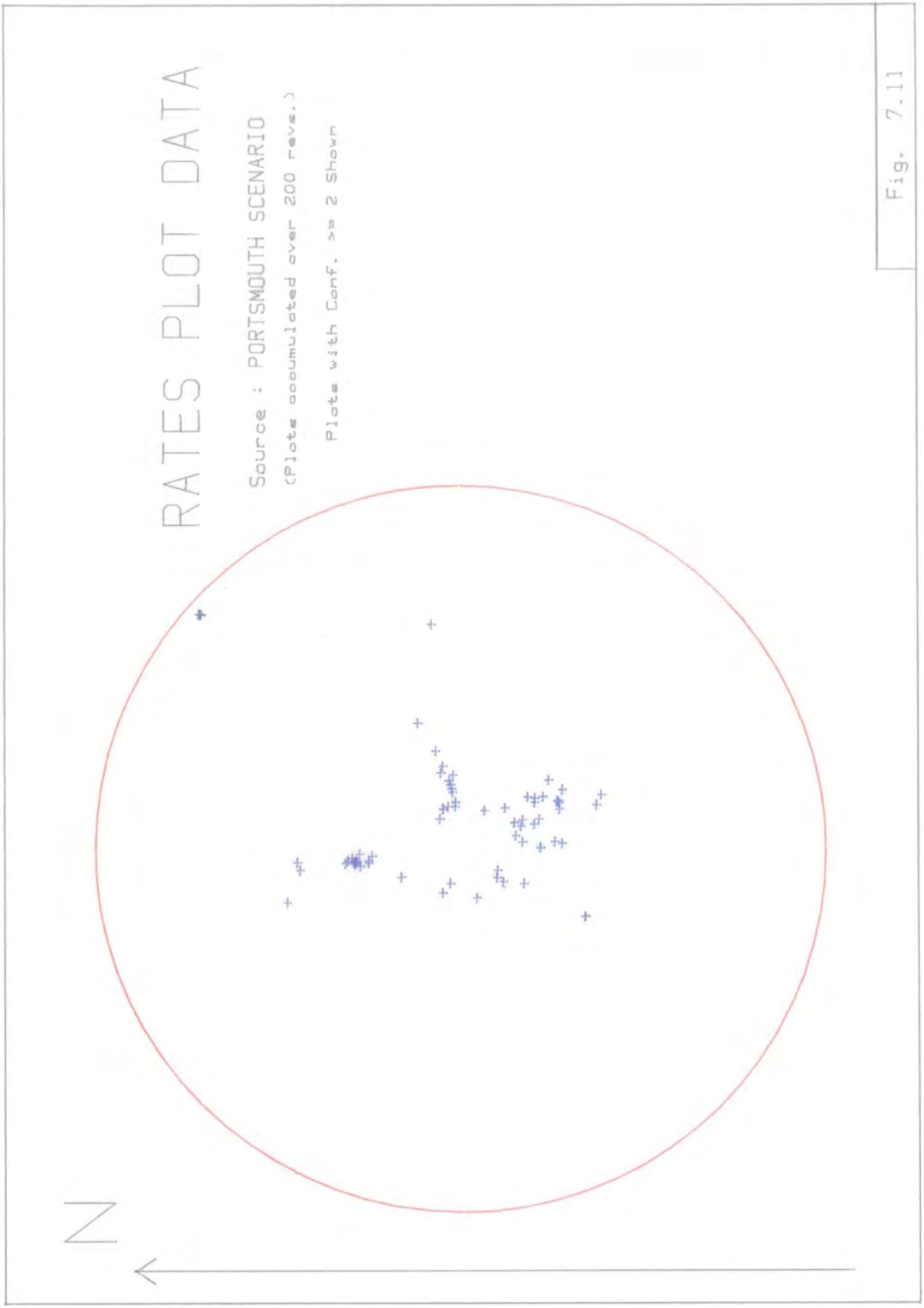

RATES PLOT DATA
Source : TEST TARGET GENERATOR
(Plots accumulated over 200 revs.)
All Stationary Plots Shown
with confidence > 2
N

Fig. 7.9


RATES PLOT DATA
Source : PORTSMOUTH SCENARIO
(Plots accumulated over 200 revs.)
All Plots Shown
Fig. 7.10
N


RATES PLOT DATA
Source : PORTSMOUTH SCENARIO
(Plots accumulated over 200 revs.)
Plots with Conf. >= 2 Shown
Fig. 7.11
N

8.0 Introduction

The arguments and principles discussed so far have been based around the use of conventional serial architectures for radar signal processing. Whilst this traditional approach has been shown to work satisfactorily, the design remains modular and would be difficult to implement in a single-chip form. There is clearly scope for a more efficient architecture to be employed and in particular one which will lend itself to a VLSI implemenation. In this respect, Systolic Arrays can be shown to function well when performing many common filtering tasks and these are therefore recommended for further research in the area of radar signal processing. The topics discussed in this chapter have been summarised in a paper presented at the I.E.E.E Radar '86 Conference [49].

8.1 The Need for a Different Architecture

One of the earliest aims of research into the RATES design was to study the possibility of implementing single-chip or highly integrated solution. In earlier chapters, a number of techniques have been discussed which could lead to the development of a VLSI-based plot extractor:

- **Use of commercial VLSI circuits** : has the advantage of low development costs but would not necessarily result in a compact solution. The design would require hardware modifications to realise future enhancements.

- **Use of custom VLSI** : Development costs are high and the final product would not be particularly flexible. Enhancements may require redesign of the complete chip.

- **Digital Signal Processors** : Potentially low development costs and considerable flexibility with regard to future enhancements. Unlikely to be a suitable architecture for a VLSI single-chip approach due to the complexity of each processor, and would therefore have to remain as a multiple-module design.

- **Microprocessors** : Extremely flexible but (within the next few years) unlikely to reach a sufficiently fast operating speed to handle, for example, one detection every 500ns.

- Bit-Slice Processors : Have the flexibility of microprocessors and DSPs due to their programmable design, combined with the speed of most commercial VSLI custom circuits. The limited logical function of typical bit-slice parts means that a highly efficient hardware architecture can be adopted without loss of adaptability to future enhancements. The disadvantage of bit-slice processors is the complexity involved in programming, as all possible gates and inputs have to be programmed.

The bit-slice processor is an ideal hardware standard for VLSI implementation, as the functional blocks used in each slice may be easily replicated and connected together. To make them viable from a development viewpoint however, it is essential to reduce their programming needs. One technique would be to increase the number of fixed hardware links between functions, thereby removing the need to programme the links. A better approach would be to design and programme the architecture to incorporate a number of units which functioned in an identical manner but on different data inputs. The units could perform one or more operations on each data stream before outputting them to other stages. The instructions, however, would be supplied from a common source and fed simultaneously to each unit.

This technique would satisfy the requirements for a reduction in the programme development time and would simplify the hardware design as a unit could be developed as an individual function and then replicated as needed. The operation of such a machine may be termed a "**Single-instruction stream, multiple data stream**" (SIMD) processor and from existing studies [59] can be shown to be based around a **Systolic Array** architecture.

In the discussions which follow, it is intended to show how the concept of systolic arrays can be applied to radar signal processing, with particular reference to the functions implemented in the RATES design. Although the systolic array has been viewed as a mainly theoretical approach, a number of more recent developments have meant that systolic arrays can now be realised in hardware and hence the importance of the systolic array should not be ignored.

8.2 Principles of Systolic Arrays

The emergence of the systolic array concept has been largely due to the pioneering work of H.T. Kung [56], [57] at Carnegie-Mellon University and is based around neural computing techniques, where a very large number of simple cellular logic units interact with their neighbours.

Kung's proposals have arisen from studies into the design of low-cost, high-speed architectures for computationally intensive tasks. Recent years have seen a dramatic reduction in the cost of implementing VLSI systems and a corresponding increase in the number of semiconductor gates that can be incorporated on a single chip. The speed of these components, however, has changed little. This is particularly well-illustrated by bit-slice processors which have seen very little change in speed of operation since their introduction several years ago. To achieve a significant improvement in speed from current VLSI circuits requires the use of a different design architecture, relying on increasing the number of processing units on a chip rather than trying to boost the speed of a single unit.

To gain full advantage from increasing the number of processing units, it is important to take account of the cost-effectiveness of the final design. The most profitable solution can be achieved through replication of a restricted number of logic cells, since every cell type added to the repertoire adds to development costs. The replication of cells can be easily achieved through use of computer-aided design methods which take advantage of a modular structure to simplify design. The only development overhead comes about through the need to connect the processing units together in the most efficient manner for the system in question. The processor network which results from such a design can be shown to have a high throughput, provided that the computations are chosen to make best use of the structure.

There are a number of problems to be considered in the design of a processor, regardless of the operations it is desired to perform. It would clearly not be worthwhile designing a high speed architecture if the data source is unable to match the input bandwidth of the processor. In this situation the system would be termed *I/O-bound*. The converse is the *Computationally-bound* system, where the input rate may be greater than the speed at which processing can be completed. In this situation, it is clearly essential to ensure that the processing speed is raised either by use of faster components or some method of splitting the computational tasks amongst several processors. The systolic array approach allows the designer to match the speed of processing to the input data rate. The design is such that a higher input rate can often be handled simply by increasing the number of processing elements - clearly a useful advantage.

8.2.1 The Systolic Element

Kung defines a systolic array as a set of interconnected elements, each of which is capable of performing some simple operation. The connections between elements are usually simple and reflect a definite pattern thereby simplifying both design and implementation. This is in contrast to multi- or parallel-processor architectures where the interconnections are often complex.

The term "systolic" is derived from the pattern of data flow between elements, which is rythmic: Data flows between elements at regular intervals, as defined by a global clock. This is analageous to the flow of blood in the body as regulated by the heart. The clock pulse also defines the execution of instructions in each element, with typically one operation per element being executed at each pulse.

Kung likens the systolic concept to a car assembly line, "where different people work on the same car at different times, and many cars are assembled simultaneously". The car assembly line may be regarded as a linear systolic array, with the periodic movement of cars from one assembly phase to the next. In this case, it is obvious that the assembly operation is faster when the operation is split into phases than if one worker attempts the complete operation. The assembly operation, however is computationally bound and restricted because only one set of parts goes into the system at a time. To speed up the process, it is necessary to introduce a parallel sequence of operations so that more than one completed car can be output at the end of the phases.

One of the key features of the systolic architecture is the ability to use the same data item several times as it passes through the structure: the car assembly line would be inefficient if, at the end of each phase, the partly-assembled car were put into a store to await the start of the next process. The only communication with the array is at its boundaries with the outside world., although the cells may involve any regular pattern of communication amongst themselves. The efficiency of the data flow in a systolic array is illustrated by **Figure 8.1**. Consider the

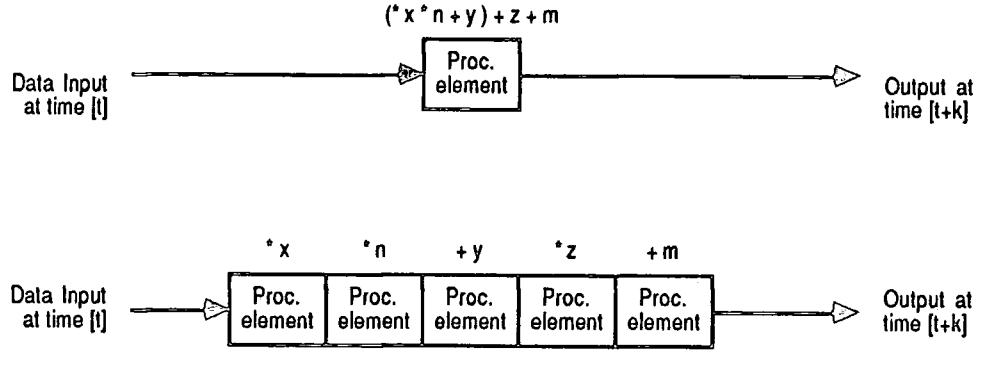


Fig. 8.1 : Principle of a Systolic Procesor

case of the single processing element, which performs five operations with each taking one processing cycle. The single processor produces an output after 5 cycles and is then ready to read the next data item.

Contrast this with the case of the 5 element processor, where each processor performs a simple task, again in one cycle, and then passes the data to the next element. The data still emerges after 5 cycles, but the throughput is increased by a factor of 5 as the first element is ready to accept the next data item after only one cycle.

The pipelining of operations between simple processors is an important feature of the systolic array, but it is equally important to provide a synchronous data path through the system, or the systolic concept is lost.

8.2.2 A Systolic Processing Example

To illustrate the principle of a systolic array processor, consider the trivial case of a matrix - vector multiplication:

$$\begin{matrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{matrix} * \begin{matrix} A_1 \\ A_2 \\ A_3 \end{matrix} = \begin{matrix} A_1X_{11} + A_2X_{12} + A_3X_{13} \\ A_1X_{21} + A_2X_{22} + A_3X_{23} \\ A_1X_{31} + A_2X_{32} + A_3X_{33} \end{matrix}$$

To perform the above calculation in software would require 9 steps, if the following algorithm were used:

```
FOR i = 1 TO 3 DO
BEGIN
   FOR j = 1 TO 3 DO
      Yi  =  Yi  +  Xij  *  Aj
END
```

By mapping the procedure onto hardware, and in particular a systolic array, we can arrive at a solution which produces a result after 5 cycles, yet uses only 3 processor elements. All that is required is that the data be presented in a specific order to the array. **Figure 8.2** illustrates the solution. In this example, each cell performs two simple operations when sequenced by the global clock:

$$Y = Y + X_{in} * A_{in}$$

$$A_{out} = A_{in}$$

Following each clock pulse, the vector is rippled through the processor array, and new input values read from the matrix. The first result appears from processor (3) after only 3 clock cycles, but a further 2 cycles are needed to accumulate the result in processor (1).

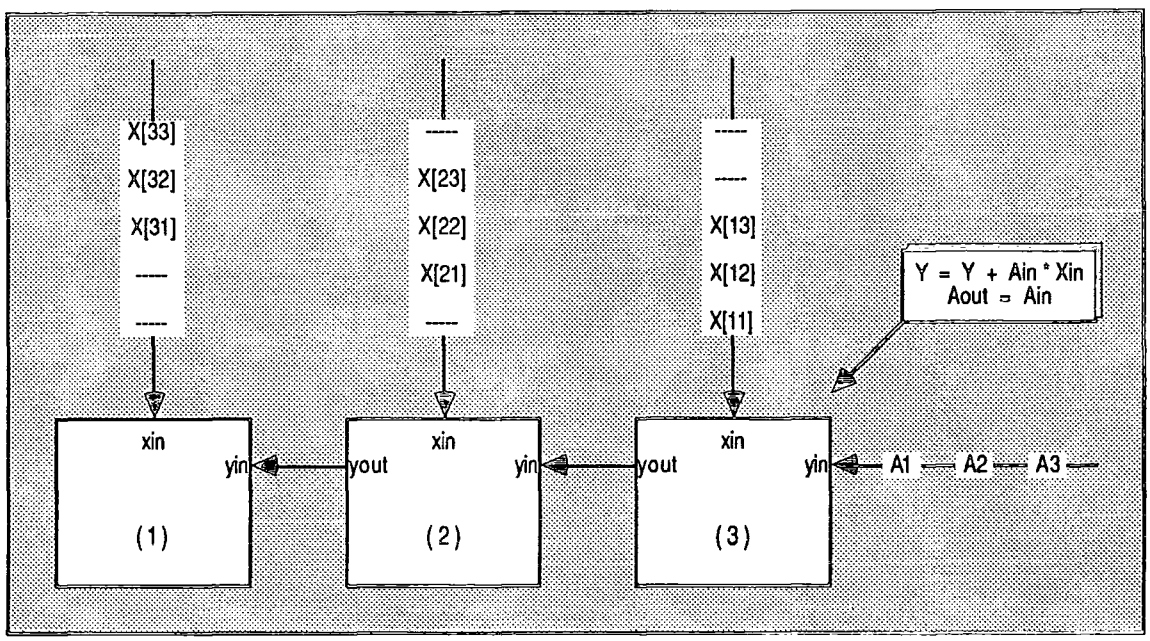


Fig. 8.2 : Example of a Systolic Matrix * Vector Processor

The two examples given so far have concerned linear processing arrays. In practice, there is no restriction on the structure or shape of the array. Kung even proposes a hexagonal array [57] to perform rapid matrix multiplication, where banded matrices are involved. In this situation, the operands and partial products are inputs to 3 sides of a hexagonal element, whilst the remaining 3 sides are used as 'ripple' outputs.

Although the simplest systolic arrays involve only one type of element, this is not a mandatory requirement for the systolic concept. It must be remembered, however, that the arrays become easier to implement as element complexity decreases and the number of different types of element is also reduced. In practice, this increases the likelihood of a radar signal processing design based around a systolic version of the RATES functional units and ultimately paves the way for a single-chip design.

8.3 Systolic Arrays in RATES

In many respects, a radar signal processor meets many of the pre-requisites for consideration of systolic architectures. The processing is often computationally intensive and frequently restricts the overall system bandwidth. The data input is regular, determined by the range clock which drives the analogue-to-digital convertor. Finally, the functions themselves are repetitive and each operates more than once on the data as it flows through the system.

To establish the optimum approach to a systolic radar signal processor, the RATES plot extractor hardware is to be considered as 3 modules: CFAR, WBI and Plot Forming. With careful design of each stage, it should be possible to merge the three modules to form a single-chip soltuion. Two types of CFAR processor will be considered - the Cell Averaging and Ordered Statistic approaches.

8.3.1 The Cell-Averaging CFAR Processor

As discussed in Chapter 2, the principle of the cell averaging CFAR is that of a window which slides over an array of range-cells. The amplitudes of the cells are averaged to determine a mean clutter level and after scaling this is compared against the amplitude of a central test cell. A target is declared if the amplitude of the test cell is greater than the mean.

If the video amplitude of the test cell is X_t then the mean value for N cells (using 2 guard cells) is given by:

$$X\sim = (\Sigma(X_{t+1} \ldots X_{t+N/2}) + \Sigma(X_{t-1} \ldots X_{t-N/2}))/(N-3)$$

A target would be identified by the CFAR if the condition:

$$X_t \geq (\Sigma(X_{t+1} \ldots X_{t+N/2}) + \Sigma(X_{t-1} \ldots X_{t-N/2}))/(N-3) * K + M$$

where K is a manually-input scalar quantity and M is due to a computer generated and(/or) manually input offset. We note the simplification:

$$\Sigma(X_{t+1} \ldots X_{t+N/2}) = \Sigma(X_{t+1} \ldots X_{t+N/2}) + X_t - X_{t+N/2-1}$$

As an example, consider **Figure 8.3**, which shows a sliding window with 12 active cells, and one guard cell on either side of the test cell. By expanding terms in the above equations, it is possible to arrive at an expression for the final mean amplitude against which the amplitude in the test cell can be compared.

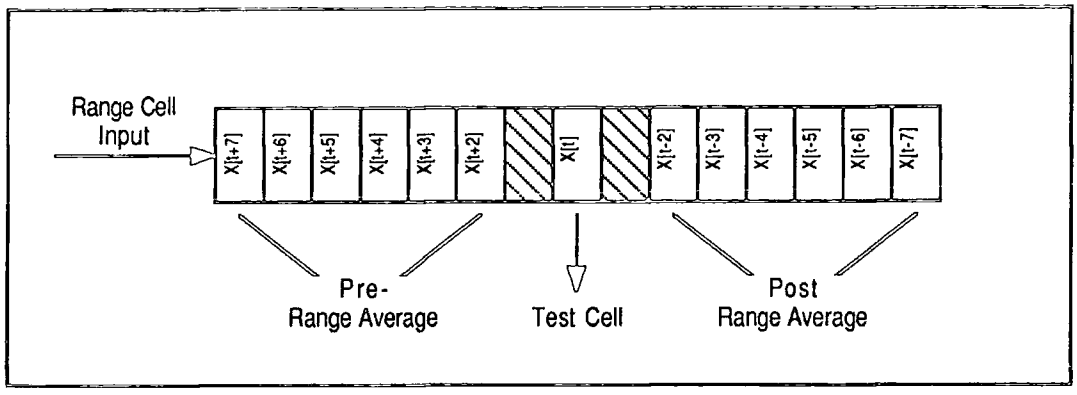


Fig. 8.3 : Example Sliding Window Detector

To establish whether a target is present in the cell of amplitude X_t, we test for the condition:

$$X_t / K - M \geq (\Sigma (X_{t+2} .. X_{t+7}) \div \Sigma (X_{t-2} .. X_{t-7})) / N$$

To maintain greatest flexibility with this test, N can either be set to the number of active cells in the whole window (12) or we can expand the expression to allow use of "greatest-of" selection, in which case N takes the value 6. This is a particularly efficient expression to transform into a systolic design and one possible solution is illustrated in **Figure 8.4.**

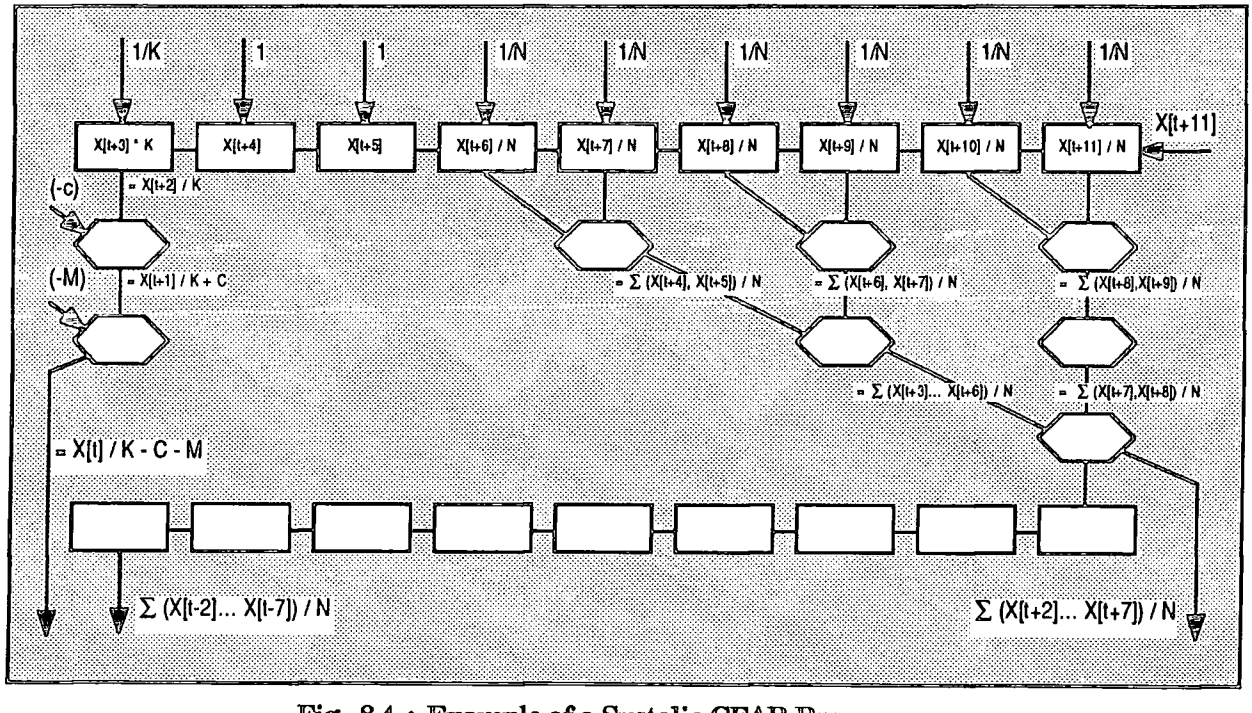


Fig. 8.4 : Example of a Systolic CFAR Processor

This design makes use of only two types of element (Figure 8.5) to produce the pre- and post- range sums and makes use of the need for built-in delays to generate a scaled value for the test cell amplitude. Three values are therefore output from the array:

$X_t / K - M - C$ (where M & C represent Manual and Computer-generated offsets)

$(\Sigma (X_{t+2}...X_{t+7}))/N$

$(\Sigma (X_{t-2}...X_{t-7}))/N$

As stated previously, a conventional Cell-Averaging CFAR can be obtained by using a value of N = 12 and adding the two sums. For a CAGO processor, a value of N = 6 is used and the function MAX((Σ (Xt+2...Xt+7))/N , (Σ (Xt-2...Xt-7))/N) applied.

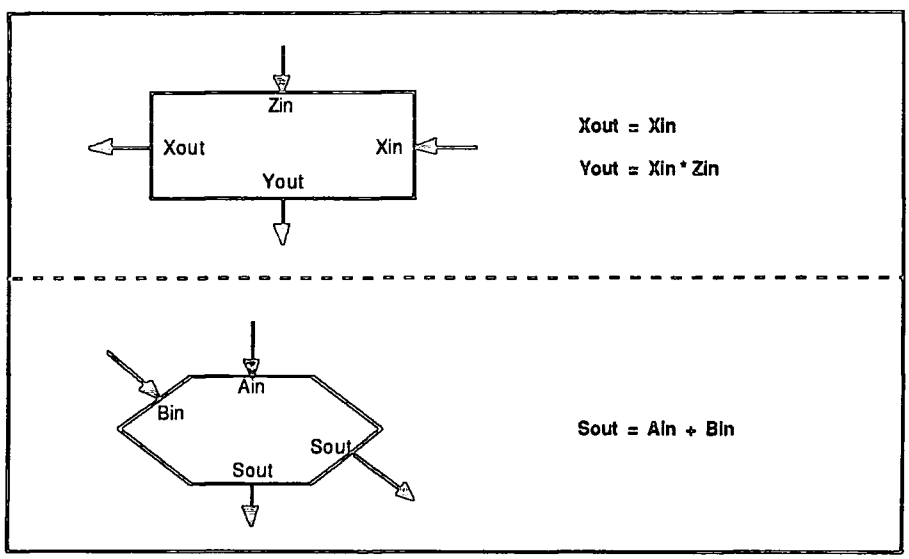


Figure 8.5 : Cells Used in the Systolic CFAR Processor

The operations indicated in Figure 8.5 are all executed following a single clock pulse. Data inputs can be considered to be latched on the rising edge of the pulse and the calculation performed & the output latched on the trailing edge. The array structure imposes a delay of only 4 cycles on the time that would be taken by a non-latched discrete logic solution.

Results of a simulation of the performance of this processor are given later.

Should a larger window size be required the array can easily be expanded, provided that the general shape is maintained (as this ensures that time delays are upheld). The structure has been chosen so that all terms appropriate to a calculation are presented simultaneously at the boundary cells.

8.3.2 The OS CFAR Processor

The ordered-statistic CFAR processor is not so well suited to a systolic design as it has no need for registers and a global clock. To extract a rank value (X_k) for use as a threshold, it is simply necessary to sort the elements in order of amplitude then extract the k^{th} item from the resulting array. The sorting procedure can be performed by asynchronous logic, as a steady state solution will eventually be reached. For completeness, an example of a systolic sort routine is illustrated in Figure 8.6.

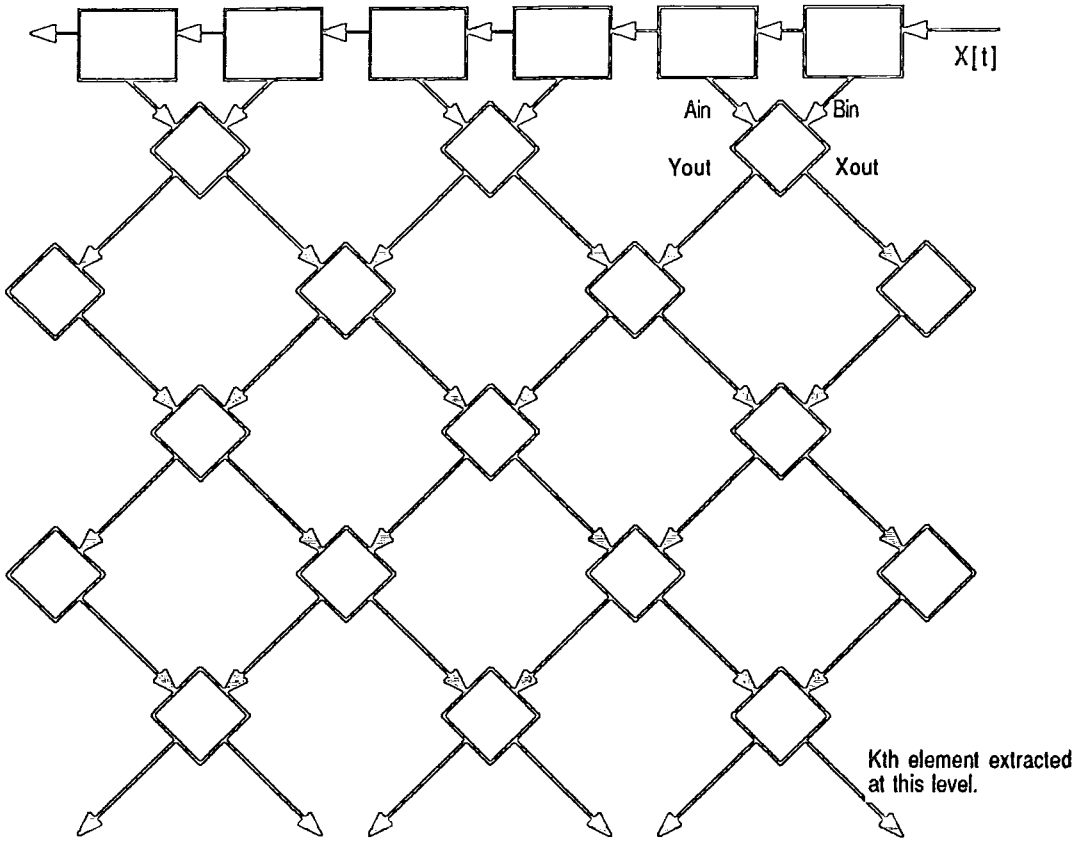


Fig. 8.6 : Example of a Systolic OS-CFAR Processor

The design uses two types of processor element: The upper row work as latches, whilst the 'diamond' elements perform the following operation:

Xout = MAX (Ain, Bin)

Yout = MIN (Ain, Bin)

The result appears after N cycles, where N is the number of cells in the window. At this point the k^{th} item can be extracted and used to calculate the rank value. The test cell should be delayed to bring it in step with the output from the sort routine before attempting to evaluate a target presence.

8.3.3 Systolic Approach to Within-Beam Integration

The problem of within-beam integration is similar to that posed by the case of the ordered-statistic CFAR, in that a satisfactory solution can be shown using asynchronous logic. Detailed studies indicate that it would be extremely difficult to implement a second threshold processor using a true systolic architecture, due to the fact that the algorithm relies on a series of logical tests to determine its outputs.

Earlier work on the idea of a systolic plot extractor [49] resulted in the suggestion of a high-level processor for the WBI function. It was based around the following algorithm (also given in chapter 4):

Let the accumulated integrator count be Y_{ij} for scan **i** and range cell **j**.

Let the binary output from the WBI at this time be W_{ij}.

The WBI is configured to use increment (INC) and decrement (DEC) values of '1' & β, as selected by the algorithm. Thresholds for upper (U) and lower (L) levels apply.

```
if X(i,j) = 1  then
{
     Y(i,j) = Y(i-1,j) + INC
}
else
{
     Y(i,j) = Y(i-1,j) - DEC
}

if Y(i,j) ≥ U then
{
     W(i,j) = 1; Y(i,j) = MIN( Y(i,j), U );
}

if ( W(i-1,j) = 0 ) & ( W(i,j) = 1 ) then
{
     INC = 1; DEC = β
}
else
{
     if ( Y(i,j) ≤ L ) then
     {
          W(i,j) = 0; INC = β; DEC = 1
     }
}

Y(i,j) = MAX( Y(i,j), 0 )
```

It is concluded that no satisfactory solution exists to implement this algorithm in a true systolic form and it is therefore recommended that, in the event of a single-chip solution being attempted, a conventional logic solution be adopted (such as proposed in chapter 4).

8.3.4 The Systolic Plot Forming Processor

The systolic concept can be used to advantage in the evaluating the centre of a plot, providing that sufficient input is made available through use of time delays. Taking the plot forming procedure as illustrated in **Figure 8.7**, a window is established around a plot area and the weighting of each cell calculated. Unlike the original RATES design, this is a continuous process but its output is only accepted if certain conditions are satisfied.

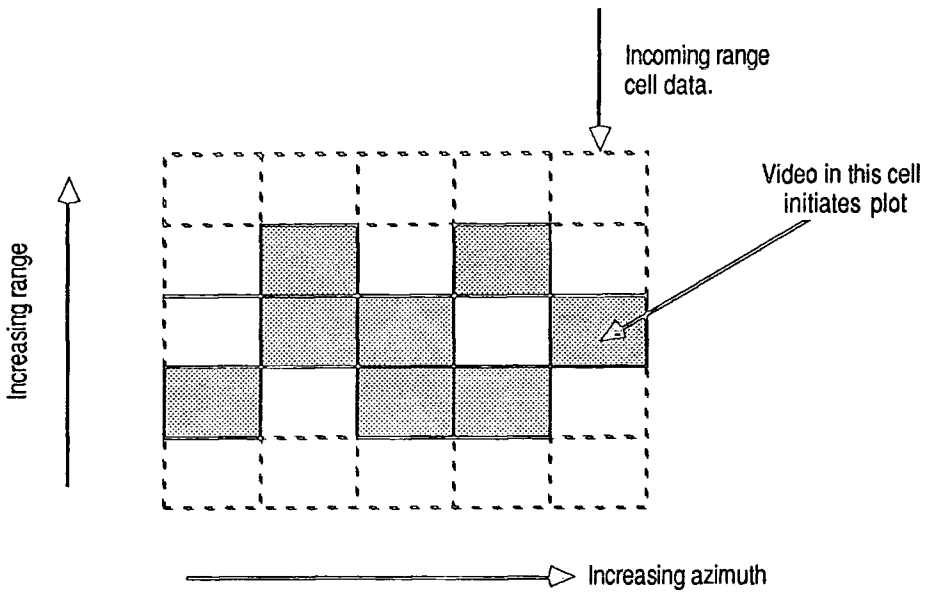


Fig. 8.7 : The Plot Window

Each cell in the window is assigned a weighting, based on its range and bearing *relative* to one particular cell. For a window 5 bearing units wide (as shown) and 3 range cells deep, the maximum bearing weight would be 5 and maximum range weight would be 3. In general, for a cell of bearing **i** and range **j** relative to a corner cell the weight is given by:

Range Weight = **(cell amplitude) * j**
Bearing Weight = **(cell amplitude) * i**

The **cell amplitude** in the plot forming processor is a binary value of either '1' or '0' depending on whether or not a target has been declared by the WBI. Calculating the weights of all cells in the window, then summing them gives a figure for the total range and bearing moments. If this is divided by the sum of the cell amplitudes, the result is the centre of the plot relative to the chosen origin.

Using this principle, we are able to design a systolic plot forming processor which produces absolute co-ordinates for the centre of the plot. The functional diagram of such a system is given in **Figure 8.8**. Only two types of cell are involved, each performing a large number of (albeit simple) operations.

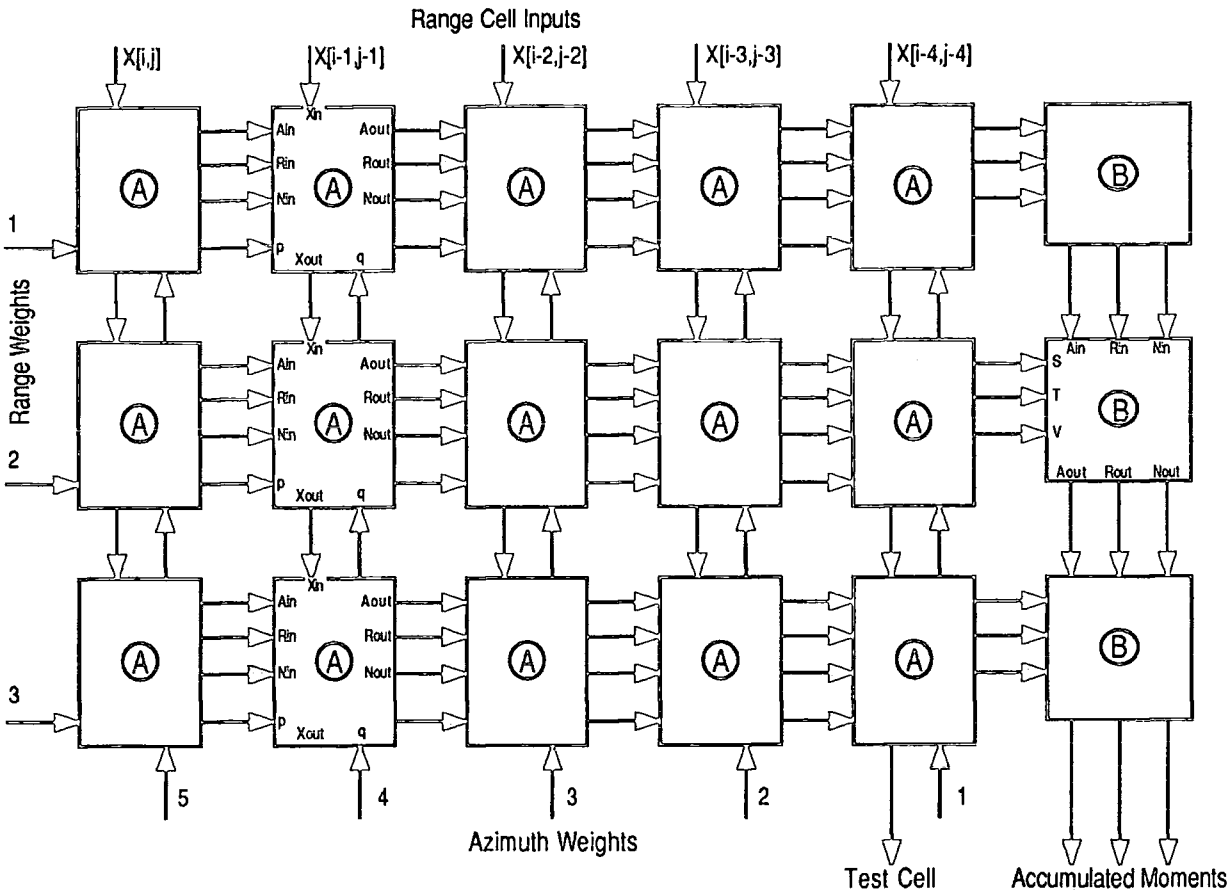


Fig. 8.8 : Systolic Plot Weight Calculator

The Type A elements calculate the following:

$$Aout = Ain + q * Xin$$
$$Rout = Rin + p * Xin$$
$$Nout = Nin + Xin$$
$$X = Xin$$
$$Xout = Xin$$

The extra delay in the value of X that is passed to the next cell is to allow for synchronisation of sums arriving at the Type 6 elements. As with the example given for the CFAR processor, data is read in on the rising edge of the global clock and the calculation performed on the trailing edge. The Type 6 elements perform only 3 operations in order to evaluate accumulated range and bearing weights and the sum of the active cells:

$$Aout = Ain + S$$
$$Rout = Rin + T$$
$$Nout = Nin + V$$

With these accumulated values, the absolute co-ordinate of the plot centre can be established. Additional logic is necessary to determine whether an output plot is valid (i.e. there is active video in the cell corresponding to range weight=2, bearing weight=1). It is also necessary to calculate the plot co-ordinates using the formulae:

$$\text{Range} = \Sigma R_w / \Sigma N \div R_0$$

$$\text{Bearing} = \Sigma B_w / \Sigma N \div B_0$$

Where ΣN is the sum of active cells in the window, ΣR_w is the sum of range moments, ΣB_w is the sum of bearing moments and R_0, B_0 are the absolute co-ordinates of the cell where **Range weight = Bearing weight = 1.**

In a practical design, it is estimated from performance of the existing RATES hardware that a window size of 3 x 5 cells is sufficient although this can be increased by adding more elements into the systolic array. The important feature of this design is that an entire window is implemented in **Type 5** elements, thereby simplifying the calculations and storage requirements.

8.4 Simulation of a Systolic Plot Extractor

To test the validity of the systolic processor designs, a number of simulations have been carried out and results compared against those that would be predicted using conventional (i.e. non-systolic) algorithms. Two simulations have been written, using a version of PASCAL running on an Apple Macintosh™ computer. One is for the cell-averaging CFAR design discussed in section 8.3.1, and the other for the plot weight calculator described in section 8.3.4. The source listing for the CFAR simulation is given in Appendix E.

The data used for the CFAR simulation is the same as used earlier in chapter 2, and this therefore provides a means of comparison with results generated by more conventional techniques. It should be noted that, not surprisingly, the numerical results that can be obtained from the systolic processor of Fig. 8.4 are identical to those that can be obtained by averaging the 6 cells either side of the guard cells. A graph of the threshold level against the test video level is shown in Figure 8.9 and clearly illustrates the points at which a target could be declared.

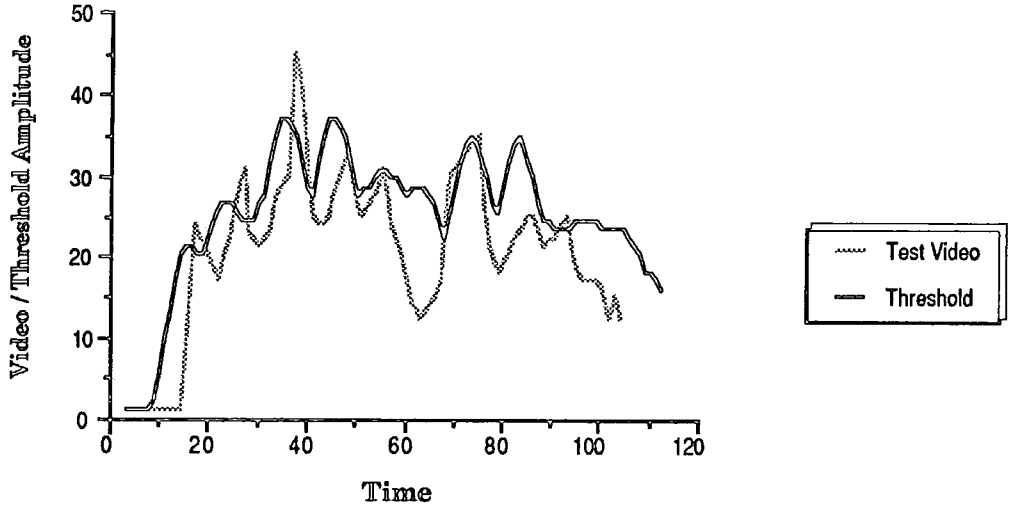


Fig. 8.9 : Systolically-generated CAGO Threshold

The plot forming processor has been tested using a series of simulated input plot windows and then checking to see if the calculated plot centre is correct. For example, the sample window shown in Figure 8.7 generates a total bearing weight of 24 and range weight of 17. Since there are 8 active cells, the processor evaluates the plot centre to coincide with the centre of the window. If the balance of active cells in the window is shifted, the centre can be seen to move accordingly.

8.5 Logic Simulation of Plot Extractor Functions

With initial results from the logical simulation of the plot extractor functions appearing encouraging, it was decided to obtain further confirmation of the viability of the systolic approach through more detailed simulation techniques.

The simulations described so far have confirmed several approaches to the design of a systolic replacement for the RATES plot extractor, but have made no attempt to assess the potential throughput of such a device. This has been remedied by taking each of the proposed systolic plot extraction functions and converting it into a detailed design based around standard 74LS-series logic (with some use of MSI and LSI components where available). Each design is then simulated at gate level, giving rise to timing diagrams for each systolic element and for each complete function. The designs are then merged to give a detailed prediction of the timing performance of a complete systolic plot extractor.

8.5.1 Simulation Software

The 8 gate-level simulations have been performed by custom-written software, running on an Apple Macintosh computer (since suitable proprietary software was not available at the time). The software provides a "black-box" simulation of standard 74LS-series TTL devices, based on average signal propagation times as quoted in semiconductor manufacturers' literature (National Semiconductor, in this case). Whilst the implied accuracy of this simulation is to the nearest nano-second, a more realistic figure would be ±10% owing to the limitations of precision of the figures in the data sheets. The simulation software takes full account of the setup and hold times on data inputs and is capable of identifying output glitches and potential race conditions.

Each circuit is configured to the simulator through an ASCII text file, in which each device is given a unique numeric reference and connections are made by linking inputs to an output device reference and pin number. A typical circuit definition file, together with the output it produces, is illustrated in Figure 8.10.

A single clock provides synchronous clocking for the systolic elements, at frequencies of 50, 25, 20, 10, 5 and 1MHz and also 500KHz. Ten separate triggers are provided, capable of starting at a known time with either a finite or infinite period. The triggers can also be used as sources for external data, their outputs changing on either rising or falling edges of the system clock. Standard output from the simulator is a timing diagram, although additional features have

been programmed to provide confirmation of the correct operation of the circuit whilst simulation is in progress.

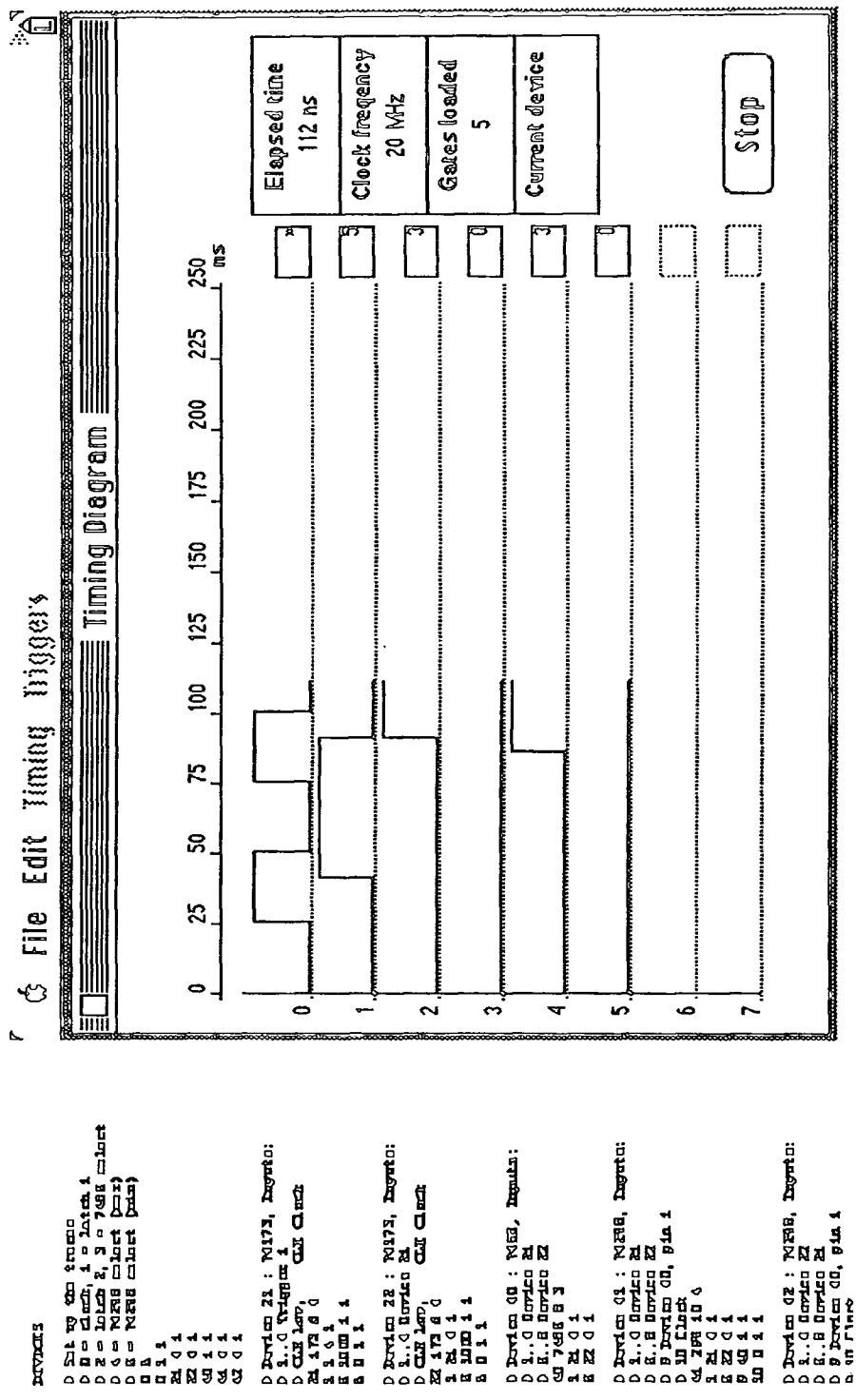


Figure 8.10 :
Logic Simulator Input File and Sample Output

8.5.2 Simulation of the Rank-Ordered CFAR

The discussion of systolic CFAR processors has identified two potential architectures for such a device. Whilst the systolic cell-averaging CFAR provides the closest approach to the existing RATES design, the Rohling Rank Ordered CFAR exhibits a number of advantages, including:

- faster response, without loss of selectivity
- greater suitability towards systolic implementation.

The critical element in this design is the maximum/minimum data selector, which outputs the greater and lesser of two inputs. The basic design has been simulated for two possible approaches. The first assumes a fully systolic element, as shown in Figure 8.11. A double output latch is required in order to eliminate potential race conditions, due to variations in device speed. By using a single output from the 74LS85 (rather than less-than and greater-than outputs) to select the required latch data, the design ensures that an output is produced from both latches, even in the case of "A=B".

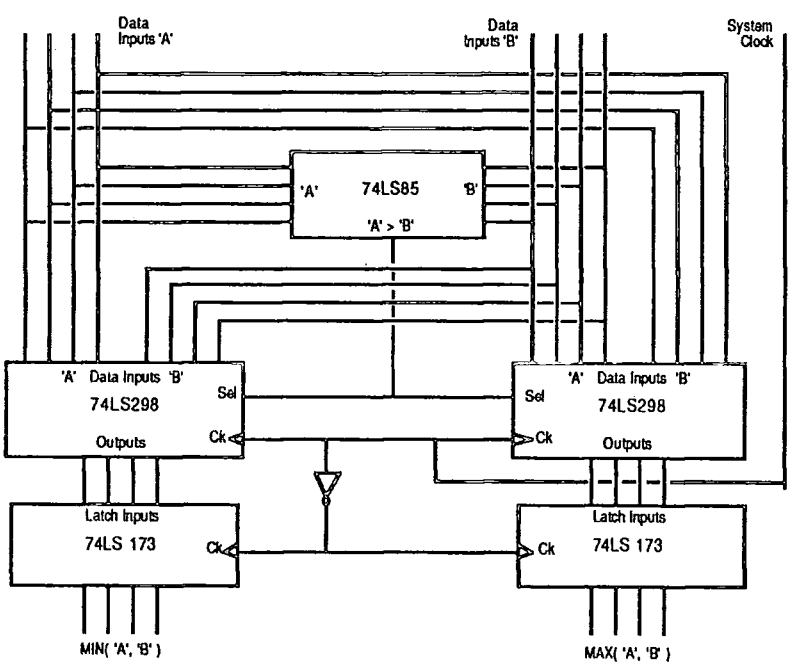


Figure 8.11
Systolic 4-bit Sorter Element

The timing diagram of this single element is illustrated below (Figure 8.12), with the clock frequency set to 20MHz. Whilst a slightly higher clock frequency could probably be accepted, the propagation delays due to the latched elements, combined with the need to utilise both rising

and falling edges of the system clock, result in this frequency being the maximum recommended for this element before the possibility of race conditions arises. The inputs applied to the element are illustrated on the diagram, with trigger data being loaded at 25ns (value = 3) and 75ns (value = 10).

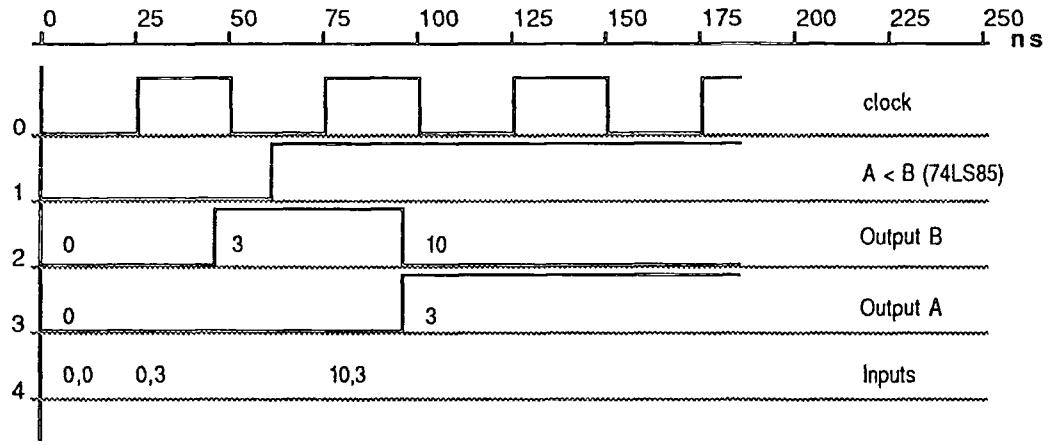


Figure 8.12
Single Sort Element - Timing Diagram

This basic element is replicated 31 times in the implementation of a 9-cell CFAR window, as illustrated in Figure 8.13.

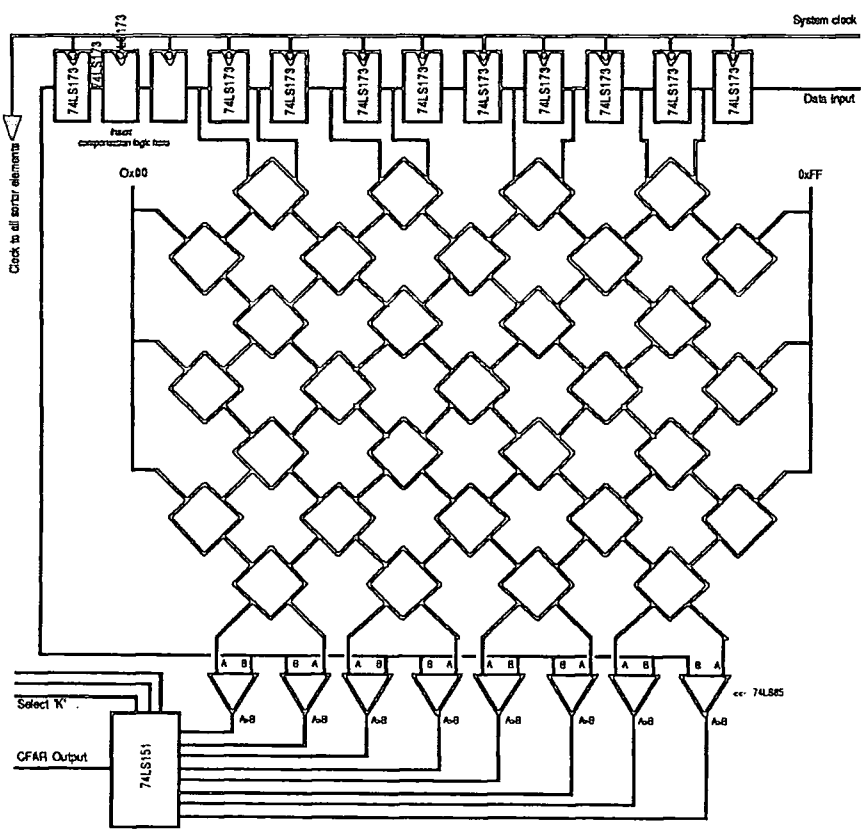


Figure 8.13
Complete Systolic Rank-Ordered CFAR Design

Because of the regularity of the design, a larger (or smaller) window could be accomodated simply by increasing (or decreasing) the dimensions of the array. The output of the central test cell does not contribute to the evaluation of the rank statistics. The output of the sorter is applied to a bank of comparators, and a simple data selector used to isolate the "K^{th} " element target status.

This design utilises a total of 174 standard TTL gates of medium complexity - it does not take account of cell scaling or offset as it is considered that these extra calculations are essentially sequential and do not contribute to the overall systolic simulation (however Figure 8.13 does indicate where the required logic should be placed).

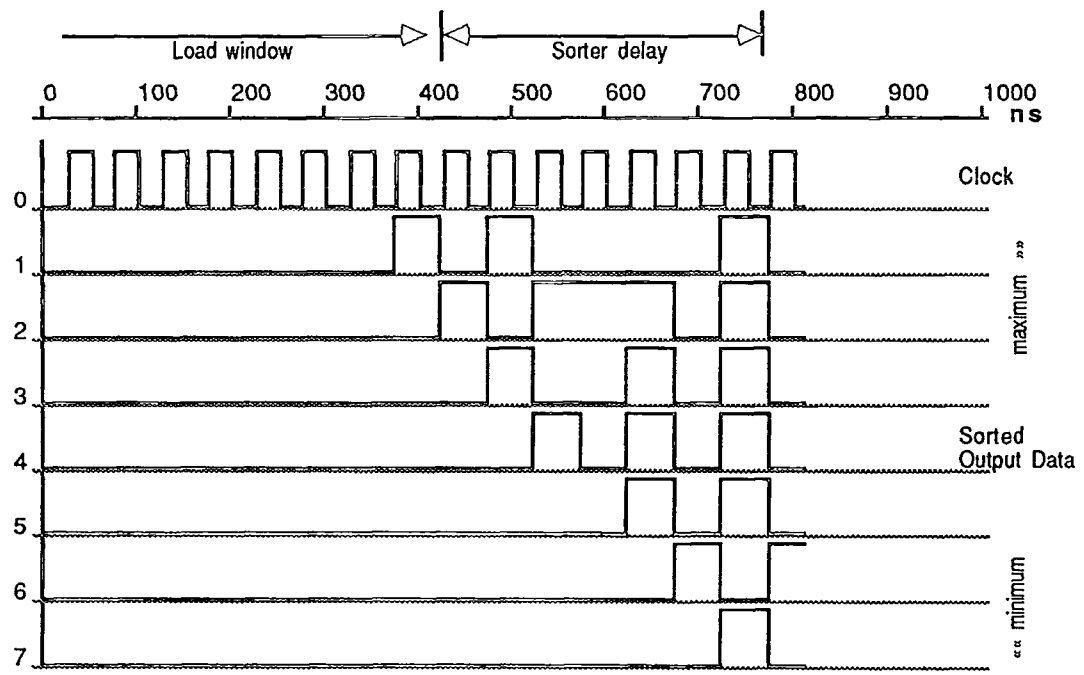


Figure 8.14
Timing Diagram for Rank-CFAR Sort Output

The timing diagram for the window and sort logic is illustrated in Figure 8.14, taking account of the response from the start of a new p.r.p (when the window is initialised) to the appearance of the first valid output data. The data used as input to this simulation is loaded on the rising edge of the system clock, and consists of the following values:

° 3 5 7 9 2 4 6 8 10

At a time (t), the window consists of the above 9 values, with '10' being the most recent input. The value '2' is in the centre of the window and therefore does not contribute to the sorted output, which appears at time (t+7), as described below.

The final outputs, represented in the timing diagram, along with the intermediate outputs at times (t-2) ... (t+6), consist of the following values:

	<<	maximum		...	minimum		>>
∘	10	9	8	7	6	5	4

(The value '3' is not displayed as the simulator is limited to a maximum of 8 concurrent timing displays).

At a frequency of 20MHz, the load takes a total of 9 clock cycles (450 ns, although the first value is preloaded and the simulation only shows 400ns) with the sorted output appearing $7\frac{1}{2}$ cycles later (another 375ns). Thereafter sorted data appears every 50ns, shortly after the trailing edge of the system clock pulse.

The output from the "K^{th}- element selector" is not included in the timing diagram, which simply attempts to illustrate the operation of the systolic sorter. An estimate of the additional delay due to the selector and comparators would suggest that the final target declaration appears 36ns after the sorter output - some 20ns after the rising edge of the system clock. From this, it is proposed that the trailing edge of the system clock be used to latch the CFAR output, prior to the output being passed to the within-beam integration circuit.

An alternative approach to data sorting is to design the array to be asynchronous. Data is clocked through the window as before, but the sorted output appears after it has rippled through a number of asynchronous sort elements. Differences between the propagation times of the sort elements will result in the final output changing many times before a state of equilibrium is reached, and it is the time taken to reach this state which defines the maximum rate at which data may be clocked through the window. Figure 8.15 illustrates one attempt at assessing the maximum clock speed of such an approach- the circuit is largely unchanged from the previous design, although those elements which perform a dummy compare-and-latch operation are now replaced with direct connections. The sample cell output is taken from the centre of the window and then used for comparison against the sorted data.

The data used in this simulation are the same as for the synchronous sorter, with a new value being loaded on each rising clock edge. In the example shown, the result of the sort operation does not match that due to the synchronous design, as the clock frequency is too high, resulting in race conditions in the component devices. To achieve satisfactory results, the clock frequency must be reduced below the 20MHz illustrated overleaf.

The difficulties of matching a suitable clock frequency and latch time clearly conflict with the supposed advantages of the systolic approach, and hence it is proposed that the systolic rank-ordered CFAR of Figure 8.13 be retained in preference to the asynchronous version described above.

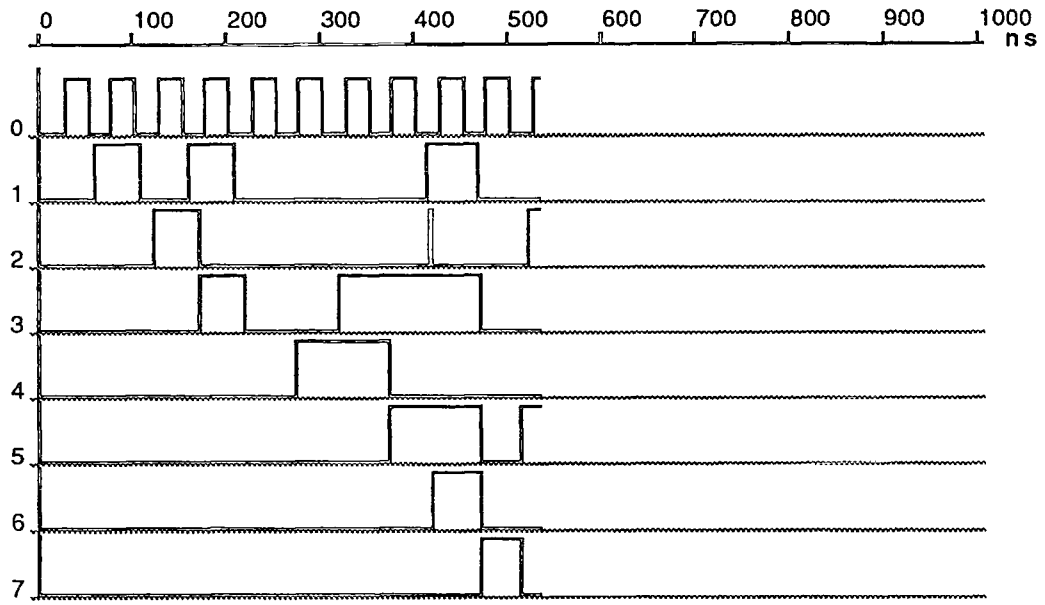


Figure 8.15
Timing Diagram of an Asynchronous Rank-CFAR

8.5.3 WBI Simulation

As discussed earlier, the within-beam-integration function cannot be readily adapted to systolic processing without some compromise or pseudo-systolic design, as the basic WBI function is entirely sequential - there is little or no advantage to be gained from forcing a parallel architecture upon it. For the purposes of this simulation exercise, however, the original RATES design has been refined to give a much simpler implementation path - the proposed circuit is illustrated below in Figure 8.16. The circuit remains sequential, with key elements clocked on the rising or falling edges of the single system clock pulse. To maintain compatibility with the preceding CFAR processor, the incoming target declarations are assumed to be latched on the falling edge of the system clock, which is supplied, inverted, to the WBI RAM address control logic. The function of the WBI has been analysed as two distinct operations:

- Selection of the increment/decrement value
- Calculation of the new sum and target status.

The increment/decrement value is dependent only upon the previous target status and incoming status, hence the value is derived through a 1-of-4 selector. Given the Marcoz and Galati choice of (+A, -1, +1, -A) and parameters T^{-1}(previous target status), Z (new target status), this choice is selected from the combination ($Z\sim T^{-1}$, $\sim Z\, T^{-1}$, $Z\, T^{-1}$, $\sim Z\, \sim T^{-1}$).

The new threshold sum is calculated from the sum of previous threshold and the above increment/ decrement, modified by the conditions described in earlier sections. The target status is evaluated from the new threshold (Σ), previous status (T^{-1}) and upper and lower threshold limits (U, L) as:

$$T = \overline{(\Sigma < U)} + (T^{-1} . \overline{(\Sigma < L)})$$

In evaluating the performance of this WBI processor, it should be clear that the performance is clearly limited by the response time of the RAM, which should be chosen to be a device optimised for a rapid update cycle (read-modify-write). The use of fast dynamic RAMs in this situation is not precluded as all active cells are guaranteed to be read-refreshed at least once during the normal refresh period (2ms).

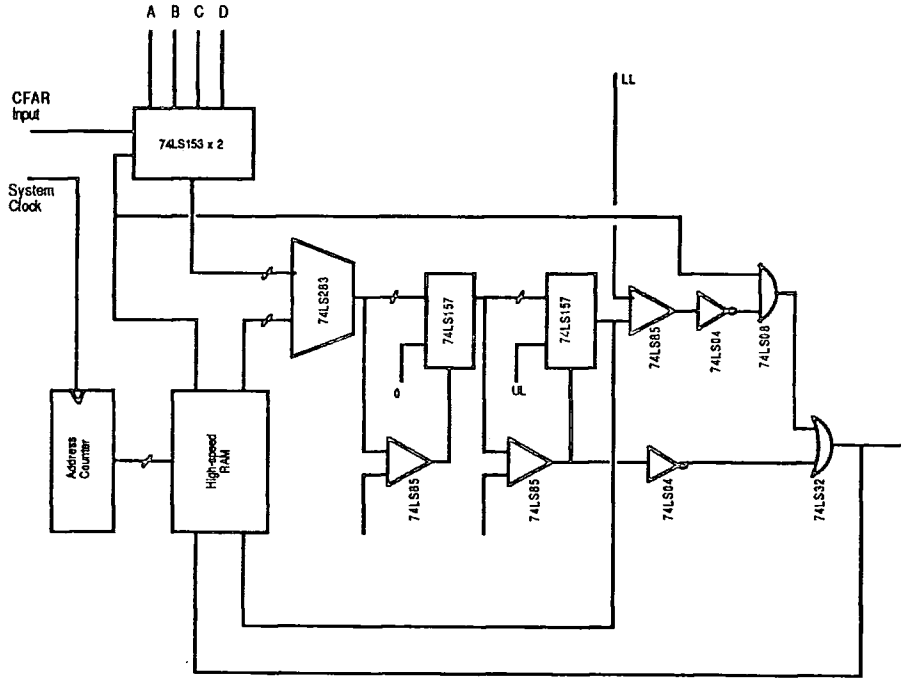


Figure 8.16
Basic Circuit of the Optimised WBI Processor

The principle of operation is as follows:

- The falling edge of the clock signifies the start of a read cycle from the RAM (trace 0)
- The previous cell count and target value are read from the RAM (trace 1)
- An increment or decrement value is selected, based upon the previous target value and incoming target status (trace 2)
- The increment/decrement is added to the cell count (trace 3)
- The new cell count is evaluated to establish whether a target exists (trace 6) or if an overflow or underflow has occured (trace 5).

The address generation logic has not been included in the simulation, as this is clearly a fixed overhead, independent of RAM speed. Without this logic, the timing diagram (Figure 8.17) indicates that a clock speed of 10MHz can be supported, whilst still leaving approximately 35ns in which to generate the RAM address (an acceptable period). This would assume that the new values of cell count and target status are clocked into the RAM on the rising edge of the inverted system clock (also signifying the start of the next read cycle, with the first operation being address generation).

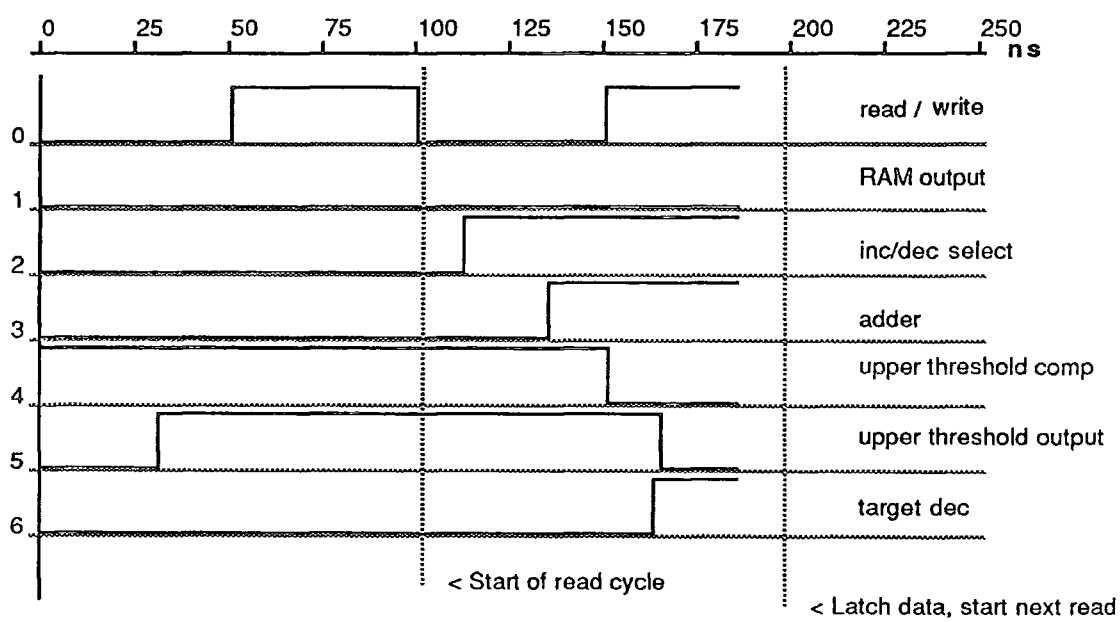


Figure 8.17
WBI Timing Diagram with 10MHz clock

In contrast to the 10MHz simulation, the effect of doubling the clock frequency is illustrated in Figure 8.18. A 20MHz frequency cannot be supported without the use of multiple clock phases or

separate input and output memories, as the propagation delay through the circuit results in the new data becoming available after the start or the next read cycle. A circuit being operated at this frequency would have to rely on known propagation delays in order to avoid race conditions.

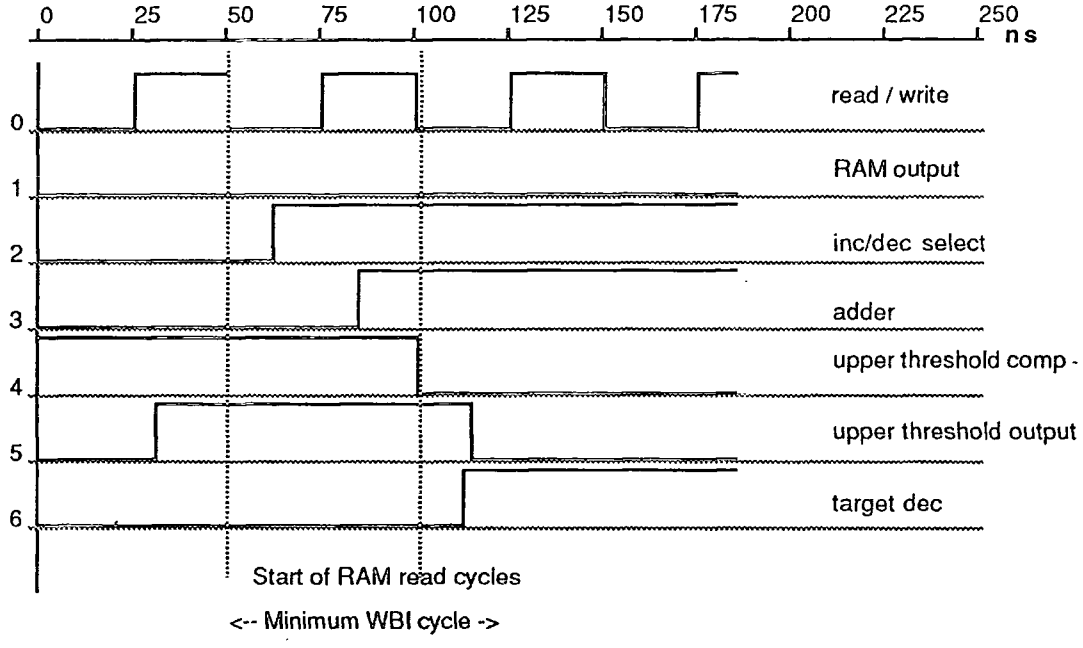


Figure 8.18
WBI Timing Diagram with 20MHz clock

The conclusion from this set of WBI simulations is that an effective single-cycle, 10MHz WBI processor could be implemented using the proposed design, although the choice of suitable components would be critical. There are a large variety of RAMs now available, offering access times from 15ns (e.g. AM10470, 4K x 1), which could be used to achieve the desired response. Alternatively, the use of fast-TTL logic would enable a longer RAM access time to be permitted, whilst still maintaining a 10MHz operating frequency.

8.5.4 Plot Forming Processor Simulation

The plot forming processor forms the third core component of the "RATES-based" systolic plot extractor. The simulation makes use of the design proposed in section 8.3.4, with additional support logic where necessary.

As discussed, the design comprises two basic element types:

- Type 'A' - each element represents one cell within the plot window
- Type 'B' - each element serves to partially-accumulate the moment of the window.

8.5.4.1 Simulation of the Type 'A' element

The Type 'A' module is implemented as illustrated below (Figure 8.19), using LSI multiplier components to reduce complexity. Through this enhancement, the plot forming circuit becomes the least complex of the three plot extractor modules. Latches are included to hold the results of the operations, provide the correct timing delays and remove the risk of race conditions. They also serve to transform an otherwise sequential element into a systolic one, driven by a single global clock.

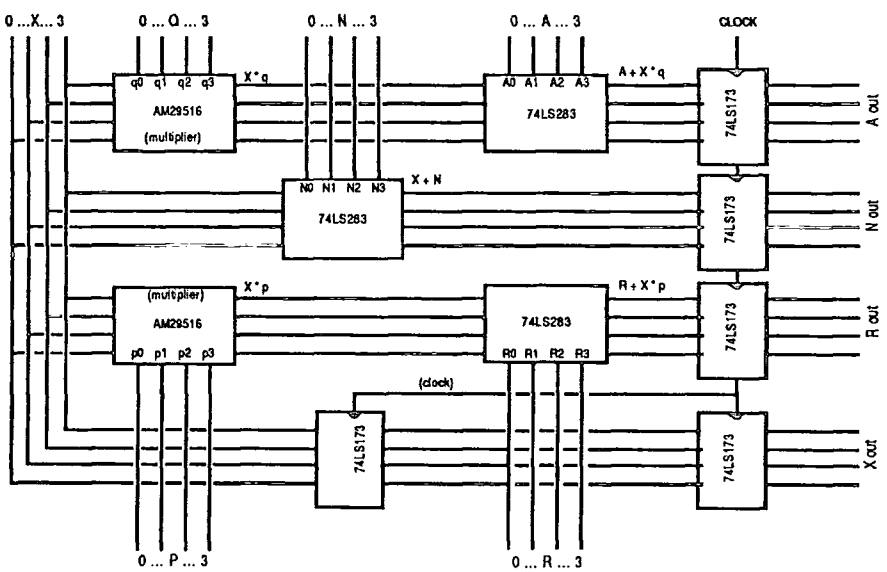


Figure 8.19
Plot Forming 'A' Element Schematic

Using the components identified above, the complete element can be driven at a frequency of 10MHz with a generous allowance for extra delays. It should be noted, however, that the

multiplier chosen for this circuit is one of the fastest currently available. From the results given in the timing diagram (Figure 8.20, below) a component with a longer propagation delay could be used in place of the AM29516 without impact on the performance, providing the 100ns total cycle time for multiplier, adder and latch data setup could be maintained.

In common with the outputs from the WBI processor, input data is loaded and the output latches clocked on the rising edge of the global clock. Double buffering (as used in the CFAR processor) is not implemented in this design owing to the lengthy propagation delays which ensure that race conditions are unlikely to arise.

As with other designs, a 4-bit data path is illustrated for the sake of clarity. Simulations have been undertaken using either 4 or 8-bit data pathes where appropriate, although the systolic principle is upheld in that the data word size can be increased by a parallel combination of elements without impact on the speed of operation or other performance criteria.

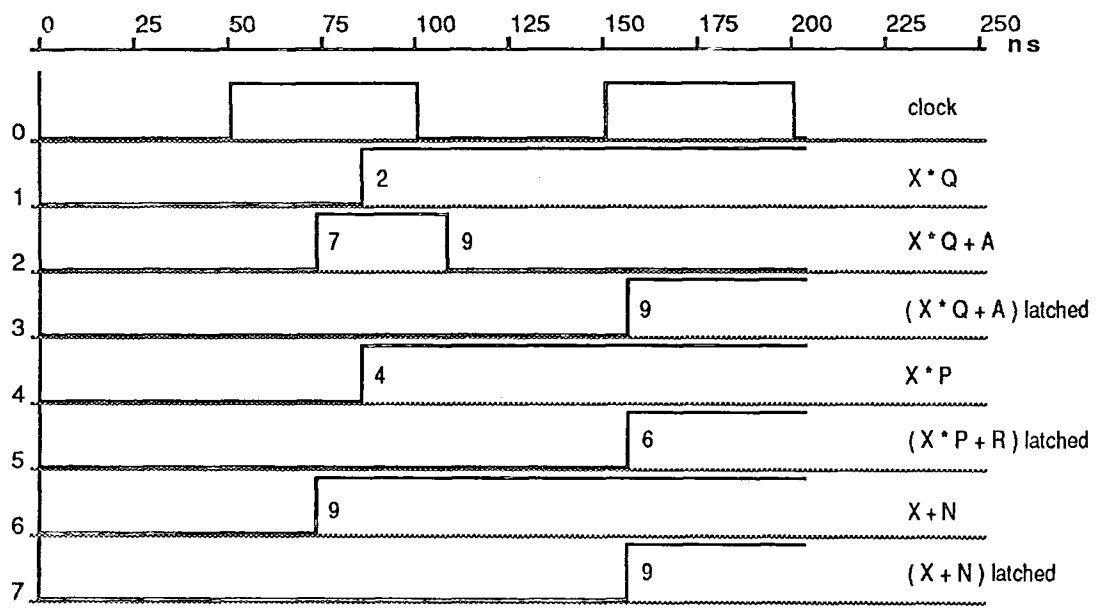


Figure 8.20
Timing Diagram for Type 'A' Element

In the case of the timing diagram illustrated above, the following data are used as input to the element:

X: 1	Q: 2	P: 4
N: 8	A: 7	R: 2

All data are presented to the element at time (t = 50), proceeding through the asynchronous

elements to the output latches, where they are loaded on the next rising edge of the system clock, to appear as outputs at time (t = 155).

As can be seen from the timing diagram of the function (X * Q + A), there is a clear 40ns between the output from the adder and the same data being latched, hence a slower multiplier could be used as discussed above. With further allowances for delays between cascaded functions, it is suggested that any device with a propagation delay of less than 65ns could be used in place of the high-speed AM29516.

The outputs from the two delay latches, illustrated in Figure 8.19, are not shown in the timing diagram as it should be clear that these components do not present a risk to the time-critical operation of the element.

8.5.4.2 Simulation of the Type 'B' Element

The type 'B' element can be implemented very easily using standard TTL components, as illustrated, in part, in Figure 8.21. This diagram shows one of the three identical adder functions of element 'B', implemented using the full eight-bit data path (a 16-bit data path could be used but is considered unnecessary due to the limitations placed upon the size of the plot window and the consequent limitation on the maximum moment that can be accumulated within it). Even using cascaded 4-bit TTL adders, followed by a data latch, the timing diagram of Figure 8.22 indicates that this element can function well within the desired 10MHz minimum throughput, and could sustain a 20MHz system clock if required.

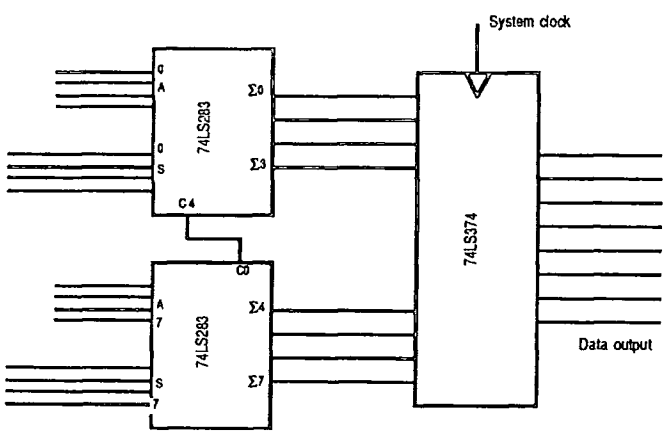


Figure 8.21
Plot Forming 'B' Element Schematic (part)

The use of the 74LS283 TTL adders provides a useful guide to the performance of this adder, as these are industry-standard parts which have been optimised for cascaded operation to enable

addition of n-bit operands. Each adder therefore includes a high-speed look ahead carry generator which produces a carry-output after only 12ns. Taking account of carry signals, an 8-bit addition can be performed by a pair of devices in 25ns (assuming use of the 74LS283 and 2KΩ, 15pf loading). Typical performance characteristics are illustrated in Figure 8.22 (assuming a 10MHz clock rate). This gives the response of the circuit to the following data, applied at a time (t = 55):

- Low order, input 1: 12
- Low order, input 2: 7
- High order, input 1: 3
- High order, input 2: 2.

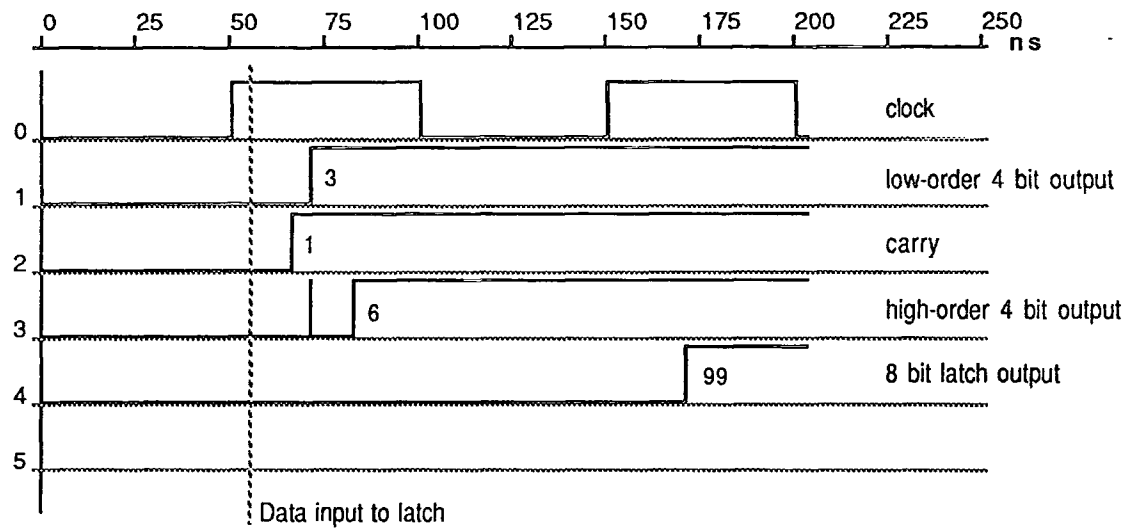


Figure 8.22
Timing Diagram for 8-bit Plot Forming Adder (part)

This timing diagram confirms manufacturers' predictions concerning the performance of a cascaded adder, with a stable output appearing after 25ns. The output data from each adder is merged into the 74LS374 and latched on the next rising edge of the 10MHz clock pulse.

8.5.4.3 The Complete Plot Forming Module

The complete plot forming module comprises as many Type 'A' elements as there are cells in a plot window of maximum width and depth. A limit must be imposed on the size of the window in order to produce meaningful results in the case of two or more closely adjacent targets - the example illustrated in Figure 8.8 is a 5 x 3 window. The window is bounded on one side by as many Type 'B' elements as the window is deep.

In operation, three sums appear from the outputs of the final Type 'B' element, some 15ns after the rising edge of the system clock pulse. The centre of the plot window is then evaluated (sequentially) from the total number of cells and the moments along each of the axes of the window. This process consists of one inversion (performed by PROM lookup) and two multiplications (35ns each, in parallel) and adds a further clock cycle delay to the calculation of the centre of gravity of the window. The operation of the plot forming processor cannot be illustrated by the logic simulator, but using the results from other simulations, the following can be deduced:

> If a WBI target declaration enters the window at time (t), the centre of gravity which can be attributed to this value is output at time (t + w + 2), where (w) is the width of the plot window.

The existing design of the plot extractor does not present an optimum approach to integrated manufacture. From Figure 8.8, it can be seen that a delay of (range_cells + 1) must be provided between the inputs of adjacent vertical groups of Type 'A' elements. The WBI processor, however, generates a similar delay through its method of operation. It would therefore seem logical to combine the two functions and take advantage of this inherent delay. The resulting module has been termed a "WBI Slice", and is discussed in greater detail in the next section.

8.5.5 The WBI Slice

The WBI slice is illustrated in Figure 8.23. The WBI section has identical timing characteristics to the conventional WBI processor, although the operation of the RAM is handled differently. The output of the WBI is applied (with only a short delay, where required) to the input of one vertical slice of Type 'A' plot forming processors.

The resulting device appears little different to the designs proposed so far, however the advantage of such an architecture can be seen if each slice were considered to be one integrated component (say, manufactured on a small gate array - the complexity is too great for a conventional PLA). A plot window of any desired width can be built-up simply by connecting devices in parallel, using the specified connections to cascade devices. The window is terminated by a single device which appears as a modified version of the elements discussed in sections 8.5.4.2 and 8.5.4.3, serving to accumulate the moments and calculate the centre of gravity of the entire plot window. With only a single cycle delay due to the WBI, a CFAR target declaration entering the WBI processor at time (t) produces a result at time (t + w + 3), (w defined as above). As with previous examples, all synchronous elements receive the same

basic clock frequency, deemed to be the highest frequency that all component modules are capable of supporting (in this case, 10Mhz).

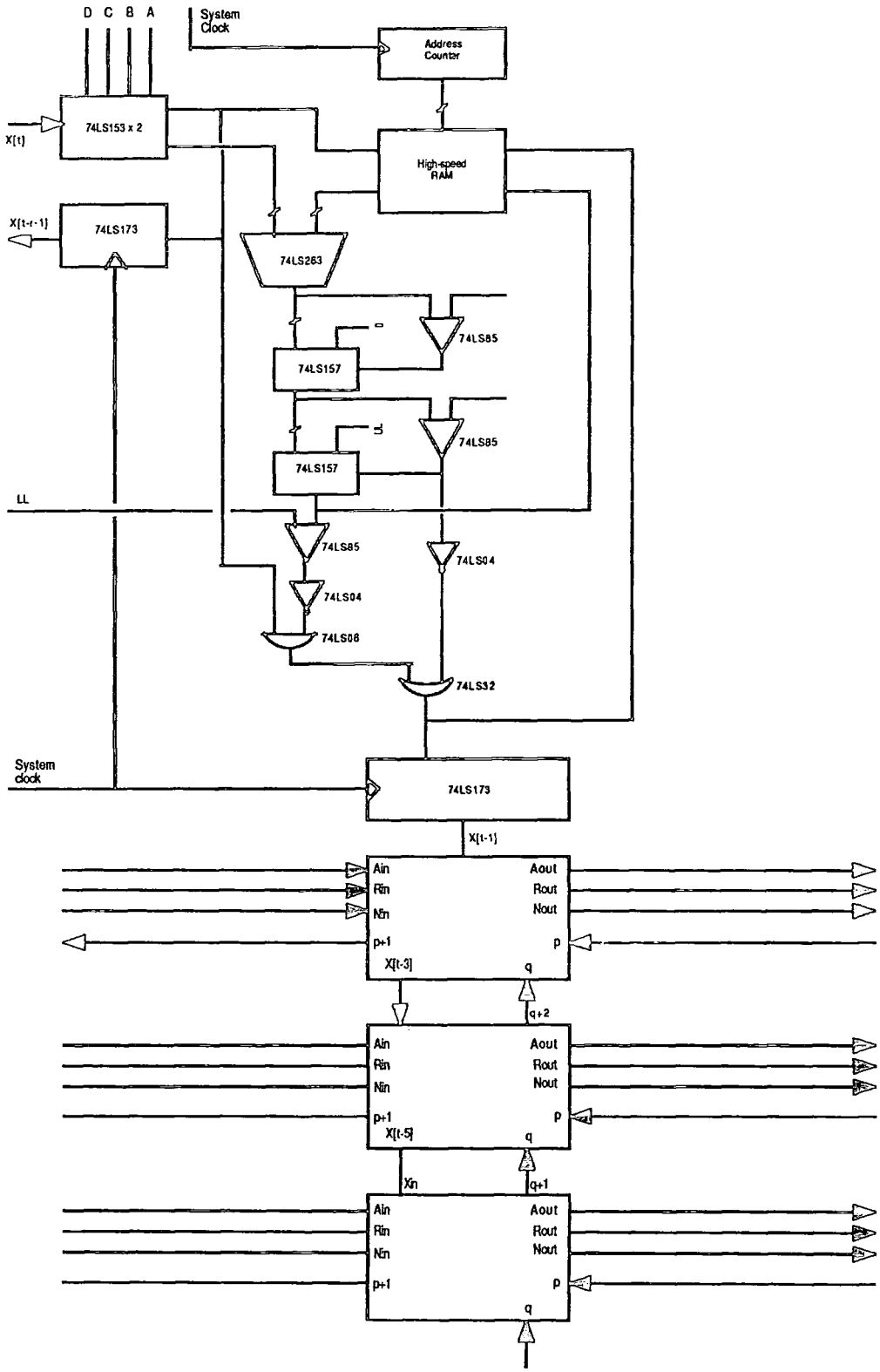


Figure 8.23
Suggested Design of a "WBI Slice"

The principle of operation of the WBI slice is straightforward: Each slice expects, as input, the target status:

$$X_{[t-n*(r-1)]} \quad \text{and generates the modified target status} \quad X_{[t-(n+1)*(r-1)]}$$

(where values of (n) in the range 0 ... (window width - 1) apply). From this information, it is established that the primary target status of any slice can be generated from the target output of the adjacent slice, and the extra delay added by the slice's latches serves to ensure that the correct values are applied to the plot forming components.

The following differences should also be noted, between the modules described earlier and those used above:

- An adder is included in each Type 'A' module to perform the function "p = p + 1". This generates the correct weighting across the width of the plot window (right to left).

- An adder is also included to perform the function "q = q + 1". This generates the correct weighting through the depth of the plot window (bottom to top).

- A latch is included at the output of the WBI section - this is needed to ensure correct "systolic" operation of the WBI function although it was not shown on earlier diagrams.

This proposed method of implementing the WBI and plot forming functions combines the two modules in a more realistic manner than previously proposed. Each slice is fully expandable and can be cascaded with any number of neighbouring units, provided a terminating slice is included. This slice is similar to the slice described above, but excludes any connection between the WBI and plot forming components, which themselves are replaced by the Type 'B' adder elements. The target input to this terminating slice is forced to zero to comply with a closure of the plot sequence.

The timing characteristics of all component elements are as for the individual cases discussed in section 8.5.4 of this document.

8.6 Summary of Simulation Results

In considering the results of these simulations, it is important to consider the advantages which may be gained from pursuing a systolic approach to the design. These have been regarded as follows:

- Greater integration of components, with the potential goal of a single board or (ideally) single-chip plot extractor.

- Greater throughput, due to reduction of interconnects between components and removal of the need for a backplane, plus the ability to use faster devices.

- Improved design architecture which can be readily adapted to test new techniques and which has built-in capabilities for future expansion.

The second of these points can be regarded as the most important for any new system, and it is this feature which is often quoted as being the key advantage of the systolic approach. The simulations which have been performed have set out to illustrate the potential for improving the throughput of the existing RATES system by adopting a systolic approach. A feature of these simulations has been to retain the basic modular approach (A-D conversion, CFAR, WBI and Plot Forming), whilst adopting different architectures where appropriate for each of the functions. The very nature of this approach has resulted in a number of problems in devising a design which meets the criteria of a "systolic system", as several functions (or parts of functions) have not proved suitable for implementation as true "systolic arrays".

The result of this work has been a design which achieves its primary objective by offering a throughput of more than twice that of the existing RATES design. All elements have been shown to function successfully at a clock speed of 10MHz, all supporting a single clock cycle between successive outputs. This effectively satisfies the requirements of a "systolic system" although the term is taken in its broadest context here, as the only true systolic arrays are those of the CFAR sorter and the plot calculator/accumulator. The advantage of the single clock frequency is clear - all elements are clocked at the frequency of the range sampling clock, and hence there is no need to derive intermediate frequencies to produce calculation results. Moreover, the delay due to the WBI and CFAR processors can be quoted as an exact offset from the time of the incoming data, and compensation included if it is desired to display all three outputs coincidentally (something which is not easily achieved in the current RATES design).

As the accuracy and resolution of radar equipment improves, the need for frequencies above

the 10MHz, which has been proven with this systolic design, will arise. The proposed design will cater for these requirements without extensive modification, as the following points should be noted when considering a practical design:

- The simulations have assumed the use of 74LS series TTL components throughout, with only brief mention of other devices. The 74LS, whilst offering low power consumption, sacrifices speed of operation in return. If higher frequencies were required, most components could be replaced with 74S, 74H or 74HC alternatives with a considerable increase in throughput (for example, the 74S283 has a lookahead carry generation delay of 6ns, compared with 11ns for its 74LS equivalent).

- Where other families of device have been used, a range of operating speeds has been quoted: the multipliers used in the plot forming section could have a propagation delay of up to 65ns without affecting the operation of the circuit. Similarly, the WBI RAM will support access times of up to 50ns in its proposed configuration. By splitting the RAM into two switched banks, used alternately for reading and writing (as in the present RATES design), it would be possible to use memories with a longer access time.

- Any likely practical implementation of this systolic design would probably aim for an integrated approach, based around a gate array or similar device. The propagation delays due to standard cells can be equated more with emitter-coupled logic than conventional TTL, and hence greater throughput could probably be achieved simply by implementing on silicon rather than using discrete devices.

- It should also be added that simulations have been performed using times based on those quoted by the manufacturers, with some bias towards the worst-case figures.

To conclude, the simulation results have proved the viability of a systolic plot extractor, based around the existing RATES modules but using new architectures offering an improved performance and increased throughput over the previous design. The use of standard "off-the-shelf" components has left considerable scope for improving the timing characteristics of the design through use of faster equivalents or a VLSI-based custom (or semi-custom) design. The availability of low-cost gate arrays should render this latter solution the prime target for the next phase of RATES development, based around the proposed systolic CFAR and "WBI Slice" modules. To quote Inmos: "*Such concurrent systems unlock the processing potential of VLSI for fifth generation applications...*".

8.6.1 A Note on Other Systolic Architectures

Although it was initially stated that this work was only intended as a theoretical discussion, there have been a number of important developments since (a) the start of research into systolic array designs and (b) the paper "The Application of Systolic Arrays to Radar Signal Processing" was written. Most importantly a family of high-speed processors directed specifically at systolic or array-processor applications has become generally available from Inmos Ltd. The **Inmos Transputer range** (of which an example is described below) are well suited to a systolic design, but are not considered sufficiently cost-effective for the proposed designs. Although they represent an acceptable price/performance ratio, they offer a much higher processing capacity than is required of conventional systolic elements. As such, they could only be used to advantage if a number of operations were merged into single processor. The use of the Transputer has not been included in the range of simulations owing to its complexity.

Another device worthy of further study is the **NCR45CG72 Geometric Arithmetic Parallel Processor**, which is described as a "CMOS systolic array with 72 processors per chip", but uses a SIMD architecture with single-bit data path. This is clearly unsuitable for a function such as the CFAR processor, where a byte-wide data path is essential and the use of multiple devices is to be avoided, however it may be more applicable to later stages of plot extraction where binary target data is the norm.

8.6.1.1 Summary of Transputer Features

The **Inmos Transputer** family is based a 1.5 micron CMOS architecture with a 16- or 32-bit data bus and typical minimum execution speed of 10 MIPS. The first product in the family was the **IMS T414**, a 32-bit processor which can best be summarised by **Figure 8.24**.

In common with the other members of the family, the T414 offers a reduced instruction set (RISC) architecture and its own specialised programming language (OCCAM). The language compiles directly to a highly optimised machine code which takes advantage of the features of the processor, but has recently come in for some criticism due to some deficiencies when compared with other high-level languages. As a result there are now several other languages available for the transputer, including 'C' and PASCAL.

The essential hardware features of the transputers, which are compatible at both pin and software levels, are as follows:

- 2K bytes of internal RAM, accessible in 50ns across a 32-bit bus.
- External memory expansion to 4G bytes with 150ns access time. Fully programmable, on-chip DRAM controller.
- Full DMA and interrupt driven interfacing.
- Four 10M bits / sec serial interfaces, supporting full-duplex communication between transputers.

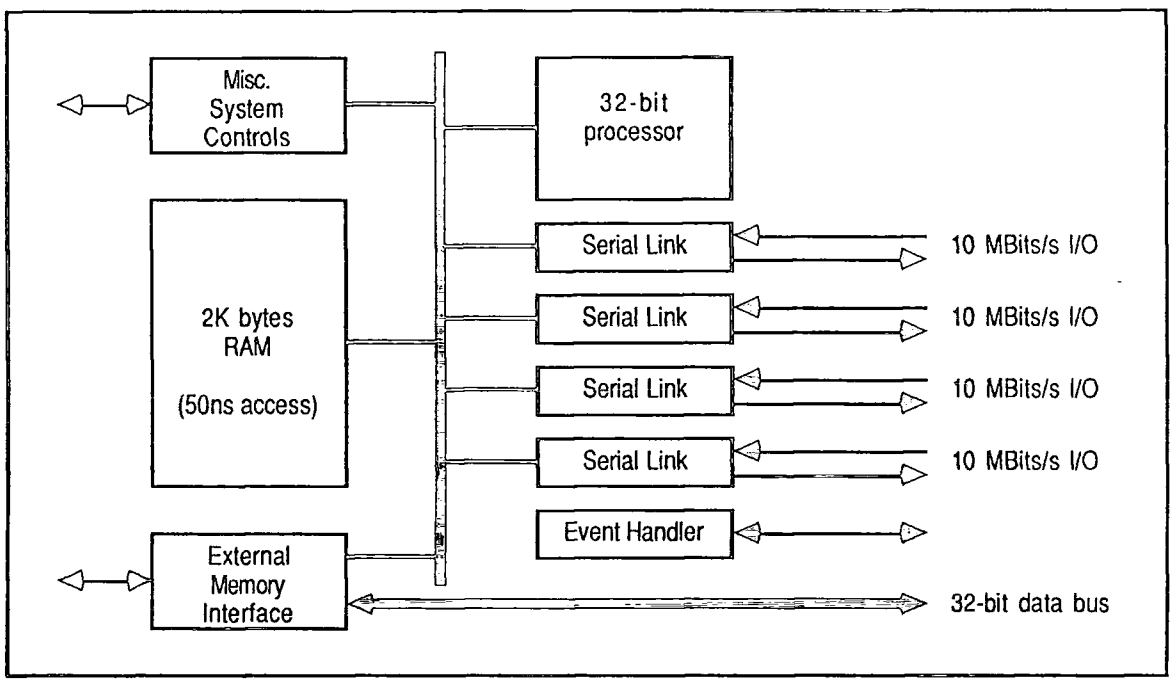


Figure 8.24
Architecture of the T414 Transputer

The ability to communicate at high-speed with other transputers is one of the key features which would permit a systolic array processor to be designed using these devices. Without this feature, the processors are less attractive and the T414 in particular exhibits a poorer performance that existing microprocessors. Transputer development, however, is ongoing and the current top-of-the-range T800 shows a greatly improved performance. This device features a hardware floating point capability which outperforms the Motorola 68020/68881 pair, provided that the true transputer-array concept is used to advantage.

In common with other major semiconductor manufacturers, Inmos have recently released a further addition to their product line - the IMS A100 digital signal processor. Primarily intended for applications which rely on high-speed multipliers, the A100 offers an array of 32 16x16-bit multiplier accumulators and may be cascaded with other DSPs to increase its performance. Whilst not appropriate to the architectures discussed in this research, the A100 offers a performance level which may be suited to future design studies.

9.0 Research Objectives

The research work involved in the production of this thesis has concentrated on a little-publicised area of radar technology, namely post-detector signal processing. To date there has been a large amount of time and effort devoted to research into new transmission and reception techniques at the expense of further work in signal processing. Design methods used in this area have failed to keep pace with advances in semiconductor technology and a number of commercial designs have failed as a result (e.g. Nimrod).

The initial aim of this research was to establish the possibility of a single-board implementation of a radar processing, filtering and tracking unit, using custom-designed integrated circuits. The system would be based around a prototype surveillance radar plot extractor with substantial modifications to take account of more modern semiconductor technology. Continuously-rotating surveillance radars are still used widely in both military and commercial installations, in spite of predictions that they would be replaced by phased-arrays, as they are relatively cheap (compared with phased-array antennæ) and have lesser signal processing demands. They can be initially interpreted using square-law or MTI detectors, although the former approach is the more attractive for a low-cost radar implementation. The post-detector radar video and synchronisation signals would be used as input to the plot extractor, whilst the output would be a series of plots (time histories of established targets). To achieve this conversion the input signal would be filtered through a double-threshold detector, weighted to establish the absolute co-ordinates of a plot then passed into a computer to be associated with previous plots.

In the course of this work, the shortcomings of the prototype system would have to be established and suitable solutions designed and tested. The important feature of these requirements would not be the replacement of small isolated modules with VLSI equivalents, but a detailed study of the functions, with a view to establishing an improved design based on a different architecture.

The research which resulted from these initial requirements can be divided into a series of specific phases:

- Familiarisation with the prototype system
- Construction of a duplicate hardware system
- Study of computer processing requirements
- Re-design of computer and interface hardware
- Programming of test software
- Study of individual hardware functions, with a view to replacement using:
  - DSP techniques
  - VLSI semi-custom designs
  - Systolic processor arrays

All of the above phases have been achieved (to a greater or lesser extent) and have been fully documented in this thesis. Conclusions have been drawn at each point, and have been discussed in detail where appropriate. They are summarised in the section which follows.

9.1 Summary and Conclusions

A study of the prototype plot extractor revealed a number of deficiencies in both hardware and software. One of the main disadvantages of the system was the requirement for an oversized computer tracking sytem which had clearly failed to keep pace with modern semiconductor technology. The plot extractor hardware also suffered in a similar fashion and, being based around SSI and MSI logic, required large amounts of power and board space. It was clearly important to concentrate on removing redundant modules from the design and improve the performance of the tracking computer.

The Durham RATES system was constructed from a duplicate set of plot extractor boards, with the computer and its interface omitted. Following investigations into suitable alternative processors, the Motorola 68000 was adopted as the replacement for the original tracking computer, and a number of interfaces designed and tested. A small software suite was developed, partly emulating the user and hardware interfaces of the prototype and allowing the performance of the new system to be assessed.

This design addressed many of the problems in the prototype, concentrating in particular on the outdated performance of the tracking computer and its associated interface. As a result of the new design, clutter cancellation is more flexible than it was before and could be used to advantage if the CFAR processor were to be enhanced. Plot forming too benefits from a direct interface to the filtering and tracking computer, and earlier problems of 'lock-up' due to input overloads have now been remedied.

Tests of the new interface have proved highly successful, showing a dramatic improvement in a number of areas:

- Considerable simplification of interfacing hardware, with a reduction from **36** boards in the prototype (plus computer) to **29** in the development system (*including* a single-board computer).

- Increased number of input plots per octant to the tracking computer

- Increased software reliability, achieved through greater control of the interface and system timing.

- Faster execution of software due to:
  - Faster instruction times
  - Improved program structure (suggested use of assembly language 'drivers' & high-level application software).
  - Less time involved in input/output-bound tasks due to efficiency of modern DMA controllers.

It is concluded that the replacement of the tracking computer should be a high priority in a redesigned system and that the use of code-compatible microprocessors, such as the Motorola 680x0 family, is strongly recommended to assist future upgrade paths. It is also suggested that a different bus structure be adopted to allow for standardisation to a recommended design. A possible candidate would be the VME bus for which a large number of 68000...68020 (& 030) -based systems are available.

Studies of alternative designs for the plot extractor hardware modules have not resulted in any successful practical implementations. The use of a standard "CFAR - > WBI - > Plot Forming" approach has been retained although there have been studies of alternative designs for the 3 modules.

The original brief to investigate the use of custom logic in automatic radar track extraction systems has been rejected due to the high costs involved in a design which simply duplicates the existing system in a VLSI form. The poor reliability of the custom VLSI designs (none of the custom-designed devices worked correctly) combined with the difficulty in producing a perfect chip from early manufacturing runs has meant that this approach has not be found to be viable.

It is suggested that the use of semi-custom logic, such as gate arrays (ULAs, ASICs) is now a more cost effective approach because of the improvements in on-site design methods (typically using IBM PCs and compatibles). The gate complexity of these devices has increased considerably in recent years and it is now also possible to obtain ULAs which can be erased and re-programmed as required. Design tools allow the programmer to work directly from TTL-based prototypes and it would thereforebe possible to produce an initial design with a minimum of cost and effort.

Software programmable devices have also been discussed with a view to their use in the plot extractor hardware. It has been concluded that current devices do not exhibit a fast enough performance for an implementation involving single processors connected serially, although there is potential for future fourth- and fifth-generation digital signal processors in this area.

9.2 Recommendations for Further Work

The failure of the Nimrod project, which used similar technology to that of the RATES prototype, has led to a dramatic change in MOD policies. These now require the use of 'off-the-shelf' computer hardware, in place of custom designs and the software itself is being passed onto software houses for implementation, thereby decreasing both the design time and the risk involved in a project. Nimrod is not the only system to be let down by its computer technology - the Tornado's Foxhunter radar has suffered similar problems, and is unable to keep track of enemy aircraft whilst filtering out unwanted signals.

A report issued by the House of Commons Defence Committee ("Implementing the Lessons of the Falklands Campaign") referred to a document issued by the US Navy, which concluded that "the British were hampered by a lack of modern radars, target identification systems, data management systems and electronic warfare equipment in their fleet."

These reports clearly promote the further development of radar systems, particularly in military situations. The general-purpose surveillance radar is likely to be an important component in future installations, particularly on grounds of cost, but only if there are improvements in current signal processing techniques and new technology is used to the full.

The Durham RATES system has shown the potential for a dramatic reduction in the size of tracking / filtering computers simply through use of modern microprocessors and it is likely that further improvements would be achieved if state-of-the-art components were used (e.g the Motorola 68020 or 030 and 68881 / 2). Further studies are also recommended in the area of VLSI semi-custom designs. With more manufacturers offering production facilities for these components, and increasing availability of workstation-based design tools there is now little reason for avoiding their use.

The use of systolic architectures is predicted to be a major development area in the future, particularly as awareness of neural computing methods increases. The emergence of a number of "systolic processor array I.C.s", together with the increasing popularity of the Inmos Transputer family confirms these observations and it is therefore considered that systolic architectures should not be overlooked. The potential of systolic arrays for easy VLSI implementation suggests that this approach may represent the ultimate solution to a single-chip plot extractor.

As a result of research at Durham, a major U.K. electronics company has taken an interest in the re-designed RATES system, and presentations have already been given to potential customers who represented the Navy of a NATO country.

Further enhancements have been suggested, particularly applicable to the military environment, including development of the ECM module or the introduction of some form of "Identification Friend or Foe" (IFF) system. This latter feature has also been recommended as an essential feature of modern radar systems in the House of Commons Defence Committee report on the Falklands Campaign.

To summarise, it is considered essential that the RATES design is developed further, as it represents an ideal low-cost solution to a general purpose surveillance radar. The functions it achieves are not restricted to military applications, and with further modification it could easily be used for commercial designs such as collision-avoidance systems, or enhanced navigational radars.

APPENDIX A

References and Bibliography

[1] HERTZ, H.
"Electric Waves", New York : Dover 1962.

[2] HULSMEYER, C.
"Hertzian-wave projecting and receiving apparatus adapted to indicate or give warning of the presence of a metallic body, such as a ship or train, in the line of projection of such waves", *British Patent no 13170*, 2nd Sept. 1904.

[3] MARCONI, G.
"Radio Telegraphy", *Proc. I.R.E. 1922*, vol 10, p215.

[4] APPLETON, E.
"The Scientific Principles of Radio-location", *J.I.E.E.*, vol. 92, pt. 1, 1945, pp 340-353.

[5] WATSON-WATT, R.
"The Evolution of Radiolocation", *J.I.E.E.*, vol. 93, pt 1, 1946, pp374 - 382.

[6] STEWART WATSON, D.
"A Survey of C.R.T. Problems in Service Applications with special reference to radar", *ibid.*, pt 3A, pp

[7] BURGESS, J.S.
"Recent Advances in Radar Technology", *Proc I.E.E. Radar-77*, p 8.

[8] SMITH, R.A.
"A Survey of the Development of Radar", *J.I.E.E.*, vol. 94, pt 1, 1947, pp 172-178.

[9] GARRATT, G.R.M.
"The Birth of Radar", *Electronic Engineering*, , March 1958, pp 140-142.

[10] DAVIES, D.H., FIELDING, C.C., GIRLING, F.E.J.
"Radar", *Proc I.E.E.*, vol. 118, no 9R, Sept. 1971, pp1071-1089.

[11] RANDALL, J.T., BOOT, H.A.H.
"Early Work on the Cavity Magnetron", *J.I.E.E.*, vol 93, pt 3A, 1946, p182.

[12] PAGE, R.M.
"1977 Pioneer Award, Monostatic Radar", *ibid.* p 557.

[13] SKOLNIK, M.I.
"Fifty Years of Radar", *Proc I.E.E.E.*, vol 73, no 2, Feb. 1985, pp 182-197.

[14] WILLIAMS, F.C.
"Introduction to Circuit Techniques for Radiolocation", *J.I.E.E.*,, vol 93, pt 3, 1946, p 289.

[15] EARLE BAILEY, A.
"MTI Research at ADRDE", *I.E.E.E. Radar '85 Conference Record Supplement*, pp S-16 - S-18.

[16] BRYSON, Sir L.
"All at Sea", *Proc I.E.E.-A*, vol 133, pt A, No 1, Jan 1986, p1.

[17] CLARKE, J., DAVIES, D.E.N., RADFORD, M.F.
"Review of United Kingdom Radar", *I.E.E.E. Trans. AES*, vol 20, no 5, Sept 1984, pp 506-519.

[18] EASTWOOD, Sir E.
"Radar: New Techniques and Applications", *Proc Royal Society London*, Ser A, vol 354, no 1677, 1977, p137.

[19] CAPENTIER, M.H.
"Radar: Yesterday, Today & Tomorrow", *Proc International Conference on Radar '78*, p1.

[20] HALL, J.S. (ed)
"Radar Aids to Navigation", *MIT Radiation Laboratory Series*, vol 2, McGraw Hill Book Company, 1947.

[21] EMERSON, R.C.
"Some Pulsed Doppler, MTI and AMTI Techniques", *Rand Corporation Technical Report*, No R-274, Mar. 1954.

[22] EMERY, M.
"Extracteur Radar Primaire et Radar Secondaire EV. 660 pour l'Aviation Civile", *L'Onde Electrique*, vol 48, no 491, Feb 1968, p 118.

[23] POINSARD, H., & GENDREU, R.
"Système de Détection d'Echoes à Fausses Alarmes Constantes", *Revue Technique Thomson-CSF*, vol 1, no 3, Sept. 1969.

[24] REBOULOT, M.
"Extracteur Microprogrammé d'Infor-mation Radar", *L'Onde Electrique*, vol 48, no 491, Feb. 1968, p138.

[25] CAHEN, R. & BERARD, H.
"La Présentation d'Images Radar en Télévision", *L'Onde Electrique*, vol 48, no 491 Feb. 1968, p125.

[26] BATH, W.G. et al
"False Alarm Control in Automated Radar Systems", (*source uncertain*).

[27] WEISS, M.
"Analysis of some Modern Cell-Averaging CFAR Processors in Multiple Target Situations", *I.E.E.E. Transactions AES*, AES-18, Jan. '82, pp 102 - 114.

[28] EBERT, H. & SIEGENTHALER, K.
"Digitale Radarzielextraktoren zur Automatischen Erkennung Fliegender Objekte", *Regelungstechnishe Praxis und Prozeß-Rechentechnik*, vol 14, pt 5, Oct 1972, p 149.

[29] SCHÖNFELD, W.H. & GILLMANN, H.
"Grundlagen für Speicherung von Radarbildern auf Magnetband", *Elektronische Rundschau*, vol 11, pt 6, June 1957, pp 165 - 167.

[30] MILNE, K.
"Principles and Concepts of Multistatic Surveillance Radar", *Proc. I.E.E.E. Radar '77*,

[31] EBERT, H.
"Radar Data Processing Using Microcomputers", *ibid*, p 90 et seq.

[32] KIME, F.W.
"Evolution of the Man-Machine Interface in Surveillance Radar Systems", *(source uncertain)*.

[33] TUNNICLIFFE, R.J.
"A Simple Automatic Radar Track Extraction System", *Proc. I.E.E.E. Radar '77*, pp 76 - 80.

[34] MASCOT SUPPLIERS ASSOCIATION.
"The Official Handbook of MASCOT", *MASCOT Suppliers Association*, Computing Standards Section, RSRE, Malvern, Dec. 1980.

[35] MILES, J.A.H. & SHEPHERD, A.
"Radar Automatic Track Extraction System (RATES), Software Description and Source Code Listings", ARE Internal Memorandum XCC 82011, Oct. 1982.

[36] SKOLNIK, M.I.
"Radar Handbook", MacGraw Hill, 1970.

[37] DILLARD, G.M.
"A Moving Window Detector for Binary Integration", *I.E.E.E. Transactions IT*, IT-13, Jan. 1967, pp 2 - 10.

[38] HANSEN, V.G. & SAWYERS, J.H.
"Detectability Loss due to Greatest-Of Selection in a Cell Averaging CFAR", *I.E.E.E. TransactionsAES*, AES-16, Jan. 1980, pp 115 - 118.

[39] MOORE, J.D. & LAWRENCE, N.B.
"Comparison of two CFAR Methods used with Square-Law Detection of Swerling 1 Targets", *Proc. I.E.E.E. Radar '80*, 1980.

[40] ROHLING, H.
"Radar CFAR Thresholding in Clutter and Multiple Target Situations", *I.E.E.E. Transactions AES*, AES-19, July 1983, p 608.

[41] ROHLING, H.
"New CFAR Processor Based on an Ordered Statistic", *Proc. I.E.E.E. Radar '85*, May 1985, pp 271 - 275.

[42] MAO, Y.H., ZHOU, Z.C., MENG, X.Y.
XONG, F.Q., ZHANG, S.Y.
"A Non-Parametric CFAR Detector Implemented with CCD Tapped Delay Line" *Proc. I.E.E.E. Radar '85*, May 1985, pp 430 - 434.

[43] VOGEL, L.E., REID, W.S. & STEICHEN, P.E.
"An Examination of Radar Signal Processing via Non-Parametric Techniques" *Proc. I.E.E.E. Radar '75*, April 1975, pp 533 - 537.

[44] DILLARD, G.M. & ANTONIAK, C.E.
"A Practical Distribution-free Detection Procedure for Multiple Range-bin Radars", *I.E.E.E. Transactions AES*, vol. AES-6, No 5, Sept. 1970, pp 629 - 635.

[45] WALKER, J.F.
"Performance Data for a Double Threshold Radar Detector", *I.E.E.E. Transactions AES*, AES-7, Jan. 1971, pp 142 - 146.

[46] SWERLING, P.
"The 'Double Threshold' Method of Detection", *RAND Corp. Report RM1008*, Dec. 1952.

[47] MARCOZ, F. & GALATI, G.
"A Suboptimal Detection Technique: the Accumulator Detector", *Alta Frequenza*, vol. XLI, Feb. 1972.

[48] SPEARMAN, R., SPRACKLEN, C.T. & MILES, J.H.
"The Application of Systolic Arrays to Radar Signal Processing", *Proc. I.E.E.E. Radar '86*, May 1986, pp 65 - 70.

[49] QUIGLEY, A.L.C. & HOLMES, J.E.
"The Development of Algorithms for the Formation and Updating of Tracks", *ARE Internal Memorandum WP-XBC-7512*, Nov. 1975.

[50] RABINOWITZ, S.J., GAGER, C.H.,
BROOKNER, E. et al
"Applications of Digital Technology to Radar", *Proc. I.E.E.E.*, Vol. 73, No 2, Feb. 1985, pp 325 - 339.

[51] FERRANTI COMPUTER SYSTEMS
"Final Report of a Study on the Application of LSI/VLSI Techniques to Radar Plot Extraction Systems, Part A", *Ferranti Report 4920*, Issue 2, Nov. '83.

[52] POLISE, A.T. & MOSKOVITZ, G.
"A Real-Time Application of Microprocessors to a Radar System", *Proc. I.E.E.E. Southeastern Conference*, April 1981, pp 26 - 31.

[53] LOPPNOW, D.H.
"Microcomputer Hardware for Radar Signal Processing with Distributed Architecture", *Proc. I.E.E.E. Radar '85*, May 1985, pp 184 - 189.

[54] HOLMES, J.E.
"Automatic Track Initiation", *ARE Internal Memorandum WP-XBC-7505*, June 1975.

[55] COWAN, D.
"Boost μP-board Memory Capacity with Simple Hardware Changes", *Electronic Design*, Oct. 1981, pp 197 - 198.

[56] KUNG, H.T.
"Why Systolic Architectures?", *I.E.E.E. Computer*, Jan. 1982, pp 37 - 46.

[57] KUNG, H.T. & LEISERSON, C.E.
"Algorithms for VLSI Processor Arrays", *in "Introduction to VLSI Systems", ed Mead & Conway*, Addison-Wesley.

[58] KUNG, S.Y.
"VLSI Array Processors", *I.E.E.E. ASSP Magazine*, July 1985, pp4 - 22.

[59] UMEO, H.
"A Class of SIMD Machines Simulated by Systolic VLSI Arrays", *in "VLSI: Algorithms and Architectures", ed Bertolazzi & Luccio*, North-Holland.

APPENDIX B

Backplane Connections

for the

Motorola KDM Development Board

Pin		Pin	
A	+5 V	1	+5 V
B	+5 V	2	+5 V
C	+5 V	3	+5 V
D	~IRQ 2	4	~HALT
E	~IRQ 7	5	~RESET
F	VMA	6	~WRITE
H	GND	7	~VPA
J	ENABLE	8	GND
K	GND	9	GND
L	MEM CLK	10	VMALDS
M	-12 V	11	-12 V
N	~BR	12	~REFREQ
P	~BG	13	REFGNT
R	~BGACK	14	~IACK
S	~LDS	15	~UDS
T	+12 V	16	+12 V
U	STANDBY	17	STANDBY
V	PWRFAIL	18	CLOCK
W	~BERR	19	VMAUDS
X	GND	20	GND
Y	GND	21	GND
Z	GND	22	GND
A'	~AS	23	~DTACK
B'	GND	24	GND
C'	~D11	25	~D9
D'	~D15	26	~D13
E'	~D10	27	~D8
F'	~D14	28	~D12
H'	~D3	29	~D1
J'	~D7	30	~D5
K'	~D2	31	~D0
L'	~D6	32	~D4
M'	A15	33	A16
N'	A14	34	A13
P'	A11	35	A12
R'	A10	36	A9
S'	A7	37	A8
T'	A6	38	A5
U'	A3	39	A4
V'	A2	40	A1
W'	GND	41	GND
X'	GND	42	GND
Y'	GND	43	GND

~D0...~D15 - (Bi-directional). Inverted data bus.

A1...A16 - (Bi-directional). Address bus. There is no A0 as upper and lower bytes are selected by separate strobe signals.

~WRITE - (Bi-directional). Defines the direction for a data transfer. Sense is not reversed for external bus master control.

~UDS, ~LDS - (Bi-directional). Upper and lower data strobes, control the transfer of data from upper and lower bytes in a word.

~AS- (Bi-directional). Address strobe, indicates a valid address on the bus.

~DTACK - (Bi-directional). Due to the asynchronous bus architecture of the 68000, this is needed to signify that a data transfer operation is complete. In a processor read, an active DTACK latches the data, whilst an active signal during a write will terminate the cycle.

~BR - (input). A request from another device to gain control of the bus.

~BG - (output). Processor confirmation that the bus will be released at the end of the current cycle.

~BGACK - (input). Confirmation that an external device holds control of the bus. The signal cannot be activated until four conditions are met:
1: Bus grant has been received.
2: Address strobe is inactive. (processor not using the bus).
3: DTACK is inactive (peripherals not using bus).
4: BGACK is inactive (no other device has claimed control).

~BERR - (input). Alerts the processor to an error in the current bus cycle.

~RESET - (Bi-directional). Indicates a software reset instruction or hardware external reset.

~HALT - (Bi-directional). As an input causes the processor to stop execution at the end of the current cycle. An output indicates a failure due to double bus fault or similar.

MEM CLK, CLOCK, ENABLE - (Bi-directional). The standard 6800 compatible enable signal, lasting for 10 68000 clock periods.

~VPA - (input). Signal to the processor that it is addressing a 6800 peripheral.

~VMA - (Bi-directional). Processor confirmation that a 6800 address is on the bus, and it is responding to ENABLE.

~IRQ2, IRQ7 - (input). Interrupt requests to the processor.

APPENDIX C

Source Code Listing
of
RATES Development Software

```
0000
0000                                    ;
0000                                     ;  "RADAR AUTOMATIC TRACK EXTRACTION SYSTEM" (RATES)
0000                                    ;  DURHAM VERSION
0000                                    ;  SOURCE CODE
0000                                    ;
0000                                    ;
0000                                    ;  PROCESSOR :       MOTOROLA 68000
0000                                    ;
0000                                    ;  FILE :            RATES_34.ASM
0000                                    ;  DATE :            9th Mar. '85
0000                                    ;  PROGRAMMER :      Richard Spearman
0000                                    ;
0000                                    ;  RELEASE VERSION :                3.4
0000                                    ;
0000                                     ;  This version implements basic plot filtering, status VDU control
0000                                    ;  and command input.
0000
0000
0000                                     ;  68000 predefined address space
0000
0068                                    IRQ2       EQU       $00068          ; Interrupt vector 2
006C                                    IRQ3       EQU       $0006c          ; Interrupt vector 3
0070                                    IRQ4       EQU       $00070          ; Interrupt vector 4
0074                                    IRQ5       EQU       $00074          ; Interrupt vector 5
0078                                    IRQ6       EQU       $00078          ; Interrupt vector 6
0080                                    TRAP0      EQU       $00080          ; Trap 0, used to emulate north IRQ.
0000
0000                                     ;  KDM board hardware addresses
0000
0003FF01                                Terminal   EQU       $3ff01          ; Address of console port
0003FF21                                Port       EQU       $3ff21          ; Address of second serial port
000200F6                                Macsbug    EQU       $200f6          ; Warm boot to monitor ROM
0000
0000                                     ;  RATES memory-mapped peripheral addresses
0000
00030000                                Cfar       EQU       $30000          ; CFAR threshold values (W)
00032000                                Synth      EQU       $32000          ; Synthetics characters (W)
00032000                                Ecm        EQU       $32000          ; ECM data (R)
00033001                                VduL       EQU       $33001          ; Status VDU, left hand side (R/W)
00033000                                VduR       EQU       $33000          ; Status VDU, right hand side (R/W)
00034000                                RollX      EQU       $34000          ; Rollball X co-ordinate (R)
00034002                                RollY      EQU       $34002          ; Rollball Y co-ordinate (R)
00035000                                Constants  EQU       $35000          ; Plot extractor constants (R/W)
00035002                                PlotCount  EQU       $35002          ; Count of plot input words (R)
00036000                                TTCoords   EQU       $36000          ; Test Targets output address (W)
00036000                                ShipMotn   EQU       $36000          ; Ships motion data (R)
0000
00038000                                DmaChan0   EQU       $38000          ; DMA Channel 0, base address
00038040                                DmaChan1   EQU       $38040          ; DMA Channel 1, base address
00038080                                DmaChan2   EQU       $38080          ; DMA Channel 2, base address
000380C0                                DmaChan3   EQU       $380c0          ; DMA Channel 3, base address
0000
0000                                     ;  Miscellaneous other equate values
0000
0020                                    BlokSize   EQU       32              ; Most efficient transfer block size
0000
0000
0000
0000                                     ;  Program base address
0000
0000    4EC0 xxxx              (PX)     Begin      BRA       StartUp
0004
0004                                     ;  Initial program constant declarations
0004
0000                                    Null       EQU       $00
000D                                    CR         EQU       $0d
000A                                    LF         EQU       $0a
0008                                    BS         EQU       $08
0007                                    BEL        EQU       $07
0018                                    CtrlX      EQU       $18
0004
0004    44 75 72 68 61 6D 20 52 41 54 45 53
                                        SysName    DC.B      'Durham RATES'
0010    53 6F 66 74 77 61 72 65 0D 0A
                                                   DC.B      'Software', CR, LF
001A    56 65 72 73 69 6F 6E 20 33 2E 34 0D 0A
                                                   DC.B      'Version 3.4', CR, LF
0027    52 61 64 61 72 20 53 65 6C 65 63 74 65 64 20 3A 20 00
                                        RadMess    DC.B      'Radar Selected : ', Null
0039
0039    08 20 08 00                     BackSp     DC.B      $08, $20, $08, Null
003D    00                                         .ALIGN    2
```

```
003E                                    ; ****  Table of permitted user commands   ****
003E                                    ;
003E                                    Commands
0040    xxxx                (R)                 DC.W        Com1 - Commands
0042    xxxx                (R)                 DC.W        Com2 - Commands
0044    xxxx                (R)                 DC.W        Com3 - Commands
0046    xxxx                (R)                 DC.W        Com4 - Commands
0048    xxxx                (R)                 DC.W        Com5 - Commands
004A    xxxx                (R)                 DC.W        Com6 - Commands
004C    xxxx                (R)                 DC.W        Com7 - Commands
004E    xxxx                (R)                 DC.W        Com8 - Commands
0050    xxxx                (R)                 DC.W        Com9 - Commands
0052    xxxx                (R)                 DC.W        Com10 - Commands
0054    xxxx                (R)                 DC.W        Com11 - Commands
0056    xxxx                (R)                 DC.W        Com12 - Commands
0058    xxxx                (R)                 DC.W        Com13 - Commands
005A    xxxx                (R)                 DC.W        Com14 - Commands
005C    xxxx                (R)                 DC.W        Com15 - Commands
005E    xxxx                (R)                 DC.W        Com16 - Commands
0060    xxxx                (R)                 DC.W        Com17 - Commands
0062    xxxx                (R)                 DC.W        Com18 - Commands
0064    xxxx                (R)                 DC.W        Com19 - Commands
0066    xxxx                (R)                 DC.W        Com20 - Commands
0068    xxxx                (R)                 DC.W        Com21 - Commands
0068
0068    44 43 00                            Com1:   DC.B        'DC', Null
006B    52 55 4E 00                         Com2:   DC.B        'RUN', Null
006F    48 41 4C 54 00                      Com3:   DC.B        'HALT', Null
0074    44 52 50 00                         Com4:   DC.B        'DRP', Null
0078    44 49 50 00                         Com5:   DC.B        'DIP', Null
007C    44 53 50 00                         Com6:   DC.B        'DSP', Null
0080    44 54 54 00                         Com7:   DC.B        'DTT', Null
0084    44 43 54 00                         Com8:   DC.B        'DCT', Null
0088    50 49 50 00                         Com9:   DC.B        'PIP', Null
008C    50 4F 50 00                         Com10:  DC.B        'POP', Null
0090    50 52 54 00                         Com11:  DC.B        'PRT', Null
0094    52 41 44 41 52 00                   Com12:  DC.B        'RADAR', Null
009A    44 50 4C 49 53 54 00                Com13:  DC.B        'DPLIST', Null
00A1    49 50 4C 49 53 54 00                Com14:  DC.B        'IPLIST', Null
00A8    53 45 54 56 41 4C 00                Com15:  DC.B        'SETVAL', Null
00AF    4C 45 56 45 4C 00                   Com16:  DC.B        'LEVEL', Null
00B5    49 4E 4A 45 43 54 00                Com17:  DC.B        'INJECT', Null
00BC    54 54 31 00                         Com18:  DC.B        'TT1', Null
00C0    54 54 32 00                         Com19:  DC.B        'TT2', Null
00C4    54 54 33 00                         Com20:  DC.B        'TT3', Null
00C8    4D 41 43 53 42 55 47 00             Com21:  DC.B        'MACSBUG', Null
00D0                                                .ALIGN      2
00D0
00D0
00D0                                    ;  Prompt strings for setting system parameters or for
00D0                                    ;  display of parameters on the system console.
00D0
00D0                                    ;  Radar Dependent Parameter Strings
00D0
00D0    41 20 20 52 65 76 20 50 65 72 69 6F 64 20 69 6E 20 4D 69 6C 6C 69 73 65 63 73 20 20 20 20 20 00
                                        Rdps        DC.B        'A  Rev Period in Millisecs      ', Null
00F0    42 20 20 52 61 64 61 72 20 52 61 6E 67 65 20 69 6E 20 44 61 74 61 6D 69 6C 65 73 20 20 20 20 00
                                                    DC.B        'B  Radar Range in Datamiles     ', Null
0110    43 20 20 52 61 6E 67 65 20 53 44 20 69 6E 20 46 65 65 74 20 20 20 20 20 20 20 20 20 20 20 20 00
                                                    DC.B        'C  Range SD in Feet             ', Null
0130    44 20 20 42 65 61 72 69 6E 67 20 53 44 20 69 6E 20 43 20 44 65 67 72 65 65 73 20 20 20 20 20 00
                                                    DC.B        'D  Bearing SD in C Degrees      ', Null
0150    45 20 20 52 61 6E 67 65 20 43 65 6C 6C 20 53 69 7A 65 20 69 6E 20 46 65 65 74 20 20 20 20 20 00
                                                    DC.B        'E  Range Cell Size in Feet      ', Null
0170    46 20 20 53 50 46 20 52 61 6E 67 65 20 43 61 70 74 2E 20 53 69 7A 65 20 69 6E 20 46 65 65 74 00
                                                    DC.B        'F  SPF Range Capt. Size in Feet', Null
0190    47 20 20 53 50 46 20 42 65 61 72 69 6E 67 20 43 61 70 74 2E 20 69 6E 20 43 20 44 65 67 73 20 00
                                                    DC.B        'G  SPF Bearing Capt. in C Degs ', Null
01B0    48 20 20 52 61 6E 67 65 20 4D 53 20 42 69 74 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 00
                                                    DC.B        'H  Range MS Bit                ', Null
01D0
01D0                                    ;  Radar Independent Parameter Strings
01D0
01D0    49 20 20 49 6E 69 74 69 61 6C 20 53 50 46 20 43 6F 6E 66 69 64 65 6E 63 65 20 20 20 20 20 20 00
                                        Rips        DC.B        'I  Initial SPF Confidence       ', Null
01F0    4A 20 20 53 50 46 20 43 6F 6E 66 69 64 65 6E 63 65 20 49 6E 63 72 65 6D 65 6E 74 20 20 20 20 00
                                                    DC.B        'J  SPF Confidence Increment     ', Null
0210    4B 20 20 4D 61 78 69 6D 75 6D 20 53 50 46 20 43 6F 6E 66 69 64 65 6E 63 65 20 20 20 20 20 20 00
                                                    DC.B        'K  Maximum SPF Confidence       ', Null
0230    4C 20 20 43 65 6E 74 69 20 53 50 46 20 53 6D 6F 6F 74 68 69 6E 67 20 46 61 63 74 6F 72 20 20 00
                                                    DC.B        'L  Centi SPF Smoothing Factor   ', Null
0250    4D 20 20 57 61 72 6D 2D 75 70 20 52 65 76 73 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 00
                                                    DC.B        'M  Warm-up Revs                 ', Null
```

```
0270    4E 20 20 43 65 6E 74 69 20 4E 6F 69 73 65 20 54 68 72 65 73 68 6F 6C 64 20 20 20 20 20 20 20 00
                                        DC.B        'N  Centi Noise Threshold        ', Null
0290    4F 20 20 43 65 6E 74 69 20 4D 61 6E 6F 65 75 76 72 65 20 54 68 72 65 73 68 6F 6C 64 20 20 20 00
                                        DC.B        'O  Centi Manoeuvre Threshold    ', Null
02B0    50 20 20 43 65 6E 74 69 20 4D 61 6E 6F 65 75 76 72 65 20 44 65 63 61 79 20 46 61 63 74 6F 72 00
                                        DC.B        'P  Centi Manoeuvre Decay Factor', Null
02D0    51 20 20 4D 61 78 20 4B 6E 6F 74 73 20 41 6C 6C 6F 77 65 64 20 20 20 20 20 20 20 20 20 20 20 00
                                        DC.B        'Q  Max Knots Allowed            ', Null
02F0    52 20 20 4D 61 78 20 42 65 61 72 69 6E 67 20 44 69 66 66 2E 20 69 6E 20 43 20 44 65 67 73 20 00
                                        DC.B        'R  Max Bearing Diff. in C Degs  ', Null
0310    53 20 20 47 53 20 70 65 72 20 31 30 30 30 20 4B 6E 6F 74 73 20 20 20 20 20 20 20 20 20 20 20 00
                                        DC.B        'S  GS per 1000 Knots            ', Null
0330    54 20 20 49 6E 69 74 69 61 6C 20 54 72 61 63 6B 20 43 6F 6E 66 69 64 65 6E 63 65 20 20 20 20 00
                                        DC.B        'T  Initial Track Confidence     ', Null
0350    55 20 20 43 6F 6E 66 69 72 6D 65 64 20 54 72 61 63 6B 20 43 6F 6E 66 69 64 65 6E 63 65 20 20 00
                                        DC.B        'U  Confirmed Track Confidence   ', Null
0370    56 20 20 4D 61 78 20 4E 75 6D 62 65 72 20 6F 66 20 4D 69 73 73 65 64 20 4C 6F 6F 6B 73 20 20 00
                                        DC.B        'V  Max Number of Missed Looks   ', Null
0390    57 20 20 53 68 69 70 73 20 4D 6F 74 69 6F 6E 20 43 6F 72 72 65 63 74 69 6F 6E 20 20 20 20 20 00
                                        DC.B        'W  Ships Motion Correction      ', Null
03B0    58 20 20 4C 6F 77 65 72 20 54 68 72 65 73 68 2E 20 43 6C 75 74 74 65 72 20 43 6F 75 6E 74 20 00
                                        DC.B        'X  Lower Thresh. Clutter Count  ', Null
03D0    59 20 20 55 70 70 65 72 20 54 68 72 65 73 68 2E 20 43 6C 75 74 74 65 72 20 43 6F 75 6E 74 20 00
                                        DC.B        'Y  Upper Thresh. Clutter Count  ', Null
03F0    5A 20 20 53 70 61 72 65 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 00
                                        DC.B        'Z  Spare                        ', Null
0410                                    .ALIGN      2
0410
0410
0410                                    ;  Initialisation and Default Parameters
0410
0410                                    ;  Radar Dependent Variables
0410
0410    0FA0 005A 00C9 0012 03D8 0514 00B4 0003
                                    RdpDef  DC.W        4000, 90, 201, 18, 984, 1300, 180, 3     ; 992
0420    09C4 002A 0032 001E 00F6 04B0 00C8 0003
                                            DC.W        2500, 42, 50, 30, 246, 1200, 200, 3      ;
0430    09C4 002A 00C9 001E 03D8 0514 00C8 0004
                                            DC.W        2500, 42, 201, 30, 984, 1300, 200, 4     ;
0440    09C4 001E 001E 000F 007B 0258 0096 0002
                                            DC.W        2500, 30, 30, 15, 123, 600, 150, 2       ;
0450    09C4 001E 005A 000F 0171 0514 0096 0004
                                            DC.W        2500, 30, 90, 15, 369, 1300, 150, 4      ;
0460    1F40 00FC 005A 0022 0171 0514 00C8 0001
                                            DC.W        8000, 252, 90, 34, 369, 1300, 200, 1     ; 1006
0470
0470    0031 0063 0032 0045 0061 0015
                                    MaxRaNum DC.W       $31, $63, $32, $45, $61, $15; Max range nos.
047C
047C    0004 0001 0002 0001 0001 0008
                                    RangeFac DC.W       $4, $1, $2, $1, $1, $8              ; Range factors
0488
0488                                    ;  Radar Independent Variables
0488
0488    0005 0005 000F 001E             RipDef  DC.W        5, 5, 15, 30
0490    0005 012C 012C 001E                     DC.W        5, 300, 300, 30
0498    09C4 01F4 0007 0004                     DC.W        2500, 500, 7, 4
04A0    0008 0006 0008 0000                     DC.W        8, 6, 8, 0
04A8    0001 0000 0000                          DC.W        1, 0, 0
04AE
04AE                                    ;  Status VDU Initialisation Values
04AE
04AE    0000 0000 0034 0024             VDUITab DC.W        $00, $00, $34, $24
04B6    0001 0001 0075 0075                     DC.W        $01, $01, $75, $75
04BE    0002 0002 0003 0003                     DC.W        $02, $02, $03, $03
04C6    003F 003F 0004 0004                     DC.W        $3f, $3f, $04, $04
04CE    0007 0007 0005 0005                     DC.W        $07, $07, $05, $05
04D6    0006 0006 0000 0000                     DC.W        $06, $06, $00, $00
04DE
04DE                                    ;  Status VDU template
04DE
04DE    xxxx xxxx xxxx          (R)     VDUTemp DCB.W       640, 0
09DE
09DE
09DE
09DE
```

```
09DE                                ; ***************************************************************
09DE                               ; *                                                             *
09DE                                ; *  MAIN PROGRAM DATA AREA - SHOULD RESIDE IN RAM              *
09DE                               ; *                                                             *
09DE                                ; ***************************************************************
09DE
09DE      0000 0000                 BuffPtr     DC.L    0           ; Address of current prompt
09E2      xx xx xx xx xx xx   (R)   InpBuff     DCB.B   82, 0       ; Console input buffer
0A34      xx xx xx xx xx xx   (R)   TmpBuff     DCB.B   82, 0       ; Temporary buffer area
0A86
0A86                                ; ****  Main incoming / outgoing data areas                 ****
0A86                               ;
0A86      xx xx xx xx xx xx   (R)   CellBlock   DCB.B   4096, 0     ; Clutter cells
1A86      xxxx xxxx xxxx      (R)   ThreshVal   DCB.W   4096, 0     ; Threshold values
3A86      xxxx xxxx xxxx      (R)   SynthVal    DCB.W   512, 0      ; Synthetics character codes
3E86      xx xx xx xx xx xx   (R)   VDUMap      DCB.B   640, 0      ; RAM-based VDU template
4106      0000 0000                 Base        DC.L    0           ; Current input plot base adr.
410A      xxxx xxxx           (R)   FirstIPA    DCB.L   2048, 0     ; Incoming plot word storage
610A
610A                                ; ****  System Commands Table - #$80 if on, #$00 if off     ****
610A                               ;
610A                               SysCommand
610A      00                        StopRun:    DC.B    0           ; Stop or run command
610B      00                        DispUnsp:   DC.B    0           ; Display unassociated plots
610C      00                        DispIpc:    DC.B    0           ; Display input plots
610D      00                        DispStat:   DC.B    0           ; Display stationary plots
610E      00                        DispTent:   DC.B    0           ; Display tentative tracks
610F      00                        DispConf:   DC.B    0           ; Display confirmed tracks
6110      00                        PlotIP:     DC.B    0           ; Plot input plots
6111      00                        PlotOP:     DC.B    0           ; Plot output plots
6112      00                        PrintTrks:  DC.B    0           ; Print tracks command
6113      00                       Thresh      DC.B    0           ; Threshold command
6114      0000                     ThreshHol   DC.W    0           ; Threshold hole size
6116      0000                     ThreshLev   DC.W    0           ; Threshold level
6118      00                       ManInit     DC.B    0           ; Manual initiation command
6119
6119                               TestTarget
6119      00                        TestNum:    DC.B    0           ; Test target number
611A      0000                      TestRange:  DC.W    0           ; Test target range
611C      0000                      TestBear:   DC.W    0           ; Test target bearing
611E      0000                      TestSpeed:  DC.W    0           ; Test target speed
6120      0000                      TestHead:   DC.W    0           ; Test target heading
6122
6122      0000                     RadarType   DC.W    0           ; Radar type code
6124      0000                     ConstWord   DC.W    0           ; 'Old' constants word
6126
6126
6126                                ; ****  Integer system parameters                                ****
6126                               ;
6126
6126      0000                     RevPermi    DC.W    0           ; Rev period in milliseconds
6128      0000                      RadRanDM:   DC.W    0           ; Radar range in datamiles
612A      0000                      RangeSDFt:  DC.W    0           ; Radar range SD in feet
612C      0000                      BearSDCd:   DC.W    0           ; Bearing SD in C degrees
612E      0000                      RangeCSFt:  DC.W    0           ; Range cell size in feet
6130      0000                      SPFRCSFt:   DC.W    0           ; SPF range capture size in feet
6132      0000                      SPFBCSCd:   DC.W    0           ; SPF bearing capture size in C degrees
6134      0000                      RangeMSBT:  DC.W    0           ; Range ms bit
6136      0000                      InitSPFC:   DC.W    0           ; Initial SPF confidence
6138      0000                      SPFConInc:  DC.W    0           ; SPF confidence increment
613A      0000                      MaxSPFCon:  DC.W    0           ; Maximum SPF confidence
613C      0000                      CenSPFSmo:  DC.W    0           ; Centi SPF smoothing factor
613E      0000                      WarmRevs:   DC.W    0           ; Warm up revs
6140      0000                      CenNoise:   DC.W    0           ; Centi noise threshold
6142      0000                      CenManThr:  DC.W    0           ; Centi manoeuvre threshold
6144      0000                      CenMandy:   DC.W    0           ; Centi manoeuvre decay factor
6146      0000                      MaxKnots:   DC.W    0           ; Max knots allowed
6148      0000                      MaxBDC:     DC.W    0           ; Max bearing difference
614A      0000                      GSPerKK:    DC.W    0           ; GS per 1000 knots
614C      0000                      InitTRL:    DC.W    0           ; Initial track likelihood
614E      0000                      ConfTRL:    DC.W    0           ; Confirmed track likelihood
6150      0000                      MaxMissed:  DC.W    0           ; Max number of missed looks
6152      0000                      ShipMotn:   DC.W    0           ; Ships motion compensation
6154      0000                      LowerThr:   DC.W    0           ; Lower threshold clutter count
6156      0000                      UpperThr:   DC.W    0           ; Upper threshold clutter count
6158
6158                                ; ****  Derived system parameters                                ****
6158                               ;
6158
6158      0000                     RevPerSec   DC.W    0           ; Rev period in seconds   (f10)
615A      0000                      ShiftSR:    DC.W    0           ; Shift for segment range (int)
615C      0000 0000                 RUPMile:    DC.L    0           ; Range units per mile (f16)
```

```
6160    0000                        RadRange:   DC.W    0          ; Radar range (int)
6162    0000                        SynRanF:    DC.W    0          ; Synthetics range factor (f16)
6164    0000                        SPFRCS:     DC.W    0          ; SPF range capture size (int)
6166    0000                        SPFBCS:     DC.W    0          ; SPF bearing capture size (int)
6168    0000                        SPFSmooth:  DC.W    0          ; SPF smoothing factor (f16)
616A    0000                        RadRanSD:   DC.W    0          ; Radar range SD (f10)
616C    0000                        RadBearSD:  DC.W    0          ; Radar bearing SD (f16)
616E    0000                        RadRadSD:   DC.W    0          ; Radar radians SD (f16)
6170    0000                        NoiseThr:   DC.W    0          ; Noise threshold (f10)
6172    0000                        ManvThr:    DC.W    0          ; Manoeuvre threshold (f10)
6174    0000                        ManDFact:   DC.W    0          ; Manoeuvre decay factor (f10)
6176    0000                        RanNoise:   DC.W    0          ; Range noise threshold (int)
6178    0000                        MaxNRE:     DC.W    0          ; Max noise range error (int)
617A    0000                        BearNoise:  DC.W    0          ; Bearing Noise Threshold (f16)
617C    0000                        MaxNBE:     DC.W    0          ; Max noise bearing error (f16)
617E    0000                        MaxRanD:    DC.W    0          ; Max range difference (int)
6180    0000                        SpeedFact:  DC.W    0          ; Speed factor (f10)
6182    0000                        MaxBearD:   DC.W    0          ; Max bearing difference (f16)
6184    0000                        MaxRadD:    DC.W    0          ; Max radians difference (f16)
6186    0000                        ShiftClut:  DC.W    0          ; Shift for clutter range (int)
6188    0000                        CRange:     DC.W    0          ; C Range
618A    0000                        CScale:     DC.W    0          ; C Scale
618C
618C    0000                        MaxRSect    DC.W    0          ; Max range sector
618E    0000                         MaxOn4:    DC.W    0          ; Max range sector / 4
6190    0000                         MaxOn7:    DC.W    0          ; Max range sector / 7
6192    0000                         MaxOn15:   DC.W    0          ; Max range sector / 15
6194    0000                         MaxBlack:  DC.W    0          ; Max black range sector
6196
6196    xxxx xxxx xxxx      (R)     ManPKnot    DCB.W   5, 0       ; Manoeuvre per knot table
61A0    xxxx xxxx xxxx      (R)     ManPG       DCB.W   5, 0       ; Manoeuvre per G table
61AA
61AA                                ; ****  System state variables                                  ****
61AA                                ;
61AA    0000                        Octant      DC.W    0          ; Currently scanned octant
61AC    0000                        PrevOct     DC.W    0          ; Previous octant
61AE    0000                        SecProct    DC.W    0          ; Second previous octant
61B0    0000                        NextOct     DC.W    0          ; Next octant
61B2    0000                        SectOctOn   DC.W    0          ; Second octant on
61B4    0000                        FrthOctOn   DC.W    0          ; Fourth octant on
61B6
61B6                                ; **** Spat state table                                         ****
61B6                                ;
61B6    0000                        RevNum      DC.W    0          ; Current rev. number
61B8    0000                         ShipHead:  DC.W    0          ; Ships heading in degrees
61BA    0000                         ShipSpeed: DC.W    0          ; Ships speed in knots
61BC    0000                         RollRange: DC.W    0          ; Rollball range in miles
61BE    0000                         RollBear:  DC.W    0          ; Rollball bearing in degrees
61C0    0000                         TotFInput: DC.W    0          ; Total filter input plots
61C2    0000                         NumStat:   DC.W    0          ; Number of stationary plots
61C4    0000                         TotFOut:   DC.W    0          ; Total filter output plots
61C6    0000                         TotUnass:  DC.W    0          ; Total unassigned plots
61C8    0000                         NumTent:   DC.W    0          ; Number of tentative tracks
61CA    0000                         NumConf:   DC.W    0          ; Number of confirmed tracks
61CC    0000                         NumFIW:    DC.W    0          ; Number of filter input words
61CE    0000                         NumFOP:    DC.W    0          ; Number of filter output plots
61D0    0000                         NumUnass:  DC.W    0          ; Number of unassociated plots
61D2    0000                         NumSynth:  DC.W    0          ; Number of synthetics chars in buffer
61D4    0000                        TrackCount  DC.W    0          ; Track count
61D6    0000                        Loading     DC.W    0          ; System loading parameter
61D8
61D8
61D8                                 ; ************************************************************
61D8                                ; *                                                          *
61D8                                 ; *   Start of main program code -                           *
61D8                                ; *          Initialisation  routines                        *
61D8                                 ; ************************************************************
61D8                                ;
61D8
61D8    0000 0000                   Zero        DC.L    $00000000
61DC
61DC    487A A804           (P)     InitData    PEA     InpBuff
61E0    421F                                    CLR.B   (SP)+              ; Nothing in input buffer
61E2    487A A850           (P)                 PEA     TmpBuff
61E6    421F                                    CLR.B   (SP)+
61E8    487A FF20           (P)                 PEA     StopRun
61EC    421F                                    CLR.B   (SP)+              ; RATES stopped.
61EE
61EE    41FA FFC6           (P)                 LEA     RevNum, A0
61F2    303C 000E                               MOVE.W  #14, D0
61F6    4258                        ClearStat   CLR.W   (A0)+
61F8    51C8 FFFC           (P)                 DBRA    D0, ClearStat      ; Clear VDU statistics
```

```
61FC    207C 0003 8000                      MOVE.L  #DmaChan0, A0       ; Program DMA controller
6202    1168 0000 0000                      MOVE.B  0(A0), 0(A0)        ; for REQ driven cycle-steal
6208    217C AA92 0400 0004                 MOVE.L  #$AA920400, 4(A0)   ; transfer from 16-bit device
6210                                                                    ; with ACK.
6210
6210    43F9 0003 8040                      LEA     DmaChan1, A1
6216    45FA A86E           (P)             LEA     CellBlock, A2
621A    1291                                MOVE.B  (A1), (A1)          ; Clear Dma status.
621C    237C 0AA1 0400 0004                 MOVE.L  #$0aa10400, 4(A1)   ; Set Dma for burst mode
6224    234A 000C                           MOVE.L  A2, $c(A1)          ; addressing clutter cells.
6228    337C 1000 000A                      MOVE.W  #4096, $a(A1)
622E    487A FFA8           (P)             PEA     Zero
6232    237A FFA4 0014      (P)             MOVE.L  Zero, $14(A1)
6238    137C 0080 0007                      MOVE.B  #$80, 7(A1)         ; Clear 4096 cells to zero
623E
623E    45FA D846           (P)             LEA     SynthVal, A2
6242    234A 000C                           MOVE.L  A2, $c(A1)
6246    337C 00FF 000A                      MOVE.W  #255, $a(A1)
624C    1291                                MOVE.B  (A1), (A1)
624E    137C 0080 0007                      MOVE.B  #$80, 7(A1)         ; Clear 255 synthetics vals.
6254
6254    4840 xxxx           (PX)            PEA     NorthProc           ; Set interrupt vectors
6258    4840 xxxx           (PX)            PEA     ReadChar
625C    4840 xxxx           (PX)            PEA     FirstOct
6260    21DF 0068                           MOVE.L  (SP)+, IRQ2         ; Octant demand IRQ
6264    21D7 006C                           MOVE.L  (SP), IRQ3
6268    21DF 0070                           MOVE.L  (SP)+, IRQ4         ; ACIA input IRQ
626C    21D7 0074                           MOVE.L  (SP), IRQ5
6270    21DF 0080                           MOVE.L  (SP)+, TRAP0        ; North marker IRQ
6274
6274    4E75                                RTS
6276
6276
6276                            ; **** Initialise the Liquid Crystal status display and  ****
6276                            ; **** set up a display template in RAM. Clear all vals. ****
6276                            ; **** to zero & display values to spaces.               ****
6276                            ;
6276
6276    0026                    VduOffset   DC.W    38          ; Table of VDU template
6278    006E                                DC.W    110         ; parameter display locns,
627A    00C4                                DC.W    196         ; offset from start of RAM-based
627C    00D6                                DC.W    214         ; VDU map.
627E    0114                                DC.W    276
6280    0126                                DC.W    294
6282    01B8                                DC.W    440
6284    0208                                DC.W    520
6286    0258                                DC.W    600
6288    02A8                                DC.W    680
628A    02F8                                DC.W    760
628C    0348                                DC.W    840
628E
628E    41FA A21E           (P)     InitVDU LEA     VDUITab, A0         ; Transfer initialisation
6292    43F9 0003 3000                      LEA     VduR, A1            ; codes to VDU controller.
6298    323C 0008                           MOVE.W  #8, D1
629C    4A29 0003                   IVWait  TST.B   3(A1)               ; Wait for VDU controller
62A0    6B FA               (P)             BMI     IVWait              ; ready to accept data.
62A2    3358 0002                           MOVE.W  (A0)+, 2(A1)
62A6    3298                                MOVE.W  (A0)+, (A1)
62A8    51C9 FFF2           (P)             DBRA    D1, IVWait          ; Loop until VDU initialised
62AC
62AC    41FA A230           (P)             LEA     VDUTemp, A0
62B0    43FA DBD4           (P)             LEA     VDUMap, A1
62B4    323C 027F                           MOVE.W  #639, D1
62B8    32D8                        IVLoop  MOVE.W  (A0)+, (A1)+        ; Copy PROM template to RAM
62BA    51C9 FFFC           (P)             DBRA    D1, IVLoop
62BE    4E75                                RTS
62C0
62C0                            ; ************************************************************
62C0                            ; *                                                          *
62C0                            ; *          Miscellaneous character I/O subroutines         *
62C0                            ; *                                                          *
62C0                            ; ************************************************************
62C0
62C0
62C0                            ; **** Output a character to the port whose base address ****
62C0                            ; **** is IOPort, from register D0.B                     ****
62C0                            ;
62C0
62C0    0815 0001                   Outch   BTST.B  #1, (A5)            ; Wait for buffer ready
62C4    67 FA               (P)             BEQ.S   Outch
62C6    1B40 0002                           MOVE.B  D0, 2(A5)           ; Transfer to port buffer
62CA    4E75                                RTS
```

```
62CC
62CC                                    ; **** Print a null terminated string on the system      ****
62CC                                    ; **** console, by calling subroutine Outch(). A6 points ****
62CC                                    ; **** to string. Uses D0 & A5.
62CC                                    ;
62CC
62CC    4BF9 0003 FF01                  PrintStr    LEA     Terminal, A5
62D2    101E                            GetChar     MOVE.B  (A6)+, D0
62D6    xx                  (R)                     BNE.S   LastChar
62D6    61 E8               (P)                     BSR.S   Outch
62D8    60 F8               (P)                     BRA.S   GetChar
62DA    4E75                            LastChar    RTS
62DC
62DC
62DC
62DC                                    ; **** Print the current prompt string on the terminal   ****
62DC                                    ;
62DC
62DC    52 41 54 45 53 20 3E 20
                                        Pr1         DC.B    'RATES > '
62E4    0D 0A 00                                    DC.B    CR, LF, Null
62E7    00                                          .ALIGN  2
62E8    3E 20 0D 0A 00                  Pr2         DC.B    '> ', CR, LF, Null
62ED    00                                          .ALIGN  2
62EE
62EE    487A FE1A           (P)         PrPrompt    PEA     StopRun
62F2    4DFA FFE8           (P)                     LEA     Pr1, A6
62F6    081F 0007                                   BTST.B  #7, (SP)+
62FA    67 D0               (P)                     BEQ     PrintStr
62FC    4DFA FFEA           (P)                     LEA     Pr2, A6
6300    60 CA               (P)                     BRA.S   PrintStr
6302
6302
6302                                    ; **** Display a carriage return, line feed on terminal  ****
6302                                    ;
6302
6302    4BF9 0003 FF01                  NewLine     LEA     Terminal, A5
6308    103C 000D                                   MOVE.B  #CR, D0
630C    61 B2               (P)                     BSR     Outch
630E    103C 000A                                   MOVE.B  #LF, D0
6312    61 AC               (P)                     BSR     Outch
6314    4E75                                        RTS
6316
6316
6316                                    ; **** Copy the status VDU template to the VDU control   ****
6316                                    ; **** board. Uses A0, A1 and D1                  ****
6316                                    ;
6316
6316    48E7 40C0                       VDUOut      MOVEM.L D1/A0-A1, -(SP)
631A    41FA DB6A           (P)                     LEA     VDUMap,A0
631E    43F9 0003 3000                              LEA     VduR,A1
6324
6324    337C 0A0A 0002                              MOVE.W  #$0A0A,2(A1)        ; Set VDU status to receive
632A    32BC 0000                                   MOVE.W  #$0000,(A1)         ; data in template form.
632E    337C 0B0B 0002                              MOVE.W  #$0B0B,2(A1)
6334    32BC 0000                                   MOVE.W  #$0000,(A1)
6338    337C 0C0C 0002                              MOVE.W  #$0C0C,2(A1)
633E    323C 027F                                   MOVE.W  #639,D1
6342
6342    4A29 0003                       OutLoop     TST.B   3(A1)               ; Wait for controller ready
6346    6B FA               (P)                     BMI.S   OutLoop
6348    3298                                        MOVE.W  (A0)+,(A1)
634A    51C9 FFF6           (P)                     DBRA    D1,OutLoop
634E    4CDF 0302                                   MOVEM.L (SP)+, D1/A0-A1
6352    4E75                                        RTS
6354
6354                                    ; **** Convert a 16 bit integer to ascii format. The     ****
6354                                    ; **** parameters are pushed onto the stack prior to the  ****
6354                                    ; **** call: integer first, buffer address second.       ****
6354                                    ;
6354
6354    48E7 6080                       IntToAsc    MOVEM.L D1/D2/A0,-(A7)
6358    206F 0010                                   MOVE.L  12 + 4(A7), A0
635C    322F 0012                                   MOVE.W  12 + 6(A7), D1
6360    4A41                                        TST.W   D1                  ; Is integer negative
6364    xx                  (R)                     BPL.S   PosValue
6364    10FC 002D                                   MOVE.B  #'-', (A0)+         ; Insert '-'
6368    4441                                        NEG.W   D1
636A
636A    343C 2710                       PosValue    MOVE.W  #10000,D2           ; Starting quotient
636E
636E    48C1                            DecDiv      EXT.L   D1
```

```
6370    84C1                            DIVU    D1, D2          ; Divide to give result &
6374    xx              (R)             BNE.S   MakeAscii       ; remainder.
6374    0C02 0001                       CMP.B   #$0001, D2
637A    xx              (R)             BNE.S   IntZero
637A
637A    0601 0030               MakeAscii ADD.B  #$30, D1       ; Convert rmdr to Ascii
637E    10C1                            MOVE.B  D1, (A0)+
6380
6380    4841                    IntZero SWAP    D1              ; Switch rmdr & result
6382    84FC 000A                       DIVU    #10, D2         ; Divide quotient by 10
6386    66 E6           (P)             BNE.S   DecDiv
6388
6388    4210                            CLR.B   (A0)            ; Null terminate
638A
638A    4CDF 0106                       MOVEM.L (A7)+,D1/D2/A0  ; restore registers
638E    2F57 0006                       MOVE.L  (A7), 6(A7)
6392    DFFC 0000 0006                  ADDA.L  #6, A7
6398    4E75                            RTS
639A
639A
639A                            ; **** Copy an Ascii string from a buffer to the status   ****
639A                            ; **** VDU RAM template using fields 6 characters wide.   ****
639A                            ; **** Needs at least 10 bytes of local variable space.   ****
639A                           ;
639A
639A    4E54 FFF6               VduMapUp LINK   A4, #-10
639E    48E7 70F0                       MOVEM.L D1-D3/A0-A3, -(SP)  ; Push registers
63A2
63A2    41FA FE12       (P)             LEA     RevNum, A0      ; Address of first integer
63A6    43FA DADE       (P)             LEA     VDUMap, A1
63AA    45FA FECA       (P)             LEA     VduOffset, A2
63AE    323C 000B                       MOVE.W  #11, D1
63B2
63B2    47EC FFF6               IntConv LEA     -10(A4), A3     ; Output buffer address
63B6    3F18                            MOVE.W  (A0)+, -(A7)
63B8    2F0B                            MOVE.L  A3, -(A7)
63BA    61 98           (P)             BSR     IntToAsc
63BC
63BC    343C 0005                       MOVE.W  #5, D2          ; Set character count
63C0    361A                    CopyChar MOVE.W (A2)+, D3       ; Get output offset
63C2    4A13                            TST.B   (A3)            ; Is string null or short ?
63C6    xx              (R)             BEQ.S   SpaceOut
63C6
63C6    139B 30 00                      MOVE.B  (A3)+, (A1, D3) ; No: Copy Ascii digit
63CA    4EC0 xxxx       (PX)            BRA     CopyInc
63CE    13BC 0020 30 00         SpaceOut MOVE.B #' ', (A1, D3)
63D4
63D4    5489                    CopyInc ADDQ.L  #2, A1
63D6    51C9 FFDA       (P)             DBRA    D1,IntConv
63DA
63DA    4CDF 0F0E                       MOVEM.L (SP)+, D1-D3/A0-A3  ; Restore registers
63DE    4E5C                            UNLK    A4
63E0    4E75                            RTS
63E2
63E2
63E2                            ; ****  End of general purpose output routines. Next one   ****
63E2                            ; ****  is for handling input from the console. It is only ****
63E2                            ; ****  called when an interrupt is generated from the     ****
63E2                            ; ****  acia port connected to the terminal.               ****
63E2                            ; ****  All affected registers are saved. If the input     ****
63E2                            ; ****  character is a CR, the command is passed to the    ****
63E2                            ; ****  parser and interpreter.                            ****
63E2                           ;
63E2
63E2    40E7                    ReadChar MOVE.W SR, -(SP)       ; Save status register
63E4    48E7 E0E6                       MOVEM.L D0-D2/A0-A2/A5-A6, -(SP)
63E8    46FC 2400                       MOVE.W  #$2400, SR      ; Mask all interrupts
63EC
63EC    41FA A5F0       (P)             LEA     BuffPtr, A0     ; Current buffer pointer
63F0    4BF9 0003 FF01                  LEA     Terminal, A5
63F6    1ABC 00D5                       MOVE.B  #$d5, (A5)      ; Set RTS high
63FA
63FA    102D 0002               InChar  MOVE.B  2(A5), D0       ; Read character from acia
63FE    0200 007F                       AND.B   #$7F,D0
6402
6402    0C00 0008                       CMP.B   #BS, D0         ; Is it a backspace
6408    xx              (R)             BNE.S   NotBS
6408
6408    B1FA A5D8       (P)             CMPA.L  InpBuff, A0     ; Can't backspace at start
640E    xx              (R)             BNE.S   EchoBS          ; of line !
640E    103C 0007                       MOVE.B  #BEL, D0
6412    61 00 FEAC      (P)             BSR     Outch
```

```
6416    4EC0 xxxx          (PX)              BRA     ExitRead
641A
641A    4DFA 9C1D          (P)     EchoBS    LEA     BackSp, A6
641E    61 00 FEAC         (P)               BSR     PrintStr
6422    5388                               SUBQ.L  #$01, A0          ; Input pointer to prev. char
6424    4EC0 xxxx          (PX)              BRA     ExitRead
6428
6428    0C00 0018                  NotBS     CMP.B   #CtrlX, D0        ; Delete line char ?
642E    xx                 (R)               BEQ.S   ClrIpBuff
642E
642E    0C00 000D                            CMP.B   #CR, D0           ; Carriage return ?
6434    xx                 (R)               BNE.S   NotCR
6434
6434    4E80 xxxx          (PX)              BSR     Interpret         ; Interpret command
6438    61 00 FEB4         (P)     ClrIpBuff BSR     PrPrompt          ; Print new prompt string
643C    41FA A5A4          (P)               LEA     InpBuff, A0       ; & clear input buffer.
6440    4EC0 xxxx          (PX)              BRA     ExitRead
6444
6444    61 00 FE7A         (P)     NotCR     BSR     Outch
6448    10C0                               MOVE.B  D0, (A0)+         ; Copy character to buffer
644A
644A    4210                       ExitRead  CLR.B   (A0)              ; Null terminate buffer
644C    487A A590          (P)               PEA     BuffPtr
6450    2EC8                               MOVE.L  A0, (SP)+         ; Save pointer
6452
6452    1ABC 0095                          MOVE.B  #$95, (A5)        ; Enable RTS again
6456    4CDF 6707                          MOVEM.L (SP)+, D0-D2/A0-A2/A5-A6
645A    46DF                               MOVE.W  (SP)+, SR
645C    4E73                               RTE
645E
645E
645E                                ; ****  Subroutine to parse the input buffer and pick out  ****
645E                                ; ****  command strings for execution. The appropriate     ****
645E                                ; ****  subroutine is then executed or an error printed.   ****
645E                                ; ****  D7 is used to communicate the command sense.       ****
645E                               ;
645E
645E    48E7 C1E0                  Interpret MOVEM.L D0-D1/D7/A0-A2, -(SP)
6462    41FA A57E          (P)               LEA     InpBuff, A0       ; Set pointer to buffer start
6466    1E3C 0080                          MOVE.B  #$80, D7          ; Command is 'active'
646A
646A    4E80 xxxx          (PX)    InterNext BSR     ReadString        ; Read command into TmpBuff
6470    xx                 (R)               BEQ.S   NoCommand
6470
6470    41FA 9BCC          (P)               LEA     Commands, A0      ; Address of commands table
6474    4240                               CLR.W   D0                ; D0 is index to keyword
6476    4281                               CLR.L   D1
6478
6478    5240                       NextKey   ADDQ.W  #$0001,D0
647A
647A    3230 00 00                         MOVE.W  (A0, D0.W), D1    ; Read offset byte
6480    xx                 (R)               BEQ.S   LastEntry
6480
6480    43F0 10 00                         LEA     (A0, D1.W), A1    ; Generate keyword address
6484    45FA A5AE          (P)               LEA     TmpBuff, A2
6488    B509                       StrCmp    CMPM.B  (A1)+, (A2)+      ; Compare key with input string
648A    67 FC              (P)               BEQ.S   StrCmp
648C
648C    4A22                               TST.B   -(A2)             ; End of command string?
6490    xx                 (R)               BEQ.S   StrMatch
6490    60 E6              (P)               BRA     NextKey           ; Try another keyword
6492
6492    4240                       LastEntry CLR.W   D0                ; No keyword match
6494
6494    D040                       StrMatch  ADD.W   D0, D0            ; Generate subroutine offset
6496    303B 00 xx         (R)               MOVE.W  JTab(D0), D0
649A    4EBB 00 xx         (R)               JSR     JTab(D0)          ; Call subroutine - A0 points
649E                                                                   ; to rest of input buffer.
649E    487A A542          (P)               PEA     InpBuff
64A2    421F                               CLR.B   (SP)+
64A4    4CDF 0783                  NoCommand MOVEM.L (SP)+, D0-D1/D7/A0-A2
64A8    4E75                               RTS
64AA
64AA                                ; ****  Jump table with pointers to command subroutines    ****
64AA                               ;
64AA
64AA                               JTab
64AC    xxxx               (R)             DC.W    Error - JTab      ; First prints error message
64AE    xxxx               (R)             DC.W    DelCmd - JTab     ; Negate command action
64B0    xxxx               (R)             DC.W    Run - JTab        ; Start the tracking software
64B2    xxxx               (R)             DC.W    Halt - JTab       ; Halt the tracker
64B4    xxxx               (R)             DC.W    DrpCom - JTab     ; Display residual plots
```

```
64B6    xxxx            (R)             DC.W    DipCom - JTab       ; Display input plots
64B8    xxxx            (R)             DC.W    DspCom - JTab       ; Display stationary plots
64BA    xxxx            (R)             DC.W    DttCom - JTab       ; Display tentative tracks
64BC    xxxx            (R)             DC.W    DctCom - JTab       ; Display confirmed tracks
64BE    xxxx            (R)             DC.W    PipCom - JTab       ; Plot input plots
64C0    xxxx            (R)             DC.W    PopCom - JTab       ; Plot output plots
64C2    xxxx            (R)             DC.W    PrtCom - JTab       ; Print tracks command
64C4    xxxx            (R)             DC.W    Radar - JTab        ; Display / set radar type
64C6    xxxx            (R)             DC.W    ListRDP - JTab      ; List radar-dep parameters
64C8    xxxx            (R)             DC.W    ListRIP - JTab      ; List radar-ind parameters
64CA    xxxx            (R)             DC.W    SetValue - JTab     ; Set tracking parameters
64CC    xxxx            (R)             DC.W    ThreshCom - JTab    ; Set threshold level
64CE    xxxx            (R)             DC.W    ManCom - JTab       ; Manual track initiation
64D0    xxxx            (R)             DC.W    TT1 - JTab          ; Set first test target
64D2    xxxx            (R)             DC.W    TT2 - JTab          ; Set second test target
64D4    xxxx            (R)             DC.W    TT3 - JTab          ; Set third test target
64D6    xxxx            (R)             DC.W    Monitor - JTab      ; Jump to KDM monitor
64D6
64D6                                    ; ****  Read a string composed of alphas and numerics only ****
64D6                                    ; ****  from the buffer pointed to by A0, and store the     ****
64D6                                    ; ****  null terminated result in the buffer TmpBuff.      ****
64D6                                    ;
64D6    48E7 C040                       ReadString  MOVEM.L  D0-D1/A1, -(SP)
64DA    43FA A558       (P)                 LEA     TmpBuff, A1         ; Address of output buffer
64DE    4241                                CLR.W   D1
64E0
64E0    1018                            ReRead      MOVE.B   (A0)+, D0          ; Read next character
64E2    0C00 0030                           CMP.B   #'0', D0
64E8    xx              (R)                 BLT.S   NotAlNum            ; Char < 'zero'
64E8    0C00 005A                           CMP.B   #'Z',D0
64EE    xx              (R)                 BGT.S   NotAlNum            ; Char > 'Z'
64EE    0C00 0039                           CMP.B   #'9', D0
64F4    xx              (R)                 BLT.S   ValidChar
64F4    0C00 0041                           CMP.B   #'A',D0
64FA    xx              (R)                 BLT.S   NotAlNum            ; '9' < char < 'A'
64FA
64FA    12C0                            ValidChar   MOVE.B   D0, (A1)+          ; Copy to output buffer
64FC    5241                                ADDQ.W  #1, D1              ; Increment output count
64FE
64FE    4A41                            NotAlNum    TST.W    D1             ; Has output been written?
6502    xx              (R)                 BNE.S   OutputDone
6502    0C00 0020                           CMP.B   #' ', D0            ; Ignore leading spaces.
6506    67 D8           (P)                 BEQ.S   ReRead
6508
6508    4211                            OutputDone  CLR.B    (A1)           ; Null terminate
650A    4CDF 0203                           MOVEM.L (SP)+, D0-D1/A1
650E    4E75                                RTS
6510      .
6510                                    ; ****  Read an integer from the input buffer pointed to by    ****
6510                                    ; ****  A0 and return the integer in register D0.              ****
6510                                    ; ****  If D2 is $FFFF, no integer was found in the buffer. ****
6510                                    ;
6510    48E7 6000                       ReadInt     MOVEM.L  D1-D2, -(SP)
6514    343C FFFF                           MOVE.W   #$ffff, D2         ; Set status
6518    1018                                MOVE.B   (A0)+, D0          ; Read first character
651C    xx              (R)                 BEQ.S   GotTermCh
651C    0C00 0020                           CMP.B   #' ', D0            ; Ignore leading spaces
6520    67 EE           (P)                 BEQ.S   ReadInt
6522
6522    4242                                CLR.W   D2
6524    4281                                CLR.L   D1                  ; Result register
6526
6526    0C00 002D                           CMP.B   #'-', D0            ; Is number negative?
652C    xx              (R)                 BNE.S   PosInt
652C    343C 8000                           MOVE.W  #$8000,D2
6530    1018                                MOVE.B  (A0)+, D0           ; Read first digit
6532
6532    0400 0030                       PosInt      SUB.B    #'0', D0           ; Remove Ascii offset
6538    xx              (R)                 BLT.S   LastDigit
6538    0C00 000A                           CMP.B   #$0a, D0            ; Check: 0 <= Char <= 9
653E    xx              (R)                 BGT.S   LastDigit
653E    C2FC 000A                           MULU    #$0a, D1            ; Sum = Sum * 10 + Digit
6542    D240                                ADD.W   D0, D1
6544    1018                                MOVE.B  (A0)+, D0           ; Next digit
6546    60 EA           (P)                 BRA     PosInt
6548
6548    4A42                            LastDigit   TST.W    D2
654C    xx              (R)                 BEQ.S   GotTermCh
654C    4441                                NEG.W   D1                  ; Negate result
654E
654E    2F01                            GotTermCh   MOVE.L   D1, -(SP)          ; Result to D0
6550    4CDF 0007                           MOVEM.L (SP)+, D0-D2
6554    4E75                                RTS
```

```
6556                                    ; ****  Print an error message on the console & clear the  ****
6556                                    ; ****  input buffer.                                      ****
6556                                    ;
6556
6556                                    ;  RATES Error Messages
6556
6558    xxxx                (R)         ErrIndex    DC.W    ITL - ErrIndex
655A    xxxx                (R)                     DC.W    IChar - ErrIndex
655C    xxxx                (R)                     DC.W    IPar - ErrIndex
655E    xxxx                (R)                     DC.W    IFile - ErrIndex
6560    xxxx                (R)                     DC.W    IName - ErrIndex
6562    xxxx                (R)                     DC.W    INum - ErrIndex
6564    xxxx                (R)                     DC.W    ICom - ErrIndex
6564    0000                                        DC.W    $0000
6566
6566    49 6E 76 61 6C 69 64 20 54 68 72 65 73 68 6F 6C 64 20 4C 65 76 65 6C 20 20 00
                                    ITL:        DC.B    'Invalid Threshold Level  ', Null
6580    41 20 2D 20 5A 20 3F 00
                                    IChar:      DC.B    'A - Z ?', Null
6588    52 44 50 20 2F 20 52 49 50 20 3F 00
                                    IPar:       DC.B    'RDP / RIP ?', Null
6594    4F 70 65 6E 20 46 69 6C 65 20 45 72 72 6F 72 20 20 00
                                    IFile:      DC.B    'Open File Error  ', Null
65A6    4E 61 6D 65 20 45 72 72 6F 72 20 20 00
                                    IName:      DC.B    'Name Error  ', Null
65B3    4E 75 6D 62 65 72 20 3F 00
                                    INum:       DC.B    'Number ?', Null
65BC    43 6F 6D 6D 61 6E 64 20 3F 00
                                    ICom:       DC.B    'Command ?', Null
65C6                                        .ALIGN  2
65C6
65C6    303C 0006                   Error       MOVE.W  #$0006, D0          ; Issue illegal command error
65CA
65CA    D040                        GenError    ADD.W   D0, D0              ; Generate correct offset
65CC    4DFA FF88           (P)                 LEA     ErrIndex, A6
65D0    3036 00 00                  MOVE.W  (A6, D0.W), D0      ; Form offset for string
65D4    4DF6 00 00                  LEA     (A6, D0.W), A6
65D8    61 00 FCF2          (P)             BSR     PrintStr            ; Print error message
65DC    61 00 FD10          (P)             BSR     PrPrompt            ; and prompt.
65E0    487A A400           (P)             PEA     InpBuff
65E4    421F                            CLR.B   (SP)+               ; Reset input pointer
65E6    4E75                            RTS
65E8
65E8                                    ; ****  Negate the sense of the command which follows it   ****
65E8                                    ;
65E8    4207                        DelCmd      CLR.B   D7                  ; Negate command sense
65EA    60 00 FE7E          (P)             BRA     InterNext
65EE
65EE                                    ; ****  Start (or stop) the operation of the tracker       ***
65EE                                    ;
65EE    487A FB1A           (P)     Run         PEA     StopRun
65F2    1EFC 0080                       MOVE.B  #$80, (SP)+
65F6    4E75                            RTS
65F8
65F8                                    ; ****  Suspend output from the tracker to the hardware    ****
65F8                                    ;
65F8    487A FB10           (P)     Halt        PEA     StopRun
65FC    421F                            CLR.B   (SP)+
65FE    4E75                            RTS
6600
6600                                    ; ****  Turn on display of unassociated plots                 ****
6600                                    ;
6600    487A FB09           (P)     DrpCom      PEA     DispUnsp
6604    1EC7                            MOVE.B  D7, (SP)+
6606    4E75                            RTS
6608
6608                                    ; ****  Turn on display of input plots                    ****
6608                                    ;
6608    487A FB02           (P)     DipCom      PEA     DispIpc
660C    1EC7                            MOVE.B  D7, (SP)+
660E    4E75                            RTS
6610
6610                                    ; ****  Turn on display of stationary plots                   ****
6610                                    ;
6610    487A FAFB           (P)     DspCom      PEA     DispStat
6614    1EC7                            MOVE.B  D7, (SP)+
6616    4E75                            RTS
6618
6618                                    ; ****  Turn on display of tentative tracks                   ****
6618                                    ;
6618    487A FAF4           (P)     DttCom      PEA     DispTent
661C    1EC7                            MOVE.B  D7, (SP)+
661E    4E75                            RTS
```

```
6620                                    ; ****  Turn on display of confirmed tracks                    ****
6620                                    ;
6620    487A FAED               (P)     DctCom      PEA     DispConf
6624    1EC7                            MOVE.B      D7, (SP)+
6626    4E75                            RTS
6628
6628                                    ; ****  Plot input plot data                                   ****
6628                                    ;
6628    487A FAE6               (P)     PipCom      PEA     PlotIp
662C    1EC7                            MOVE.B      D7, (SP)+
662E    4E75                            RTS
6630
6630                                    ; ****  Plot output plot data                                  ****
6630                                    ;
6630    487A FADE               (P)     PopCom      PEA     PlotIp
6634    1EC7                            MOVE.B      D7, (SP)+
6636    4E75                            RTS
6638
6638                                    ; ****  Print track data                                       ****
6638                                    ;
6638    487A FAD8               (P)     PrtCom      PEA     PrintTrks
663C    1EC7                            MOVE.B      D7, (SP)+
663E    4E75                            RTS
6640
6640
6640                                    ; ****  Print the current radar type on the screen             ****
6640                                    ; ****  followed by the prompt string.
6640                                    ;
6640
6640    4DFA 99E5               (P)     Radar       LEA     RadMess, A6         ; Radar type - prompt string
6644    61 00 FC86              (P)                 BSR     PrintStr
6648    303A FAD8               (P)                 MOVE.W  RadarType, D0       ; Get current radar type
664C    0640 0041                                   ADD.W   #'A', D0
6650    61 00 FC6E              (P)                 BSR     Outch               ; Print type letter
6654    61 00 FCAC              (P)                 BSR     NewLine
6658    4E75                                        RTS
665A
665A
665A                                    ; ****  Print the Radar Dependent or Independent parameters     ****
665A                                    ; ****  on the console port.                                   ****
665A                                    ;
665A
665A    61 E4                   (P)     ListRDP     BSR     Radar               ; First line of output
665C    323C 0007                                   MOVE.W  #$0007, D1          ; 8 other parameters
6660    4282                                        CLR.L   D2
6662    41FA FAC2               (P)                 LEA     RevPermi, A0
6666    43FA 9A68               (P)                 LEA     Rdps, A1
666A
666A    4DF1 20 00                      NextRDP     LEA     (A1, D2.W), A6
666E    0682 0000 0020                              ADD.L   #$00000020, D2      ; Increment to next string
6674    61 00 FC56              (P)                 BSR     PrintStr
6678
6678    3F18                                        MOVE.W  (A0)+, -(SP)        ; Push parameter value
667A    487A A3B8               (P)                 PEA     TmpBuff             ; and string address
667E    61 00 FCD4              (P)                 BSR     IntToAsc
6682    4DFA A3B0               (P)                 LEA     TmpBuff, A6
6686    61 00 FC44              (P)                 BSR     PrintStr            ; Convert to Ascii & print
668A    61 00 FC76              (P)                 BSR     NewLine
668E    51C9 FFDA               (P)                 DBRA    D1, NextRDP         ; Next Parameter
6692    4E75                                        RTS
6694
6694
6694                                    ; ****  Print radar independent parameter strings & values.    ****
6694                                    ;
6694
6694    41FA FAA0               (P)     ListRIP     LEA     InitSpfc, A0        ; Address of parameter 1
6698    43FA 9B36               (P)                 LEA     Rips, A1
669C    323C 0010                                   MOVE.W  #$0010,D1           ; out of a total of 17.
66A0    4282                                        CLR.L   D2
66A2    61 00 FC5E              (P)                 BSR     NewLine
66A6
66A6    4DF1 20 00                      NextRIP     LEA     (A1, D2.W), A6
66AA    0682 0000 0020                              ADD.L   #$00000020, D2      ; Next parameter string
66B0    61 00 FC1A              (P)                 BSR     PrintStr
66B4
66B4    3F18                                        MOVE.W  (A0)+, -(SP)        ; Push parameter value
66B6    487A A37C               (P)                 PEA     TmpBuff             ; and output address.
66BA    61 00 FC98              (P)                 BSR     IntToAsc
66BE    4DFA A374               (P)                 LEA     TmpBuff, A6         ; Set pointer to result
66C2    61 00 FC08              (P)                 BSR     PrintStr
66C6    61 00 FC3A              (P)                 BSR     NewLine
66CA    51C9 FFDA               (P)                 DBRA    D1, NextRIP         ; Next parameter
66CE    4E75                                        RTS
```

```
66D0                                ; ****  Set a value for any one of the tracking parameters ****
66D0                               ;
66D0
66D0      43FA FA54          (P)   SetValue    LEA      RevPermi, A1
66D4      4240                               CLR.W    D0
66D6      1018                               MOVE.B   (A0)+, D0          ; Read next char in buffer
66DA      xx                 (R)             BEQ.S    ValueError         ; Next char is null
66DA      0C00 0020                          CMP.B    #' ', D0
66DE      67 F0              (P)             BEQ.S    SetValue           ; Ignore leading spaces
66E0
66E0      0400 0041                          SUB.B    #'A', D0
66E6      xx                 (R)             BMI.S    ValueError
66E6      0C00 001A                          CMP.B    #26,D0             ; > 'Z' ?
66EC      xx                 (R)             BGT.S    ValueError
66EC
66EC      D040                               ADD.W    D0, D0             ; Convert letter to index
66EE      43F1 00 00                         LEA      (A1, D0.W), A1
66F2      61 00 FE1C         (P)             BSR      ReadInt
66F8      xx                 (R)             BNE.S    StoreValue
66F8
66F8      303C 0005                          MOVE.W   #5, D0             ; No parameter value given
66FC      60 00 FECC         (P)             BRA      GenError
6700
6700      303C 0001          ValueError      MOVE.W   #1, D0             ; Error in parameter code
6704      60 00 FEC4         (P)             BRA      GenError
6708
6708      3280               StoreValue      MOVE.W   D0, (A1)           ; Write new value
670A      4E80 xxxx          (PX)            BSR      DerivePars         ; Recalculate other values
670E      4E75                               RTS
6710
6710                                ; ****  Set a threshold level in the CFAR clutter map       ****
6710                                ; ****  Destroys contents of D0, D1 & A1. A0 points to the  ****
6710                                ; ****  input buffer, D7 is set to command sense.          ****
6710                               ;
6710
6710      43FA FA01          (P)   ThreshCom   LEA      Thresh, A1
6714      4A07                               TST.B    D7                 ; Test command sense
6718      xx                 (R)             BEQ.S    NoLevel
6718
6718      61 00 FDF6         (P)             BSR      ReadInt            ; Read hole size
671E      xx                 (R)             BEQ.S    ThreshErr
671E      3200                               MOVE.W   D0, D1             ; Store hole size
6720      61 00 FDEE         (P)             BSR      ReadInt
6726      xx                 (R)             BEQ.S    LevelErr
6726      4A40                               TST.W    D0
672A      xx                 (R)             BMI.S    LevelErr           ; Can't be negative
672A
672A      0C40 007F                          CMP.W    #127, D0           ; Or > 127
6730      xx                 (R)             BGT.S    LevelErr
6730
6730      12FC 0080                          MOVE.B   #$80, (A1)+
6734      32C1                               MOVE.W   D1, (A1)+          ; Restore hole size
6736      3280                               MOVE.W   D0, (A1)           ; Save new level
6738      4EC0 xxxx          (PX)            BRA      ThreshDer          ; Go derive new parameters
673C
673C      4211               LevelErr        CLR.B    (A1)
673E      303C 0000                          MOVE.W   #0, D0             ; Set error code
6742      60 00 FE86         (P)             BRA      GenError
6746
6746      303C 0005          ThreshErr       MOVE.W   #5, D0             ; Command error code
674A      60 00 FE7E         (P)             BRA      GenError
674E
674E      4211               NoLevel         CLR.B    (A1)
6750
6750      4E80 xxxx          (PX)  ThreshDer   BSR      DerivePars         ; Calculate derived parameters
6754      4E75                               RTS
6756
6756                                ; ****  Manually initiate a track                          ****
6756                               ;
6756      487A F9C0          (P)   ManCom      PEA      ManInit
675A      1EC7                               MOVE.B   D7, (SP)+
675C      4E75                               RTS
675E
675E
675E                                ; ****  Set the parameters for one of the test targets     ****
675E                                ; ****  A0 points to input buffer, D7 is command sense.    ****
675E                                ; ****  destroys the contents of D0, A1.                   ****
675E
675E      303C 0001                  TT1           MOVE.W   #$01, D0
6762      4EC0 xxxx          (PX)                  BRA      TTProc
6766
6766      303C 0002                  TT2           MOVE.W   #$02, D0
676A      4EC0 xxxx          (PX)                  BRA      TTProc
```

```
676E    303C 0003                   TT3         MOVE.W   #$03, D0
6772
6772    43FA F9A5           (P)     TTProc      LEA      TestNum, A1
6776    4A07                                    TST.B    D7                 ; Test command sense
677A    xx                  (R)                 BNE.S    TTSet
677A
677A    4400                                    NEG.B    D0                 ; Target generation off
677C    1280                                    MOVE.B   D0, (A1)
677E    4EC0 xxxx           (PX)                BRA      TTEnd
6782
6782    12C0                        TTSet       MOVE.B   D0, (A1)+
6784    61 00 FD8A          (P)                 BSR      ReadInt            ; Read target range
678A    xx                  (R)                 BEQ.S    TTEnd
678A    32C0                                    MOVE.W   D0, (A1)+
678C
678C    61 00 FD82          (P)                 BSR      ReadInt            ; Read target bearing
6792    xx                  (R)                 BEQ.S    TTEnd
6792    32C0                                    MOVE.W   D0, (A1)+
6794
6794    61 00 FD7A          (P)                 BSR      ReadInt            ; Read target speed
679A    xx                  (R)                 BEQ.S    TTEnd
679A    32C0                                    MOVE.W   D0, (A1)+
679C
679C    61 00 FD72          (P)                 BSR      ReadInt            ; Read target heading
67A2    xx                  (R)                 BEQ.S    TTEnd
67A2    3280                                    MOVE.W   D0, (A1)
67A4
67A4    4E75                        TTEnd       RTS
67A6
67A6
67A6                                 ; ****  Jump back to KDM board monitor                    ****
67A6                                ;
67A6    4BF9 0003 FF01              Monitor     LEA      Terminal, A5
67AC    61 00 FB54          (P)                 BSR      NewLine
67B0    1ABC 0015                               MOVE.B   #$15, (A5)         ; Clear interrupt status
67B4    21FC 0002 1706 0068                     MOVE.L   #$21706, IRQ2
67BC    21FC 0002 1706 006C                     MOVE.L   #$21706, IRQ3      ; Reset vectors
67C4    21FC 0002 1706 0070                     MOVE.L   #$21706, IRQ4
67CC    4EF9 0002 00F6                          JMP      MacsBug
67D2
67D2
67D2                                  ; ***************************************************************
67D2                                ; *                                                             *
67D2                                 ; *  Routines for performing calculations on fixed point data  *
67D2                                 ; *  & for deriving parameters from the basic radar dependent  *
67D2                                ; *  and independent values.                                    *
67D2                                ; *                                                             *
67D2                                  ; ***************************************************************
67D2                                ;
67D2    48E7 40C0                   CopyParam   MOVEM.L  D1/A0-A1, -(A7)    ; Push registers
67D6    41FA 9C38           (P)                 LEA      RdpDef, A0
67DA    487A F946           (P)                 PEA      RadarType
67DE    321F                                    MOVE.W   (SP)+, D1
67E0    E949                                    LSL.W    #4, D1             ; Generate index to pars.
67E2    41F0 10 00                              LEA      (A0, D1.W), A0
67E6    43FA F93E           (P)                 LEA      RevPermi, A1
67EA    323C 0007                               MOVE.W   #7, D1             ; 8 parameters to copy
67EE
67EE    32D8                        CopyNext    MOVE.W   (A0)+, (A1)+       ; Copy into RAM
67F0    51C9 FFFC           (P)                 DBRA     D1, CopyNext
67F4    41FA 9C92           (P)                 LEA      RipDef, A0
67F8    323C 0010                               MOVE.W   #16, D1            ; 16 independent parameters
67FC
67FC    32D8                        CopyMore    MOVE.W   (A0)+, (A1)+       ; Copy to variables
67FE    51C9 FFFC           (P)                 DBRA     D1, CopyMore
6802
6802    4CDF 0302                               MOVEM.L  (A7)+, D1/A0-A1    ; Restore registers
6806    4E75                                    RTS
6808
6808
6808                                 ; ****  Perform a fixed point divide operation such that   ****
6808                                ; ****  Integer D0 / Integer D1  gives result in the form :
6808                                ; ****      (16 bit integer & 16 bit fractional part)       ****
6808                                ;
6808
6808    80C1                        Divide16    DIVU     D1, D0             ; Generate integer & rmdr
680A    3F00                                    MOVE.W   D0, -(SP)          ; Push integer part
680C    4240                                    CLR.W    D0
680E    80C1                                    DIVU     D1, D0             ; Divide rmdr by quotient
6810    4840                                    SWAP     D0
6812    301F                                    MOVE.W   (SP)+, D0          ; Replace rmdr with integer
6814    4840                                    SWAP     D0                 ; Swap bytes
6816    4E75                                    RTS
```

```
6818                                    ; ****  Multiply two decimal numbers together, where the   ****
6818                                    ; ****  numbers each have a 16 bit integer part and 16 bit ****
6818                                    ; ****  fractional part. The numbers start in D1 and D2.   ****
6818                                    ; ****  The result is left in register D0                  ****
6818                                    ; ****  Uses: a.b * x.y = (ax + 0.bx + 0.ay + (0.b)(0.y))  ****
6818                                   ;
6818      48E7 3800                    Mult16      MOVEM.L  D2-D4, -(SP)
681C      4284                                     CLR.L    D4                  ; Clear result register
681E      2401                                     MOVE.L   D1, D2
6820      3601                                     MOVE.W   D1, D3              ; Fraction to D3
6822
6822      4842                                     SWAP     D2
6824      C6C0                                     MULU     D0, D3              ; (0.b)(0.y)
6826      4843                                     SWAP     D3
6828      D843                                     ADD.W    D3, D4              ; Store partial result
682A
682A      3602                                     MOVE.W   D2, D3
682C      C6C0                                     MULU     D0, D3              ; (0.b)x
682E      D883                                     ADD.L    D3, D4              ; Accumulate result
6830
6830      4840                                     SWAP     D0
6832      3601                                     MOVE.W   D1, D3
6834      C6C0                                     MULU     D0, D3              ; (0.y)a
6836      D883                                     ADD.L    D3, D4              ; Accumulate result
6838
6838      C0C2                                     MULU     D2, D0              ; ab
683A      4840                                     SWAP     D0                  ; Rearrange words
683C      4240                                     CLR.W    D0
683E      D084                                     ADD.L    D4, D0              ; Accumulate result
6840
6840      4CDF 001C                                MOVEM.L  (SP)+, D2-D4
6844      4E75                                     RTS
6846
6846                                    ; ****  Derive parameters based on the radar dependent     ****
6846                                    ; ****  parameters. These are only calculated occasionally ****
6846                                   ;
6846      48E7 F8F0                    DeriveFix   MOVEM.L  D0-D4/A0-A3, -(SP)
684A      41FA F90C            (P)                 LEA      RevPerSec, A0
684E
684E      303C 000A                                MOVE.W   #10, D0
6852      323A F8D2            (P)                 MOVE.W   RevPermi, D1        ; Get rev period in millisecs
6856      48C1                                     EXT.L    D1
6858      E1A9                                     LSL.L    D0, D1
685A      82FC 03E8                                DIVU     #1000, D1
685E      30C1                                     MOVE.W   D1, (A0)+           ; Rev period in secs (f10)
6860
6860      907A F8D2            (P)                 SUB.W    RangeMsBt, D0       ; ShiftSR = 10 - RangeMsBit
6864      30C0                                     MOVE.W   D0, (A0)+           ; Shift for segment range
6866
6866      203C 0000 BB80                           MOVE.L   #48000, D0
686C      323A F8C0            (P)                 MOVE.W   RangeCSFt, D1       ; RUPM = RCSFt / 48000
6870      61 96                (P)                 BSR      Divide16
6872      20C0                                     MOVE.L   D0, (A0)+           ; Range units / mile (f16)
6874
6874      4281                                     CLR.L    D1
6876      323A F8B0            (P)                 MOVE.W   RadRanDM, D1
687A      4841                                     SWAP     D1
687C      61 9A                (P)                 BSR      Mult16              ; RadarRan = RRDM * RUPM
687E      4840                                     SWAP     D0
6880      30C0                                     MOVE.W   D0, (A0)+           ; Radar range (integer)
6882
6882      4281                                     CLR.L    D1
6884      323C 03FF                                MOVE.W   #1023, D1
6888      4841                                     SWAP     D1                  ; SynRF = 1023 / RadarRan
688A      82C0                                     DIVU     D0, D1
688C      30C1                                     MOVE.W   D1, (A0)+           ; Synthetics range fact (f16)
688E
688E      303A F8A0            (P)                 MOVE.W   SPFRCSFt, D0
6892      C0FC 0008                                MULU     #8, D0              ; SPFRCS = SPFRCSFt * 8 / RCSFT
6896      48C0                                     EXT.L    D0
6898      80FA F894            (P)                 DIVU     RangeCSFt, D0
689C      30C0                                     MOVE.W   D0, (A0)+           ; Spf range capt. size (int)
689E
689E      303A F892            (P)                 MOVE.W   SPFBCSCd, D0
68A2      C0FC E902                                MULU     #$E902, D0
68A6      4840                                     SWAP     D0
68A8      30C0                                     MOVE.W   D0, (A0)+           ; Spf bearing capt. size (int)
68AA
68AA      4280                                     CLR.L    D0
68AC      303A F88E            (P)                 MOVE.W   CenSPFSmo, D0       ; SPFS = CSPFS / 100
68B0      323C 0064                                MOVE.W   #100, D1
68B4      61 00 FF52           (P)                 BSR      Divide16
68B8      30C0                                     MOVE.W   D0, (A0)+           ; Spf smoothing factor (f16)
```

```
68BA    4280                            CLR.L   D0
68BC    303A F86C           (P)         MOVE.W  RangeSDFt, D0
68C0    223C 0000 1770                  MOVE.L  #6000,D1
68C6    61 00 FF40          (P)         BSR     Divide16
68CA    223A F890           (P)         MOVE.L  RUPMile, D1
68CE    61 00 FF48          (P)         BSR     Mult16              ; RRSD = RSDF * 6000 / RUPM
68D2    2600                            MOVE.L  D0, D3              ; Store RRSD in f16 format
68D4    EC88                            LSR.L   #6, D0
68D6    30C0                            MOVE.W  D0, (A0)+           ; Radar range sd   (f10)
68D8                                                                    °
68D8    303A F852           (P)         MOVE.W  BearSDCd, D0
68DC    C0FC 003A                       MULU    #$003A, D0
68E0    E848                            LSR.W   #4, D0
68E2    30C0                            MOVE.W  D0, (A0)+           ; Radar bearing sd (f16)
68E4
68E4    48C0                            EXT.L   D0
68E6    223C 0003 243F                  MOVE.L  #$0003243F, D1
68EC    61 00 FF2A          (P)         BSR     Mult16
68F0    30C0                            MOVE.W  D0, (A0)+           ; Radar radians sd (f16)
68F2
68F2    4280                            CLR.L   D0
68F4    303A F84A           (P)         MOVE.W  CenNoise, D0
68F8    323C 0064                       MOVE.W  #100, D1
68FC    61 00 FF0A          (P)         BSR     Divide16
6900    2800                            MOVE.L  D0, D4              ; Store NoiseThr in (f16)
6902    EC88                            LSR.L   #6, D0
6904    30C0                            MOVE.W  D0, (A0)+           ; Noise threshold (f10)
6906
6906    4280                            CLR.L   D0
6908    303A F838           (P)         MOVE.W  CenManThr, D0
690C    323C 0064                       MOVE.W  #100, D1
6910    61 00 FEF6          (P)         BSR     Divide16
6914    EC88                            LSR.L   #6, D0
6916    30C0                            MOVE.W  D0, (A0)+           ; Manoeuvre threshold (f10)
6918
6918    4280                            CLR.L   D0
691A    303A F828           (P)         MOVE.W  CenMandy, D0
691E    323C 0064                       MOVE.W  #100, D1
6922    61 00 FEE4          (P)         BSR     Divide16
6926    30C0                            MOVE.W  D0, (A0)+           ; Manoeuvre decay factor (f10)
6928
6928    2003                            MOVE.L  D3, D0              ; Restore RadRanSD
692A    2204                            MOVE.L  D4, D1              ; Restore NoiseThr
692C    61 00 FEEA          (P)         BSR     Mult16
6930    4840                            SWAP    D0
6932    30C0                            MOVE.W  D0, (A0)+           ; Range noise threshold (int)
6934    4840                            SWAP    D0
6936    223C 0002 72F1                  MOVE.L  #$000272F1, D1
693C    61 00 FEDA          (P)         BSR     Mult16
6940    4840                            SWAP    D0
6942    30C0                            MOVE.W  D0, (A0)+           ; Max noise range error (int)
6944
6944    4280                            CLR.L   D0
6946    303A F824           (P)         MOVE.W  RadBearSD, D0
694A    2204                            MOVE.L  D4, D1
694C    61 00 FECA          (P)         BSR     Mult16
6950    30C0                            MOVE.W  D0, (A0)+           ; Bearing noise threshold (f16
6952    223C 0002 72F1                  MOVE.L  #$000272F1, D1
6958    61 00 FEBE          (P)         BSR     Mult16
695C    30C0                            MOVE.W  D0, (A0)+           ; Max noise bearing error (f16)
695E
695E    4280                            CLR.L   D0
6960    303A F7E4           (P)         MOVE.W  MaxKnots, D0
6964    323C 0E10                       MOVE.W  #3600, D1
6968    61 00 FE9E          (P)         BSR     Divide16
696C    223A F7EE           (P)         MOVE.L  RUPMile, D1
6970    61 00 FEA6          (P)         BSR     Mult16              ; MaxRanDiff = MaxKnots / 3600
6974    4281                            CLR.L   D1                  ;       * RUPMile * RevPerSec
6976    323A F7E0           (P)         MOVE.W  RevPerSec, D1
697A    ED89                            LSL.L   #6, D1
697C    61 00 FE9A          (P)         BSR     Mult16
6980    4840                            SWAP    D0
6982    30C0                            MOVE.W  D0, (A0)+           ; Max range difference (int)
6984
6984    3200                            MOVE.W  D0, D1
6986    4280                            CLR.L   D0
6988    303A F7BC           (P)         MOVE.W  MaxKnots, D0
698C    61 00 FE7A          (P)         BSR     Divide16
6990    EC88                            LSR.L   #6, D0
6992    30C0                            MOVE.W  D0, (A0)+           ; Speed factor (f10)
6994
6994    303A F7B2           (P)         MOVE.W  MaxBDC, D0
6998    C0FC 003A                       MULU    #$003A, D0          ; MaxBearDiff = MaxBDC * 5.5e-5
```

```
699C    E848                                LSR.W   #4, D0
699E    30C0                                MOVE.W  D0, (A0)+           ; Max bearing difference (f16)
69A0
69A0    223C 0003 243F                      MOVE.L  #$0003243F, D1
69A6    61 00 FE70          (P)             BSR     Mult16
69AA    30C0                                MOVE.W  D0, (A0)+           ; Max radians difference (f16)
69AC
69AC    4280                                CLR.L   D0
69AE    303C 00AF                           MOVE.W  #$00AF, D0
69B2    223A F7A8           (P)             MOVE.L  RUPMile, D1
69B6    61 00 FE60          (P)             BSR     Mult16
69BA    2A00                                MOVE.L  D0, D5              ; D5 = half G
69BC    41FA F7E2           (P)             LEA     ManPKnot + 10, A0
69C0    43FA F7E8           (P)             LEA     ManPG + 10, A1
69C4    3C3C 0004                           MOVE.W  #4, D6              ; D6 = missed look count
69C8
69C8    3206                        CalcFeas    MOVE.W  D6, D1
69CA    5241                                ADDQ.W  #1, D1
69CC    C2FA F78A           (P)             MULU    RevPerSec, D1       ; Delay = (MLC+1) * RPS
69D0    ED89                                LSL.L   #6, D1
69D2    2001                                MOVE.L  D1, D0
69D4    61 00 FE42          (P)             BSR     Mult16              ; DS = delay * delay
69D8    2800                                MOVE.L  D0,D4               ; D4 = delay squared
69DA
69DA    4281                                CLR.L   D1
69DC    323A F76C           (P)             MOVE.W  GSPerKK, D1
69E0    4841                                SWAP    D1
69E2    61 00 FE34          (P)             BSR     Mult16
69E6    4240                                CLR.W   D0
69E8    4840                                SWAP    D0
69EA    223C 0000 03E8                      MOVE.L  #1000, D1
69F0    61 00 FE16          (P)             BSR     Divide16            ; FTS = DS * GSPerKK / 1000
69F4    2205                                MOVE.L  D5,D1
69F6    61 00 FE20          (P)             BSR     Mult16
69FA    3100                                MOVE.W  D0, -(A0)           ; Manoeuvre per knot[]
69FC
69FC    2205                                MOVE.L  D5, D1
69FE    2004                                MOVE.L  D4, D0
6A00    61 00 FE16          (P)             BSR     Mult16
6A04    4840                                SWAP    D0
6A06    3300                                MOVE.W  D0, -(A1)           ; Manoeuvre per G[]
6A08
6A08    51CE FFBE           (P)             DBRA    D6, CalcFeas
6A0C
6A0C
6A0C                                ; **** Routines to derive the variable parameters - needed    ****
6A0C                                ; **** more frequently than the above subroutine.          ****
6A0C                               ;
6A0C    48E7 F080                   DerivePars  MOVEM.L D0-D3/A0, -(A7)     ; Get register space
6A10    3639 0003 5000                          MOVE.W  Constants, D3
6A16
6A16    3203                                    MOVE.W  D3, D1
6A18    0241 0078                               AND.W   #$0078, D1
6A1C    487A F706           (P)                 PEA     ConstWord
6A20    3EC1                                    MOVE.W  D1, (SP)+
6A22    0243 0018                               AND.W   #$0018, D3
6A26    E64B                                    LSR.W   #3, D3              ; Isolate C Range Shift
6A28    343C 0001                               MOVE.W  #1, D2
6A2C    E76A                                    LSL.W   D3, D2
6A2E    487A F758           (P)                 PEA     CRange
6A32    3EC2                                    MOVE.W  D2, (SP)+           ; Save C Range
6A34
6A34    0241 0060                               AND.W   #$0060, D1
6A38    EA49                                    LSR.W   #5, D1              ; Isolate C Scale & save
6A3A    487A F74E           (P)                 PEA     CScale
6A3E    3EC1                                    MOVE.W  D1, (SP)+
6A40
6A40    323A F6E0           (P)                 MOVE.W  RadarType, D1
6A44    41FA 9A36           (P)                 LEA     RangeFac, A0        ; Start of range factors table
6A48    3030 10 00                              MOVE.W  0(A0,D1.W), D0      ; Load range factor
6A4C    41FA 9A22           (P)                 LEA     MaxRaNum, A0
6A50    C0F0 10 00                              MULU    0(A0, D1.W), D0
6A54    E248                                    LSR.W   #1, D0              ; Adjust for binary point
6A56    E668                                    LSR.W   D3, D0              ; Shift right by C Range Shift
6A58
6A58    41FA F732           (P)                 LEA     MaxRSect, A0
6A5C    30C0                                    MOVE.W  D0, (A0)+           ; Max range sector
6A5E
6A5E    48C0                                    EXT.L   D0
6A60    2200                                    MOVE.L  D0, D1              ; Store max range sector
6A62    E488                                    LSR.L   #2, D0
6A64    30C0                                    MOVE.W  D0, (A0)+           ; Max range sector / 4
6A66
```

```
6A66    2001                                MOVE.L  D1, D0
6A68    80FC 0007                           DIVU    #7, D0
6A6C    30C0                                MOVE.W  D0, (A0)+           ; Max sector / 7
6A6E
6A6E    2001                                MOVE.L  D1, D0
6A70    80FC 000F                           DIVU    #15, D0
6A74    30C0                                MOVE.W  D0, (A0)+           ; Max sector / 15
6A76
6A76    303A F69C           (P)             MOVE.W  ThreshHol, D0
6A7A    C0C1                                MULU    D1, D0
6A7C    323A F6AA           (P)             MOVE.W  RadRanDM, D1
6A80    80C1                                DIVU    D1, D0
6A82    30C0                                MOVE.W  D0, (A0)+           ; Max black range sector
6A84
6A84    323A F6D4           (P)             MOVE.W  ShiftSR, D1
6A88    D243                                ADD.W   D3, D1
6A8A    5341                                SUBQ.W  #1, D1
6A8C    487A F6F8           (P)             PEA     ShiftClut
6A90    3EC1                                MOVE.W  D1, (SP)+           ; Shift for clutter range
6A92    4E75                                RTS                         ; End of derive parameters
6A94
6A94                                ; ****  Test to see if constants word has changed. If no   ****
6A94                                ; ****  change then no need to recalculate parameters.     ****
6A94                               ;
6A94
6A94    3039 0003 5000              CheckConst  MOVE.W  Constants, D0
6A9A    0240 0078                           AND.W   #$0078, D0
6A9E    B07A F684           (P)             CMP.W   ConstWord, D0
6AA2    66 00 FF68          (P)             BNE     DerivePars
6AA6    4E75                                RTS
6AA8
6AA8                                  ; ***************************************************************
6AA8                                ; *                                                             *
6AA8                                 ; *  End of miscellaneous calculation routines.                    *
6AA8                                 ; *  The routines which follow form the main part of the        *
6AA8                                 ; *  interface to the plot forming hardware.                    *
6AA8                                ; *                                                             *
6AA8                                 ; *  The first routine is the main driver program, called when  *
6AA8                                 ; *  starting the RATES software. The software loops in this    *
6AA8                                 ; *  routine until the tracker is started.                      *
6AA8                                ; *                                                             *
6AA8                                  ; ***************************************************************
6AA8                                ;
6AA8
6AA8    46FC 2700                   StartUp     MOVE.W  #$2700, SR          ; Mask off all interrupts
6AAC
6AAC    61 00 F72E          (P)             BSR     InitData            ; Initialise data areas
6AB0    61 00 F7DC          (P)             BSR     InitVDU             ; Initialise status VDU
6AB4
6AB4    4DFA 954E           (P)             LEA     SysName, A6         ; Print RATES intro
6AB8    61 00 F812          (P)             BSR     PrintStr
6ABC    3039 0003 5000                      MOVE.W  Constants, D0       ; and current radar type.
6AC2    0240 0007                           AND.W   #$0007, D0
6AC6    487A F65A           (P)             PEA     RadarType
6ACA    3EC0                                MOVE.W  D0, (SP)+
6ACC    61 00 FB72          (P)             BSR     Radar
6AD0    61 00 FD00          (P)             BSR     CopyParam           ; Copy dependent parameters
6AD4    61 00 FD70          (P)             BSR     DeriveFix
6AD8
6AD8    1ABC 0095                           MOVE.B  #$95, (A5)          ; Activate interupts on ACIA
6ADC    46FC 2200                           MOVE.W  #$2200, SR          ; and wait for the tracker to
6AE0                                                                    ; be started.
6AE0    41FA F628           (P)             LEA     StopRun, A0
6AE4    4A10                        Wait4Go     TST.B   (A0)
6AE6    67 FC               (P)             BEQ     Wait4Go
6AE8
6AE8    61 00 F804          (P)             BSR     PrPrompt
6AEC    46FC 2700                           MOVE.W  #$2700, SR          ; Mask off interupts again
6AF0
6AF0    3039 0003 5000              SysInit     MOVE.W  Constants, D0
6AF6    0240 0380                           AND.W   #$0380, D0          ; Mask out all but octant
6AFA    0C40 0280                           CMP.W   #$0280, D0
6AFE    66 F0               (P)             BNE.S   SysInit             ; Wait until octant 5
6B00
6B00    46FC 2500                           MOVE.W  #$2500,SR           ; Enable north interrupt
6B04    41FA F638           (P)             LEA     WarmRevs, A0
6B08    4E80 xxxx           (PX)    WarmWait    BSR     MipDa           ; Initialise variables
6B0C    4A50                                TST.W   (A0)
6B0E    66 F8               (P)             BNE.S   WarmWait            ; Warm up delay period.
6B10
6B10    3039 0003 5002                      MOVE.W  PlotCount, D0       ; Enable the DMA Interface
6B16    4E72 2000                   TryAgain    STOP    #$2000          ; Enable all interupts
6B1A    60 FA               (P)             BRA.S   TryAgain            ; and wait for north.
```

```
6B1C                                    ; ************************************************************
6B1C                                  ; *                                                          *
6B1C                                    ; *  This routine is called whenever an OCTANT DEMAND irq is  *
6B1C                                    ; *  received. FirstOct is only called once - on subsequent   *
6B1C                                    ; *  occasions the vector points to OctDemand. Both routines  *
6B1C                                    ; *  clear the stack to a known state (a stack reset), then   *
6B1C                                    ; *  push the interrupt return address and status register. At *
6B1C                                    ; *  the end of the subroutine, execution jumps to the chosen *
6B1C                                    ; *  interrupt return address.                               *
6B1C                                  ; *                                                          *
6B1C                                    ; ************************************************************
6B1C                                  ;
6B1C
6B1C      46FC 2700                     FirstOct    MOVE.W   #$2700, SR          ; Mask off all interupts
6B20      2E7C 0000 0800                            MOVEA.L  #$0800, A7          ; Reset stack pointer
6B26      41C0 xxxx           (PX)                 LEA      FirstTrack, A0
6B2A      2F08                                     MOVE.L   A0, -(A7)           ; Push start address of tracker
6B2C      4EC0 xxxx           (PX)                 BRA      OctProcess
6B30
6B30      46FC 2700                     OctDemand   MOVE.W   #$2700, SR          ; Mask interrupts
6B34      2E7C 0000 0800                            MOVEA.L  #$0800, A7          ; Reset stack pointer
6B3A      41C0 xxxx           (PX)                 LEA      Tracker, A0
6B3E      2F08                                     MOVE.L   A0, -(A7)           ; Normal tracker address
6B40
6B40      3F3C 2000                     OctProcess  MOVE.W   #$2000, -(A7)       ; Push correct value of SR
6B44      4E80 xxxx           (PX)                 BSR      DataOut             ; Output calculated data
6B48      4E80 xxxx           (PX)                 BSR      ResetOctant         ; Reset octant parameters
6B4C      4A43                                     TST.W    D3
6B50      xx                  (R)                  BNE.S    NotNorth
6B50
6B50      41FA F66E           (P)                  LEA      TotFInput, A0
6B54      4258                                     CLR.W    (A0)+               ; On north only, clear data
6B56      4258                                     CLR.W    (A0)+
6B58      4258                                     CLR.W    (A0)+
6B5A      4258                                     CLR.W    (A0)+
6B5C      4258                                     CLR.W    (A0)+
6B5E
6B5E      4E80 xxxx           (PX)      NotNorth   BSR      MipDa
6B62      4E73                                     RTE
6B64
6B64
6B64                                    ; ****  Routine called after an octant demand interrupt.   ****
6B64                                    ; ****  FirstTrack is only called once to print prompt and  ****
6B64                                    ; ****  reset the interrupt vector. Routines are called to  ****
6B64                                    ; ****  output the clutter cell and threshold values.      ****
6B64                                  ;
6B64
6B64      2E 2E 2E 54 72 61 63 6B 65 72 20 69 6E 69 74 69 61 74 65 64 20 00
                                        InitStr     DC.B     '...Tracker initiated ', Null
6B7A                                              .ALIGN   2
6B7A
6B7A      4DFA FFE8           (P)       FirstTrack  LEA      InitStr, A6         ; Print initialised message
6B7E      61 00 F74C          (P)                   BSR      PrintStr
6B82      61 00 F76A          (P)                   BSR      PrPrompt
6B86      487A FFA8           (P)                   PEA      OctDemand
6B8A      21DF 0068                                 MOVE.L   (SP)+, IRQ2         ; Replace octant irq vector
6B8E
6B8E      4E80 xxxx           (PX)      Tracker     BSR      ClutterOut
6B92      4E80 xxxx           (PX)                  BSR      ThreshOut
6B96      46FC 2000                                 MOVE.W   #$2000, SR
6B9A
6B9A      487A F63A           (P)       WaitLoop    PEA      Loading
6B9E      525F                                      ADD.W    #1, (SP)+           ; When idle, increment value
6BA0                                                                            ; to assess loading factor.
6BA0      323C 03E8                                 MOVE.W   #1000, D1
6BA4      51C9 FFFE           (P)       DelayLoop   DBRA     D1, DelayLoop
6BA8
6BA8      60 F0               (P)                   BRA      WaitLoop
6BAA
6BAA
6BAA                                    ; ************************************************************
6BAA                                  ; *                                                          *
6BAA                                    ; *  This routine is called whenever an NORTH irq is received. *
6BAA                                    ; *  It updates the clutter map and checks to see if any of   *
6BAA                                    ; *  the control-panel settings have changed, calculating new *
6BAA                                    ; *  derived parameters if necessary.                        *
6BAA                                  ; *                                                          *
6BAA                                    ; ************************************************************
6BAA                                  ;
6BAA
6BAA      46FC 2600                     NorthProc   MOVE.W   #$2600, SR          ; Mask most interrupts
6BAE      48E7 FFFE                                 MOVEM.L  D0-D7/A0-A6, -(A7)  ; Push all registers to stack
6BB2      487A F556           (P)                   PEA      StopRun
```

```
6BB6    4A1F                            TST.B   (SP)+
6BBA    xx              (R)             BEQ.S   TrackStop
6BBA
6BBA    61 00 F7DE      (P)             BSR     VDUMapUp            ; Update the status VDU
6BBE    487A F5F6       (P)             PEA     RevNum
6BC2    525F                            ADD.W   #1, (SP)+
6BC4    41FA F578       (P)             LEA     WarmRevs, A0
6BC8    4A50                            TST.W   (A0)                ; Is tracker running?
6BCC    xx              (R)             BEQ.S   Running
6BCC
6BCC    5350                            SUB.W   #1, (A0)            ; Decrement warm-up count
6BCE
6BCE    61 00 FEC4      (P)     Running BSR     CheckConst
6BD2
6BD2    61 00 F742      (P)     TrackStop BSR   VDUOut
6BD6    4CDF 7FFF                       MOVEM.L (A7)+, D0-D7/A0-A6  ; Restore registers
6BDA    4E73                            RTE
6BDC
6BDC
6BDC                                ; ************************************************************
6BDC                               ; *                                                          *
6BDC                                ; *  Transfer data, block by block, to the appropriate buffers *
6BDC                                ; *  in the plot extractor hardware. This version is designed  *
6BDC                                ; *  around an 8MHz CPU rate with normal DTACK delays, and     *
6BDC                                ; *  completes the operation using 32 loops.                      *
6BDC                               ; *                                                          *
6BDC                                ; ************************************************************
6BDC                               ;
6BDC
6BDC    48E7 F0E0               DataOut MOVEM.L D0-D3/A0-A2, -(A7)
6BE0    43FA CEA4       (P)             LEA     SynthVal, A1        ; Load address pointers
6BE4    45FA AEA0       (P)             LEA     ThreshVal, A2
6BE8    41FA 9E9C       (P)             LEA     CellBlock, A0
6BEC
6BEC    303A F5BC       (P)             MOVE.W  Octant, D0          ; Set correct output octant
6BF0    323C 000A                       MOVE.W  #10, D1
6BF4    E368                            LSL.W   D1, D0              ; Generate octant value
6BF6    D0C0                            ADDA.W  D0, A0              ; and add to form CFAR outpu
6BF8    D4C0                            ADDA    D0, A2              ; and threshold addresses.
6BFA
6BFA    363C 001F                       MOVE.W  #31, D3             ; Set transfer block count
6BFE    4A79 0003 5000          ScanWait TST.W  Constants
6C04    6A F8           (P)             BPL.S   ScanWait            ; Wait for scan period
6C06
6C06    323C 000F                       MOVE.W  #15,D1              ; Set transfer block size
6C0A    33D9 0003 2000          TxScan  MOVE.W  (A1)+, Synth
6C10    51C9 FFF8       (P)             DBRA    D1, TxScan          ; Transfer synthetics data
6C14
6C14    323C 000F                       MOVE.W  #15,D1              ; Set transfer block size
6C18    4A79 0003 5000          InterWait TST.W Constants
6C1E    6B F8           (P)             BMI.S   InterWait           ; Wait for interscan
6C20
6C20    30DA                    TxInter MOVE.W  (A2)+, (A0)+        ; Copy threshold words to B3
6C22    51C9 FFFC       (P)             DBRA    D1, TxInter
6C26
6C26    51CB FFD6       (P)             DBRA    D3, ScanWait        ; Decrement block count
6C2A    4CDF 070F                       MOVEM.L (A7)+, D0-D3/A0-A2
6C2E    4E75                            RTS
6C30
6C30
6C30                                ; ************************************************************
6C30                               ; *                                                          *
6C30                                ; *  This routine is called after every octant demand to reset *
6C30                                ; *  the octant-dependent parameters.                        *
6C30                               ; *                                                          *
6C30                                ; ************************************************************
6C30                               ;
6C30
6C30    48E7 E080               ResetOctant MOVEM.L D0-D2/A0, -(SP)
6C34    487A F598       (P)             PEA     NumFOP
6C38    425F                            CLR.W   (SP)+               ; Clear filter output word
count
6C3A    3039 0003 5000                  MOVE.W  Constants, D0
6C40    3400                            MOVE.W  D0, D2
6C42    E54A                            LSL.W   #2, D2
6C44    0642 0200                       ADD.W   #$0200, D2
6C48    0242 0E00                       AND.W   #$0E00, D2          ; Calculate cell address offse
6C4C    41FA 9E38       (P)             LEA     CellBlock, A0
6C50    D0C2                            ADDA.W  D2, A0
6C52
6C52    343C 00FF                       MOVE.W  #255,D2
6C56
6C56    4298                    ClearCell CLR.L (A0)+               ; Clear clutter cells
```

```
6C58    51CA FFFC           (P)             DBRA    D2, ClearCell
6C5C    41FA F54C           (P)             LEA     Octant, A0
6C60
6C60    323C 0007                           MOVE.W  #7, D1
6C64    E268                                LSR.W   D1, D0
6C66    0240 0007                           AND.W   #7, D0              ; Generate the octant number
6C6A    3600                                MOVE.W  D0, D3
6C6C    30C0                                MOVE.W  D0, (A0)+           ; and save.
6C6E
6C6E    5340                                SUBQ.W  #1, D0
6C70    0240 0007                           AND.W   #7, D0
6C74    30C0                                MOVE.W  D0, (A0)+           ; Previous octant
6C76
6C76    5340                                SUBQ.W  #1, D0
6C78    0240 0007                           AND.W   #7, D0
6C7C    30C0                                MOVE.W  D0, (A0)+           ; Second previous octant
6C7E
6C7E    5640                                ADDQ.W  #3, D0
6C80    0240 0007                           AND.W   #7, D0
6C84    30C0                                MOVE.W  D0, (A0)+           ; Next octant
6C86
6C86    5240                                ADDQ.W  #1, D0
6C88    0240 0007                           AND.W   #7, D0
6C8C    30C0                                MOVE.W  D0, (A0)+           ; Second octant on
6C8E
6C8E    5440                                ADDQ.W  #2, D0
6C90    0240 0007                           AND.W   #7, D0
6C94    30C0                                MOVE.W  D0, (A0)+           ; Fourth octant on
6C96
6C96    4CDF 0107                           MOVEM.L (SP)+, D0-D2/A0
6C9A    4E75                                RTS
6C9C
6C9C                                    ; ************************************************************
6C9C                                    ; *                                                          *
6C9C                                    ; *  This routine is called to terminate previous DMA operatn *
6C9C                                    ; *  and restart it with a new word count and transfer address *
6C9C                                    ; *  It is called every octant after data output and parameter *
6C9C                                    ; *  reset.                                                  *
6C9C                                    ; *                                                          *
6C9C                                    ; ************************************************************
6C9C                                    ;
6C9C    48E7 C0E0                       MipDa   MOVEM.L D0-D1/A0-A2, -(A7)
6CA0    41F9 0003 8000                          LEA     DmaChan0, A0        ; Load DMA base address
6CA6    43FA D45E           (P)                 LEA     Base, A1
6CAA    0828 0003 0000                          BTST.B  #3, 0(A0)           ; Is it still running?
6CB2    xx                  (R)                 BEQ.S   NoDmaAct
6CB2
6CB2    117C 0010 0007                          MOVE.B  #$10, 7(A0)         ; Software abort DMA ch 0
6CB8    1168 0000 0000              NoDmaAct    MOVE.B  0(A0), 0(A0)        ; Reset DMA status register
6CBE    2028 000C                               MOVE.L  $C(A0), D0          ; Read next transfer address
6CC2    9089                                    SUB.L   A1, D0              ; Subtract base address
6CC4    E448                                    LSR.W   #2, D0
6CC6    487A F504           (P)                 PEA     NumFIW
6CCA    3EC0                                    MOVE.W  D0, (SP)+           ; Divide by 2 to give no of
6CCC    487A F4F2           (P)                 PEA     TotFInput           ; plots read in.
6CD0    D15F                                    ADD.W   D0, (SP)+
6CD2
6CD2    45FA D436           (P)                 LEA     FirstIPA, A2
6CD6    303A F4D2           (P)                 MOVE.W  Octant, D0
6CDA    0800 0000                               BTST.B  #0, D0
6CE0    xx                  (R)                 BNE.S   EvenOct             ; Is this an even octant?
6CE0    D5FC 0000 2000                          ADDA.L  #$2000, A2
6CE6
6CE6    228A                        EvenOct     MOVE.L  A2, (A1)            ; New base address
6CE8    214A 000C                               MOVE.L  A2, $C(A0)          ; Store in dma controller
6CEC    317C 1000 000A                          MOVE.W  #$1000, $A(A0)      ; Set count value
6CF2    117C 0080 0007                          MOVE.B  #$80, 7(A0)         ; and start operation
6CF8    3039 0003 5002                          MOVE.W  PlotCount, D0
6CFE
6CFE    4CDF 0703                               MOVEM.L (A7)+, D0-D1/A0-A2  ; Restore registers
6D02    4E75                                    RTS
6D04
6D04                                    ; ************************************************************
6D04                                    ; *                                                          *
6D04                                    ; *  Update the values in the clutter cell table, based on the *
6D04                                    ; *  presence of targets in a cell causing an increment in the *
6D04                                    ; *  clutter count for that location. A maximum of 512 synth.  *
6D04                                    ; *  characters are also sent to the output buffer is DispIPC  *
6D04                                    ; *  is turned on.                                           *
6D04                                    ; *                                                          *
6D04                                    ; ************************************************************
```

```
6D04    48E7 FCF0              ClutterOut  MOVEM.L  D0-D5/A0-A3, -(SP)
6D08    207A D3FC      (P)                 MOVEA.L  Base, A0          ; Load start of plots addres
6D0C    43FA CD78      (P)                 LEA      SynthVal, A1      ; and synthetics address
6D10    45FA 9D74      (P)                 LEA      CellBlock, A2     ; and clutter byte address
6D14    303A F470      (P)                 MOVE.W   ShiftClut, D0     ; Load shift for clutter range
6D18
6D18    323A F4B8      (P)                 MOVE.W   NumSynth, D1      ; D1 is no. synthetics * 2
6D1C    D241                               ADD.W    D1, D1
6D1E    D2C1                               ADDA.W   D1, A1            ; Form synthetics address
6D20
6D20    343A F4AA      (P)                 MOVE.W   NumFIW, D2
6D24    5342                               SUBQ.W   #1, D2            ; D1 is no. FIW - 1
6D28    xx             (R)                 BMI.S    NoClutter
6D28
6D28    3610                   MoreClutr   MOVE.W   (A0), D3          ; Load plot bearing
6D2A    3828 0002                          MOVE.W   2(A0), D4         ; and range
6D2E    E06C                               LSR.W    D0, D4            ; Shift plot range
6D30    0C44 003F                          CMP.W    #63, D4
6D36    xx             (R)                 BLE.S    CheckCell
6D36
6D36    383C 003F                          MOVE.W   #63, D4
6D3A    47FA F458      (P)     CheckCell   LEA      MaxOn15 + 2, A3
6D3E    3A3C 0002                          MOVE.W   #2, D5
6D42    B863                   NoMRSMatch  CMP.W    -(A3), D4         ; Compare against max range
6D44    5ECD FFFC      (P)                 DBGT     D5, NoMRSMatch    ; sector values.
6D48
6D48    5845                               ADDQ.W   #4, D5            ; Offset comparison result.
6D4A    E86B                               LSR.W    D4, D3            ; Re-scale plot bearing
6D4C    0243 01C0                          AND.W    #$01C0, D3
6D50    D843                               ADD.W    D3, D4            ; Add range & bearing to form
6D52    1A32 40 00                         MOVE.B   0(A2, D4.W), D5   ; index into clutter cell table
6D56    0C05 000F                          CMP.B    #15, D5
6D5C    xx             (R)                 BGE.S    AtMaxVal
6D5C
6D5C    5232 00 00                         ADDQ.B   #1,0(A2,D0.W)     ; Increment clutter cell value
6D60    487A F3AA      (P)     AtMaxVal    PEA      DispIPC
6D64    081F 0007                          BTST.B   #7, (SP)+
6D6A    xx             (R)                 BEQ.S    EndCUpdate
6D6A
6D6A    0C41 0400                          CMP.W    #1024, D1         ; Clutter cell count at 1024?
6D6E    6C 00 xxxx     (R)                 BGE      EndCUpdate
6D72
6D72    5841                               ADDQ.W   #4, D1
6D74    3618                               MOVE.W   (A0)+, D3         ; Load bearing
6D76    3818                               MOVE.W   (A0)+, D4         ; Load range
6D78    E64B                               LSR.W    #3, D3            ; Scale bearing down
6D7A    C8FA F3E6      (P)                 MULU     SynRanF, D4       ; Multiply by synthetics r.f
6D7E    4844                               SWAP     D4
6D80    0244 03FF                          AND.W    #$03FF, D4
6D84    0044 0800                          OR.W     #$0800, D4
6D88    32C3                               MOVE.W   D3, (A1)+         ; Save bearing & range values
6D8A    32C4                               MOVE.W   D4, (A1)+         ; as synthetics address.
6D8C
6D8C    51CA FF9A      (P)     EndCUpdate  DBRA     D2, MoreClutr     ; Next set of input words
6D90    E249                               LSR.W    #1, D1
6D92    0241 01FE                          AND.W    #510, D1
6D96    487A F43A      (P)                 PEA      NumSynth
6D9A    3EC1                               MOVE.W   D1, (SP)+         ; Store number of synthetics
6D9C
6D9C    4CDF 0F3F              NoClutter   MOVEM.L  (SP)+, D0-D5/A0-A3
6DA0    4E75                               RTS
6DA2
6DA2
6DA2
6DA2                           ; ************************************************************
6DA2                           ; *                                                          *
6DA2                           ; *  This routine calculates and stores the threshold values  *
6DA2                           ; *  based on the clutter level within a given area: If      *
6DA2                           ; *  is high, the level is doubled & if low is halved. Upper  *
6DA2                           ; *  and lower thresholds are imposed.                       *
6DA2                           ; *                                                          *
6DA2                           ; ************************************************************
6DA2                           ;
6DA2
6DA2    48E7 F880              ThreshOut   MOVEM.L  D0-D4/A0, -(SP)
6DA6    303A F402      (P)                 MOVE.W   Octant, D0
6DAA    323C 000A                          MOVE.W   #10, D1
6DAE    E368                               LSL.W    D1, D0            ; Calculate octant part
6DB0
6DB0    323A F3E2      (P)                 MOVE.W   MaxBlack, D1      ; D1 is max black range sector
6DB4    4242                               CLR.W    D2                ; D2 is bearing part
6DB6
6DB6    41FA ACCE      (P)     NextBear    LEA      ThreshVal, A0
```

```
6DBA    D0C0                                ADDA.W   D0, A0
6DBC    D0C2                                ADDA.W   D2, A0              ; Form cell address
6DBE
6DBE    4243                                CLR.W    D3                  ; D3 is range sector
6DC0    30FC 007F                ClrBlack   MOVE.W   #127, (A0)+         ; Set black range cell values
6DC4    5243                                ADD.W    #1, D3
6DC6    B243                                CMP.W    D3, D1
6DC8    6F F6            (P)                BLE.S    ClrBlack
6DCA
6DCA    3F03                     NextRange  MOVE.W   D3, -(SP)           ; Push range part
6DCC    3800                                MOVE.W   D0, D4
6DCE    D842                                ADD.W    D2, D4
6DD0    E24C                                LSR.W    #1, D4
6DD2    3F04                                MOVE.W   D4, -(SP)           ; Push bearing part
6DD4    4E80 xxxx        (PX)               BSR      CalThrVal
6DD8    30DF                                MOVE.W   (SP)+, (A0)+        ; Pop new threshold value
6DDA    5243                                ADD.W    #1, D3
6DDC    B67A F3AE        (P)                CMP.W    MaxRSect, D3
6DE0    66 E8            (P)                BNE.S    NextRange
6DE2
6DE2    0642 0080                           ADD.W    #128, D2
6DE6    0C42 0400                           CMP.W    #1024, D2
6DEA    66 CA            (P)                BNE.S    NextBear
6DEC
6DEC    4CDF 011F                           MOVEM.L  (SP)+, D0-D4/A0
6DF0    4E75                                RTS
6DF2
6DF2
6DF2
6DF2                                  ; ***************************************************************
6DF2                                ; *                                                             *
6DF2                                 ; *  Calculate new threshold values for a given cell. The      *
6DF2                                 ; *  routine is passed the range part & bearing part as        *
6DF2                                 ; *  parameters. It returns the threshold. This may be equal   *
6DF2                                 ; *  to a fixed value if calculation is turned off.           *
6DF2                                ; *                                                             *
6DF2                                  ; ***************************************************************
6DF2                                ;
6DF2
6DF2    48E7 F0C0                CalThrVal  MOVEM.L  D0-D3/A0-A1, -(SP)
6DF6    487A F31B        (P)                PEA      Thresh
6DFA    081F 0007                           BTST.B   #7, (SP)+
6E00    xx               (R)                BNE.S    CalcLev
6E00
6E00    343A F314        (P)                MOVE.W   ThreshLev, D2       ; Return fixed value
6E04    4EC0 xxxx        (PX)               BRA      EndCalcs
6E08
6E08    322F 001C                CalcLev    MOVE.W   24 + 4(A7), D1      ; Pop bearing part
6E0C    302F 001E                           MOVE.W   24 + 6(A7), D0      ; Pop range part
6E10
6E10    43FA F382        (P)                LEA      MaxOn15 + 2, A1
6E14    343C 0002                           MOVE.W   #2,D2
6E18    B061                     TestLev    CMP.W    -(A1), D0           ; Find the range shift value
6E1A    5EC8 FFFC        (P)                DBGT     D0, TestLev
6E1E    5242                                ADDQ.W   #1, D2
6E20
6E20    3601                                MOVE.W   D1, D3              ; D1 is full bearing part
6E22    E46B                                LSR.W    D2, D3              ; Shift bearing part
6E24    0243 0FC0                           AND.W    #$0FC0, D3
6E28    D640                                ADD.W    D0, D3              ; add threshold part.
6E2A
6E2A    43FA 9C5A        (P)                LEA      CellBlock, A1
6E2E    1631 30 00                          MOVE.B   0(A1, D3.W), D3     ; Load clutter density
6E32    4883                                EXT.W    D3
6E34
6E34    43FA AC50        (P)                LEA      ThreshVal, A1
6E38    D041                                ADD.W    D1, D0
6E3A    E348                                LSL.W    #1, D0
6E3C    3431 00 00                          MOVE.W   0(A1, D0.W), D2     ; Read present threshold level
6E40
6E40    B67A F314        (P)                CMP.W    UpperThr, D3        ; Compare upper threshold with
6E44    6F 00 xxxx       (R)                BLE      LTUpper
6E48
6E48    4A42                                TST.W    D2                  ; Is present level zero?
6E4C    xx               (R)                BNE.S    NonZero
6E4C    343C 0001                           MOVE.W   #1, D2              ; Result is '1'
6E50    4EC0 xxxx        (PX)               BRA      EndCalcs
6E54
6E54    0C42 0040                NonZero    CMP.W    #64, D2
6E5A    xx               (R)                BLT.S    LT64
6E5A    343C 007F                           MOVE.W   #127, D2            ; Set to max threshold
6E5E    4EC0 xxxx        (PX)               BRA      EndCalcs
6E62
```

```
6E62    D442                            LT64        ADD.W   D2, D2              ; Double present threshold
6E64    4EC0 xxxx           (PX)                    BRA     EndCalcs
6E68
6E68    B67A F2EA           (P)         LTUpper     CMP.W   LowerThr, D3
6E6E    xx                  (R)                     BGE.S   EndCalcs
6E6E    E24A                                        LSR.W   #1, D2              ; Halve present threshold
6E70
6E70    3F42 0028                       EndCalcs    MOVE.W  D2, 32 + 8(SP)
6E74    4CDF 013F                                   MOVEM.L (SP)+, D0-D5/A0
6E78    2F57 0002                                   MOVE.L  (A7), 2(A7)
6E7C    DFFC 0000 0002                              ADDA.L  #2, A7
6E82    4E75                                        RTS
6E84
```

APPENDIX D

Source Listing
for
Moving Target Generator

(Written in ADA, runs under CP/M on Cifer 2684)

```
B>TYPE  TARGET.ADA

PROCEDURE test_target  IS

      -- FIRSTLY SET CONSTANTS FOR TYPE 992 RADAR

 radar_range : CONSTANT FLOAT := 4390.0;
 rupm : CONSTANT FLOAT := 48.78;
 inv_speed_factor : CONSTANT FLOAT := 5.42E-2;

 TYPE command IS (highlight,double_width,graph,blink,old_clear);
 esc : CONSTANT CHARACTER := CHARACTER'VAL(27);

 atan,ident : INTEGER;
 tt_range : ARRAY(1..3) OF INTEGER;
 tt_bearing : ARRAY(1..3) OF INTEGER;
 tt_along : ARRAY(1..3) OF FLOAT;
 tt_cross : ARRAY(1..3) OF FLOAT;
 tt_available : ARRAY(1..3) OF BOOLEAN;
 distance,bearing,heading,speed : INTEGER;
 trig : ARRAY (0..80) OF FLOAT;
 sin,cos,hypot : FLOAT;


------------------------------------------------------------
--  Subroutine to manipulate the graphics capabilities of
--  the screen. 'OP' is one of 'highlight' 'double_width'
--  'blink', 'graph' or 'old_clear'.'DIR' is '1' to
--  turn the effect on or '0' to turn it off.
------------------------------------------------------------


PROCEDURE screen(op : command; dir : INTEGER) IS

BEGIN

   PUT(esc);
   CASE op IS
       WHEN highlight =>
           IF dir = 1 THEN PUT('N');
           ELSE PUT('O'); PUT(esc); PUT('+'); END IF;
       WHEN double_width =>
           IF dir = 1 THEN PUT('E'); ELSE PUT('I'); END IF;
       WHEN graph =>
           PUT('*'); PUT(CHARACTER'VAL(dir + 64));
       WHEN blink =>
           IF dir = 1 THEN PUT('F'); ELSE PUT('G'); END IF;
       WHEN old_clear =>
           PUT('M');  screen(double_width,0);
   END CASE;

END screen;
------------------------------------------------------------

--  Draw box and then annotate the user display

------------------------------------------------------------


PROCEDURE screen_initialise IS

   PROCEDURE horizontal IS
     -- Draw horizontal lines
   BEGIN
       screen(highlight,1);
       screen(graph,9); PUT(' ');
       FOR i IN 1..5 LOOP
           screen(graph,11);
           FOR j IN 1..12 LOOP
               PUT(' ');
           END LOOP;
           screen(graph,12); PUT(' ');
       END LOOP;
       new_line;
   END horizontal;

   PROCEDURE vertical( line,code : INTEGER ) IS
     --  Draw vertical lines
   BEGIN
```

```
        screen(graph,9); PUT(' ');
        FOR i IN 1..5 LOOP
            screen(graph,0);
            FOR j IN 1..12 LOOP
                PUT(' ');
            END LOOP;
            screen(graph,9); PUT(' ');
        END LOOP;
        new_line;
    END vertical;


 BEGIN
    put(esc); put('J');
    put(esc); put('V');
    new_line;
    screen(old_clear,0);
    screen(highlight,1);
    screen(double_width,1);
    put("RATES        992  TEST TARGET GENERATOR");
    screen(highlight,0);
    screen(double_width,0);
    new_line; new_line; new_line;
    horizontal;
    screen(graph,9); put(' '); screen(graph,0); put("NUMBER");
    screen(graph,9); PUT(' '); screen(graph,0); PUT("RANGE");
    screen(graph,9); PUT(' '); screen(graph,0); PUT("BEARING");
    screen(graph,9); PUT(' '); screen(graph,0); PUT("SPEED");
    screen(graph,9); PUT(' '); screen(graph,0); PUT("HEADING");
    screen(graph,9); PUT(' ');
    new_line;
    FOR i IN 1..3 LOOP
        horizontal;
        FOR j IN 1..3 LOOP
            vertical(j,i);
        END LOOP;
    END LOOP;
    screen(graph,9); PUT(' ');
    FOR i IN 1..5 LOOP
        screen(graph,14);
        FOR j IN 1..12 LOOP PUT(' '); END LOOP;
        screen(graph,13); PUT(' ');
    END LOOP;
    screen(graph,0);
    screen(highlight,0);
END screen_initialise;

-------------------------------------------------------------
--  Direct cursor to appropriate input position
-------------------------------------------------------------

PROCEDURE position(target_number,item : INTEGER) IS

BEGIN
    PUT(esc); PUT('P');
    PUT(CHARACTER'VAL(item*13+36));
    PUT(CHARACTER'VAL(target_number*4+36));
END position;

-------------------------------------------------------------
--  Turn blinking target i.d. on (or off) if selected
-------------------------------------------------------------

PROCEDURE number(flash,target_number : INTEGER) IS

BEGIN
    screen(highlight,1);
    position(target_number,0);
    IF flash = 1 THEN screen(blink,1); END IF;
    screen(double_width,1);
    PUT("TT"); PUT(target_number);
    screen(double_width,0);
    screen(blink,0);
END number;

-------------------------------------------------------------
--  Clear display of chosen target parameters
-------------------------------------------------------------
```

```
PROCEDURE clear_line(target_number : INTEGER) IS

BEGIN
   FOR i IN 1..4 LOOP
      position(target_number,i);
      PUT("        ");
   END LOOP;
END clear_line;

------------------------------------------------------------
--  Use direct keyboard input to read chosen target number
------------------------------------------------------------

PROCEDURE get_target_number(code : OUT INTEGER) IS

BEGIN
   code := bdos(6,255) - 48;
   IF code > 0 AND code < 4 THEN
      clear_line(code);
      number(1,code);
   ELSE
      IF code = -21 THEN
         PUT(esc); PUT('W');
         BDOS(0);
      ELSE code := 0;
      END IF;
   END IF;
END get_target_number;

------------------------------------------------------------
--  Read range, bearing, speed & heading for chosen target
------------------------------------------------------------

PROCEDURE read_data(code : INTEGER) IS

BEGIN
   IF code > 0 AND code < 4 THEN
      FOR j IN 1..4 LOOP
         position(code,j);
         CASE j IS
            WHEN 1 => GET(distance);
                      IF distance = 0 THEN exit; END IF;
            WHEN 2 => GET(bearing);
            WHEN 3 => GET(speed);
            WHEN 4 => GET(heading);
         END CASE;
      END LOOP;
      IF distance /= 0 THEN tt_available(code) := true;
      ELSE tt_available(code) := false; END IF;
      number(0,code);
   END IF;

END read_data;

------------------------------------------------------------
--          START OF MAIN CALCULATING ROUTINES  --
------------------------------------------------------------

PROCEDURE set_constants IS

BEGIN
   trig(0):=0.0;  trig(1):=1606.0;  trig(2):=3196.0;
   trig(3):=4756.0;  trig(4):=6270.0;  trig(5):=7723.0;
   trig(6):=9102.0;  trig(7):=10394.0;  trig(8):=11585.0;  trig(9):=12665.0;
   trig(10):=13623.0;  trig(11):=14449.0;  trig(12):=15137.0;  trig(13):=15679.0;
   trig(14):=16069.0;  trig(15):=16305.0;  trig(16):=16384.0;
   FOR i IN 0..16 LOOP trig(i):=trig(i)*0.6104E-4; END LOOP;
   FOR i IN 0..15 LOOP trig(32-i) := trig(i); END LOOP;
   FOR i IN 0..31 LOOP trig(64-i) := -trig(i); END LOOP;
   FOR I IN 1..3 LOOP
      tt_available(i) := false; number(0,i);
   END LOOP;

END set_constants;

------------------------------------------------------------
--  Calculate a square root by Newton's approximation
------------------------------------------------------------
```

```
FUNCTION sqrt(dist : FLOAT) RETURN FLOAT IS

   guess,delt : FLOAT;

BEGIN

   IF dist = 0.0 THEN RETURN 0.0;
   ELSIF dist = 1.0 THEN RETURN 1.0;
   ELSE
      guess := dist / 2.0;
      LOOP
         delt:=guess - (guess * guess - dist)/(2.0 * guess);
         EXIT WHEN ABS(guess - delt) < 0.5;
         guess := delt;
      END LOOP;
      RETURN delt;
   END IF;

END sqrt;

------------------------------------------------------------
--  Calculate sin and cosine using a look-up table
------------------------------------------------------------

PROCEDURE sin_and_cos (angle : INTEGER) IS

   rads_per_degree : CONSTANT FLOAT := 1.74533E-2;
   pi_by_32 : CONSTANT FLOAT := 9.81748E-2;
   element : INTEGER;
   x,s,c,sin_rads,cos_rads : FLOAT;

BEGIN
   x := FLOAT(angle) * rads_per_degree;
   element := INTEGER (x / pi_by_32);
   s := trig(element);
   c := trig(16+element);
   sin_rads := x - FLOAT(element) * pi_by_32;
   cos_rads := 1.0 - sin_rads * sin_rads * 0.5;
   sin := s * cos_rads + c* sin_rads;
   cos := c * cos_rads - s * sin_rads;

END sin_and_cos;

------------------------------------------------------------
--  Use Pythagoras to find hypotenuse of 'Q' & 'R' then use
--  an approximation to establish the arctangent
------------------------------------------------------------

PROCEDURE atan_and_hypot(q,r : FLOAT) IS

   x,x2,angle : FLOAT;

BEGIN

   hypot := sqrt(q*q + r*r);
   x := q/r;
   IF x < -0.4142136 THEN
      x := (x + 0.4142136) / (x * (-0.4142136) + 1.0);
      angle := -0.125;
   ELSIF x < 0.4142136 THEN
      x := (x - 0.4142136)/(x*0.4142136+1.0);
      angle := 0.125;
   ELSE angle := 0.0;
   END IF;
   x2 := x*x;
   atan  :=  INTEGER(((((0.016733)*x2+0.169562)*x2+0.4462159)/(0.1401829e1+x2)*x+angle)*0.32768e5);

END atan_and_hypot;

------------------------------------------------------------
--  Initialise a new target with appropriate along & cross vectors
------------------------------------------------------------

PROCEDURE set_target(code : INTEGER) IS

   beta : INTEGER;
   dist_per_rev : FLOAT;
```

```
BEGIN

    IF code > 0 AND code < 4 THEN
        tt_range(code) := INTEGER(FLOAT(distance) * rupm);
        tt_bearing(code):=INTEGER(FLOAT(bearing) * 0.182044E3);
        beta := heading - bearing;
        IF beta < 0 THEN
            beta := beta + 360;
        END IF;
        sin_and_cos(beta);
        dist_per_rev := FLOAT(speed) * inv_speed_factor;
        tt_along(code) := dist_per_rev * cos;
        tt_cross(code) := dist_per_rev * sin;
    END IF;

END set_target;

------------------------------------------------------------
--  Transfer each of the targets in turn to the test target
--  generator, calculating new position & updating stored
--  parameters before output.
------------------------------------------------------------

PROCEDURE output_targets IS

    along_range,ratio : FLOAT;

BEGIN

    FOR n IN 1..3 LOOP
        IF tt_available(n) THEN
            along_range := FLOAT(tt_range(n)) + tt_along(n);
            atan_and_hypot(tt_cross(n),along_range);
            IF hypot > radar_range THEN
                tt_along(n) := - tt_along(n);
                tt_cross(n) := - tt_cross(n);
            ELSE
                tt_bearing(n) := tt_bearing(n) + atan;
                ratio := (FLOAT(tt_range(n)) - 1.0) / hypot;
                tt_along(n) := hypot - along_range * ratio;
                tt_cross(n) := tt_cross(n) * ratio;
                tt_range(n) := INTEGER(hypot + 0.5);
                PUT(tt_range(n)); PUT("     ");
                PUT(tt_bearing(n));new_line;
            END IF;
        END IF;
    END LOOP;

END output_targets;

------------------------------------------------------------
--  The main program
------------------------------------------------------------

BEGIN
    screen_initialise;
    set_constants;
    LOOP
        get_target_number(ident);
        read_data(ident);
        set_target(ident);
        output_targets;
    END LOOP;
END;
```

APPENDIX E

Source Listing
for
Systolic CFAR Simulation

(Written in PASCAL, runs on Apple Macintosh)

```
program simulate;
 const
  width = 9;
  depth = 5;
  N = 6;

 type
  cell_type = record
    case code : integer of
     0 : (
       Xin, Xout : real;
       Zin, Yout : real;
       Xptr, Zptr : ^real
     );
     1 : (
       Ain, Bin, Sout : real;
       Aptr, Bptr : ^real
     );
   end;


 var
  cell : array[1..width, 1..depth] of cell_type;
  xval, cago : real;
  inv_n, one, manual, comp, inv_k, zero : real;
  source, res : text;
  pulse, id : integer;


{ Connect the elements of the array together }

 procedure connect;
  var
   x, y : integer;
 begin

  for x := 1 to 9 do
   for y := 2 to 4 do
    begin
     cell[x, y].code := 1;
     cell[x, y].Aptr := nil;
     cell[x, y].Bptr := nil;
    end;

  for x := 1 to 9 do
   begin
    cell[x, 1].code := 0;
    cell[x, 5].code := 0;
    cell[x, 5].Zptr := @one;
    if x < 7 then
     cell[x, 1].Zptr := @inv_n
    else
     cell[x, 1].Zptr := @one;
    if x > 1 then
     begin
      cell[x, 1].Xptr := @cell[x - 1, 1].Xout;
      cell[x, 5].Xptr := @cell[x - 1, 5].Xout;
     end
    else
     begin
      cell[x, 1].Xptr := @xval;
      cell[x, 5].Xptr := @cell[1, 4].Sout;
     end;
   end;

  for x := 1 to 3 do
   begin
    cell[(2 * x - 1), 2].Aptr := @cell[(2 * x - 1), 1].Yout;
    cell[(2 * x - 1), 2].Bptr := @cell[(2 * x), 1].Yout;
   end;
  cell[1, 3].Aptr := @cell[1, 2].Sout;
  cell[1, 3].Bptr := @zero;
  cell[3, 3].Aptr := @cell[3, 2].Sout;
  cell[3, 3].Bptr := @cell[5, 2].Sout;
  cell[1, 4].Aptr := @cell[1, 3].Sout;
  cell[1, 4].Bptr := @cell[3, 3].Sout;
```

```
  cell[9, 2].Aptr := @cell[9, 1].Yout;
  cell[9, 2].Bptr := @zero;
  cell[9, 3].Aptr := @cell[9, 2].Sout;
  cell[9, 3].Bptr := @zero;
 end;


{  Calculate the new value of the supplied element }

 procedure calculate (var element : cell_type);
 begin
  case element.code of

   0 :
    begin
     element.Xout := element.Xin;
     element.Yout := element.Xin * element.Zin;
    end;
   1 :
    element.Sout := element.Ain + element.Bin;
  end;
 end;


{ Read the new inputs to the chosen element }

 procedure shift (var element : cell_type);
 begin
  with element do
   begin
    case code of
     0 :
      begin
       if Xptr <> nil then
        Xin := Xptr^
       else
        Xin := 0;
       if Zptr <> nil then
        Zin := Zptr^
       else
        Zin := 0;
      end;
     1 :
      begin
       if Aptr <> nil then
        Ain := Aptr^
       else
        Ain := 0;
       if Bptr <> nil then
        Bin := Bptr^
       else
        Bin := 0;
      end;
    end;
   end;
 end;


{ Simulate rising edge of global clock }

 procedure move_data;
  var
   x, y : integer;
 begin
  for x := 1 to 9 do
   for y := 1 to 5 do
    begin
     shift(cell[x, y]);
    end;
 end;


{ Simulate falling edge of global clock }

 procedure do_calcs;
  var
   x, y : integer;
 begin
```

```
  for x := 1 to 9 do
   for y := 1 to 5 do
    begin
     calculate(cell[x, y]);
    end;
 end;


 function max (a, b : real) : real;
 begin
  if (a > b) then
   max := a
  else
   max := b;
 end;


{ Display / print the results }

 procedure show;
  var
   x, y : integer;
   cago_sum : real;
 begin
  cago_sum := 0;
  for x := 4 to 9 do
   begin
    cago_sum := cago_sum + cell[x, 1].Xout;
   end;
  cago_sum := cago_sum / 6;
  write(res, trunc(max(cell[1, 4].Sout, cell[9, 5].Xout)), chr(9));
  writeln(res, trunc(cell[9, 3].Sout));
  writeln('Systolic CAGO gives: ', max(cell[1, 4].Sout, cell[9, 5].Xout));
  writeln('Test cell: ', cell[9, 3].Sout);
  writeln;

 end;



{ The clock is split into two phases : the rising edge and the }
{  falling edge. On the rising edge, data is copied to the     }
{  element inputs. On the falling edge calculations are done   }

 procedure clock;
 begin
  pulse := pulse + 1;
  move_data;
  show;
  do_calcs;
 end;


 procedure initialise;
 begin
  open(source, 'Simulation Data');
  open(res, 'Results');
  rewrite(res);
  reset(source);
  pulse := 0;
  inv_n := 1 / N;
  one := 1.0;
  zero := 0.0;
 end;

begin
 initialise;
 connect;
 while not eof(source) do
  begin
   readln(source, id);
   readln(source, xval);
   readln(source, cago);
   clock;
  end;
 close(source);
 close(res)
end.
```

APPENDIX F

The Application of Systolic Arrays to Radar Signal Processing

(Published in I.E.E.E. Conference Proceedings, Radar '86)

The Application of Systolic Arrays To Radar Signal Processing

R.Spearman† C.T.Spracklen† J.H.Miles°

Abstract

Durham University, Department of Applied Physics and Electronics, have for many years been involved in the development of digital signal processing techniques for radar signals. More recently they have carried out major research projects for various semiconductor manufacturers on the design and simulation of very large scale integrated circuits and have become involved in the area of knowledge representation using massively parallel processors. This has naturally led to a realisation of the dramatic advantages to be gained from the incorporation of systolic arrays into digital systems and, in this paper, into digital radar signal processors.

Introduction

In this paper we present the design of a systolic array processor radar system and compare it to a conventional radar processor. It is shown that the use of very large scale integrated circuit techniques, in the form of systolic arrays, leads to the dramatic performance improvements normally associated with emitter coupled logic, while retaining the space and power saving advantages associated with very large scale integration and CMOS technology.

We show how systolic arrays can be used to replace the boards of high speed logic normally associated with a high performance radar implementing all of the normal processing functions associated with a general purpose digital processor, but at the speed associated with the fixed function logic arrays.

Modern radar systems demand both high performance and portability, and many improvements have been made in recent years, including the use of digital techniques. The gradual change from small scale integration to very large scale integrated (VLSI) techniques, and the use of microprocessor and special purpose digital signal processing elements have led to dramatic improvements in speed, size, cost and power consumption. Since, in most cases, production volumes are very low the use of general purpose processors is important. Although in recent years, sophisticated computer aided design tools have made it relatively straightforward to design semi-custom VLSI devices, it is only in the area of arrays of replicated structures that the necessary circuit density improvements have been made that will allow components of the complexity necessary to implement a radar processor.

In this paper an alternative approach to the general purpose programmable processor is proposed, based on the systolic array approach developed originally at Carnegie Mellon University.

Multi-Processor Architectures

The design of digital signal processing systems has increased in complexity over the past few years, beyond the point where a single processing element is used to perform a variety of sequential operations on an incoming data stream. To achieve a greater throughput, the same element can be replicated to spread the load more evenly. The significant point about the enhancements is that the single element performs many operations on the data before an output is given. Typically, the processor would be a standard microprocessor or possibly a bit-slice device - however these both suffer from the disadvantage of a sizeable instruction set and a correspondingly low throughput and I/O rates.

An ideal compromise would be some form of dedicated hardware following a constant instruction sequence, such as the multiplier-accumulator devices. It is at this point that we introduce the concept of a systolic array. Such an array consists of a set of interconnected cells, each of which performs some elemental operation. Data flows between cells in a rythmic (i.e. systolic) fashion. The array does not necessarily produce a final result from each element, as partial results usually flow between the elements.

There is no instruction/operand FETCH and we are able to use each data item several times - i.e. we are not I/O bound. The use of extensive pipelining and concurrency means that many of the simple cells will be performing computations at the same time.

Radar Signal Processing

In attempting to design a systolic array radar signal processor it is necessary to consider the design of a basic radar signal processing system. The architecture is based around a monostatic surveillance tracking radar with log video input and appropriate synchronisation data.

An analogue to digital converter (A/D) is coupled to a constant false alarm rate processor that evaluates the first threshold crossings to produce binary target data. This is fed to a second threshold process which integrates the signal over azimuth. The resulting declaration is then averaged in terms of both range and azimuth to produce a centre of plot prediction.

† Durham University, Dept. of Applied Physics & Electronics. ° Admiralty Research Establishment, Portsmouth.

CFAR Operation

The most commonly used form of CFAR detector uses a sliding window over the full range of the radar with a target being declared if the video level in a central cell exceeds the averages of the cells on either side of it. (Fig. 1).

In this example a target would be identified if :

$$X_{t-6} \geq MAX(\Sigma x_{t-}x_{t-4},\ \Sigma x_{t-8}-x_{t-12})/N^{\circ}K+M$$

where N is number of cells in sum, K,M are constants. This type of calculation may be simplified if the summation of the cells is considered as :

$$\Sigma x_{t-}x_{t-4} = \Sigma(x_{t-1}-x_{t-5}) + x_t - x_{t-5}$$

thus reducing the computation from 4 additions to 1 addition and 1 subtraction.

For a standard first threshold processor, the above calculations would be performed by either dedicated logic or using a programmable device. Neither method would normally involve pipelining at any stage hence the maximum throughput of the processor would be proportional to the sum of the times for each individual calculation.

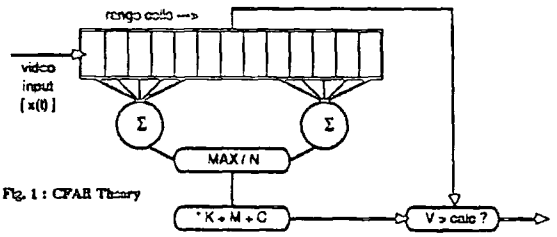


Fig. 1 : CFAR Theory

Using general MSI components, it is estimated that the time taken to evaluate a target detection is :

$$T = [\ 2 \circ T_{sum} + 2 \circ T_{mult} + 2 \circ T_{sum} + T_{cmp}\ ]$$

which gives a mean time of 120nS and hence a maximum digitisation rate of 8 MHz.

For modern radar signal processing applications a more flexible solution, combined with higher throughput is clearly required and one way to achieve this could be through the use of a pipelined architecture for the first and second threshold processors, since it is these sections which immediately affect the maximum digitisation rate.

The proposed systolic architecture makes use of high levels of pipelining between processing cells, with data flow occuring only at regular intervals as defined by the global clock. The block diagram of such a system is shown in Fig. 2. The complete first threshold section makes use of only 2 basic processor cells with a minimum of interconnection to external circuits. The ease of implementation of this circuit in VLSI form has been paramount in its design, accounting for the use of large numbers of replicated structures.

All processor cells latch input data on the first half of each clock cycle, whilst the second half of the cycle results in the following actions :

Type 0 :
Sin = Sin + Xin - Yin,
Sout = Sin,
Yout = Yin,
Xout = Xin

Type 1 :
Sin = Sin * w + z,
Sout = Sin,
Yout = Yin,
Xout = Xin

Type 2 :
Sin = Sin,
Sout = Sin,
Xout = Xin,
Yout = Yin

Type 3 :
Sin = Xin + Yin,
Sout = Sin,
Zout = Zin

It may be observed from these descriptions that types 1 & 2 are essentially equivalent as are types 0 & 3, hence the basic structure consists of only 2 distinct units.

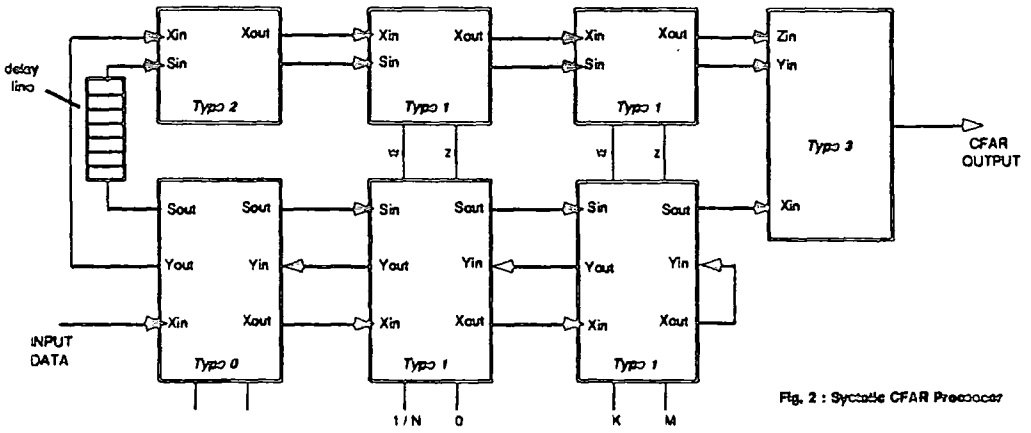


Fig. 2 : Systolic CFAR Processor

The time taken for a result to appear at the boundry of the type 3 cell is now equal to the time taken by the partial result propagating from cell 0 through the 3 intervening cells and is equivalent to a total of 5 clock cycles. In this case, however, the clock rate can be significantly higher than in a non-pipelined solution and is restricted only by the maximum processing time of any individual cell.

Typically the type 1 cell (a multiplier-accumulator) would have the longest processing time of approx. 50nS, resulting in a maximum data rate of 20MHz. The output data would be delayed for 5 range cells as against 1 range cell for the non-pipelined case, however total data throughput is higher. In addition, a dramatic improvement may be achieved by expanding the systolic block into an array, and adding other processing functions, whilst retaining the same global clock rate.

Evaluation of 2nd Threshold.

Initial research into the systolic concept resulted in techniques for evaluating the first threshold, however further work has suggested that an solution to complete plot extraction may be achieved using a larger number of basic cells and an array of the original processor blocks.

The second threshold is an azimuth-orientated process, involving integration of the first-threshold detections over sequential pulse repetitions, until a certain level (the second threshold) is reached whereupon a target is deemed to exist in terms of both range and bearing. The calculations make use of a principle described by Marcoz and Galati and involve simple integration using predefined increments, with a maximum returned value such that targets can die away if not detected on successive pulse repetitions.

Assuming a binary output x_t from the first threshold detector, arrays of count values and target declarations accumulated on range scan i : Y_i, Z_i, thresholds M, L and increment / decrement values ß, ∂ (ß > ∂) then :

if $x_t = 1$ then $Y_I = Y_{I-1} + INC$
else $Y_I = Y_{I-1} - DEC$

if $Y_I \geq M$ then $Z_I = 1$, $Y_I = MIN(Y_I, M)$

if ($Z_{I-1} = 0$) & ($Z_I = 1$) then INC = ß, DEC = ∂
else
if ($Y_I \leq L$) then $Z_I = 0$, INC = ∂, DEC = ß

$Y_I = MAX(Y_I, 0)$

This is a formal statement of the Marcoz and Galati algorithm, which may be implemented in several systolic forms, although as shown in Fig. 3 provides the simplest solution. Whilst adaption to an array of block processors with each adding to the integral from the previous azimuth sweep, seems obvious, this does not lead to the optimum solution since there is no guarantee that a target will be declared within the width of the processor window. A better solution can be achieved with each block being subject to separate second threshold processing, producing an array of 'hits' which may be fed in parallel to a systolic plot extractor (Fig. 6).

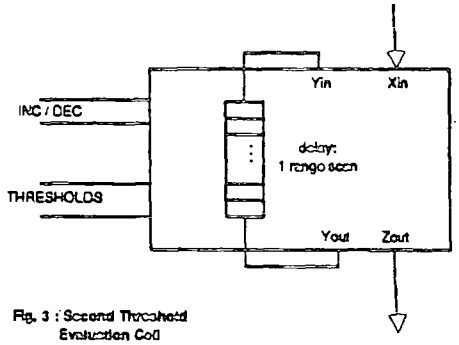


Fig. 3 : Second Threshold Evaluation Cell

The output from the second threshold circuit takes the form of a binary target declaration which may be fed directly to a plot extraction process, the purpose of which is to establish a central coordinate for the target in question, by integration over a predefined area and suitable weighting of the accumulated 'hits'. The most readily adaptable form of this algorithm involves taking moments of the plot about range and azimuth, averaging these and hence computing the absolute coordinate of the target. (Fig. 4).

Given a area size of M azimuth pulses and N range cells, for each M the corresponding range moment is equal to :

$$\Sigma R_m = \Sigma \{I=1,N\} \; I x_{I,J}$$

Similarly the azimuth moment for each range depth is :

$$\Sigma \emptyset_n = \Sigma \{J=1,M\} \; J x_{I,J}$$

The absolute centre of gravity of the plot may now be taken as :

$$\emptyset = (\Sigma \{I=1,N\} \; \emptyset_I) / \Sigma N + \emptyset_0$$
$$R = (\Sigma \{J=1,M\} \; R_I) / \Sigma N + R_0$$

where $\emptyset_0$ and R_0 are base coordinates of the plot and ΣN is the sum of active plot cells given by :

$$\Sigma N = \Sigma \{I=1,N; J=1,M\} \; x_{I,S}$$

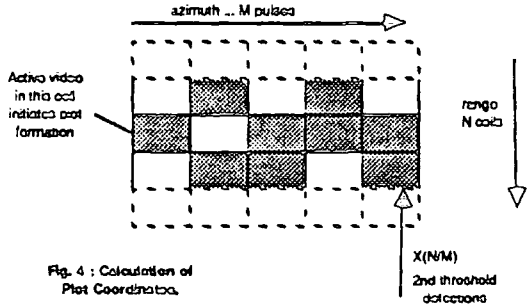


Fig. 4 : Calculation of Plot Coordinates

This extractor may be successfully implemented in systolic form, using a single processing element for each cell in the plot window. The configuration, as shown in Fig. 6 may be expanded to cover any area and only requires the appropriate range and azimuth weightings to be input to the boundry cells.

This systolic array has been subject to simulation and has produced results comparable with those achieved using a conventional plot extractor. The increase in speed in the final stage of the unit gives rise to a considerable increase in the number of plots which may be handled per sweep, and confirms the advantages of the systolic concept.

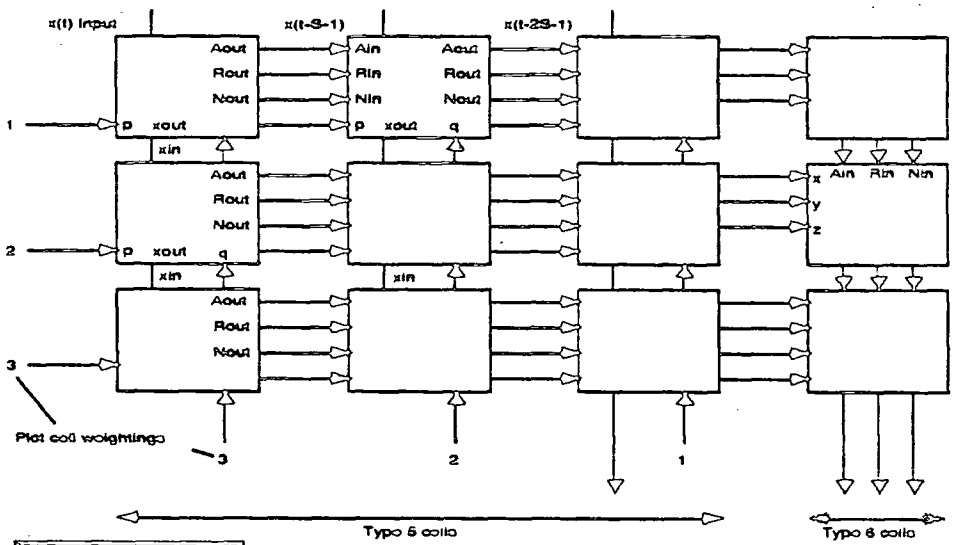


Fig. 5 : Systolic Plot Extractor

The output from the plot extractor takes the form of accumulated range and bearing moments(ΣR, $\Sigma \emptyset$) and an active cell count (ΣN). It is only necessary to divide the moments by the cell count and add the results to base coordinates to determine the plot centre. It is suggested that this function be evaluated outside the systolic array, in order to preserve the regular structure. In addition, gating is required to prevent plots being declared in very close proximity to each other. Since the whole plot extractor is clocked globally, there is no need for any reference range and bearing to be included until the final stage of calculation, as these may be determined from the fixed delays which occur within the structure.

For the plot extractor section, data is again latched on the first half of a clock cycle, whilst the following takes place on the second half of the cycle :

Type 5 :

Aout = Ain + q * Xin
Rout = Rin + p * Xin
Nout = Nin + Xin
Xout = X
X = Xin

Type 6 :

Aout = Ain + x
Rout = Rin + y
Nout = Nin + z

The extra delay in the Type 5 cells is required to synchronise the calculations in the final cells.

It is hoped that a hardware implementation of the plot extractor will be produced shortly, and further work is to be done to produce a VLSI implementation of the complete system.

Conclusion

We have shown how systolic arrays can be used to replace the boards of high speed logic normally associated with a high performance radar and can be used to implement all of the normal processing functions associated with such a system. In particular, we have presented multi-function systolic arrays that have the flexibility associated with a general purpose digital processor, but the speed associated with fixed function logic arrays.

It is concluded that systolic arrays offer the radar engineer a means of making dramatic savings in power consumption and space requirements, while maintaining the speed associated with conventional high speed logic techniques and the flexibility associated with general purpose digital processors.

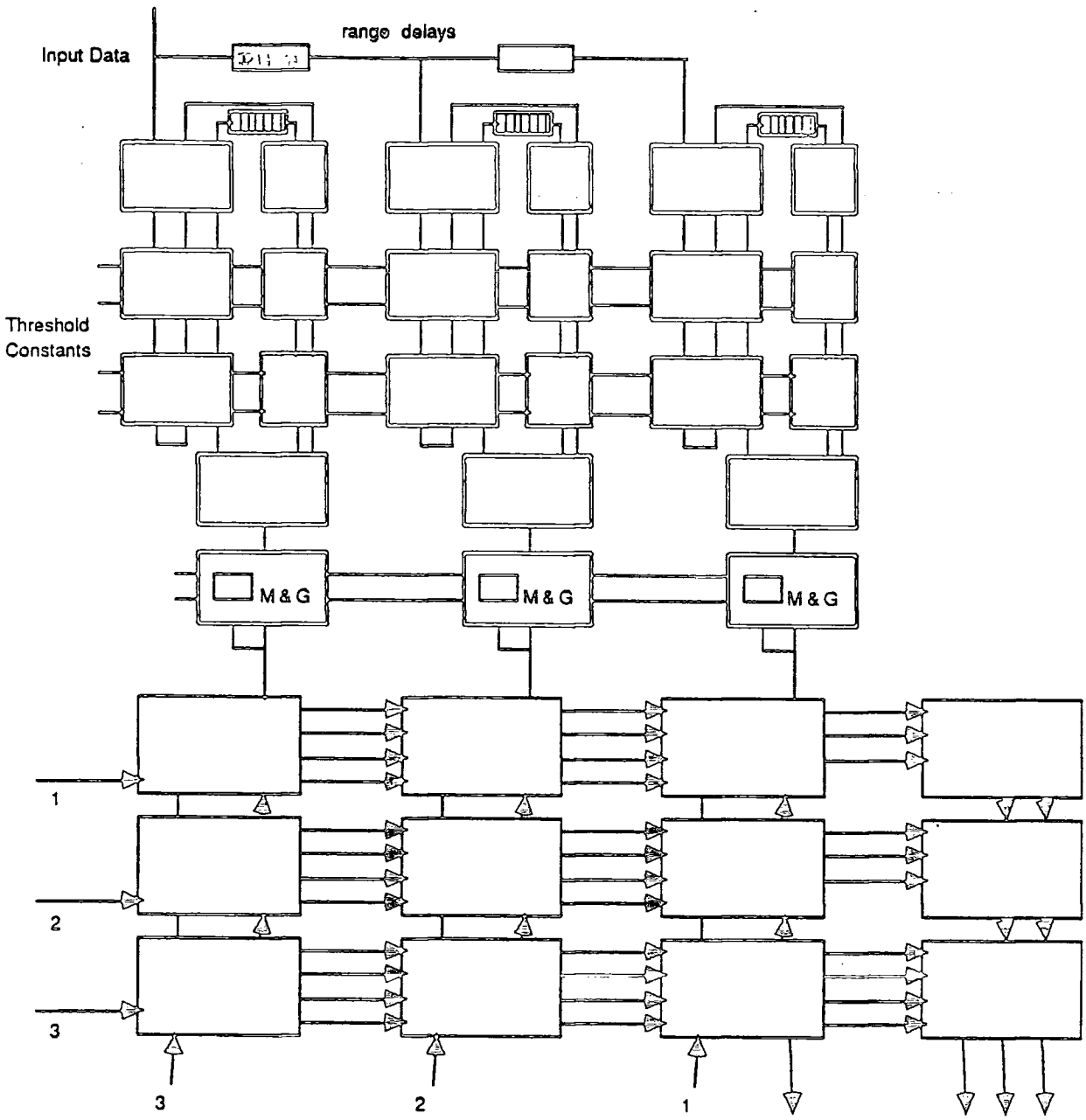


Fig. 6 : Complete Systolic Plot Extractor

For Further Reference: